



**HAL**  
open science

# Foundations of Reliable Cooperation under Asynchrony, Byzantine Faults, and Message Adversaries

Timothé Albouy

► **To cite this version:**

Timothé Albouy. Foundations of Reliable Cooperation under Asynchrony, Byzantine Faults, and Message Adversaries. Distributed, Parallel, and Cluster Computing [cs.DC]. Université de Rennes, 2024. English. NNT: . tel-04764046v1

**HAL Id: tel-04764046**

**<https://inria.hal.science/tel-04764046v1>**

Submitted on 3 Nov 2024 (v1), last revised 6 Dec 2024 (v2)

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

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



Distributed under a Creative Commons Attribution 4.0 International License

# THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601

*Mathématiques, Télécommunications, Informatique,  
Signal, Systèmes, Électronique*  
Spécialité : *Informatique*

Par

**Timothé Albouy**

## **Foundations of Reliable Cooperation under Asynchrony, Byzantine Faults, and Message Adversaries**

Fondations de la Coopération Fiable avec de l'Asynchronie,  
des Pannes Byzantines, et des Adversaires de Messages

Thèse présentée et soutenue à Rennes, le 16 décembre 2024

Unité de recherche : IRISA (UMR 6074)

### **Composition du jury :**

|                 |                         |                             |                            |
|-----------------|-------------------------|-----------------------------|----------------------------|
| Rapporteurs :   | Alessia Milani          | Professeure des universités | Université d'Aix-Marseille |
|                 | Giuliano Losa           | Ingénieur chercheur         | Stellar Foundation         |
| Examineurs :    | Sara Tucci-Piergiovanni | Cheffe de laboratoire       | CEA                        |
|                 | Cédric Tedeschi         | Professeur des universités  | Université de Rennes       |
| Dir. de thèse : | François Taïani         | Professeur des universités  | Université de Rennes       |
|                 | Davide Frey             | Chargé de recherche         | Inria Rennes               |



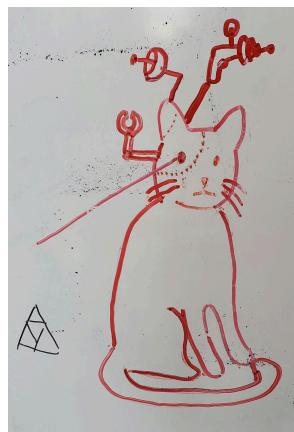
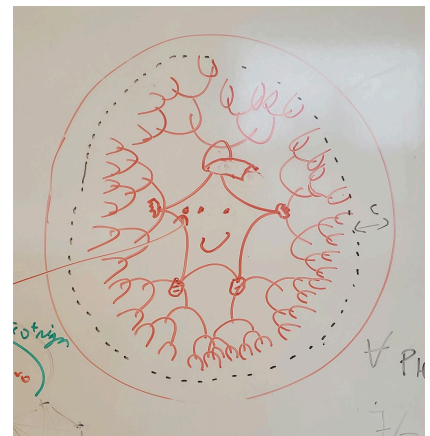
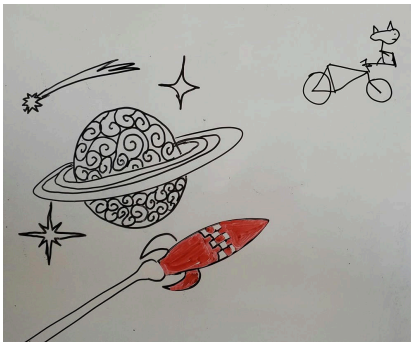
*À ma mère*



# REMERCIEMENTS

Les remerciements apparaîtront dans la version finale de la thèse.

*The acknowledgments will appear in the final version of the thesis.*





# RÉSUMÉ EN FRANÇAIS

---

Le calcul distribué (ou réparti) est la science de la coopération : il intervient dès que plusieurs agents travaillent ensemble pour atteindre un objectif commun, malgré le fait que chacun de ces agents n'a qu'une connaissance partielle de l'environnement global. À ce titre, le calcul distribué apparaît dans la vie de tous les jours, et formalise des problématiques qui existent depuis les premières civilisations, que ce soit s'assurer du secret et de l'authenticité de la communication, ou encore s'adapter à la lenteur ou la non-fiabilité de la propagation de l'information. Avec le début des micro-ordinateurs et l'avènement d'Internet dans les années 1980, les serveurs informatiques et la fibre optique ont progressivement remplacé les généraux d'armée et les émissaires royaux, mais il est intéressant de constater que les choses n'ont pas fondamentalement changé depuis l'Antiquité, elles se sont simplement accéléré !

Des messageries instantanées aux systèmes de paiement, en passant par le vote électronique et le *cloud computing*, les systèmes distribués *informatiques* sont omniprésents dans notre monde moderne interconnecté. Cependant, avec le développement de la cyber-menace et les risques de bogues logiciels et de dysfonctionnements matériels, il est aujourd'hui capital d'avoir des garanties de robustesse et de disponibilité pour nos systèmes distribués actuels. Cette thèse explore donc des outils théoriques et des techniques permettant de rendre les systèmes distribués modernes plus résilients à toutes ces sources d'incertitude.

## La modélisation de systèmes distribués

Afin de comprendre la complexité du monde réel, il est primordial de créer des simplifications de la réalité dans lesquelles il est plus facile de raisonner : c'est ce qu'on appelle des *modèles*. Le défi derrière la création d'un "bon" modèle réside dans la tension inhérente au fait de simplifier la réalité tout en y restant suffisamment fidèle, afin que le modèle fournisse un avantage déductif conséquent, mais que ses réponses restent quand même applicables aux conditions réelles.

Ainsi, plusieurs modèles facilitant les analyses formelles se sont progressivement imposés dans le domaine du calcul distribué. Considérons tout d'abord qu'un système réparti est composé de plusieurs processus (parfois appelés nœuds, agents ou encore processeurs) qui exécutent un algorithme et communiquent en échangeant des données. Dans la suite, nous explorerons deux familles de modèles permettant de représenter les échanges de données entre les processus d'un système distribué : les modèles d'interaction et les modèles de communication.

**Interaction : le passage de messages et la mémoire partagée** On distingue deux principaux paradigmes d'interaction entre les processus d'un système distribué : le passage de messages et la mémoire partagée. Le passage de messages implique l'envoi et la réception



de messages sur un réseau entre les processus, tandis que la mémoire partagée permet aux processus de lire et écrire directement dans un espace mémoire commun. Le *web* est un exemple de système à passage de messages de par son architecture client-serveur, tandis que les microprocesseurs multi-cœurs s'apparentent à des systèmes à mémoire partagée. Bien que ces deux modèles présentent de nombreuses similitudes, ce manuscrit se concentrera sur les systèmes répartis à passage de messages.

**Communication : la synchronie et l'asynchronie** En plus des modèles standards d'interaction inter-processus, le calcul réparti discerne plusieurs modèles de communication différents, qui portent sur les latences de communication. Le modèle *synchrone* est le plus fort de ces modèles : on suppose que les messages échangés entre les processus ont un délai de communication maximum, et que ce délai est connu de tout le monde dans le système. Ce modèle correspond à des conditions de communication idéales, et permet donc de résoudre de nombreux problèmes distribués. Cependant, ce modèle peut aussi s'avérer irréaliste dans des systèmes comme Internet, où les congestions, les pics de latence ou les partitionnements peuvent (temporairement) violer les hypothèses de synchronie. Pour répondre à cette problématique, l'autre modèle de communication majeur, le modèle *asynchrone*, ne fait aucune hypothèse sur le délai des messages : ces derniers peuvent rester en transit un temps arbitrairement long, du moment qu'ils sont *à terme* reçus par leur destinataire. Le modèle asynchrone étant le plus sévère, il permet donc de créer des systèmes distribués très résilients aux variations de latence du réseau. En effet, si un système fonctionne dans un environnement asynchrone, alors il peut de surcroît fonctionner dans un environnement synchrone. Pour cette raison, nous nous intéresserons exclusivement au modèle asynchrone dans cette thèse.

## Les différents modèles de panne

Après avoir examiné les différents modèles d'interaction et de communication du calcul réparti, il est essentiel de considérer un autre aspect crucial des systèmes distribués : les défaillances. Dans la section suivante, nous explorerons les différents modèles de panne qui peuvent affecter ces systèmes : les pannes relatives aux processus (participants) du système, et les pannes relatives au réseau de communication.

**Pannes de processus : les crashes et byzantins** Les pannes des processus peuvent être classées selon une hiérarchie allant des *crashes* (arrêts brutaux) aux *défaillances byzantines* [93,110], où les processus "fautifs" (dits *byzantins*) adoptent un comportement déviant arbitrairement de l'algorithme prescrit, que ce soit à cause d'une erreur d'implémentation ou d'une intention malveillante (cyberattaque). Nous pouvons observer de manière assez immédiate que le modèle byzantin est plus agressif que le modèle crash, étant donné que les byzantins peuvent "simuler" des crashes (simplement en restant silencieux), tandis qu'un processus qui

crashe ne peut pas envoyer deux messages contradictoires à deux destinataires différents afin de compromettre le système, comme pourrait le faire un byzantin.

Par convention, nous appelons *processus corrects* tous les processus non défailants (que ce soit des crashes ou des défaillances byzantines).

**Pannes réseau : l’adversaire de messages** Les pannes réseau, quant à elles, ont été formalisées par le modèle de l’*Adversaire de Messages* (en anglais, *Message Adversary*, ou *MA*), qui introduit des fautes “mobiles” pouvant affecter différents liens de communication au cours de l’exécution du système distribué [126,127]. Ces défaillances peuvent être de plusieurs type : omissions de réception, omission d’envoi, corruptions, duplication, envoi erroné... L’adversaire de messages est donc un attaquant extérieur ayant un certain pouvoir de perturbation sur les canaux de communication du système. Ce modèle théorique permet de prendre en compte des phénomènes courants dans les systèmes distribués pratiques, comme les interférences de communication, les déconnexions temporaires de processus, ou encore les réseaux à topologie dynamique.

**Pannes hybrides** Les deux modèles vus dans cette section sont irréductibles l’un à l’autre, c’est-à-dire qu’on ne peut pas émuler des pannes réseau avec des pannes de processus, et inversement. Par exemple, un adversaire de message ne pouvant que supprimer des messages ne peut pas simuler un processus byzantin, et à l’inverse, un processus byzantin ne peut pas supprimer des messages envoyés entre deux processus corrects (non-byzantins). C’est pour cette raison que des modèles de pannes hybrides, combinant pannes de processus et pannes réseau, permettent de considérer des scénarios importants que ces paradigmes ne pourraient pas couvrir individuellement. En un sens, “*le tout est plus grand que la somme des parties*”.

Pour illustrer les pannes hybrides, considérons un système de stockage distribué. Des pannes de processus peuvent survenir lorsqu’un serveur tombe en panne ou est compromis par un attaquant (devenant ainsi byzantin). Simultanément, des pannes réseau peuvent se produire en raison de congestions du réseau ou d’attaques par déni de service, entraînant des pertes de messages entre les serveurs encore opérationnels. Cette combinaison de pannes de processus et de réseau peut créer des scénarios de pannes hybrides complexes, que cette thèse vise à adresser.

## Les multiples facettes du problème de l’accord

Maintenant que nous avons établi un cadre pour comprendre les différents types de défaillances dans les systèmes distribués, nous pouvons nous pencher sur les défis fondamentaux que ces systèmes doivent relever. Dans le calcul réparti, tous les problèmes impliquent la coopération de plusieurs participants vers un but commun. Que ce soit pour concevoir un système d’élection, un service de noms de domaine ou un outil d’édition collaboratif, les algorithmes distribués cherchent à atteindre une certaine forme d’*accord* entre les participants. Cependant, comme nous allons voir dans la section suivante, l’accord est un problème protéiforme pouvant se man-

ifester de diverses façons. Dans cette section, nous mettrons l'accent sur deux des variantes les plus connues de ce problème : le *consensus* [98,116] et la *diffusion fiable* [32,93].

**Le consensus** Le *consensus* est un problème primordial dans lequel tous les processus du système distribué doivent se mettre d'accord sur une même valeur parmi celles qu'ils ont proposées. Il s'agit d'un des problèmes les plus connus de l'informatique distribuée, si ce n'est son problème le plus connu. Le consensus a de nombreuses applications industrielles, notamment dans le contexte de la réplication de machines d'état (technologie essentielle de l'informatique en nuage), les bases de données distribuées, la synchronisation d'horloges, ou encore les systèmes de *blockchain*.

Cependant, le consensus est également contraint par un théorème d'impossibilité fondamental du calcul réparti, appelé couramment théorème FLP [70], qui démontre qu'il est impossible de résoudre le consensus dans un système asynchrone en présence d'un seul crash de processus. Pour contourner cette impossibilité, il est courant que les concepteurs de systèmes distribués renforcent leur modèle en rajoutant des hypothèses (de synchronie, typiquement), ou affaiblissent le problème en implémentant un consensus probabiliste (et donc non-déterministe).

En plus de cette contrainte théorique, les implémentations pratiques du consensus font également couramment face à des problèmes de passage à l'échelle, dûs au fort niveau de synchronisation entre les processus imposé par cette puissante primitive d'accord. Mais il est intéressant de remarquer que ce haut niveau de synchronisation n'est en fait pas nécessaire pour beaucoup d'applications distribuées, comme par exemple le transfert d'argent. En effet, comme détaillé ci-dessous, pour implémenter un système de transfert d'argent, des primitives d'accord plus faibles suffisent, comme par exemple la *diffusion fiable*.

**La diffusion fiable** La *diffusion fiable* (en anglais, *reliable broadcast*, ou *RB*) est une autre primitive fondamentale qui permet à un processus émetteur de diffuser une valeur au reste du système, et ce en garantissant des propriétés de sûreté et de disponibilité bien définies, même en présence de pannes de processus. Intuitivement, la diffusion fiable est une primitive d'accord "tout ou rien" : soit tout le monde accepte la même valeur diffusée par l'émetteur (et uniquement cette valeur), soit personne n'accepte de valeur.

La diffusion fiable byzantine (en anglais, *Byzantine reliable broadcast*, ou *BRB*) est une généralisation naturelle du RB dans des contextes byzantins. En effet, le BRB garantit que tout le monde voit à terme la *même chose* (soit la même valeur, soit aucune valeur), et ce en dépit d'un émetteur initial potentiellement byzantin qui peut faire preuve de duplicité en envoyant des messages contradictoires à différentes parties du réseau.

Même si la diffusion fiable est une primitive plus faible que le consensus, dans le sens qu'elle permet de résoudre moins de problèmes distribués, elle possède néanmoins de nombreuses applications concrètes intéressantes. Par exemple, il a été démontré que le BRB était suffisant

pour concevoir des systèmes de paiement distribués tolérants aux processus byzantins malveillants [21,23,55,79]. En outre, l'utilisation du BRB à la place du consensus possède de nombreux avantages. Tout d'abord, le BRB n'est pas soumis à l'impossibilité FLP, et contrairement au consensus, il peut être réalisé dans des environnements asynchrones sujets aux pannes de processus. De plus, les implémentations du BRB sont typiquement plus légères que celles du consensus, du fait des contraintes de synchronisation plus faibles. De ce fait, de manière générale, les systèmes basés uniquement sur le BRB montent mieux en charge que les systèmes utilisant du consensus.

En raison de ses nombreux bénéfices, nous nous intéresserons particulièrement au problème de la diffusion fiable dans cette dissertation.

## **Thèse**

Ayant exploré les différentes facettes du problème de l'accord, nous sommes maintenant en mesure de formuler la thèse centrale de ce travail, qui se concentre sur la résolution efficace de la diffusion fiable dans des environnements sujets à des pannes hybrides. Dans ce manuscrit, nous nous appuyerons donc sur l'hypothèse suivante.

- **Hypothèse**

*Les modèles de pannes hybrides peuvent représenter avec précision les conditions réelles des systèmes distribués asynchrones à grande échelle.*

Dans ce contexte, la présente dissertation se concentrera principalement sur le cas de la diffusion fiable et défendra la thèse suivante.

- **Thèse**

*Certains problèmes intéressants, tels que la diffusion fiable, peuvent être efficacement résolus dans des environnements asynchrones sujets à des pannes hybrides.*

**Contributions** Pour étayer la thèse précédente, ce manuscrit présente les contributions suivantes.

1. *Un nouveau modèle de système distribué capturant les pannes hybrides.* Ce modèle permet une représentation plus fidèle des défaillances complexes dans les systèmes réels, ouvrant la voie à des algorithmes plus robustes. En particulier, étant donné un système de  $n$  processus, ce modèle combine au plus  $t$  défaillances byzantines avec un adversaire de messages qui peut supprimer  $d$  messages envoyés par n'importe quel processus durant une étape de communication sur le réseau. L'adversaire de messages est défini indépendamment de toute hypothèse de synchronie ou structure d'algorithme. Par conséquent, ce modèle de panne s'applique naturellement à n'importe quel algorithme s'exécutant dans un système asynchrone à passage de messages.

2. *Une nouvelle définition formelle de la diffusion fiable (MBRB).* En se servant du modèle hybride précédent, cette thèse définit une nouvelle abstraction, appelée “diffusion fiable byzantine tolérante aux adversaires de messages” (en anglais, *Message-Adversary Byzantine Reliable Broadcast*, ou *MBRB*), généralise la diffusion fiable byzantine classique (BRB) et permet d’aborder des scénarios de défaillance plus larges, notamment ceux intégrant des pannes mobiles pouvant causer des pertes de messages. Spécifiquement, le MBRB impose une borne inférieure explicite (notée  $\ell_{MBRB}$  et appelée “*puissance de livraison*”) sur le nombre de processus corrects qui livrent les valeurs diffusées. Cela est dû au fait que, contrairement au BRB, il n’est plus possible de garantir que la totalité des processus livrent la valeur diffusée en présence d’un adversaire de message, car ce dernier peut totalement isoler du réseau au plus  $d$  processus corrects.
3. *Un théorème d’optimalité pour le MBRB.* Nous démontrons ensuite que, dans le cadre théorique que nous avons défini, le MBRB peut être implémenté si et seulement si la condition  $n > 3t + 2d$  est respectée. Intuitivement, cette borne combine la condition  $n > 3t$ , nécessaire et suffisante pour résoudre les problèmes d’accord dans des contextes asynchrones sujets aux pannes byzantines, avec la condition  $n > 2d$ , qui empêche la présence de partitionnement réseau. Ce résultat théorique établit des limites fondamentales et guide la conception d’algorithmes optimaux.
4. *Une implémentation optimale et simple du MBRB.* En s’appuyant sur les contributions précédentes, nous proposons un nouvel algorithme, basé sur les signatures électroniques, qui démontre de manière simple la faisabilité pratique du MBRB, avec une résilience et une puissance de livraison maximales. En effet, cet algorithme ne suppose que  $n > 3t + 2d$ , démontré précédemment comme étant la borne optimale, et sa puissance de livraison est de  $\ell_{MBRB} = c - d$ , où  $c$  est le nombre de processus effectivement corrects dans le système ( $n - t \leq c \leq n$ ).
5. *L’abstraction  $k2\ell$ -cast.* Bien que l’algorithme précédent fournit une implémentation du MBRB avec une résilience et une puissance de livraison optimales, il nécessite des signatures, qui ne fonctionnent en pratique que de manière probabiliste. Afin d’explorer comment le MBRB peut être implémenté sans cette hypothèse, nous introduisons une nouvelle primitive *plusieurs-vers-plusieurs* appelée  $k2\ell$ -cast, qui permet de garantir que si une masse critique de  $k$  processus corrects diffusent une même valeur, alors au moins  $\ell$  processus corrects vont à terme livrer cette valeur. Ainsi, le  $k2\ell$ -cast capture un mécanisme de construction de quorums omniprésent dans la conception d’algorithmes distribués. Particulièrement, nous démontrons que le  $k2\ell$ -cast facilite la construction d’algorithmes MBRB sans signature, en arrivant à transformer des algorithmes classiques de BRB sans signatures, tels que celui de Bracha ou celui d’Imbs-Raynal, en les rendant non seulement tolérants aux pannes hybrides, mais aussi

plus rapides d'exécution. Cela nous permet donc d'élargir l'applicabilité de nos travaux aux systèmes sans cryptographie.

6. *Une implémentation du MBRB utilisant des codes d'effacement.* En contribution finale, nous revisitons l'implémentation du MBRB avec signatures du point de vue de la complexité de communication. Bien que notre premier algorithme MBRB avec signatures fournisse une résilience aux pannes et une puissance de livraison optimales, il présente un coût de communication élevé. En comparaison, notre nouvel algorithme amélioré, appelé *MBRB Codé*, atteint une complexité de communication quasi-optimale en utilisant des techniques de codes d'effacement, de signatures à seuil et d'engagements vectoriels. En effet, le MBRB Codé possède un coût de communication de  $O(|v| + n\lambda)$  bits envoyés par processus correct, où  $|v|$  est la taille de la valeur diffusée, et  $\lambda$  est le paramètre de sécurité des primitives cryptographiques. Cela rend le MBRB Codé optimal à  $\lambda$  près, étant donné que la borne de communication inférieure se situe à  $\Omega(|v| + n)$  bits envoyés par processus correct. En somme, le MBRB Codé est un algorithme plus efficace pour les déploiements sur les systèmes où la bande passante est une ressource précieuse, comme par exemple les systèmes blockchain ou encore les bases de données répliquées.

## Organisation

Le reste du manuscrit est organisé comme suit.

- Le Chapitre 1 introduit la dissertation, fournissant le contexte essentiel pour comprendre les enjeux de cette thèse.
- Le Chapitre 2 présente l'état de l'art, positionnant notre travail dans le paysage scientifique actuel.
- Le Chapitre 3 définit notre modèle de système hybride et le problème du MBRB, posant les bases théoriques de nos contributions.
- Le Chapitre 4 présente une implémentation simple du MBRB, démontrant sa faisabilité pratique.
- Le Chapitre 5 introduit le  $k2\ell$ -cast, élargissant la portée de nos résultats aux systèmes sans signatures.
- Le Chapitre 6 optimise notre approche, rendant le MBRB plus efficace en communication pour les déploiements à grande échelle.
- Enfin, le Chapitre 7 synthétise nos contributions et ouvre des perspectives pour de futures recherches.
- Pour des questions de présentation, certains développements apparaissent en Annexes A-D, comme des détails supplémentaires sur le problème du consensus et des preuves de correction.



# TABLE OF CONTENTS

---

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>1</b>  |
| 1.1      | Distributed computing: definition and comparison            | 1         |
| 1.2      | Algorithmics: a few basic notions                           | 4         |
| 1.3      | Models of distributed computing                             | 6         |
| 1.3.1    | Communication: synchrony, asynchrony, and partial synchrony | 6         |
| 1.3.2    | Faults: processes and network                               | 7         |
| 1.4      | The multiple flavors of agreement                           | 9         |
| 1.4.1    | Consensus: A fundamental problem...                         | 10        |
| 1.4.2    | Reliable broadcast (RB)                                     | 12        |
| 1.5      | Thesis  | 14        |
| 1.6      | Organization  | 16        |
| <b>2</b> | <b>Background</b>   | <b>17</b> |
| 2.1      | The standard system model                                   | 17        |
| 2.1.1    | Process model   | 18        |
| 2.1.2    | Communication model   | 18        |
| 2.1.3    | Cryptographic schemes                                       | 19        |
| 2.2      | Byzantine reliable broadcast (BRB)                          | 20        |
| 2.2.1    | Asynchronous signature-free BRB                             | 21        |
| 2.2.2    | Asynchronous communication-efficient asynchronous BRB       | 24        |
| 2.2.3    | Synchronous BRB   | 25        |
| 2.2.4    | Summary   | 25        |
| 2.3      | Hybrid fault models   | 25        |
| 2.3.1    | Summary   | 27        |
| 2.4      | Conclusion  | 27        |
| <b>3</b> | <b>A Theoretical Framework for Hybrid Fault Tolerance</b>   | <b>29</b> |
| 3.1      | A common computing model                                    | 30        |
| 3.2      | MA-tolerant Byzantine reliable broadcast (MBRB)             | 33        |
| 3.3      | A necessary and sufficient condition for MBRB               | 35        |
| 3.4      | Conclusion  | 38        |
| <b>4</b> | <b>A Simple Signature-Based MBRB Implementation</b>         | <b>39</b> |
| 4.1      | Preliminaries   | 40        |
| 4.2      | Algorithm   | 41        |



|          |   |            |
|----------|---|------------|
| 4.3      | Correctness proof of Algorithm 3 .....                                      | 42         |
| 4.4      | Theoretical performance analysis of Algorithm 3 .....                       | 46         |
| 4.5      | Conclusion .....  | 58         |
| <b>5</b> | <b><i>k2l</i>-cast: A Modular Abstraction for Signature-Free MBRB .....</b> | <b>59</b>  |
| 5.1      | The <i>k2l</i> -cast abstraction: Definition .....                          | 60         |
| 5.2      | Signature-free <i>k2l</i> -cast .....                                       | 62         |
| 5.2.1    | Proof intuition of Algorithm 4 .....  | 65         |
| 5.3      | <i>k2l</i> -cast in action: From classical BRB to MBRB algorithms .....     | 69         |
| 5.3.1    | Bracha’s BRB algorithm (1987) reconstructed .....                           | 70         |
| 5.3.2    | Imbs-Raynal’s BRB algorithm (2016) reconstructed .....                      | 71         |
| 5.3.3    | Numerical evaluation of the MBRB algorithms .....                           | 72         |
| 5.4      | Signature-based <i>k2l</i> -cast .....                                      | 75         |
| 5.4.1    | Algorithm .....   | 75         |
| 5.4.2    | Proof .....   | 77         |
| 5.5      | Conclusion .....  | 81         |
| <b>6</b> | <b>A MBRB Implementation with Near-Optimal Communication .....</b>          | <b>83</b>  |
| 6.1      | Overview and intuition .....  | 85         |
| 6.2      | Preliminaries .....   | 87         |
| 6.2.1    | Error correction codes (ECC) .....  | 87         |
| 6.2.2    | Cryptographic primitives .....  | 88         |
| 6.3      | The Coded MBRB algorithm .....  | 90         |
| 6.3.1    | Algorithm description .....   | 91         |
| 6.3.2    | The error-correction code in use .....                                      | 95         |
| 6.3.3    | Intuition of Coded MBRB’s Global-delivery property .....                    | 96         |
| 6.3.4    | Communication analysis of Coded MBRB .....                                  | 99         |
| 6.4      | Discussion .....  | 100        |
| 6.4.1    | Selection of <i>k</i> .....   | 101        |
| 6.4.2    | Supporting multiple instances and multiple senders .....                    | 101        |
| 6.4.3    | Using Bracha’s BRB on hash values under a message adversary .....           | 102        |
| 6.5      | Conclusion .....  | 103        |
| <b>7</b> | <b>Conclusion .....</b>   | <b>105</b> |
| 7.1      | Summary of contributions .....  | 105        |
| 7.2      | Future directions .....   | 107        |
| 7.3      | Final words .....   | 109        |
| <b>A</b> | <b>Circumventing the FLP impossibility .....</b>                            | <b>111</b> |

|          |  |            |
|----------|--|------------|
| <b>B</b> | <b>Proof of the Signature-Free <math>k2\ell</math>-cast Implementation (Algorithm 4)</b> | <b>113</b> |
| B.1      | Safety of Algorithm 4  | 113        |
| B.2      | Liveness of Algorithm 4  | 115        |
| <b>C</b> | <b>Proof of the Signature-Free MBRB Implementations</b>                                  | <b>123</b> |
| C.1      | Proof of Bracha’s reconstructed MBRB (Algorithm 5)                                       | 123        |
| C.1.1    | Instantiating the parameters of the $k2\ell$ -cast objects                               | 123        |
| C.1.2    | Proof of satisfaction of the assumptions of Algorithm 4                                  | 125        |
| C.1.3    | MBRB proof of Algorithm 5  | 130        |
| C.2      | Proof of Imbs-Raynal’s reconstructed MBRB (Algorithm 6)                                  | 135        |
| C.2.1    | Instantiating the parameters of the $k2\ell$ -cast object                                | 135        |
| C.2.2    | Proof of satisfaction of the assumptions of Algorithm 4                                  | 136        |
| C.2.3    | MBRB proof of Algorithm 6  | 140        |
| <b>D</b> | <b>Correctness Proof of Coded MBRB (Algorithms 8-9)</b>                                  | <b>145</b> |
|          | <b>Bibliography</b>  | <b>157</b> |



# INDEX OF FIGURES

---

|  |    |
|--|----|
| Figure 1: A parallel program strives to divide the problem into several independent chunks processed in parallel by multiple workers and reassembled in a unified result .....   | 3  |
| Figure 2: In consensus, all participants can propose a value, and everyone eventually decides one of the proposed values (here, the majority rule is used) .....   | 10 |
| Figure 3: From Bob’s point of view, the case where Alice sent a slow message and the case where Alice crashed are indistinguishable in an asynchronous system .....  | 11 |
| Figure 4: Reliable broadcast guarantees that everyone eventually <i>delivers</i> the same value from the sender, even in the presence of failures (on the sender or other processes) .....   | 12 |
| Figure 5: Communication between the different sets of processes of execution $E$ leading to a MBRB-No-duplicity violation (the red crosses represent the messages suppressed by the message adversary) .....                               | 37 |
| Figure 6: Distribution of signatures among processes of $A$ and $B$ two rounds after $p_i$ mbrb-broadcast $(v, sn)$ .....  | 47 |
| Figure 7: Worst-case distribution of signatures received by correct processes ( $n = 100$ ), after a <code>mbrb_broadcast()</code> execution by a correct process, at the end of rounds 2 (in orange), 3 (in green), and 4 (in blue) ..... | 50 |
| Figure 8: Maximum values of $d$ depending on $t$ for a termination in at most 2 rounds (in black), 3 rounds (in orange), 4 rounds (in green), or 5 rounds (in blue), assuming $n = 100$ and $c = n - t$ .....                              | 56 |
| Figure 9: From the system parameters to a $k2\ell$ -cast implementation .....  | 63 |
| Figure 14: Required values of $k$ for $obj_W$ in the reconstructed Imbs-Raynal BRB algorithm with varying values of $t$ and $d$ within the ranges that satisfy IR16-Assumption .....   | 73 |
| Figure 15: Distribution of distinct BUNDLE messages received by correct processes; combining Observation 36.1 and c-MBRB-Assumption shows that $ B_{\text{send}}  > k - 1$ .....   | 97 |



# INDEX OF TABLES

---

|  |    |
|--|----|
| Table 1: Acronyms and notations used in Chapter 2 .....  | 17 |
| Table 2: Acronyms and notations used in Chapter 3 .....  | 30 |
| Table 3: Acronyms and notations used in Chapter 4 .....  | 40 |
| Table 4: Acronyms and notations used in Chapter 5 .....  | 60 |
| Table 5: Bracha's original version vs. $k2\ell$ -cast-based reconstruction when $d = 0$ .....      | 71 |
| Table 6: Imbs-Raynal's original version vs. $k2\ell$ -cast-based reconstruction when $d = 0$ ..... | 72 |
| Table 7: Acronyms and notations used in Chapter 6 .....  | 84 |



# INDEX OF ALGORITHMS

---

|  |    |
|--|----|
| Algorithm 1: Multi-shot version of Bracha’s BRB algorithm ( $n > 3t$ , code for $p_i$ ) .....  | 22 |
| Algorithm 2: Multi-shot version of Imbs-Raynal’s BRB algorithm ( $n > 5t$ , code for $p_i$ ) .....   | 23 |
| Algorithm 3: A signature-based implementation of the MBRB communication abstraction ( $n > 3t + 2d$ , code for $p_i$ ) .....   | 42 |
| Algorithm 4: Signature-free $k2\ell$ -cast (code for $p_i$ ) .....   | 63 |
| Algorithm 5: Multi-shot $k2\ell$ -cast-based reconstruction of Bracha’s BRB algorithm (code for $p_i$ ) .....  | 70 |
| Algorithm 6: Multi-shot $k2\ell$ -cast-based reconstruction of Imbs-Raynal’s BRB algorithm (code for $p_i$ ) .....   | 71 |
| Algorithm 7: Signature-based $k2\ell$ -cast (code for $p_i$ ) .....  | 76 |
| Algorithm 8: Helper functions and <code>mbrb_broadcast</code> operation of the Coded MBRB algorithm (code for $p_i$ ) .....  | 91 |
| Algorithm 9: Phases of the Coded MBRB algorithm (code for $p_i$ , single-shot, single-sender, $n > 3t + 2d$ , $k \leq n - t - 2d$ , threshold for the TS scheme $\tau = \lfloor \frac{n+t}{2} \rfloor + 1$ ) ..... | 91 |





# LIST OF ACRONYMS AND NOTATIONS

| Acronyms          | Meaning   |
|-------------------|---|
| BRB               | Byzantine reliable broadcast                                |
| MA                | Message adversary   |
| MBRB              | MA-tolerant BRB   |
| General notations | Meaning   |
| $n$               | nb of processes in the system                               |
| $t$               | max nb of Byzantine processes                               |
| $d$               | power of the MA   |
| $c$               | effective nb of correct processes ( $n - t \leq c \leq n$ ) |
| $\lambda$         | security parameter of the cryptographic primitives          |
| $p_i$             | process of the system with identity $i$                     |
| $v$               | applicative value   |
| $sn$              | sequence number   |
| $m$               | implementation message                                      |
| $\ell_{MBRB}$     | min nb of correct processes that mbrb-deliver a value       |
| $\delta$          | time cost of MBRB   |
| $\mu$             | message cost of MBRB  |
| $\kappa$          | communication cost of MBRB                                  |
| $p_s$             | sender process of MBRB                                      |
| $\star$           | unspecified value   |
| $\perp$           | sentinel (null) value                                       |
| $\varepsilon$     | negligible value  |

| Notations of Chapter 5 | Meaning   |
|------------------------|---|
| $k$                    | min nb of correct processes that $k2\ell$ -cast a value                         |
| $\ell$                 | min nb of correct processes that $k2\ell$ -deliver a value                      |
| $k'$                   | min nb of correct $k2\ell$ -casts if there is a correct $k2\ell$ -delivery      |
| $\delta$               | <b>true</b> if no-duplicity is guaranteed, <b>false</b> otherwise               |
| $q_d$                  | size of the $k2\ell$ -delivery quorum   |
| $q_f$                  | size of the forwarding quorum   |
| $single$               | <b>true</b> iff only 1 value can be endorsed, <b>false</b> otherwise            |
| Notations of Chapter 6 | Meaning   |
| $k$                    | reconstruction threshold of the erasure code ( $k$ out of $n$ )                 |
| $\tilde{v}_i$          | $i^{\text{th}}$ fragment of value $v$   |
| $\Sigma$               | threshold signature (TS)  |
| $\tau$                 | threshold of the TS scheme (set to $\tau = \lfloor \frac{n+t}{2} \rfloor + 1$ ) |
| $\sigma_i$             | signature share of the TS scheme by process $p_i$                               |
| $sig_i, sigs$          | $(\sigma_i, i)$ pair, set of $sig_i$ pairs                                      |
| $C$                    | vector commitment (VC)  |
| $\pi_i$                | proof of inclusion of fragment $\tilde{v}_i$ in a VC                            |



# LIST OF ARTICLES

---

## Publications included in this manuscript

[14] Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani (2023), “Asynchronous Byzantine reliable broadcast with a message adversary.” *Theoretical Computer Science*, 17 pages, DOI: 10.1016/J.TCS.2023.114110.

► **Conference version (Invited paper) [11]:**

Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani (2021), “Byzantine-tolerant reliable broadcast in the presence of silent churn,” *Proc. 23rd Int’l Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS’21)*, 13 pages, DOI: 10.1007/978-3-030-91081-5\_2.

[12] Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani (2022), “A modular approach to construct signature-free BRB algorithms under a message adversary,” *Proc. 26th Int’l Conference on Principles of Distributed Systems (OPODIS)*, 23 pages, DOI: 10.4230/LIPIcs.OPODIS.2022.26.

► **Full arXiv version [13]:**

Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani (2022), “A modular approach to construct signature-free BRB algorithms under a message adversary,” [arXiv:2204.13388](https://arxiv.org/abs/2204.13388).

[7] Timothé Albouy, Davide Frey, Ran Gelles, Carmit Hazay, Michel Raynal, Elad Michael Schiller, François Taïani, and Vassilis Zikas (2024), “Near-optimal communication Byzantine reliable broadcast under a message adversary,” *Proc. 28th Int’l Conference on Principles of Distributed Systems (OPODIS)*.

► **Brief announcement [8]:**

Timothé Albouy, Davide Frey, Ran Gelles, Carmit Hazay, Michel Raynal, Elad Michael Schiller, François Taïani, and Vassilis Zikas (2024), “Brief announcement: Towards optimal communication Byzantine reliable broadcast under a message adversary,” *Proc. 38th Int’l Symposium on Distributed Computing (DISC’24)*, 7 pages, DOI: 10.4230/LIPIcs.DISC.2024.13.

► **Full arXiv version [9]:**

Timothé Albouy, Davide Frey, Ran Gelles, Carmit Hazay, Michel Raynal, Elad Michael Schiller, François Taïani, and Vassilis Zikas (2024), “Near-optimal communication Byzantine reliable broadcast under a message adversary,” [arXiv:2312.16253](#).

## **Publications not included in this manuscript**

- [5] Timothé Albouy, Antonio Fernández Anta, Chryssis Georgiou, Mathieu Gestin, Nicolas Nicolaou, and Junlang Wang (2024), “AMECOS: A modular event-based framework for concurrent object specification,” *Proc. 28th Int’l Conference on Principles of Distributed Systems (OPODIS)*.

► **Full arXiv version [6]:**

Timothé Albouy, Antonio Fernández Anta, Chryssis Georgiou, Mathieu Gestin, Nicolas Nicolaou, and Junlang Wang (2024), “AMECOS: A modular event-based framework for concurrent object specification,” [arXiv:2405.10057](#).

- [15] Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani (2024), “Good-case early-stopping latency of synchronous Byzantine reliable broadcast: the deterministic case,” *Distributed Computing*, 23 pages, DOI: [10.1007/s00446-024-00464-6](#).

► **Conference version [13]:**

Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani (2022), “Good-case early-stopping latency of synchronous Byzantine reliable broadcast: the deterministic case,” *Proc. 36th Int’l Symposium on Distributed Computing (DISC)*, 22 pages, DOI: [10.4230/LIPIcs.DISC.2022.4](#).

## **Other articles currently under submission**

- [10] Timothé Albouy, Davide Frey, Mathieu Gestin, Michel Raynal, and François Taïani (2024), “Context-adaptive cooperation,” [arXiv:2311.08776](#).
- [4] Timothé Albouy, Emmanuelle Anceaume, Davide Frey, Mathieu Gestin, Arthur Rauch, Michel Raynal, and François Taïani (2024), “Asynchronous BFT asset transfer: quasi-anonymous, light, and consensus-free,” [arXiv:2405.18072](#).

# INTRODUCTION

---

Je n'ai fait cette lettre plus longue que parce que je n'ai pas eu le loisir de la faire plus courte.

---

*Blaise Pascal, Les Provinciales, lettre 16, 1656*

Distributed computing is the science of cooperation: it arises as soon as multiple participants work together to achieve a common goal despite having only partial knowledge of their environment [119]. As such, distributed computing appears in everyday life, and formalizes considerations that have existed since the dawn of civilization, *e.g.*, ensuring the secrecy and authenticity of communication, or coping with the slowness and unreliability of information propagation. With the rise of micro-computers and the launch of the Internet in the 1980s, servers and optic fiber have progressively supplanted army generals and royal emissaries, but it is interesting to note that things have not intrinsically changed since ancient times, only sped up!

From messaging apps and online payments to e-voting and cloud computing, geographically distributed *computer* systems pervade our modern interconnected world. However, with the increasing prevalence of cyber threats and the lingering risk of software bugs and hardware malfunctions, the imperative for robustness and availability guarantees in distributed systems is clear. Hence, this dissertation will explore theoretical tools and techniques for making modern distributed systems more resilient against all these sources of unpredictability.

## 1.1 Distributed computing: definition and comparison

A distributed *computer* system comprises multiple *processes*<sup>1</sup> that execute an algorithm and coordinate their activities by exchanging information. The literature distinguishes two primary kinds of communication mediums: *message passing* and *shared memory*. These two models differ fundamentally in how they enable communication among the processes.

---

1. Many different terms are commonly used to refer to the participants of a distributed system (agents, nodes, peers, sensors, *etc.*). For consistency, we will stick to *processes* in this manuscript.

- In message-passing systems, processes communicate by explicitly sending and receiving messages through a network, that could be implemented by protocols such as TCP/IP. Perhaps the most important and famous distributed computing problem is *consensus*, which was historically introduced in the message-passing model: all processes must reach an agreement on one of the values they proposed [98]. Message-passing systems are also studied from the standpoint of networking or cryptography, under the term *multi-party computation (MPC)*.
- In shared-memory systems, processes can directly read from and write to a common memory space, as if they were accessing their local memory. One of the oldest and most important problems relating to this model is *mutual exclusion* (or *mutex*) [60], *i.e.*, making sure that no two different processes can concomitantly access a shared resource, whether physical (*e.g.*, an office printer) or logical (*e.g.*, the critical section of a program). In operating systems research, the case of shared memory is also studied under the angles of *scheduling* or *cache coherence*.

Although multiple works show the many similarities that exist between these two communication models (it is sometimes believed that consensus and mutual exclusion are two faces of the same coin [120]<sup>2</sup>), the present manuscript will mainly focus on message-passing distributed systems. In the sequel, we underline the differences between distributed computing and other models of computation.

**Sequential, parallel and distributed computing** Because distributed computing typically involves independent sequential processes executing concurrently, it is closely related to both sequential and parallel computing, yet should not be confused with either of these two major fields.

*Sequential computing* refers to the traditional model of computation, where a single processing unit (such as a CPU core) executes instructions one after the other in a sequence, following the order defined by the program. Sequential computing has been the dominant model for decades, mainly thanks to the “free lunch” allowed by Moore’s law [103], which predicted a rapid increase in available computing power through transistor miniaturization, and enabled faster programs without optimization.

However, at the beginning of the century, the increasing difficulty of maintaining the growth of microprocessors’ clock speeds led manufacturers to progressively switch to multicore chips [37,109]. Harnessing this new trend, *parallel computing* rose to prominence as a way to execute programs efficiently, by distributing the workload on several computing units. As a simplified example, in a parallel program, the input data can be divided into distinct chunks,

---

2. Let us note that consensus and mutex were historically linked to specific interaction models (message passing and shared memory, respectively), but their formal definitions are, in fact, oblivious to the underlying communication paradigm.

that can then be independently processed by multiple workers (see Figure 1). Parallel programs involve important synchronization issues (*e.g.*, threads, locks, or barriers), making them notoriously harder to write than sequential programs. However, with respect to computability power, parallel computing is an equivalent of sequential computing primarily interested in efficiency. Indeed, given enough time, every problem that can be solved in parallel can also be solved sequentially.

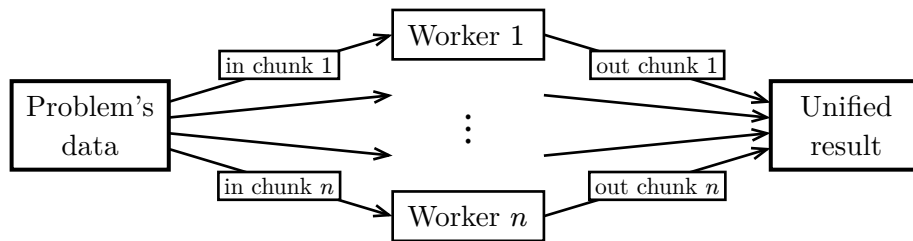


Figure 1: A parallel program strives to divide the problem into several independent chunks processed in parallel by multiple workers and reassembled in a unified result

*Distributed computing* studies issues similar to parallel (and sequential) computing, but from a different angle. Unlike parallel computing, where the environment is typically under the programmer's control (no failures), in distributed computing, the (physically) distributed nature of the system is imposed on the system designer [119]. The challenge in distributed systems lies in mastering the uncertainties created by this environment: concurrency (*i.e.*, the actions of other processes in the system), information speed (*e.g.*, message delays, memory latency, *etc.*), or failures (process- or network-related). Interestingly, one can observe that these uncertainties all stem from the impossibility of instantaneous communication.

**Computability power and impossible problems** Sequential and distributed computing also differ fundamentally in terms of *computability power*, *i.e.*, the classes of problems that are solvable in these two models.<sup>3</sup>

Indeed, our current understanding of the computability limits of sequential machines<sup>4</sup> are determined by the Church-Turing thesis, which conjectures that everything that can be mechanically computed can be done so by a Turing machine. However, in 1936, A.M. Turing famously demonstrated in his paper “*On computable numbers, with an application to the Entscheidungs-*

---

3. Computability power is not to be confused with *computing power*, which refers to how fast some machine can make calculations, and which is typically measured in *floating-point operations per second* (or *FLOPS*).

4. We consider the sequential machines of the Chomsky-Schützenberger hierarchy of automata [48–50]: Finite state automata  $\subset$  Pushdown automata  $\subset$  Turing machines. Even though this hierarchy was initially presented as a classification of formal grammars, it also applies to sequential computation in the context of symbol manipulation.



*sproblem*” that some problems cannot be solved by Turing machines [135].<sup>5</sup> This impossibility result<sup>6</sup> lies at the foundation of modern informatics and is sometimes regarded as the “birth certificate” of the discipline [132].

Similar fundamental impossibility results also exist in distributed computing [19], such as the CAP [75] or FLP [70] theorems (see Section 1.4.1), but as stated by Herlihy, Rajsbaum, and Raynal [83], “*these limits to computability reflect the difficulty of making decisions in the face of ambiguity, and have little to do with the inherent computational power of individual participants.*” Indeed, even if all processes of the distributed system were stronger than a Turing machine (which is inconceivable according to the Church-Turing thesis), these problems would remain unsolvable.

## 1.2 Algorithmics: a few basic notions

Before proceeding further into this dissertation, it is essential to explain a few notions relating to (distributed) algorithmics, namely the concepts of *specification*, *algorithm* and *proof*, and the distinction between *application* and *implementation*.

**Specification, algorithm, and proof** As we show next, answering questions about the computability of distributed problems requires systematic reasoning centered on specifications, algorithms, and proofs.

All theoretical problems considered in algorithmics (such as graph coloring or array sorting) are formally defined using *specifications* (or *abstractions*), *i.e.*, structured descriptions of the problem’s properties on its inputs and outputs. Formal specifications are central to *declarative programming*, one of the two main programming paradigms, which focuses on describing the problem to solve, *i.e.*, *what* we want (a.k.a. the black box approach), rather than giving precise instructions on *how* to solve it (a.k.a. the white box approach). These specifications are often given as a list of *preconditions* (*i.e.*, requirements on the problem’s input) and *postconditions* (*i.e.*, guarantees on the problem’s outputs). Let us consider the example of an operation “`sort_array(unsorted_array) → sorted_array`” that takes a (potentially) unsorted number array as input and outputs the associated sorted array. The precondition (requirement) is that *unsorted\_array* is a well-formed array of numbers (which can be verified at the type level), while the postcondition (guarantee) is that *sorted\_array* is indeed a sorted number array, *i.e.*, every number in the array is greater or equal to each of its predecessors.

---

5. Although Alonzo Church wrote a similar proof of undecidability a few months before [51,52], Alan Turing’s is the most well-known today due to the elegance and simplicity of his Turing machine model (compared to Church’s  $\lambda$ -calculus).

6. Turing’s proof follows from the unsolvability of some decision problems, one of which is the famous *halting problem*.

Specifications are then implemented using *algorithms*, which are sequences of instructions describing the steps to *solve* (or *implement*) the problem. An algorithm can be seen as an instance of *imperative programming*, the other main programming paradigm, which focuses on describing *how* the program should operate to solve the problem, by providing its exact instructions and control flow. For array sorting, these instructions may involve swapping two array numbers, for instance. Multiple algorithms can implement a given specification: for example, merge sort and bubble sort are two popular algorithms that implement the previous `sort_array()` specification, although the former is more efficient than the latter in terms execution time. Using complexity theory, different implementations of the same problem may be formally compared in terms of performance or cost: *time complexity* characterizes the algorithm’s execution duration (typically, its asymptotic number of executed instructions), while *space complexity* characterizes its storage usage (*i.e.*, the asymptotic number of bits it needs to store).

Finally, a *proof* uses formal reasoning to show that a given algorithm implements a given specification, in which case the algorithm is said to be *correct*. Correctness is an essential feature of dependable or critical systems. As stated by Maurice Herlihy: “*Correctness may be theoretical, but incorrectness has a practical impact.*”

The same pipeline also applies to distributed algorithms: distributed problems are first specified, then implementations are proposed, and finally, the implementations are proven correct. In contrast to sequential algorithms, distributed algorithms are executed simultaneously by all system processes, and use additional instructions for inter-process communication, such as `send()/receive()` for exchanging messages or `read()/write()` on a shared memory. Moreover, specifications in distributed computing (and more generally in system design) tend to distinguish two kinds of guarantees: *safety* properties (*i.e.*, “nothing bad ever happens”) and *liveness* properties (*i.e.*, “something good eventually happens”). For instance, in an automated train system, safety ensures that train wrecks cannot happen (derailment, collision, ...), while liveness guarantees that the train eventually reaches its destination on time. Although safety is always a priority, liveness is particularly important, as it precludes trivial implementations (*e.g.*, a train that does not move has no accident) and captures the “dynamic” aspect of informatics that mathematics lacks.

**Implementation and application, messages and values** When considering an abstraction (*i.e.*, formal specification), one must distinguish between the *implementation* and the *applicative* levels. As outlined previously, the implementation refers to the lower-level algorithm that satisfies the abstraction. In contrast, the application refers to the upper-level client of the abstraction, which could be the system’s end user, or another algorithm that leverages the abstraction to construct more complex behaviors. In this context, the specification acts as the interface between the implementation and the application. The interplay between implementation and application is instrumental in building modular complex systems.

From a terminology point of view, we also distinguish two complementary notions: *messages* and *values*. In a message-passing distributed system, the word *message* refers to a message sent by an algorithm on the network level to implement an abstraction. Messages are *sent* and *received*. On the other hand, the word *value* refers to a payload that a client (application) seeks to disseminate via an abstraction. In message-passing algorithms, values are typically contained in messages, but these two notions should not be conflated. In short, messages relate to the implementation, while values relate to the application.

The terms denoting the dissemination and acceptance of some values at the application level depend on the abstraction. For instance, in consensus, we say that values are *proposed* and *accepted* (see Section 1.4.1), while in reliable broadcast, we say that they are *rb-broadcast* and *rb-delivered* (see Section 1.4.2).

### 1.3 Models of distributed computing

Having established these fundamental concepts of algorithmics, we can now explore how they are applied in the modeling of distributed systems. Like many other scientific disciplines (pure vs. applied math, theoretical vs. experimental physics, *etc.*), research in distributed computing spans a spectrum going from theory to practice. As researchers, we are always someone's theoretician and someone else's practitioner. One of the theoreticians' goals is to design formal models, *i.e.*, simplifications of reality that are easy to reason about. Practitioners can then use these theoretical frameworks to build trusted solutions, applications, and systems. This section presents the *de facto* standard models of distributed computing regarding communication (a)-synchrony and fault tolerance.

#### 1.3.1 Communication: synchrony, asynchrony, and partial synchrony

In message-passing distributed computing, it is assumed that an adversary can control (to some extent) the delay of messages transiting in the network, thus introducing some unpredictability. Hence, the subsequent communication models define the limits of the adversary's power to delay messages.

In the *synchronous* model, all messages have a maximum delay  $\Delta$  known by all processes. Thus, the adversary cannot delay a message for more than  $\Delta$  time units. The synchronous model can also be represented by round-based communication: all processes go at the same pace and exchange messages during communication rounds of duration  $\Delta$ , and all messages sent during a round are received in the same round.

By contrast, in the *asynchronous* model, messages can have any arbitrary (but finite) delay. Hence, there is no upper bound  $\Delta$  on the maximum delay, but the adversary must eventually let messages be received.<sup>7</sup>

The *partially synchronous* model was first introduced by Dwork and Lynch in [68] as a middle ground between the two previous models. It originates from the observation that existing large-scale distributed systems (such as the Internet) are not completely synchronous or asynchronous. Indeed, these systems are usually synchronous (network latencies are stable), but they can sometimes experience phases of asynchrony (where latency spikes can appear) due to congestion or denial-of-service attacks, for instance. Partially synchronous algorithms are typically designed to be always *safe* (even when synchrony assumptions are temporarily violated), but they may stop progressing during asynchronous periods. However, their termination is guaranteed as soon as the network becomes synchronous again. Two models have been proposed to represent partial synchrony: in the first one, there is a *global synchronization time* (or *GST* for short), unknown to processes, before which the system is asynchronous and after which it is synchronous; and in the second one, there is a maximum delay of messages  $\Delta$ , but that is not known by processes [43].

This manuscript focuses on the asynchronous model. This model is weaker than its (partially) synchronous counterparts: as no assumption is made on network delays, fewer problems are solvable under asynchrony (see Section 1.4.1). However, asynchronous algorithms have the advantage of being more robust: they keep their guarantees no matter the message delays and tolerate (partially) synchronous environments, whereas (partially) synchronous algorithms may break under asynchrony. Additionally, asynchronous algorithms do not need to wait for the end of communication rounds to advance; they progress as soon as their conditions on received messages are satisfied, making them more reactive and, therefore, faster in practice than (partially) synchronous algorithms.

### 1.3.2 Faults: processes and network

A distributed system can be subject to faults that impact its processes or its communication medium (*i.e.*, network in our case). Faults usually correspond to involuntary defects in the software or hardware, but they can also be caused by a malicious adversary trying to jeopardize the system (in the case of a cyberattack). As we shall see next, these two types of failures are irreducible to one another, in the sense that one cannot simulate network faults using process faults, and *vice versa*.

---

7. As the duration of local computations by processes is often negligible compared to message delays, one typically considers that the former is “absorbed” in the latter.

**Process faults** Models for process faults can be classified in the following hierarchy, sorted by increasing expressive power: Crash faults  $\subset$  Omission faults  $\subset$  Byzantine faults. The non-faulty processes of the system that obediently follow the algorithm are called *correct* processes.

*Crash faults* (also called fail-stop) are the least expressive model of the hierarchy, in the sense that they only represent the specific case where a process that previously followed the algorithm suddenly becomes silent (*i.e.*, stops communicating) after a specific instant (the crash).

*Omissions* are meant to generalize crashes. As with crashes, a process  $p$  subject to omissions follows the algorithm. But, unbeknownst to  $p$ , an adversary may block some (or all) of the messages that  $p$  sends (in the case of send omissions) or that are intended to  $p$  (in the case of receive omissions). We can observe that omissions can trivially emulate the crash of a process if the adversary removes all its incoming and outgoing messages. However, compared to crashes, omissions can also represent other (more favorable) settings where only a subset of these messages are deleted.

Finally, *Byzantine faults* are the most expressive model for process faults, as they can consider every possible behavior from faulty processes [93,110]. Indeed, a Byzantine process may arbitrarily deviate from the algorithm and collude with other Byzantine processes to fool correct processes. As the most general model for process failures, Byzantine fault tolerance (BFT) is often regarded as the golden standard for building robust distributed systems.

Although the BFT model is one of the most aggressive, some attack vectors cannot be represented only by Byzantine processes. For example, for several decades, theoretical research in distributed computing revolved mainly around *static* (or *permissioned/closed*) systems, where the set of participating processes never changes, and new identities cannot be created or acquired. But in the early 2000s, the advent of large-scale peer-to-peer services (*e.g.*, Napster, BitTorrent [53,100], Tor [61], *etc.*) brought interest to *dynamic* (or *permissionless/open*) systems, where processes can join and leave the network (or simply disconnect/reconnect), a phenomenon known as *churn*. In particular, open systems must cope with Sybil attacks, in which an adversary tries to compromise the network by flooding it with fake identities [66]. Sybil resistance constitutes an even harder challenge for distributed systems designers, as the adversary thresholds often assumed in closed settings (typically, less than one-third of processes are Byzantine) cannot continue to hold in open settings.

**Network faults** In addition to process faults, a practical distributed system may also experience failures related to its communication medium. Indeed, messages transiting on the network can be subject to deletions (omissions), corruptions, or spurious creations due to interference or lossy channels, for instance.

This family of network failures has been formalized by the *Message Adversary (MA)* model, first introduced by Santoro and Widmayer in [126]. Unlike Byzantine or omission faults, which

are *static*, *i.e.*, they are pinned to specific processes, an MA removes this constraint and introduces *mobile* faults, which may target different communication links during the execution. In particular, these link failures may happen between two correct processes, a scenario ignored in the standard Byzantine model (unless one assumes all processes to be potentially Byzantine, which considerably limits what can be achieved in practice). Hence, the MA model can cover significant phenomena, such as correct processes’ disconnections, dynamicity, and churn.

**Hybrid fault models** As we have just seen, process and network failures are orthogonal fault models, in the sense that they cannot emulate each other. For example, a message adversary suppressing messages cannot simulate a Byzantine process; conversely, a Byzantine process cannot delete messages exchanged between two correct processes.

Remarkably, the seminal papers that introduced these two models both received the Dijkstra Prize, the most prestigious award in distributed computing<sup>8</sup>: Pease, Shostak, and Lamport’s 1980 paper “*Reaching agreement in the presence of faults*” [110], which introduced Byzantine failures, received the 2005 citation, and Santoro and Widmayer’s 1989 paper “*Time is not a healer*” [126], which introduced message adversaries, received the 2024 citation.

Combining the two types of faults into one fault model allows for considering significant scenarios these paradigms could not cover individually: the whole is greater than the sum of the parts. For instance, hybrid failures can represent systems prone to both malicious processes and recoverable message omissions for correct processes (*e.g.*, due to transient disconnections or process mobility).

To illustrate hybrid faults, consider a distributed storage system. Process faults can occur when a server crashes or is compromised by an attacker (becoming Byzantine). Simultaneously, network faults can occur due to network congestion or denial-of-service attacks, resulting in message losses between still-operational servers. This combination of process and network faults yields complex hybrid fault scenarios that this thesis aims to address.

## 1.4 The multiple flavors of agreement

Having established the fundamental models and concepts of distributed computing, we can turn our attention to the central challenges that distributed systems must address. In distributed computing, all problems involve the cooperation of multiple participants towards a common goal. Whether for building a system for leader election, a naming service, or a collaborative editing tool, distributed algorithms typically seek to achieve some form of *agreement*. However, as we shall see, the agreement problem can manifest itself in many different forms. The following section will focus on two key variants of this problem: *consensus* and *reliable broadcast*.

---

8. In distributed computing, the Dijkstra Prize is only surpassed in prestige by a few awards covering all disciplines in informatics, the most notorious being the Turing Prize (considered the “Nobel of informatics”).

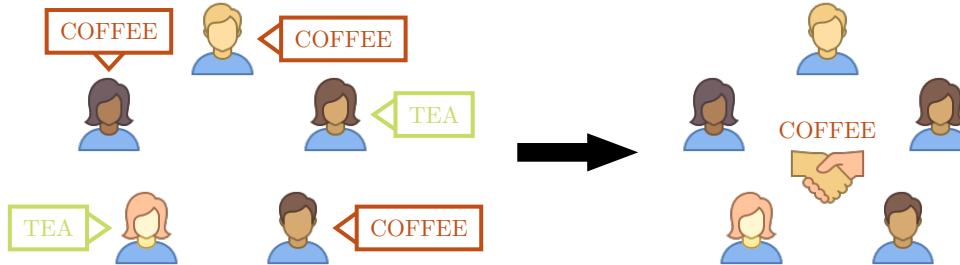


Figure 2: In consensus, all participants can propose a value, and everyone eventually decides one of the proposed values (here, the majority rule is used)

#### 1.4.1 Consensus: A fundamental problem...

Consensus is a fundamental abstraction of distributed computing with a simple premise: all participants of a distributed system must propose a value, and all participants must eventually agree on the same value among the ones that have been proposed [98] (see Figure 2).

The industrial applications of consensus are manifold: cloud computing, distributed databases, satellite navigation systems, clock synchronization, blockchain systems, *etc.* In particular, consensus is critical in the context of *state-machine replication (SMR)*, that is, making a system of multiple processes that can individually fail appear to external observers (clients) as one single entity that is never subject to failures. The importance of consensus is further exemplified by the fact that it is a *universal problem* [84]: roughly speaking, consensus can be used to solve any other distributed problem (specified as a concurrent object with a sequential specification) [82].

**...with a fundamental impossibility** It has been famously proven in [70] that asynchronous consensus cannot be solved even with one process crash. This impossibility result, colloquially known as the *FLP theorem* (for the initials of its authors: Fischer, Lynch, Patterson), is fundamental: it assumes a very general communication medium (asynchrony) while considering a very weak fault model (only one crash).

Intuitively, this impossibility arises from the fact that a process of the system (Bob) may think that some other process (Alice) sent a message on the network that influenced the decisions of others. However, Bob cannot distinguish between the case where Alice's message to Bob is delayed due to asynchrony and the case where Alice crashed (see Figure 3). Indeed, in asynchrony, it is impossible to detect crashes.<sup>9</sup>

9. Let us notice that a crashed process can be detected under synchrony by simply sending a message to it and waiting for a response for 2 times the maximum delay.

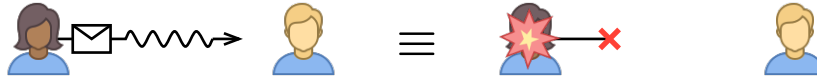


Figure 3: From Bob’s point of view, the case where Alice sent a slow message and the case where Alice crashed are indistinguishable in an asynchronous system

The implications of this impossibility are profound: even if we consider a very large network of billions of computers, a single crash can prevent consensus. The FLP theorem can be seen as the formalization of Lamport’s famous quote, which humorously describes the nature of distributed computing: “A *distributed [computer] system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable*” [95].

Other impossibility proofs of *asynchronous resilient consensus* were later presented. The initial FLP proof was circumscribed to the message-passing model, but Loui and Abu-Amara later showed that this impossibility also holds in shared memory [97]. Some other proofs are based on combinatorial topology (such as in the asynchronous computability theorem by Herlihy and Shavit [81]), and others follow an axiomatic approach [5,133]: the notion of asynchronous resilient consensus is defined as a system of axioms which is then proven inconsistent, *i.e.*, it contains a contradiction. Constructive proofs follow another interesting approach [73,136]: they explicitly describe how a non-terminating execution of consensus can be constructed.

**Circumventing the impossibility** Multiple solutions circumvent the FLP impossibility by enriching the underlying model with additional assumptions or relaxing the consensus guarantees. Some of these solutions require adding partial synchrony assumptions [38,96], while others rely on randomization [25,104]. A detailed review of some of the most notable solutions to avoid FLP is presented in Appendix A.

**The other penalty of consensus: performance** Distributed systems relying on consensus typically suffer substantial performance overheads due to strong synchronization costs between the processes. For example, most decentralized cryptocurrencies (*e.g.*, Bitcoin [106]) use consensus<sup>10</sup> to implement a blockchain, a replicated append-only database storing all system transactions in a chain of blocks. Blocks containing new transactions are regularly added to the chain, thus forming a *total order* of transactions: all participants process all transactions sequentially and in the same order (block by block). In practice, achieving total order in a decentralized and open system (such as a cryptocurrency) is particularly costly. For example, Bitcoin’s throughput is capped at a dozen transactions per second *worldwide* [46], while mainstream (centralized) payment processing networks, such as Visa or Mastercard, can handle several thousand trans-

10. In blockchains, consensus is typically implemented with probabilistic safety guarantees (*i.e.*, safety properties might be temporarily violated in unfavorable cases, which is sometimes referred to as *Nakamoto consensus*), contrary to the traditional approach for implementing consensus in closed (or permissioned) systems which provides probabilistic liveness but ensures deterministic safety.



actions per second. This technical difficulty has been captured in the *Blockchain Trilemma* [90], which conjectures that there is a necessary trade-off between three critical aspects of blockchain technology: *Security* (resilient against attack), *Decentralization* (open, no central authority), and *Scalability* (high throughput, low latency and operational costs).

However, contrary to common belief, total order and strong agreement (such as consensus) are unnecessary in many applications, particularly money-transfer/cryptocurrency systems. Indeed, to tackle the performance issue of consensus/blockchain-based cryptocurrencies, weaker abstractions can be exploited, such as *reliable broadcast* [21,23,55,79].

#### 1.4.2 Reliable broadcast (RB)

Introduced in the mid-eighties, *reliable broadcast (RB)* is another fundamental communication abstraction that lies at the center of many fault-tolerant distributed systems. Formally defined in the synchronous setting by Lamport, Shostak, and Pease in 1982 (under the name *Byzantine generals problem*) [93], and then in the asynchronous setting by Bracha and Toueg in 1985 [31,32], RB allows each process to broadcast values with well-defined properties in the presence of process failures. In turn, these properties make it possible to design provably correct distributed software for upper-layer applications, such as distributed file systems, event notification, or replication. Notably, reliable broadcast plays a crucial role in money and asset transfer systems, as discussed below.

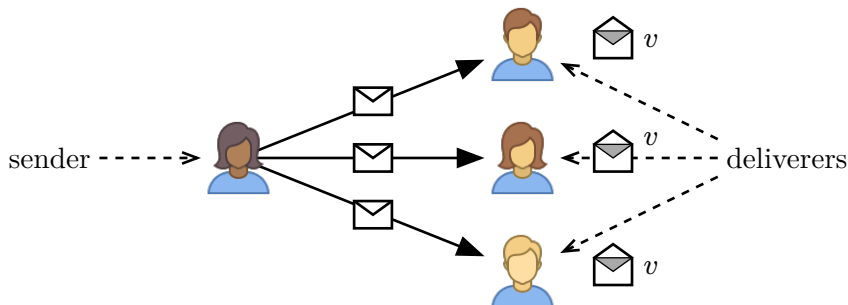


Figure 4: Reliable broadcast guarantees that everyone eventually *delivers* the same value from the sender, even in the presence of failures (on the sender or other processes)

Like other members of the broadcast family, reliable broadcast involves a leader (the sender) who disseminates some value to the entire network (see Figure 4). The multi-sender generalization of reliable broadcast allows all processes to be senders, but each broadcast instance by some process runs “in isolation” (it does not have side effects with broadcasts from other processes). Informally, reliable broadcast guarantees that the correct processes *deliver* (*i.e.*, accept) the same set of values, which includes at least all the values they broadcast.<sup>11</sup> This set may also

11. Additionally, unlike other primitives such as CRDT [72,108,129] or gossip [69,89], reliable broadcast guarantees that at most one value is delivered from each sender per broadcast instance.

contain values broadcast by faulty processes. The fundamental property of reliable broadcasting is that no two correct processes deliver different sets of values [35,116], despite the potential failure of the sender or other processes in the system (hence *reliable*). In RB, values are *rb-broadcast* and *rb-delivered*.

**Byzantine reliable broadcast (BRB)** Designing a reliable broadcast algorithm that tolerates any number of crashes is quite simple: the sender sends her value to everyone, and every receiver forwards this value to everyone (in case the sender crashed in the middle of the execution). However, by contrast, implementing reliable broadcast in the presence of Byzantine failures is far from trivial, because Byzantine processes may, for instance, dissemble and send or forward contradicting values to different recipients. Such an algorithm is called *Byzantine reliable broadcast (BRB)*, and we say that a process *brb-broadcasts* and *brb-delivers* values. The most famous BRB algorithm is due to Bracha [32] (1987), and assumes that the system comprises at least two times more correct processes than Byzantine ones (which is optimal in terms of Byzantine fault tolerance [116]).

**Comparison with consensus** As we can observe, Byzantine reliable broadcast (BRB) is a *one-to-many* primitive (one sender brb-broadcasts, everyone brb-delivers), while consensus is a *many-to-many* primitive (everyone proposes, everyone decides). As explained previously, BRB can be naturally generalized to a multi-sender setting. However, unlike in consensus, two values brb-broadcast by two distinct processes are not in competition with each other: each BRB instance runs “in isolation,” whereas in consensus, the decision of some value prevents the decision of values proposed by other processes. This sole difference allows BRB to be implemented in asynchronous and fault-prone environments [32]. In contrast, consensus cannot be achieved in the same setting due to the FLP impossibility (see Section 1.4.1).

On the flip side, BRB is weaker than consensus: all problems solved by BRB can also be solved by consensus, but the reverse is not true. For instance, consensus can solve the leader election problem, but BRB cannot. However, BRB still has many interesting and useful applications, such as *money transfer*.

**Example of application: money transfer** As hinted at at the end of Section 1.4.1, consensus or, equivalently, total order of transactions are unnecessary when constructing decentralized electronic payment systems. Intuitively, we can understand this fact by observing that real-world bookkeeping never attempted to record all the world’s transactions, let alone order them: two unrelated transactions do not need to be ordered relative to each other. The fact that consensus is not necessary to implement money transfer was stated one of the first times in [80], and then proven in [79]. More formally, the only condition required to prevent double-spending in a payment system is for the processes to agree on the order in which transactions are issued from each individual account: if some account issues two conflicting transactions spending the

same funds, the rest of the network, and especially the corresponding creditors, have to settle on which transaction (if any) is correct. This relaxed, per-account transfer ordering can be obtained by communication primitives weaker than consensus, such as BRB.<sup>12</sup>

Following the seminal result of [79], multiple concomitant papers have presented money transfer systems based on BRB [21,23,55]. The first paper [21] is more theoretical: it presents the first concurrent specification of money transfer, an algorithm, and its correctness proof. Surprisingly, it also shows that money transfer is a weaker problem than implementing read/write registers. The other two papers introduced *Astro* [55] and *FastPay* [23], two practical implementations of money transfer, along with performance benchmarks. These two broadcast-based systems claim Visa-level scalability regarding transaction throughput and latency. *Astro* was later extended to open environments into the *Pastro* (permissionless *Astro*) system [92], and *FastPay* was also extended to provide confidential and untraceable transfers into the *Zef* system [24].

This line of work demonstrates that BRB-based systems can provide correctness, resilience, scalability, and privacy guarantees on par with (and sometimes greater than) their consensus-based counterparts.

## 1.5 Thesis

Having introduced the different facets of the agreement problem, we are now able to formulate the central thesis of this work, which focuses on the efficient resolution of reliable broadcast in environments subject to hybrid failures. In this manuscript, we will therefore rely on the following assumption:

- **Assumption**

*Hybrid fault models that combine Byzantine faults with a network-level message adversary, can accurately represent real-world conditions of large-scale asynchronous distributed systems.*

In this setting, the present dissertation mainly focuses on the case of reliable broadcast. Indeed, while significant progress has been made in understanding and implementing reliable broadcast in various contexts, there remain open questions and opportunities for improvement, particularly in systems subject to hybrid failures. This work aims to address these gaps and advance the state of the art in reliable broadcast implementations. Therefore, this dissertation will defend the following thesis.

- **Thesis**

*Interesting problems, such as reliable broadcast, can be efficiently implemented in asynchronous environments prone to hybrid process-and-network failures.*

---

12. This is only the case for money transfer systems where each account is owned by a single process. In systems where accounts can have multiple owners, consensus can be required between the processes owning a given account.

**Contributions** To support this thesis, the dissertation presents the following contributions.

1. *A new distributed system model capturing hybrid failures.* This model allows for a more faithful representation of complex failures in real systems, paving the way for more robust algorithms. Specifically, in a system of  $n$  processes, this model combines at most  $t$  Byzantine process failures with a message adversary that can delete  $d$  messages sent by any process during a communication step on the network. The message adversary is defined independently of any synchrony assumption or any specific structure of the algorithm. As a result, this fault model naturally applies to any algorithm executing on an asynchronous message-passing system.
2. *A new formal definition of reliable broadcast (MBRB).* Using the previous hybrid model, this thesis defines a new abstraction called *Message-Adversary Byzantine Reliable Broadcast (MBRB)*, that generalizes simple Byzantine reliable broadcast (BRB) and allows addressing broader failure scenarios, particularly those involving mobile faults that can cause message losses. Specifically, MBRB explicitly imposes a lower bound (noted  $\ell_{MBRB}$  and called “*delivery power*”) on the number of correct processes that must deliver individual broadcast values. This is because, unlike with BRB, it is no longer possible to guarantee that all processes deliver the broadcast value in the presence of a message adversary, as the latter can completely isolate at most  $d$  correct processes from the network.
3. *An optimality theorem for MBRB.* We demonstrate that, within the theoretical framework we have defined, MBRB can be implemented if and only if the condition  $n > 3t + 2d$  is respected. Intuitively, this bound combines the condition  $n > 3t$ , necessary and sufficient to solve agreement problems in asynchronous contexts subject to Byzantine failures, with the condition  $n > 2d$ , which prevents the presence of network partitioning. This theoretical result establishes fundamental limits and guides the design of optimal algorithms.
4. *A simple and optimal implementation of MBRB.* Building on the previous insights, we propose a novel algorithm based on digital signatures, that simply demonstrates the practical feasibility of MBRB, with maximum resilience and delivery power. Indeed, this algorithm only assumes the optimal resilience bound  $n > 3t + 2d$ , and its delivery power is  $\ell_{MBRB} = c - d$ , where  $c$  is the actual number of correct processes in the system ( $n - t \leq c \leq n$ ).
5. *The  $k2\ell$ -cast abstraction.* Although the previous algorithm provides an implementation of MBRB with optimal resilience and delivery power, it requires signatures, which in practice are only guaranteed to hold with a given probability. In order to explore how MBRB can be implemented without such an assumption, we introduce a novel many-to-many abstraction termed  $k2\ell$ -cast, which ensures that if a critical mass of  $k$  correct processes broadcast the same value, then at least  $\ell$  correct processes will eventually deliver the value. Thus,  $k2\ell$ -cast captures a quorum construction mechanism omnipresent in the design of distributed

algorithms. Particularly, we demonstrate that  $k2\ell$ -cast facilitates the construction of signature-free MBRB algorithms by transforming classic signature-free BRB algorithms, such as Bracha’s or Imbs-Raynal’s, making them not only tolerant to hybrid failures but also faster in execution. This allows us to broaden the applicability of our work to systems without cryptography.

6. *An implementation of MBRB using erasure coding.* In a final contribution, we revisit the signature-based implementation of MBRB from a communication complexity perspective. Although our first signature-based MBRB algorithm provides optimal fault resilience and delivery power, it exhibits a high communication cost. By contrast, our new enhanced algorithm, called Coded MBRB, achieves near-optimal communication complexity using erasure coding techniques, threshold signatures, and vector commitments. Indeed, Coded MBRB has a communication cost of  $O(|v| + n\lambda)$  bits sent per correct process, where  $|v|$  is the size of the broadcast value, and  $\lambda$  is the security parameter of the cryptographic primitives. This makes Coded MBRB optimal up to  $\lambda$ , given that the lower communication bound is  $\Omega(|v| + n)$  bits sent per correct process. In summary, Coded MBRB is a more efficient algorithm for deployments on large-scale systems where bandwidth is a precious resource, such as blockchain systems or replicated databases.

## 1.6 Organization

The rest of this manuscript is organized as follows.

- Chapter 2 presents the state of the art, positioning our work in the current scientific landscape.
- Chapter 3 defines our hybrid system model and the MBRB problem, laying the theoretical foundations for our contributions.
- Chapter 4 presents our simple implementation of MBRB, demonstrating its practical feasibility.
- Chapter 5 showcases  $k2\ell$ -cast, expanding the scope of our results to signature-free systems.
- Chapter 6 optimizes our approach, making MBRB more communication-efficient for large-scale deployments.
- Finally, Chapter 7 synthesizes our findings and opens perspectives for future research.
- For the sake of presentation, some developments appear in Appendices A-D, such as additional details on the consensus problem and correctness proofs.

# BACKGROUND

---

If I have seen further, it is by standing on the shoulders of Giants.

---

Isaac Newton, *letter to Robert Hooke, 1675*

This chapter provides the foundational knowledge and state-of-the-art context necessary to understand the research landscape in which this thesis is situated. We begin by defining the standard system model (Section 2.1) underpinning the existing solutions discussed in this chapter. This model sets the stage for our exploration on two key areas: *Byzantine reliable broadcast (BRB)* and *hybrid fault models*. In Section 2.2, we delve into the state-of-the-art in *Byzantine reliable broadcast*, a fundamental primitive in distributed systems that ensures reliable cooperation in the presence of Byzantine faults. We examine both asynchronous and synchronous implementations, highlighting key algorithms and recent advancements in efficiency and fault tolerance. Section 2.3 explores *hybrid fault models*, which combine traditional Byzantine process failures with dynamic link failures. By the end of this chapter, readers will have a comprehensive understanding of the current state of research in fault-tolerant distributed systems, setting the stage for the novel contributions presented in subsequent chapters.

Table 1 summarizes the acronyms and notations used in this chapter.

## 2.1 The standard system model

The formal design and analysis of distributed algorithms requires a precise model of computation. Many models exist that cover the capacities of interacting agents (*Do they have an identity? Do they know each other? Is their setup static or dynamic?*), the type of interactions (*e.g.*, shared memory or message passing, see Section 1.1), the fault model (crash faults, Byzantine faults, see Section 1.3.2), or temporal assumptions (synchrony vs. asynchrony, see Section 1.3.1) [35,98,114,116]. In what follows, we describe one such system model that became a *de facto* standard in many distributed computing works. This is the common model assumed by all state-of-the-art solutions presented in Section 2.2 and Section 2.3.

| Acronyms  | Meaning  |
|-----------|--|
| BRB       | Byzantine-tolerant reliable broadcast                                    |
| MA        | Message adversary  |
| Notations | Meaning  |
| $n$       | number of processes in the system  |
| $t$       | upper bound on the number of Byzantine processes                         |
| $c$       | effective number of correct processes in a run ( $n - t \leq c \leq n$ ) |
| $\lambda$ | security parameter of the cryptographic primitives                       |
| $\star$   | unspecified value  |

Table 1: Acronyms and notations used in Chapter 2

### 2.1.1 Process model

The different solutions presented in this chapter assume a static message-passing distributed system comprising  $n$  processes, denoted  $p_1, \dots, p_n$ . Each process  $p_i$  has an identity, and all the identities are different and known by all processes. To simplify, we assume that  $i$  is the identity of  $p_i$ .

Regarding failures, up to  $t < n$  processes can be Byzantine, where a Byzantine process is a process whose behavior does not follow the code specified by its algorithm [93,110]. Byzantine processes can collude to fool the non-Byzantine processes (also called correct processes). Let us also notice that, in this model, the premature stop (crash) of a process is a Byzantine failure.

Moreover, given an execution of the algorithm,  $c$  denotes the number of processes that effectively behave correctly in that execution. We always have  $n - t \leq c \leq n$ . While this number remains unknown to correct processes, it is used in the following to analyze and characterize (more precisely than using its lower bound  $n - t$ ) the guarantees provided by the proposed algorithm.

### 2.1.2 Communication model

The processes communicate through a fully connected point-to-point communication network. This network is assumed to be reliable, in the sense that it neither corrupts, loses, nor creates messages. This chapter discusses both asynchronous and synchronous algorithms, so no synchrony assumption on the network is made at this stage.

Let `MSG` be a message type and  $v$  the associated value. A process can invoke the unreliable operation `broadcast MSG( $v$ )`, which is a shorthand for “**for all**  $i \in \{1, \dots, n\}$  **do send** `MSG( $v$ )` **to**  $p_j$  **end for**.” It is assumed that all the correct processes invoke `broadcast` to send messages. As we can see, the operation `broadcast MSG( $v$ )` is unreliable. For example, if the invoking process

crashes during its invocation, some correct processes might receive the implementation message  $\text{msg}(v)$  while others do not. Moreover, a Byzantine process can, by its very nature, send messages without using the macro-operation **broadcast**.

From a terminology point of view, at the implementation/network level, we say that messages are *broadcast* and *received*. Let us remind that we distinguish *messages* (which relate to the implementation level) from *values* (which refer to the applicative payloads).

### 2.1.3 Cryptographic schemes

Some of the solutions presented in this manuscript use cryptographic schemes, such as secure hash functions, asymmetric signatures, or erasure codes.

**Computationally-bounded cryptographic adversary** Except for very specific *theoretically unbreakable* cryptographic schemes (*e.g.*, one-time pads<sup>13</sup>), cryptography notoriously relies on probabilistic assumptions, such as the intractability of reversing one-way functions (*e.g.*, the prime factorization or discrete log problems). To this end, it is typically assumed that the adversary is computationally bounded (what is sometimes formalized as a *probabilistic polynomial-time (PPT)* adversary in the literature [88]). This adversary tries to compromise the private information of honest participants (*e.g.*, secret keys, messages, *etc.*) based on the information publicly transiting on the network (*e.g.*, public keys, signatures, *etc.*).

**Security parameter  $\lambda$**  Following the cryptography literature, a cryptographic scheme's security level is characterized by an abstract *security parameter*, which we denote  $\lambda$ : the higher this parameter, the more secure the scheme is [88].

However, for each cryptographic scheme, a trade-off exists between its security level (represented by  $\lambda$ ) and its storage and computational costs. A cryptographic scheme outputs cryptographic structures (such as hashes or signatures) and has associated storage costs (the size of the structures) and computational costs (the time taken to generate or verify the structures). But to increase the scheme's resistance to attacks, we typically increase the size of its structures, which also increases its storage and computational cost.

For example, if we want a higher security level for a hash function (*i.e.*, better resistance against collision and preimage attacks), then the hashes must be longer, incurring higher storage and computational overhead. In fact,  $\lambda$  is proportional to the scheme's number of security bits (the effective number of bits one has to brute-force to break it). It is typically assumed that the communication and computational costs of cryptographic schemes is in  $O(\lambda)$  (*e.g.*, a hash has  $O(\lambda)$  bits).

---

13. One-time pads however introduce costly assumptions to function, making them seldom used in practice.



Moreover, it is commonly assumed that, in a system of  $n$  processes, we have  $\lambda = \Omega(\log n)$ . Indeed, as public keys (or hashes of public keys, which have  $O(\lambda)$  bits) are typically used to identify the system's processes, the keyspace must be at least as large as the minimum number of bits needed to identify each process (*i.e.*,  $\log n$ ). This constraint also follows from the assumption that an adversary controlling a constant fraction of a system of size  $n$  cannot brute-force the cryptographic schemes too quickly, or more formally, that her computing power is subexponential in the security parameter  $\lambda$ .

## 2.2 Byzantine reliable broadcast (BRB)

The system model presented in Section 2.1 was initially proposed in the eighties and has since motivated a large body of work regarding its inherent computability limits, as well as the fundamental primitives it can support. One such primitive is reliable broadcast, whose goal and importance have been highlighted in Section 1.4.2. Due to its fundamental nature, Byzantine reliable broadcast (BRB) has been addressed by many authors, as we will see in the following. But first, let us formally define the BRB abstraction.

**BRB operations** The BRB communication abstraction comprises two matching operations, denoted `brb_broadcast()` and `brb_deliver()`. It considers that each value  $v$  disseminated using BRB has a unique identity  $(sn, i)$  (sequence number, sender identity). Furthermore, it assumes (as a precondition) that any two invocations of the `brb_broadcast()` by a correct process provide different sequence numbers. Sequence numbers are one of the most natural ways to design “multi-shot” reliable broadcast algorithms, that is, algorithms where the `brb_broadcast()` operation can be invoked multiple times with different values.

When, at the application level, a process  $p_i$  invokes `brb_broadcast( $v, sn$ )`, where  $v$  is the value and  $sn$  the associated sequence number, we say  $p_i$  “brb-broadcasts  $(v, sn)$ .” Similarly, when  $p_i$  invokes `brb_deliver( $v, sn, j$ )`, where  $p_j$  is the sender process, we say  $p_i$  “brb-delivers  $(v, sn, j)$ .” The `brb_deliver()` operation is, in fact, a callback issued by the abstraction to notify the client that a value has been delivered. To summarize, we say that the values are *brb-broadcast* and *brb-delivered* (while, as said in Section 2.1.2, the messages of the underlying algorithm are *broadcast* and *received*).

**BRB properties** The BRB abstraction is defined by the following safety and liveness properties [32,116].

- **Safety.**
  - **BRB-Validity (no spurious message).** If a correct process  $p_i$  brb-delivers a value  $v$  from a correct process  $p_j$  with sequence number  $sn$ , then  $p_j$  brb-broadcast  $v$  with sequence number  $sn$ .

- ▶ **BRB-No-duplication.** A correct process  $p_i$  brb-delivers at most one value  $v$  from a process  $p_j$  with sequence number  $sn$ .
- ▶ **BRB-No-duplicity.** No two correct processes brb-deliver different values from a process  $p_i$  with the same sequence number  $sn$ .
- **Liveness.**
  - ▶ **BRB-Local-delivery.** If a correct process  $p_i$  brb-broadcasts a value  $v$  with sequence number  $sn$ , then at least one correct process  $p_j$  eventually brb-delivers  $v$  from  $p_i$  with  $sn$ .
  - ▶ **BRB-Global-delivery.** If a correct process  $p_i$  brb-delivers a value  $v$  from a process  $p_j$  with sequence number  $sn$ , then all  $c$  correct processes brb-deliver  $m$  from  $p_j$  with sequence number  $sn$ .

Given two correct processes  $p_i$  and  $p_j$ , a value  $v$  and a sequence number  $sn$ , BRB guarantees that “ $p_i$  brb-delivers  $v$  from  $p_j$  with  $sn$  if and only if  $p_j$  has brb-broadcast  $v$  with  $sn$ ” (the “if and only” translates to a double implication that entails both BRB-Validity and BRB-Local-delivery). Furthermore, given a correct process  $p_i$ , a (correct or Byzantine) process  $p_j$ , a value  $v$  and a sequence number  $sn$ , BRB also ensures that “if  $p_i$  brb-delivers  $v$  from  $p_j$  with  $sn$ , then, all correct processes brb-deliver only  $v$  for the value identity  $(sn, j)$ ” (this captures BRB-No-duplicity and BRB-Global-delivery, and the “only” guarantees BRB-No-duplication).

### 2.2.1 Asynchronous signature-free BRB

The power of digital signatures for implementing distributed algorithms was recognized very early. For example, Lamport, Shostak, and Pease showed in 1982 [93] how *signed* messages can drastically improve fault tolerance over *oral* (*i.e.*, unsigned) messages<sup>14</sup>, just a few years after the public description of RSA [122], the first secure public-key cryptosystem for encryption and digital signatures. However, using signatures imposes additional assumptions, *e.g.*, the existence of a public-key infrastructure (PKI) or a limit on the computing power of the adversary (see Section 2.1.3). Since the introduction of the Byzantine reliable broadcast (BRB) problem, researchers have therefore repeatedly striven to propose solutions that do not rely on digital signatures. Hence, this section presents two of the most prominent signature-free BRB algorithms: Bracha’s BRB [32], and Imbs-Raynal’s BRB [87].

In the following algorithms and throughout this manuscript, the  $\star$  symbol is used as a wildcard (any value can be matched).

---

14. In particular, Lamport, Shostak, and Pease showed that, in a synchronous context, it is possible to implement BRB under any arbitrary number of Byzantine processes using signatures; but without them, it is only possible to tolerate less than one-third of Byzantine processes [93].

**Bracha’s BRB (1987)** One of the most famous and earliest asynchronous BRB algorithm, described in Algorithm 1, is due to Bracha [32]. For a value disseminated by a correct sender, this algorithm gives rise to three sequential communication steps and up to  $(n - 1)(2n + 1)$  messages sent by correct processes. This algorithm requires  $n > 3t$ , which is optimal in terms of fault tolerance. The versatility of Bracha’s algorithm has been analyzed in [85,117].

Like many other distributed agreement algorithms, Bracha’s BRB leverages the concept of *quorums* [99], which refers to a subset of processes that (at the implementation level) “vote” for the same value. This definition takes quorums in their ordinary sense: in a deliberative assembly, a quorum is the minimum number of members that must vote the same way for an irrevocable decision to be taken.

As detailed in Algorithm 1, Bracha’s BRB works in three communication phases, each associated with a particular message type: `INIT`, `ECHO`, and `READY`. The initial BRB sender uses the `INIT` type to broadcast her value to everyone in the network at line 1. Next, all (correct) receivers of the `INIT` message broadcast an `ECHO` message at line 4 containing the first value  $v$  they received for the associated *identity*  $(sn, i)$ .

When some correct process observes a quorum of `ECHO` messages for the first time, it sends a `READY` message backing the same value at line 9. Finally, when a correct receives *enough*

```

1 operation brb_broadcast( $v, sn$ ) is broadcast INIT( $v, sn$ ).
2 when INIT( $v, sn$ ) is received from  $p_j$  do
3   | if  $p_i$  has not already broadcast some ECHO( $\star, sn, j$ ) then
4   |   | broadcast ECHO( $v, sn, j$ ).
5 when ECHO( $v, sn, j$ ) is received from strictly more than  $\frac{n+t}{2}$  processes do
6   | if  $p_i$  has not already broadcast some ECHO( $\star, sn, j$ ) then
7   |   | broadcast ECHO( $v, sn, j$ );
8   | if  $p_i$  has not already broadcast some READY( $\star, sn, j$ ) then
9   |   | broadcast READY( $v, sn, j$ ).
10 when READY( $v, sn, j$ ) is received from at least  $t + 1$  processes do
11   | if  $p_i$  has not already broadcast some READY( $\star, sn, j$ ) then
12   |   | broadcast READY( $v, sn, j$ ).
13 when READY( $v, sn, j$ ) is received from at least  $2t + 1$  processes do
14   | if  $p_i$  has not already brb-delivered some ( $\star, sn, j$ ) then
15   |   | brb_deliver( $v, sn, j$ ).

```

Algorithm 1: Multi-shot version of Bracha’s BRB algorithm ( $n > 3t$ , code for  $p_i$ )

READY messages for the same value, it brb-delivers this value at line 15. By construction, it is impossible to observe two different quorums of strictly more than  $\frac{n+t}{2}$  ECHO messages backing two different values  $v \neq v'$ .

Informally, the ECHO phase guarantees BRB-No-duplicity (at most one value can be brb-delivered), while the READY phase guarantees BRB-Global-delivery (if some process brb-delivers, everyone brb-delivers).

**Imbs-Raynal’s BRB (2016)** Addressing efficiency issues, Imbs and Raynal proposed another signature-free BRB algorithm [87], described in Algorithm 2. This algorithm implements the reliable broadcast of a value by a correct process with only two communication steps (which is optimal) and up to  $n^2 - 1$  messages sent by correct processes. The price for this gain in efficiency is a weaker  $t$ -resilience than Bracha’s BRB, namely  $n > 5t$ . Hence, this algorithm and Bracha’s algorithms differ in their trade-off between  $t$ -resilience and message/time efficiency.

```

1 operation brb_broadcast( $v, sn$ ) is broadcast INIT( $v, sn$ ).
2 when INIT( $v, sn$ ) is received from  $p_j$  do
3   if  $p_i$  has not already broadcast some WITNESS( $\star, sn, j$ ) then
4     broadcast WITNESS( $v, sn, j$ ).
5 when WITNESS( $v, sn, j$ ) is received from at least  $n - 2t$  processes do
6   if  $p_i$  has not already broadcast some WITNESS( $\star, sn, j$ ) then
7     broadcast WITNESS( $v, sn, j$ ).
8 when WITNESS( $v, sn, j$ ) is received from at least  $n - t$  processes do
9   if  $p_i$  has not already brb-delivered some ( $\star, sn, j$ ) then
10  brb_deliver( $v, sn, j$ ).

```

Algorithm 2: Multi-shot version of Imbs-Raynal’s BRB algorithm ( $n > 5t$ , code for  $p_i$ )

As detailed in Algorithm 2, Imbs-Raynal’s BRB works in only two communication phases, each associated with a particular message type: INIT and WITNESS. Like in Bracha’s BRB, the initial sender in Imbs-Raynal’s BRB broadcasts her value to everyone in a INIT message at line 1. Then, every correct receiver broadcasts a WITNESS message backing the first received value from the sender line 4. A first threshold of  $n - 2t$  received WITNESS messages for the same value corresponds to the situation where a correct process can broadcast a new WITNESS message for this value at line 7, if it has not already done so previously. A second threshold of  $n - t$  received WITNESS messages for the same value corresponds to a brb-delivery of this value at line 10.

Intuitively, the WITNESS message of Imbs-Raynal’s algorithm fulfills the functions of both the ECHO and READY messages in Bracha’s algorithm. Indeed, the first threshold is large enough to

guarantee BRB-No-duplicity (there cannot be two different quorums of  $n - 2t$  messages if  $n > 5t$ ), while the *forwarding* mechanism and the second threshold ensure BRB-Global-delivery.

### 2.2.2 Asynchronous communication-efficient asynchronous BRB

Besides improving the resilience and round complexity (*i.e.*, latency) of BRB, reducing its communication cost (*i.e.*, the number of bits sent in the network overall) is also an important research direction. Here are a few recent results. Similarly to Bracha’s algorithm, all these algorithms assume an underlying fully connected asynchronous reliable network.

An efficient algorithm for BRB with long inputs of  $b$  bits using lower costs than  $b$  single-bit instances is presented in [107]. This algorithm, which assumes  $t < \frac{n}{3}$ , achieves the best possible communication complexity of  $\Theta(nb)$  input sizes. This article also presents an authenticated extension of this solution.

Scalable BRB is addressed in [78]. This work aims to avoid paying the  $O(n^2)$  message complexity price by using a non-trivial message-gossiping approach. This strategy makes it possible to design a sophisticated BRB algorithm satisfying probability-dependent properties.

To minimize the dissemination costs associated with message transmission across the network, some solutions rely on digital signatures and coding techniques [16,33,34,67,102,138]. Instead of communicating a value  $v$  of  $|v|$  bits directly, the sender first encodes the value using an error-correction code and “splits” the resulting codeword between the processes, so that each process receives one fragment of size  $O(|v|/n)$  bits. Any process that has received a *sufficient* number of fragments can reconstruct the entire value, thus ensuring the ability to reconstruct data in the event of process failures or adversarial compromises. Using this technique, each process needs to broadcast only its fragment of the value rather than the entire value. This reduced per-process communication effectively reduces the overall communication for disseminating the value itself to  $n|v|$  bits.

Another problem close to BRB is *Verifiable Information Dispersal* (or *VID*) [34]. VID implementations aim to ensure the consistency of the received information across the network, while reducing the required bandwidth usage, typically by using error-correcting codes. In this setting, significant contributions have been made by Cachin and Tessaro [34] as well as Cachin and Poritz in SINTRA [33], followed by its successors such as Honey Badger [102], BEAT [67], or DispersedLedger [138].

From a theoretical perspective, a lower bound on the overall communication complexity of BRB is  $\Omega(n|v| + n^2)$  bits, because every correct process must receive the entire value  $v$ , and because the reliable broadcast of a single bit necessitates at least  $\Omega(n^2)$  messages, as implied by the Dolev-Reischuk lower bound [62]. In this context, Alhaddad *et al.* [16] proposed several *balanced* BRB algorithms relying on erasure coding (*balanced* means that the communication cost is the

same for the sender and the rest of the processes). Notably, Alhaddad *et al.* proposed two *near-optimal* balanced BRB algorithms: one signature-free in  $O(n|v| + n^2 \log n)$  bits overall and one signature-based in  $O(n|v| + n\lambda + n^2 \log n)$  bits overall, where  $\lambda$  denotes the security parameter (see Section 2.1.3).

### 2.2.3 Synchronous BRB

The Synchronous Byzantine Reliable Broadcast problem was first introduced in [110] by Lamport, Shostak, and Pease, who proposed in [93] a deterministic solution based on signature chains, which tolerates any arbitrary number  $t < n$  of Byzantine processes present in the system. This solution requires  $t + 1$  rounds both in good cases (where the sender is correct) and bad cases (where the sender is Byzantine). Dolev and Strong showed this worst-case round complexity of  $t + 1$  rounds to be optimal for deterministic algorithms [63].

In recent years, substantial progress has been made in circumventing the bound of  $t + 1$  rounds for deterministic BRB algorithms, by exploiting *randomization* or only considering the good case where the BRB sender is correct [2,71,137]. In the deterministic case, it has also been recently shown that a good-case latency (*i.e.*, latency when the sender is correct) for synchronous BRB lower than  $t + 1$  rounds can be achieved using a deterministic algorithm subject to an arbitrary number of Byzantine faults [15] (in particular, this algorithm has a good-case latency of  $\max(2, t + 3 - c)$  synchronous rounds).

### 2.2.4 Summary

In this section, we have explored the state-of-the-art in Byzantine reliable broadcast, from foundational algorithms like Bracha’s to recent advancements in communication efficiency and synchronous implementations. These algorithms provide the basis for reliable communication in Byzantine-prone systems, but they are limited to static failure models where Byzantine behavior is confined to a fixed set of processes. As we will see in the next section and throughout this thesis, extending these concepts to more dynamic failure scenarios is crucial for addressing the challenges of modern distributed systems.

## 2.3 Hybrid fault models

While Byzantine reliable broadcast provides robust communication in the presence of Byzantine process failures, real-world distributed systems often face more complex failure scenarios. This brings us to our next topic: *hybrid fault models*. These models extend our understanding of system failures beyond just Byzantine processes, incorporating dynamic link failures to more accurately represent the challenges faced in real-world distributed systems.

Byzantine process failures cover many adversarial behaviors but remain bound to a specific set of processes. Santoro and Widmayer proposed an alternative fault model for synchronous networks [126,127], which considers *mobile* (or dynamic) *link failures* between correct processes.

In this model, failures are no longer bound to particular processes. Instead, a *message adversary* (MA) may corrupt or delete some of the  $n(n - 1)$  possible transmissions during one synchronous round, where  $n$  is the number of processes in the system. This notion of MA was implicitly introduced in [126] (under the name *transient faults* and *ubiquitous faults*) and then used (sometimes implicitly) in many works (e.g., [3,44,113,127,134]). A short tutorial on message adversaries is presented in [115]. The works of [26,58,127] focus on the case of reliable broadcast in the presence of such link failures.

Generalizing further, mobile link failures and Byzantine processes may be combined to produce a *hybrid fault model*, in which some failures are pinned to specific processes while others affect links dynamically [28,76,112,131]. Biely, Schmid, and Weiss [28] have, in particular, proposed an extensive hybrid fault model for synchronous systems that includes both a range of process failures (including Byzantine behaviors) along with mobile link failures between correct processes. As in Santoro and Widmayer’s model, link failures are mobile in that they might impact different processes in different (synchronous) rounds. The model is constrained by limiting the number of link failures a process might experience during a synchronous round both as a sender (*send* link failures) and as a recipient (*receive* link failures).

This model was extended by Schmid and Fetzer [128]<sup>15</sup> to *asynchronous round-based algorithms*. Schmid and Fetzer present, in particular, a Simulated Authenticated Broadcast algorithm (a weak form of Byzantine broadcast allowing duplicity by Byzantine senders) and a Randomized Consensus algorithm that work provided both send and receive link failures remain limited to specific patterns, ensuring in particular that no correct<sup>16</sup> process ever gets entirely disconnected from other correct processes.

Another approach to address Byzantine processes coupled with faulty links is to consider the edge connectivity of incomplete communication networks. In this setting, Pelc proved that robust communication is feasible over graphs whose edge-connectivity is more than  $2f$ , assuming the number of Byzantine *links* is bounded by  $f$  [111], which is also implied by the work of Dolev [65]. Intuitively, if at least half of the communication channels of each process can be severed, then an adversary can partition the network. Censor-Hillel, Cohen, Gelles, and Sela [40] showed that any computation can be performed when all links suffer arbitrary substitution faults (but no crashes), provided the network is 2-edge connected. When all links suffer corruption, but the overall amount of corruption is restricted, any computation can be reliably performed using the solution by Hoza and Schulman [86], for synchronous networks where the topology is known, or the solution by Censor-Hillel, Gelles, and Haeupler [41], for asynchronous networks with unknown topology. Bonomi, Decouchant, Farina, Rahli, and Tixeuil also considered the

15. Although Schmid and Fetzer’s work [128] predates the publication of Biely, Schmid, and Weiss [28], it cites an earlier version of [28] available as a technical report.

16. The model uses a finer notion of *obedient* process, which is ignored here for simplicity.

case of BRB in a synchronous multi-hop (*i.e.*, partially connected) communication network [29]. However, as it is classically assumed in distributed graph algorithms, the previous works either assume synchronous communication or static network topology, thus precluding the study of dynamic and mobile link faults under asynchrony.

### 2.3.1 Summary

This section provided an overview the evolution of fault models in distributed systems, from traditional Byzantine faults to more complex hybrid models that incorporate dynamic link failures. These hybrid fault models represent a significant advancement in our ability to design and analyze robust distributed systems. By accounting for both process and link failures, they enable the development of algorithms and protocols that can maintain reliability and consistency in more diverse and challenging environments.

## 2.4 Conclusion

This chapter has provided an overview of the foundational concepts and state-of-the-art research underpinning this thesis. We introduced the standard system model for distributed computing and explored Byzantine Reliable Broadcast (BRB), a crucial primitive for ensuring reliable cooperation in Byzantine-prone systems. Our examination of both asynchronous and synchronous BRB implementations highlighted the ongoing challenges in balancing fault tolerance, efficiency, and scalability. We also discussed the evolution of fault models, from traditional Byzantine faults to hybrid models that incorporate dynamic link failures. These hybrid models represent a significant step towards more realistic representations of failure scenarios in distributed systems. However, throughout this exploration, several key challenges emerged:

- The need for simple hybrid fault models accounting for both Byzantine processes and unreliable network links, that can apply to any asynchronous message-passing algorithm (independently of its use of rounds or not);
- Designing BRB algorithms that can operate in more dynamic network conditions;
- Balancing resilience, communication efficiency, and latency in BRB implementations.

These challenges set the stage for the novel contributions presented in this thesis. In the following chapters, we will introduce our new hybrid fault model and the Message-Adversary-tolerant Byzantine Reliable Broadcast (MBRB) abstraction, addressing many of the identified challenges and advancing our understanding of fault-tolerant distributed systems.





# A THEORETICAL FRAMEWORK FOR HYBRID FAULT TOLERANCE

---

The single biggest problem in communication is the illusion that it has taken place.

---

*George Bernard Shaw*

The increasing complexity and scale of modern distributed systems have exposed limitations in traditional fault models, particularly when dealing with hybrid failures that affect both processes and network communications. This chapter introduces a novel computing model that addresses these challenges by combining Byzantine process failures with a message adversary in an asynchronous setting.

We begin by presenting the details of this new hybrid fault model (Section 3.1), which will serve as the foundation for all subsequent algorithms and analyses in this thesis. This model assumes an asynchronous distributed system of  $n$  processes, out of which at most  $t$  may be Byzantine, and where a message adversary (MA) may remove  $d$  copies of a message broadcast by a correct process. This represents a particularly challenging environment, as the MA may target different correct processes every time the network is used, or focus indefinitely on the same (correct) victims. Further, the Byzantine processes may collude with the MA for maximal impact.

Building on this model, we introduce the *Message-Adversary-tolerant Byzantine Reliable Broadcast (MBRB)* abstraction (Section 3.2), a powerful primitive that extends traditional reliable broadcast to environments prone to both Byzantine and network failures. Finally, we establish fundamental bounds on the implementability of MBRB in asynchronous systems (Section 3.3), providing crucial insights into the limits and possibilities of fault-tolerant communication in challenging distributed environments.

| Acronyms      | Meaning  |
|---------------|--|
| MA            | Message adversary  |
| MBRB          | MA-tolerant Byzantine reliable broadcast   |
| Notations     | Meaning  |
| $n$           | number of processes in the system  |
| $t$           | upper bound on the number of Byzantine processes                                       |
| $d$           | power of the message adversary   |
| $c$           | effective number of correct processes in a run ( $n - t \leq c \leq n$ )               |
| $\ell_{MBRB}$ | minimal number of correct processes that mbrb-deliver a value                          |
| $\delta$      | time complexity of MBRB  |
| $\mu$         | message complexity of MBRB   |
| $\kappa$      | communication complexity of MBRB<br>(number of bits sent during the execution overall) |

Table 2: Acronyms and notations used in Chapter 3

By developing this comprehensive theoretical framework, we aim not only to advance our understanding of fault-tolerant distributed systems, but also to pave the way for more resilient algorithms capable of operating in increasingly complex and unpredictable network conditions.

Table 2 summarizes the acronyms and notations used in this chapter.

### 3.1 A common computing model

In this section, we introduce a new hybrid system model capturing Byzantine and link faults, that will be the common theoretical framework for the rest of this thesis. This model explicitly considers the disconnection of correct processes and applies to any message-passing algorithm, whether it uses rounds or not.<sup>17</sup> (In particular, no algorithm presented in this thesis uses explicit rounds.)

Our motivation for this novel hybrid fault model originated from our research on the reconciliation of local process states in distributed Byzantine-tolerant money transfer systems (a.k.a. cryptocurrencies), in which processes become temporarily disconnected. As in the model of Schmid and Fetzer [128] (see Section 2.3), our model combines two types of adversary in an asynchronous network: some processes may be *Byzantine*, but in addition, a *message adversary* may also remove network messages between correct processes.<sup>18</sup> However, contrary to Schmid

17. In Schmid and Fetzer’s model [128], rounds are essential to defining *receive* link failures.

18. Schmid and Fetzer’s model also encompasses arbitrary link failures, which may corrupt messages.

and Fetzer’s approach, we do not limit the number of receive link failures. As a consequence, our model allows correct processes to become disconnected for arbitrarily long periods of time. This design also allows us to eschew the notion of rounds entirely in the definition of our fault model.

**Process model** The following process model largely follows that of the standard system model, presented in Section 2.1. The system comprises  $n$  asynchronous sequential processes denoted  $p_1, \dots, p_n$ . Out of these  $n$  processes, up to  $t$  can be Byzantine [93,110]. For a more fine-grained analysis, we also consider  $c$ , the effective number of correct processes in the system for one run of the system, such that  $n - t \leq c \leq n$ . The number  $c$  is unknown to participants.

**Communication model (comm and broadcast operations)** As in the standard model (Section 2.1), processes communicate through a fully connected asynchronous point-to-point communication network. Although this network is assumed to be reliable—in the sense that it neither corrupts, duplicates, nor creates messages—it may nevertheless lose messages due to the actions of a message adversary (defined below).

Correct processes communicate by using the transmission macro  $\mathbf{comm}(m_1, \dots, m_n)$ , that sends message  $m_j$  to  $p_j$  for every  $j \in [1..n]$ . The message  $m_j$  can also be empty, in which case nothing will be sent to  $p_j$ . Therefore, let us note that the  $\mathbf{comm}(\cdot)$  primitive can simulate any kind of multicast communication between processes of the system (whether it be unicast or broadcast, for different messages or for the same message). For simplicity, processes also have access to a **broadcast** operation, which is a shorthand for  $\mathbf{broadcast} m = \mathbf{comm}(m, m, \dots, m)$ , and which simply sends the same message  $m$  to all processes. Byzantine processes may deviate arbitrarily from the correct implementation of  $\mathbf{comm}(\cdot)$ . For instance, they may unicast messages to only a subset of processes.

**Message adversary** Let  $d$  be an integer such that  $0 \leq d < c$ . An adversary controls, to some extent, the communication network and eliminates messages sent by processes. More precisely, when a correct process invokes  $\mathbf{comm}(m_1, \dots, m_n)$ , the message adversary has the discretion to choose up to  $d$  messages of the set  $\{m_1, \dots, m_n\}$  and eliminate them from the corresponding communication channels they were queued. This means that, despite the sender being correct, up to  $d$  correct processes may miss their intended message  $m_j$ .<sup>19</sup> Let us remark that, given that the **broadcast**  $m$  macro-operation is defined using the  $\mathbf{comm}(\cdot)$  primitive, it is also subject to the action of the MA: when a correct process invokes **broadcast**  $m$ , the MA can arbitrarily suppress up to  $d$  copies of message  $m$ .

For example, consider a set  $D$  of correct processes, where  $1 \leq |D| \leq d$ , such that during some period of time, the MA suppresses all the messages sent to them. It follows that, during this

---

19. A close but different notion was introduced by Dolev in [65] (and explored in subsequent works, such as [29]), which considers static  $\kappa$ -connected networks. If the adversary selects statically for each correct sender  $d$  correct processes that do not receive this sender’s messages, the proposed model includes Dolev’s model with  $\kappa = n - d$ .

period of time, this set of processes appears as being (unknowingly) input-disconnected from the other correct processes. Depending on the strategy of the MA, the set  $D$  may vary with time, and it is never known by the correct processes. Let us notice that  $d = 0$  corresponds to the weakest possible message adversary: it boils down to a classical static system where some processes are Byzantine, but no message is lost (the network is fully reliable).

Remark that this type of message adversary is stronger and, therefore, covers the more specific case of *silent churn*, in which processes may decide to disconnect from the network. While disconnected, such a process silently pauses its algorithm (a legal behavior in our asynchronous model) and is implicitly moved (by the adversary) to the  $D$  adversary-defined set. Upon returning, the node resumes its execution and is removed from  $D$  by the adversary.<sup>20</sup>

Informally, in a silent churn environment, a correct process may miss messages sent by other processes while disconnected from the network. The adjective “silent” in *silent churn* expresses that processes do not send notifications on the network whenever they leave or join the system. There is no explicit “attendance list” of connected processes, and processes are given no information on their peers’ status (connected/disconnected). In this regard, the silent churn model diverges from the classical approach when designing dynamic distributed systems, in which processes send messages on the network notifying their connection or disconnection [77]. The silent churn model is a good representation of real-life large-scale peer-to-peer systems, where peers can leave the network silently (*i.e.*, without warning other peers).<sup>21</sup>

Let us also observe that silent churn allows us to model input-disconnections due to process mobility. When a process moves from one location to another, the sender’s broadcasting range may not be large enough to ensure the moving process remains input-connected. An even more prosaic example is when a user simply turns off her device or disables its Internet connection, preventing it from receiving or sending any further messages. In this context, the MA removes all the incoming messages from the corresponding process until the device reconnects.

Let us mention that the loss of messages caused by a message adversary may be addressed using a reliable unicast protocol. These protocols were originally introduced to provide reliable channels on top of an unreliable network subject to message losses. The principle is simple: the sender keeps sending idempotent messages through an unreliable channel until it receives an acknowledgment from the receiver. This principle notoriously lies at the core of the Transmission Control Protocol (TCP), although with important practical adaptations (TCP uses timeouts to close a malfunctioning or otherwise idle connection, typically after a few minutes).

---

20. So, the notion of a message adversary implicitly includes the notion of message omission failures.

21. For more insights on this topic, see [91], which presents an in-depth study of distributed computation in dynamic networks.

But because there is no way to detect that a process has crashed or disconnected in an asynchronous environment, an ideal reliable unicast protocol (*i.e.*, one that keeps on re-transmitting until success) needs to treat disconnected processes the same way as slow processes or as if there were packet losses in the network: the sender will thus potentially send infinitely many messages to a disconnected receiver. To overcome this issue, some solutions leverage causal dependencies to avoid resending old messages: if the sender receives an acknowledgment for a given message, it can stop resending the messages that causally precede this message and that have not been acknowledged yet (*e.g.*, [56]). However, this approach still assumes that, eventually, every communication channel lets some messages pass. However, our fault model does not guarantee this property, as the MA can permanently sever up to  $d$  channels per correct process.

### 3.2 MA-tolerant Byzantine reliable broadcast (MBRB)

Having established our new hybrid fault model, which captures both Byzantine process failures and the actions of a message adversary, we now look into the challenge of reliable communication in such an environment. The following section introduces the *Message-Adversary-tolerant Byzantine Reliable Broadcast (MBRB)* abstraction, a powerful primitive designed to operate within the constraints of our hybrid fault model. MBRB extends traditional reliable broadcast concepts to account for the additional complexities introduced by the message adversary, providing a robust foundation for building fault-tolerant distributed applications. Several researchers have indeed pointed out the fundamental role that broadcast abstractions play in Byzantine money transfer systems (for instance, see [21,54–56,77,78]). This crucial role naturally leads to considering how Byzantine reliable broadcast can be expanded to more volatile and dynamic settings, thus motivating our proposal to combine traditional Byzantine faults with a message adversary.

**MBRB operations** The MBRB communication abstraction comprises two matching operations, denoted `mbrb_broadcast` and `mbrb_deliver`. It considers that each value is associated with an identity  $(sn, i)$  (sender identity, sequence number) and assumes that any two values `mbrb_broadcast` by the same correct process have different sequence numbers.

When, at the application level, a process  $p_i$  invokes `mbrb_broadcast`( $v, sn$ ), where  $v$  is the value and  $sn$  the associated sequence number, we say  $p_i$  “mbrb-broadcasts ( $v, sn$ )”. Similarly, when  $p_i$  invokes `mbrb_deliver`( $v, sn, j$ ), where  $p_j$  is the sender process, we say  $p_i$  “mbr-delivers ( $v, sn, j$ )”. We say that the values are *mbrb-broadcast* and *mbrb-delivered*.

**MBRB properties** Because of the message adversary, we cannot always guarantee that a value `mbrb-delivered` by a correct process is eventually `mbrb-delivered` by all correct processes. Hence, in the MBRB specification, we introduce a variable  $\ell_{MBRB}$  which indicates the strength of the primitive’s global delivery guarantee: if one correct process `mbrb-delivers` a value, then

$\ell_{MBRB}$  correct processes eventually mbrb-deliver this value.<sup>22</sup> The MBRB abstraction is defined by the following properties.

- **Safety.**
  - **MBRB-Validity (no spurious message).** If a correct process  $p_i$  mbrb-delivers a value  $v$  from a correct process  $p_j$  with sequence number  $sn$ , then  $p_j$  mbrb-broadcast  $v$  with sequence number  $sn$ .
  - **MBRB-No-duplication.** A correct process  $p_i$  mbrb-delivers at most one value  $v$  from a process  $p_j$  with sequence number  $sn$ .
  - **MBRB-No-duplicity.** No two different correct processes mbrb-deliver different values from a process  $p_i$  with the same sequence number  $sn$ .
- **Liveness.**
  - **MBRB-Local-delivery.** If a correct process  $p_i$  mbrb-broadcasts a value  $v$  with sequence number  $sn$ , then at least one correct process  $p_j$  eventually mbrb-delivers  $v$  from  $p_i$  with  $sn$ .
  - **MBRB-Global-delivery.** If a correct process  $p_i$  mbrb-delivers a value  $v$  from a process  $p_j$  with sequence number  $sn$ , then at least  $\ell_{MBRB}$  correct processes mbrb-deliver  $v$  from  $p_j$  with sequence number  $sn$ .

It is implicitly assumed that a correct process does not use the same sequence number twice. Let us observe that since, at the implementation level, the message adversary can always suppress all the messages sent to a fixed set  $D$  of  $d$  processes, the best-guaranteed value for messages is  $c - d$ . Furthermore, notice that the constraint  $n > 2d$  prevents the message adversary from partitioning the system.

**Performance metrics** In addition to the correctness specification, we define three metrics that capture the performance of an MBRB algorithm:  $\delta$ ,  $\mu$ , and  $\kappa$ , which respectively denote the time complexity (in number of communication rounds), the message complexity, and the (bit-)communication complexity of the algorithm. They are defined as follows.

- **MBRB-Time-cost.** If a correct process  $p_i$  mbrb-broadcasts a value  $v$  with sequence number  $sn$ , then  $\ell_{MBRB}$  correct processes mbrb-deliver  $v$  from  $p_i$  with sequence number  $sn$  in at most  $\delta$  communication rounds.

As in similar analyses [36], a *communication round* is defined by assuming that the asynchronous algorithm executes in a synchronous model. In this model, all processes execute in lock-step synchronous rounds. Each synchronous round comprises a computation step followed by

---

22. If there is no message adversary (*i.e.*,  $d = 0$ ), we should have  $\ell_{MBRB} = c \geq n - t$ .

a communication step. In a computation step, invoked operations (*e.g.*, `mbrb_broadcast`) are executed; pending messages (if any) are processed (**when ... do** statements); and sent messages are buffered (but not delivered). In the subsequent communication step, all buffered messages not suppressed by the message adversary get delivered to their destination. These messages are then processed in the computation step of the following synchronous round.

- **MBRB-Message-cost.** The `mbrb-broadcast` of a value  $v$  by a correct process  $p_i$  entails the sending of at most  $\mu$  messages by correct processes overall.
- **MBRB-Communication-cost.** The `mbrb-broadcast` of a value  $v$  by a correct process  $p_i$  entails the sending of at most  $\kappa$  bits by correct processes overall.

**Byzantine Reliable Broadcast (BRB)** If  $\ell_{MBRB} = c$  (obtained when  $d = 0$ ), the previous specification boils down to Bracha’s seminal specification [32], which defines the Byzantine reliable broadcast (BRB) communication abstraction. Hence, the BRB abstraction is a sub-case of MBRB.

### 3.3 A necessary and sufficient condition for MBRB

With the MBRB abstraction defined, a natural question arises: under what conditions can this primitive be implemented in an asynchronous system subject to our hybrid fault model? The following section addresses this question by presenting a fundamental theorem on the necessary and sufficient conditions for implementing MBRB (Theorem 1), namely: in an asynchronous  $n$ -process system with at most  $t$  Byzantine processes and a message adversary of power  $d$ , MBRB can be implemented if and only if  $n > 3t + 2d$ . This result not only establishes the boundaries of what is achievable within our model but also provides crucial guidance for the design of optimal MBRB algorithms, which we will explore in subsequent chapters.

**Theorem 1** (MBRB-Tightness). *Considering an asynchronous  $n$ -process system in which up to  $t$  processes can be Byzantine and where a  $d$ -message adversary can suppress messages, the condition  $n > 3t + 2d$  is necessary and sufficient for implementing MBRB.*

Intuitively,  $n > 3t$  comes from the traditional condition for implementing Byzantine fault tolerant consistent broadcast (a weak form of reliable broadcast) in an asynchronous system [116], while  $n > 2d$  imposes that the message adversary cannot partition the network. Hence,  $n > 3t + 2d$  is simply the conjunction of these two well-known bounds. This theorem follows from two sub-results, one proving that the  $n > 3t + 2d$  condition is necessary, and the other proving that it is sufficient.

The proof that  $n > 3t + 2d$  is sufficient for implementing MBRB is given by Theorem 2 (page 42) and Theorem 6 (page 96), which respectively show that Algorithm 3 (Original MBRB) and



Algorithms 8-9 (Coded MBRB) implement the MBRB abstraction under this resilience bound. Consequentially, it also entails that these two MBRB implementations are optimal with respect to their Byzantine- and Message-Adversary resilience.

The proof that  $n > 3t + 2d$  is a necessary condition for implementing MBRB is given by Lemma 1, which states that no *event-driven* algorithm can implement MBRB in an asynchronous  $n$ -process system with at most  $t$  Byzantine processes and a message adversary of power  $d$  if  $n \leq 3t + 2d$ . In this context, *event-driven* means that the algorithm sends messages only after receiving messages or input from the upper-layer client (Definition 1). This *event-driven* characteristic can seem restrictive at first, as it forbids processes from updating their state and sending messages when no message arrives, but we can convince ourselves that this does not decrease the power of the model, since a process cannot gain any knowledge in an asynchronous system until a message arrives [136].

**Definition 1** (Event-driven algorithm). *An algorithm implementing a broadcast communication abstraction is event-driven if, as far as the correct processes are concerned, only (i) the invocation of the broadcast operation that is provided to the application by the broadcast communication abstraction, or (ii) the reception of a message—sent by a correct or a Byzantine process—can generate the sending of messages.*

We proceed to show the following impossibility condition for implementing MBRB.

**Lemma 1** (MBRB-Necessary-condition). *When  $n \leq 3t + 2d$ , there is no event-driven (signature-free or signature-based) algorithm implementing the MBRB communication abstraction on top of an  $n$ -process asynchronous system in which up to  $t$  processes may be Byzantine and where a message adversary may suppress up to  $d$  copies of each message broadcast by a correct process.<sup>23</sup>*

**Proof.** Without loss of generality, the proof considers the case  $n = 3t + 2d$ . Let us partition the  $n$  processes into five sets  $Q_1, Q_2, Q_3, D_1$ , and  $D_2$ , such that  $|D_1| = |D_2| = d$  and  $|Q_1| = |Q_2| = |Q_3| = t$ .<sup>24</sup> So, when considering the sets  $Q_1, Q_2$ , and  $Q_3$ , there are executions in which all the processes of either  $Q_1$  or  $Q_2$  or  $Q_3$  can be Byzantine, while the processes of the two other sets are not.

23. Without loss of generality, we consider that processes communicate through the unreliable **broadcast** operation defined in Section 3.1. However, let us remark that the following rationale also holds if the underlying communication medium is the generalized **comm** operation (Section 3.1), that can disseminate different messages to different recipients.

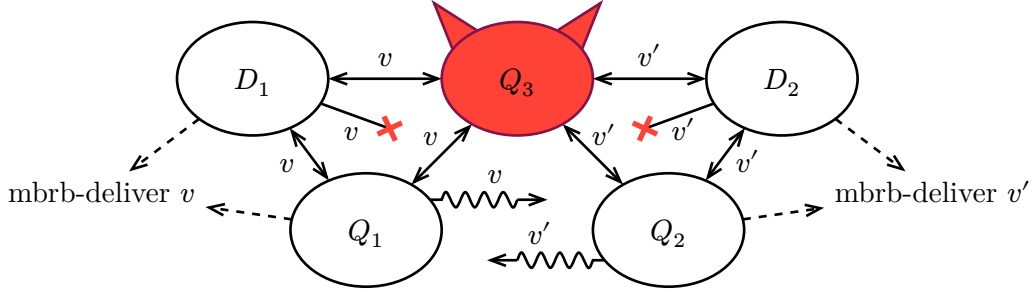


Figure 5: Communication between the different sets of processes of execution  $E$  leading to a MBRB-No-duplicity violation (the red crosses represent the messages suppressed by the message adversary)

The proof is by contradiction. So, assuming that there is an event-driven algorithm  $A$  that builds the MBRB abstraction for  $n = 3t + 2d$ , let us consider an execution  $E$  of  $A$  in which the processes of  $Q_1$ ,  $Q_2$ ,  $D_1$ , and  $D_2$  are not Byzantine while all the processes of  $Q_3$  are Byzantine.

Let us observe that the message adversary can isolate up to  $d$  processes by preventing them from receiving any message. Without losing generality, let us assume that the adversary isolates a set of  $t$  correct processes that does not contain the message sender. As  $A$  is event-driven, these  $t$  isolated processes do not send messages during the execution  $E$  of  $A$ . As a result, no correct process can expect messages from more than  $(n - t - d)$  different processes without risking being blocked forever. Thanks to the assumption  $n = 3t + 2d$ , this translates as “no correct process can expect messages from more than  $(2t + d)$  different processes without risking being blocked forever.”

In the execution  $E$ , the (Byzantine) processes of  $Q_3$  simulate the mbrb-broadcast of a value such that this value appears as being mbrb-broadcast by one of them and is mbrb-delivered as the value  $v$  to the processes of  $Q_1$  (hence the processes of  $Q_3$  appear, to the processes of  $Q_1$ , as if they were correct) and as the value  $v' \neq v$  to the processes of  $Q_2$  (hence, similarly to the previous case, the processes of  $Q_3$  appear to the processes of  $Q_2$  as if they were correct). Let us call  $v$ -messages (resp.,  $v'$ -messages) the messages generated by the event-driven algorithm  $A$  that entails the mbrb-delivery of  $v$  (resp.,  $v'$ ). Moreover, the execution  $E$  is such that we have the following.

- Concerning the  $v$ -messages: the message adversary suppresses all the  $v$ -messages sent to the processes of  $D_2$ , and asynchrony delays the reception of all the  $v$ -messages sent to  $Q_2$  until some time  $\tau$  defined below.<sup>25</sup> So, as  $|Q_1 \cup D_1 \cup Q_3| = n - t - d = 2t + d$ , Algorithm  $A$  will cause the processes of  $Q_1$  and  $D_1$  to mbrb-deliver  $v$ .<sup>26</sup>
- Concerning the  $v'$ -messages: the message adversary suppresses all the  $v'$ -messages sent to the processes of  $D_1$ , and the asynchrony delays the reception of all the  $v'$ -messages sent to

$Q_1$  until time  $\tau$ . As previously, as  $|Q_2 \cup D_2 \cup Q_3| = n - t - d = 2t + d$ , Algorithm  $A$  will cause the processes of  $Q_2$  and  $D_2$  to mbrb-deliver  $v'$ .

- Finally, the time  $\tau$  occurs after the mbrb-delivery of  $v$  by the processes of  $D_1$  and  $Q_1$ , and after the mbrb-delivery of  $v'$  by the processes of  $D_2$  and  $Q_2$ .

It follows that different non-Byzantine processes mbrb-deliver different values for the same mbrb-broadcast (or a fraudulent simulation of it) issued by a Byzantine process (with possibly the help of other Byzantine processes). This contradicts the MBRB-No-Duplicity property, which concludes the proof of the theorem.  $\square$

### 3.4 Conclusion

This chapter introduced a comprehensive computing model for addressing hybrid (*i.e.*, process- and network-related) failures in distributed systems, that will serve as the theoretical framework for the algorithms and analyses presented in the remainder of this thesis. This model defines an asynchronous message-passing network of  $n$  processes, out of which at most  $t$  may be Byzantine, and where a message adversary (MA) may remove  $d$  copies of a message disseminated by a correct process. Our model goes beyond earlier approaches, by considering hybrid faults under asynchrony while remaining oblivious to the algorithm's underlying structure (in particular, if it uses rounds or not), thus offering greater flexibility and realism.

Building on this new system model, the chapter then presented a novel hybrid-fault-tolerant reliable broadcast abstraction, called *Message-adversary-tolerant Byzantine reliable broadcast*, or *MBRB* (Section 3.2). MBRB extends the traditional notion of Byzantine Reliable Broadcast to account for the additional complexities introduced by the message adversary.

Finally, the chapter proved an optimality theorem regarding asynchronous MBRB, namely that it can be implemented if and only if we have  $n > 3t + 2d$ . The proof of this condition offers valuable insights into the interplay between Byzantine processes and the message adversary, highlighting the challenges of achieving agreement in such complex environments.

In the following chapters, we will leverage this theoretical foundation to present concrete implementations of MBRB, exploring both signature-based (Chapter 4) and signature-free (Chapter 5) approaches, as well as techniques for optimizing communication efficiency (Chapter 6). As our hybrid computing model is particularly aggressive regarding its weak synchrony assumptions and complex failures, these MBRB algorithms are designed to function in very challenging conditions.

24. For the case  $n < 3t + 2d$ , the partition is such that  $\max(|Q_1|, |D_2|) \leq d$  and  $\max(|Q_1|, |Q_2|, |Q_3|) \leq t$ .

25. Equivalently, we could also say that asynchrony delays the reception of all the  $v$ -messages sent to  $D_2 \cup Q_2$  until time  $\tau$ . The important point is here that, due to the assumed existence of Algorithm  $A$ , the processes of  $Q_1$  and  $D_1$  mbrb-deliver  $v$  with  $v$ -messages from at most  $2t + d$  different processes.

26. Let us notice that this is independent of whether the processes in  $Q_3$  are Byzantine or not.

# A SIMPLE SIGNATURE-BASED MBRB IMPLEMENTATION

---

Some single mind must be master, else there will  
be no agreement in anything.

---

*Abraham Lincoln*

This chapter presents an algorithm that implements the MBRB communication abstraction (presented in the previous chapter) in an asynchronous setting under the constraint  $n > 3t + 2d > 0$ . This algorithm uses digital signatures, and is optimal both in terms of Byzantine resilience and delivery power ( $\ell_{MBRB} = c - d$ ).

Furthermore, when considering  $d = 0$  (*i.e.*, no message adversary), this algorithm provides both an optimal  $t$ -resilience (as in Bracha’s BRB [32], Algorithm 1) and an optimal latency (as in Imbs-Raynal’s BRB [87], Algorithm 2): it only assumes  $n > 3t$ , and guarantees mbrb-delivery of a value in only two communication rounds, which is optimal.<sup>27</sup> Signatures can help save one round compared to classical signature-free BRB algorithms that assume  $n > 3t$ . Algorithm 3 fulfills the MBRB-Global-delivery property with a maximal delivery power<sup>28</sup> of  $\ell_{MBRB} = c - d$  under the following assumption (“sb” stands for “signature-based”).

**sb-MBRB-Assumption.**  $n > 3t + 2d$ .

Table 3 summarizes the acronyms and notations of this chapter.

---

27. Signature-based BRB in only two rounds is a known result [2], however, to the best of our knowledge, no existing BRB algorithm tolerates message adversaries as well as ours.

28. Recall from Section 3.2 that this is the best possible delivery power for MBRB.

**Roadmap** Section 4.1 presents preliminary notions for this chapter. Section 4.2 presents Algorithm 3, a simple signature-based MBRB implementation, whose correctness proof and performance analysis are given in Section 4.3 and Section 4.4, respectively. Finally, Section 4.5 concludes the chapter.

## 4.1 Preliminaries

**Message types** The algorithm uses only one message type, **BUNDLE**, that carries the signatures backing a given value  $v$ , along with  $v$ 's content, sequence number, and emitter. **BUNDLE** messages propagate through the network using controlled flooding.

**Local data structures** Each (correct) process saves locally the valid signatures (*i.e.*, the signed fixed-size digests of some data) that it has received from other processes using **BUNDLE** messages. Each signature “endorses” a certain triplet  $(v, sn, j)$ . When certain conditions are met (described below), a process further broadcasts in a **BUNDLE** message all signatures it knows for a given triplet  $(v, sn, j)$ . A correct process  $p_i$  saves at most one signature for a  $(v, sn, j)$  triplet per signing process  $p_k$ .

**Time measurement** For the proofs related to MBRB-Time-cost (Lemmas 8 to 15), we assume that the duration of local computations is negligible compared to that of message transfer delays, and consider them to take zero time units. As the system is asynchronous, time is measured under the traditional assumption that all messages have the same transfer delay [36].

| Acronyms      | Meaning  |
|---------------|--|
| MA            | Message adversary  |
| MBRB          | MA-tolerant Byzantine reliable broadcast   |
| Notations     | Meaning  |
| $n$           | number of processes in the system  |
| $t$           | upper bound on the number of Byzantine processes                                       |
| $c$           | effective number of correct processes in a run ( $n - t \leq c \leq n$ )               |
| $\lambda$     | security parameter of the cryptographic primitives                                     |
| $\star$       | unspecified value  |
| $\ell_{MBRB}$ | minimal number of correct processes that mbrb-deliver a value                          |
| $\delta$      | time complexity of MBRB  |
| $\mu$         | message complexity of MBRB   |
| $\kappa$      | communication complexity of MBRB<br>(number of bits sent during the execution overall) |

Table 3: Acronyms and notations used in Chapter 4

**Digital signatures** We assume the availability of an asymmetric cryptosystem to sign data (in practice, messages) and verify its authenticity. We assume that signatures are secure and, therefore, that the computing power of the adversary is bounded. Every process in the network has a public/private key pair. We suppose that the public keys are known to everyone and that the private keys are kept secret by their owner. Everyone also knows the mapping between any process' identity  $i$  and its public key. Additionally, we suppose that each process can produce at most one signature per message.

The signatures are used to cope with the net effect of the Byzantine processes, and the fact that messages broadcast (sent) by correct processes can be eliminated by the message adversary. A noteworthy advantage of signatures is that, despite the unauthenticated nature of the point-to-point communication channels, signatures allow correct processes to verify the authenticity of messages that have not been directly received from their initial sender but rather relayed through intermediary processes. Signatures provide us with a *network-wide* non-repudiation mechanism: if a Byzantine process issues two conflicting messages to two different subsets of correct processes, then the correct processes can detect the malicious behavior by disclosing to each other the Byzantine signed messages.<sup>29</sup>

**Size of values and signatures** To analyse the communication cost of Algorithm 3 (Lemma 17), we must characterize the size (in bits) of the structures used in the algorithm, in particular values and digital signatures. We denote by  $|v|$  the size of a value  $v$  disseminated through the MBRB abstraction. We further assume that the size of signatures is linear in the security parameter  $\lambda$  of the cryptographic primitives (see Section 2.1.3), *i.e.*, each signature has  $O(\lambda)$  bits. Lastly, as traditionally assumed, we assume that  $\lambda = \Omega(\log n)$ .

## 4.2 Algorithm

At a high level, Algorithm 3 works by producing, forwarding, and accumulating *witnesses* of an initial `mbrb_broadcast` operation until a large enough quorum is observed by at least one correct process, which propagates this quorum of signatures in one final unreliable `broadcast` operation.

Witnesses take the form of signatures for a given triplet  $(v, sn, i)$ , where  $v$  is the value,  $sn$  its associated sequence number, and  $i$  the identity of the sender  $p_i$  (which also produces a signature for  $(v, sn, i)$ ). Signatures serve to ascertain the provenance and authenticity of these propagated `BUNDLE` messages, thus providing a key ingredient to tolerate the limited reliability of the underlying network. They also authenticate the invoker of the `mbrb_broadcast` operation. Finally, in the last phase of the algorithm, they allow the propagation of a cryptographic proof that a quorum has been reached, thereby ensuring that enough correct processes eventually `mbrb-deliver` the value that was `mbrb-broadcast`.

---

29. The fact that the algorithm uses signed messages does not mean that MBRB requires signatures to be implemented, see Chapter 5.

```

1 operation mbrb_broadcast( $v, sn$ ) is
2   | save signature for ( $v, sn, i$ ) by  $p_i$ ;
3   | broadcast BUNDLE( $v, sn, i, \{\text{all saved signatures for } (v, sn, i)\}$ ).

4 when BUNDLE( $v, sn, j, sigs$ ) is received do
5   | if  $p_i$  already mbrb-delivered some  $(\star, sn, j)$  then return;
6   | if  $sigs$  does not contain the valid signature of ( $v, sn, j$ ) by  $p_j$  then return;
7   | save all unsaved valid signatures for ( $v, sn, j$ ) of  $sigs$ ;
8   | if  $p_i$  did not already sign some  $(\star, sn, j)$  then
9   |   | save signature for ( $v, sn, j$ ) by  $p_i$ ;
10  |   | broadcast BUNDLE( $v, sn, j, \{\text{all saved signatures for } (v, sn, j)\}$ );
11  | if strictly more than  $\frac{n+t}{2}$  signatures for ( $v, sn, j$ ) are saved then
12  |   | broadcast BUNDLE( $v, sn, j, \{\text{all saved signatures for } (v, sn, j)\}$ );
13  |   | mbrb_deliver ( $v, sn, j$ ).

```

Algorithm 3: A signature-based implementation of the MBRB communication abstraction ( $n > 3t + 2d$ , code for  $p_i$ )

In more detail, when a (correct) process  $p_i$  invokes `mbrb_broadcast( $v, sn$ )`, it builds and signs the triplet  $(v, sn, i)$  to guarantee its non-repudiation, and saves locally the resulting signature (line 2). Next,  $p_i$  broadcasts the `BUNDLE` message containing the signature it just produced (line 3).

When a correct process  $p_i$  receives a `BUNDLE( $v, sn, j, sigs$ )` message, it first checks that no value has already been mbrb-delivered for the given sequence number  $sn$  and sender  $p_j$  (line 5), and if the sender  $p_j$  signed the value (line 6). If this condition is satisfied,  $p_i$  saves all the new valid signatures inside the  $sigs$  set (line 7). Next,  $p_i$  creates and saves its own signature for  $(v, sn, j)$  and then broadcasts it in a `BUNDLE` message, if it has not already done so previously (lines 8-10). Finally, if  $p_i$  has saved a quorum of strictly more than  $\frac{n+t}{2}$  signatures for the same triplet  $(v, sn, j)$ , it broadcasts a `BUNDLE` message containing all these signatures and mbrb-delivers the triplet (lines 11-13).<sup>30</sup>

**Remark** The reader can notice that the system parameters  $n$  and  $t$  appear in the algorithm, whereas the system parameter  $d$  does not. Naturally, they all explicitly appear in the proof.

### 4.3 Correctness proof of Algorithm 3

This section proves the correctness properties of MBRB.

30. The pseudo-code presented in Algorithm 3 favors readability and is therefore not fully optimized. For instance, in some cases, a process might unreliably broadcast exactly the same content at lines 8 and 12. This could be avoided by using an appropriate flag or tracking and preventing the repeated broadcast of identical `BUNDLE` messages.

**Theorem 2** (MBRB-Correctness). *If sb-MBRB-Assumption is satisfied, Algorithm 3 implements MBRB with the guarantee  $\ell_{\text{MBRB}} = c - d$ .*

The proof follows from the next lemmas.

**Lemma 2** (MBRB-Validity). *If a correct process  $p_i$  mbrb-delivers  $v$  from a correct process  $p_j$  with sequence number  $sn$ , then  $p_j$  has previously mbrb-broadcast  $v$  with sequence number  $sn$ .*

**Proof.** If a correct process  $p_i$  mbrb-delivers  $(v, sn, j)$  (where  $p_j$  is correct) at line 13, then it has passed the condition at line 6, which means that it must have witnessed a valid signature for  $(v, sn, j)$  by  $p_j$ . Since signatures are secure, the only way to create this signature is for  $p_j$  to execute the instruction at line 2, during the `mbrb_broadcast`( $v, sn$ ) invocation.  $\square$

**Lemma 3** (MBRB-No-duplication). *A correct process  $p_i$  mbrb-delivers at most one value from a process  $p_j$  with a given sequence number  $sn$ .*

**Proof.** This property derives trivially from the condition at line 5.  $\square$

**Lemma 4** (MBRB-No-duplicity). *No two different correct processes mbrb-deliver different values from a process  $p_i$  with the same sequence number  $sn$ .*

**Proof.** Let us consider two correct processes  $p_a$  and  $p_b$  which respectively mbrb-deliver  $(v, sn, i)$  and  $(v', sn, i)$ . Due to the condition at line 11,  $p_a$  and  $p_b$  must have saved (and thus received) two sets  $Q_a$  and  $Q_b$  containing strictly more than  $\frac{n+t}{2}$  signatures for  $(v, sn, i)$  and  $(v', sn, i)$ , respectively. We thus have  $|Q_a| > \frac{n+t}{2}$  and  $|Q_b| > \frac{n+t}{2}$ .

As, for any two sets  $A$  and  $B$ , we have  $|A \cap B| = |A| + |B| - |A \cup B| \geq |A| + |B| - n > 2 \times \frac{n+t}{2} - n = t$ ,  $A$  and  $B$  have at least one correct process  $p_k$  in common, which must have signed both  $(v, sn, i)$  and  $(v', sn, i)$ . But before signing  $(v, sn, i)$  at line 2 or at line 9,  $p_k$  checks that it did not sign a different value from the same sender and with the same sequence number, whether it be implicitly during a `mbrb_broadcast`( $v, sn$ ) invocation or at line 8. Thereby,  $v$  is necessarily equal to  $v'$ .  $\square$



**Lemma 5** (MBRB-Local-delivery). *If a correct process  $p_i$  mbrb-broadcasts a value  $v$  with sequence number  $sn$ , then at least one correct process  $p_j$  mbrb-delivers  $m$  from  $p_i$  with sequence number  $sn$ .*

**Proof.** If a correct process  $p_i$  mbrb-broadcasts  $(v, sn)$ , then it broadcasts its own signature  $sig_i$  for  $(v, sn, i)$  in a  $\text{BUNDLE}(v, sn, i, \{sig_i\})$  message at line 3. As  $p_i$  is correct, it does not sign another triplet  $(v', sn, i)$  where  $v' \neq m$ , therefore it is impossible for a correct process to mbrb-deliver  $(v', sn, i)$  at line 13, because it cannot pass the condition at line 6.

Let us denote by  $K$  the set of correct processes that receive a message  $\text{BUNDLE}(v, sn, i, \{sig_i, \dots\})$  at least once. Note that because  $p_i$  executes line 3, by definition of the message adversary,  $K$  contains at least  $c - d$  processes. The assumption  $n > 3t + 2d$  further yields that  $c - d > 2t + d \geq 0$ , and therefore  $K \neq \emptyset$ . The first one of such  $\text{BUNDLE}$  messages that a process of  $K$  receives can be the one  $p_i$  initially broadcast at line 3, but it can also be a  $\text{BUNDLE}$  message broadcast by a correct process at line 10 or at line 12, or it can even be a  $\text{BUNDLE}$  message sent by a Byzantine process. In any case, the first time the processes of  $K$  receive such a  $\text{BUNDLE}$  message, they pass the conditions at lines 5 and 6, and they also pass the condition at line 8, except for  $p_i$  if it belongs to  $K$ . Consequently, each process  $p_k$  of  $K$  necessarily broadcasts its own signature  $sig_k$  for  $(v, sn, i)$  in a  $\text{BUNDLE}(v, sn, i, \{sig_k, sig_i, \dots\})$  message.

By construction of the algorithm, the set  $K$  of correct processes that ever receive a  $\text{BUNDLE}(v, sn, i, \{sig_i, \dots\})$  message is therefore included into the set  $K'$  of correct processes  $p_k$  that ever broadcast a  $\text{BUNDLE}(v, sn, i, \{sig_k, sig_i, \dots\})$ ,  $K \subseteq K'$ . This inclusion in turn implies

$$|K| \leq |K'|. \quad (1)$$

By the definition of the message adversary, a message  $\text{BUNDLE}(v, sn, i, \{sig_k, sig_i, \dots\})$  broadcast by a correct process  $p_k \in K'$  is eventually received by at least  $c - d$  correct processes. Because  $K$  is the set of processes that ever receives  $sig_i$  in a  $\text{BUNDLE}$  message, these  $c - d$  correct processes belong to  $K$  by construction. Hence, the minimum number of signatures for  $(v, sn, i)$  made by processes of  $K'$  that are also received by processes of  $K$  globally is  $|K'|(c - d)$ . Using  $K \neq \emptyset$  and therefore  $|K| \neq 0$ , it follows that a given process of  $K$  individually receives on average the distinct signatures of at least  $|K'| \frac{c-d}{|K|}$  processes of  $K'$ .

Using (1) yields  $|K'| \frac{c-d}{|K|} \geq c - d$ . From sb-MBRB-Assumption, we have  $n > 3t + 2d \iff 2n > n + 3t + 2d \iff 2n - 2t - 2d > n + t \iff c - d \geq n - t - d > \frac{n+t}{2}$  (as  $n - t \leq c$ ). As a result, by the pigeonhole principle, at least one process  $p_j$  of  $K$  (ergo one correct process) receives a set  $S$  (in possibly multiple  $\text{BUNDLE}$  messages) of strictly more than  $\frac{n+t}{2}$  valid distinct signatures for  $(v, sn, i)$ . When  $p_j$  receives the last signature of  $S$ , there are two cases:

- Case if  $p_j$  does not pass the conditions at lines 5 and 6.

As processes of  $K$  are correct, then when they broadcast a `BUNDLE`( $v, sn, i, sigs$ ) message, they necessarily include  $sig_i$  in  $sigs$ , which implies that  $sig_i$  is necessarily in  $S$ . Therefore, if  $p_j$  does not pass the condition at line 5, it is because  $p_j$  already mbrb-delivered some  $(\star, sn, i)$ . But let us remind that, as  $p_i$  is correct, it is impossible for  $p_j$  to mbrb-deliver anything different from  $(v, sn, i)$ . Therefore,  $p_j$  has already mbrb-delivered  $(v, sn, i)$ .

- Case if  $p_j$  passes the conditions at lines 5 and 6.

Process  $p_j$  then saves all signatures of  $S$  at line 7, and after it passes the condition at line 11 (as  $|S| > \frac{n+t}{2}$ ) and finally mbrb-delivers  $(v, sn, i)$  at line 13.  $\square$

**Lemma 6** (MBRB-Global-delivery). *If a correct process  $p_i$  mbrb-delivers a value  $v$  from  $p_j$  with sequence number  $sn$ , then at least  $\ell_{MBRB} = c - d$  correct processes mbrb-deliver  $v$  from  $p_j$  with sequence number  $sn$ .*

**Proof.** If a correct process  $p_i$  mbrb-delivers  $(v, sn, j)$  at line 13, it must have saved a set  $sigs$  of strictly more than  $\frac{n+t}{2}$  valid distinct signatures because of the condition at line 11. Let us remark that  $sigs$  necessarily contains the signature for  $(v, sn, j)$  by  $p_j$  because of the condition at line 6. Additionally,  $p_i$  must also have broadcast `BUNDLE`( $v, sn, i, sigs$ ) at line 12, that, by definition of the message adversary, is received by a set  $K$  of at least  $c - d$  correct processes. For each process  $p_k$  of  $K$ , we have the following.

- If  $p_k$  does not pass the conditions at lines 5 and 6, it is necessarily because it has already mbrb-delivered some  $(\star, sn, j)$  at line 13. But because of MBRB-No-duplication,  $p_k$  has necessarily mbrb-delivered  $(v, sn, j)$ .
- If  $p_k$  passes the conditions at lines 5 and 6, then it saves all signatures of  $sigs$  at line 7 and then passes the condition at line 11 and finally mbrb-delivers  $(v, sn, j)$  at line 13.

Therefore, all processes of  $K$  (which, as a reminder, are at least  $c - d = \ell_{MBRB}$ ) necessarily mbrb-deliver  $(v, sn, j)$  at line 13.  $\square$

**Remark** Neither Lemmas 2, 3, 4 or 6 use the assumption  $n > 3t + 2d$  (sb-MBRB-Assumption). As a result, if Algorithm 3 is used in a situation when the message adversary can partition the system ( $n \leq 2d$ ), the safety properties of the algorithm (MBRB-Validity, MBRB-No-duplication, and MBRB-No-duplication) and the MBRB-Global-delivery property continue to hold. In case of partition, however, the MBRB-Local-delivery property might get violated, as a value mbrb-broadcast by a correct process might fail to gather a quorum of signatures at line 11 to trigger mbrb-delivery at line 13.

---

## 4.4 Theoretical performance analysis of Algorithm 3

In this section, we show the following theorem.

**Theorem 3** (MBRB-Performance). *If sb-MBRB-Assumption is satisfied and the sender is correct, Algorithm 3 provides the following MBRB guarantees:*

- $\delta = \left. \begin{array}{l} 2 \text{ if } d=0, \text{ else} \\ 3 \text{ if } d < c - \sqrt{c \times \frac{n+t}{2}}, \text{ else} \\ 4 \text{ if } d < c - \frac{(n+t+2c)^2}{16c}, \text{ else} \\ 5 \text{ otherwise} \end{array} \right\} \text{ communication rounds,}$
- $\mu = 2n^2$  messages sent overall,
- $\kappa = O(n^2|v| + n^3\lambda)$  bits sent overall.

The proof of this theorem follows from the subsequent lemmas. For analyzing the time cost of our asynchronous algorithm, *i.e.*, the number of *communication rounds* it requires to terminate, let us remember from Section 3.2 that we rely on the common approach which assumes that the algorithm executes in a synchronous environment [36].

**Lemma 7.**  $c - d > \frac{n+t}{2}$ .

**Proof.** We have the following:

$$\begin{aligned}
 c - d &\geq n - t - d = \frac{2n - 2t - 2d}{2} && \text{(by definition of } c) \\
 &> \frac{n + 3t + 2d - 2t - 2d}{2} && \text{(by sb-MBRB-Assumption)} \\
 &> \frac{n + t}{2}. && \square
 \end{aligned}$$

**Lemma 8.** *If a correct process  $p_i$  mbrb-broadcasts  $(v, sn)$ , then at least  $c - d - \left\lfloor d - \frac{\lfloor \frac{n+t}{2} \rfloor}{c - d - \lfloor \frac{n+t}{2} \rfloor} \right\rfloor$  correct processes mbrb-deliver  $(v, sn, i)$  at most two rounds later.*

**Proof.** If a correct process  $p_i$  mbrb-broadcasts  $(v, sn)$ , then it broadcasts its own signature  $sig_i$  for  $(v, sn, i)$  in a  $\text{BUNDLE}(v, sn, i, \{sig_i\})$  message at line 3. Let us denote by  $L_1$  the set of correct processes that receive this  $\text{BUNDLE}(v, sn, i, \{sig_i\})$  message from  $p_i$  by the end of the

first round, and let  $\ell_1$  be the number of processes in  $L_1$ , such that  $c - d \leq \ell_1 = |L_1| \leq c$  (by definition of the message adversary). By construction of the algorithm, every process  $p_x$  of  $L_1$  passes the condition at line 6, and therefore broadcasts a  $\text{BUNDLE}(v, sn, i, \{sig_x, sig_i\})$  message, whether it be at line 3 for  $p_i$ , or at line 10 for any other process of  $L_1$ .

Let  $A$  and  $B$  define two partitions of the set of all correct processes ( $A \cup B$  is the set of all correct processes, and  $A \cap B = \emptyset$ ).  $A$  denotes the set of correct processes that receive strictly more than  $\frac{n+t}{2}$  signatures for  $(v, sn, i)$  from processes of  $L_1$  two rounds after  $p_i$  mbrb-broadcast  $(v, sn)$ , while  $B$  denotes the set of remaining correct processes of  $L_1$  that receive at most  $\frac{n+t}{2}$  signatures for  $(v, sn, i)$  from processes of  $L_1$  two rounds after  $p_i$  mbrb-broadcast  $(v, sn)$ . Let  $a$  be the size of  $A$ :  $a = |A|$ . By construction,  $|B| = c - a$ . Let  $s_A$  and  $s_B$  respectively denote the number of signatures for  $(v, sn, i)$  from processes of  $L_1$  received by processes of  $A$  and  $B$  at most two rounds after  $p_i$  mbrb-broadcast  $(v, sn)$ . Figure 6 represents the distribution of such signatures among the processes of  $L_1$ , sorted by decreasing number of signatures received. Each processes of  $A$  can receive at most  $\ell_1$  signatures (that is, all signatures) from processes of  $L_1$ , while each process of  $B$  can receive at most  $\lfloor \frac{n+t}{2} \rfloor$  signatures from processes of  $L_1$  two rounds after  $p_i$  mbrb-broadcasts  $(v, sn)$ . In some parts of this proof, we use  $q$  as a shorthand for  $\lfloor \frac{n+t}{2} \rfloor$  for concision.

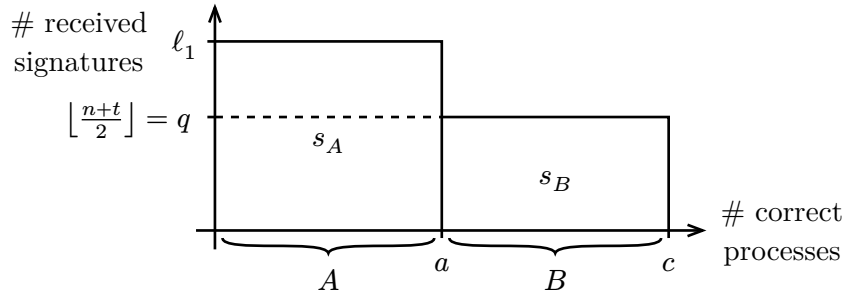


Figure 6: Distribution of signatures among processes of  $A$  and  $B$  two rounds after  $p_i$  mbrb-broadcast  $(v, sn)$

From these observations, we infer the following inequalities:

$$\begin{aligned} \ell_1 a &\geq s_A, \\ (c - a)q &\geq s_B. \end{aligned}$$

By the definition of the message adversary, a  $\text{BUNDLE}(v, sn, i, \{sig_x, sig_i\})$  message broadcast by a correct process  $p_x$  is eventually received by at least  $c - d$  correct processes. As a consequence, in total, the minimum number of signatures for  $(v, sn, i)$  collectively received by correct processes as a result of broadcasts by processes in  $L_1$  in the first two asynchronous rounds is  $\ell_1(c - d)$ . We thus have:

$$s_A + s_B \geq \ell_1(c - d).$$

By combining the previous inequalities, we obtain:

$$\begin{aligned} \ell_1 a + (c - a)q &\geq \ell_1(c - d), \\ \ell_1 a + cq - aq &\geq \ell_1(c - d), \\ \ell_1 a - aq &\geq \ell_1(c - d) - cq, \\ a(\ell_1 - q) &\geq \ell_1(c - d) - cq. \end{aligned} \tag{2}$$

By Lemma 7, we know that  $\ell_1 \geq c - d > \lfloor \frac{n+t}{2} \rfloor = q$ , so we can rewrite (2) into:

$$a \geq \frac{\ell_1(c - d) - cq}{\ell_1 - q}. \tag{3}$$

To find the lowest guaranteed value for  $a$  depending on  $\ell_1 \in [c - d, c]$ , let us study the derivative on  $\ell_1$  of the right-hand side of (3):

$$\begin{aligned} \frac{\partial(\ell_1(c - d) - cq)/(\ell_1 - q)}{\partial \ell_1} &= \frac{(c - d)(\ell_1 - q) - (\ell_1(c - d) - cq)}{(\ell_1 - q)^2}, \\ &= \frac{(c - d)(\ell_1 - q) - \ell_1(c - d) + cq}{(\ell_1 - q)^2} = \frac{qc - q(c - d)}{(\ell_1 - q)^2} = \frac{qd}{(\ell_1 - q)^2}. \end{aligned}$$

As  $q$  and  $d$  are by definition positive, we know that the above derivative is positive, or null when  $d = 0$ . Therefore,  $\frac{\ell_1(c-d)-cq}{\ell_1-q}$  is monotonically increasing on  $\ell_1 \in [c - d, c]$ , and its minimum value can be found when  $\ell_1$  is also minimum, that is, when  $\ell_1 = c - d$ . Thus, when we replace  $\ell_1$  by  $c - d$  in (3), we obtain:

$$\begin{aligned} a &\geq \frac{(c - d)(c - d) - cq}{c - d - q} = \frac{(c - d)(c - d - q) - qd}{c - d - q}, \\ &\geq c - d - \frac{qd}{c - d - q}. \end{aligned} \tag{4}$$

Let us denote by  $a_{\min}$  the lower bound on the number of correct processes that receive a quorum of strictly more than  $\frac{n+t}{2}$  valid distinct signatures for  $(v, sn, i)$  two rounds after  $p_i$  mbrb-broadcast  $(v, sn)$ , such that  $a_{\min} \leq a = |A|$ . As the right-hand side of (4) is not always an integer, we have:

$$\begin{aligned} a_{\min} &= \left\lceil c - d - \frac{qd}{c - d - q} \right\rceil = c - d + \left\lceil -\frac{qd}{c - d - q} \right\rceil, \\ &= c - d - \left\lfloor \frac{qd}{c - d - q} \right\rfloor, \end{aligned} \quad (\text{as } \forall x \in \mathbb{R}, \lceil -x \rceil = -\lfloor x \rfloor)$$

$$= c - d - \left\lfloor \frac{d \lfloor \frac{n+t}{2} \rfloor}{c - d - \lfloor \frac{n+t}{2} \rfloor} \right\rfloor. \quad (\text{by definition of } q)$$

Hence, at least  $a_{\min} = c - d - \left\lfloor \frac{d \lfloor \frac{n+t}{2} \rfloor}{c - d - \lfloor \frac{n+t}{2} \rfloor} \right\rfloor$  processes of  $L_1$  receive strictly more than  $\frac{n+t}{2}$  valid distinct signatures for  $(v, sn, i)$  two rounds after  $p_i$  mbrb-broadcasts  $(v, sn)$ . For every process  $p_y$  of  $A$ :

- If  $p_y$  does not pass the condition at lines 5 and 6 after receiving the last signature of the quorum in a `BUNDLE` message, it is necessarily because  $p_y$  already mbrb-delivered some  $(\star, sn, i)$ , since processes of  $L_1$  are correct and all their `BUNDLE` messages include the signature for  $(v, sn, i)$  by  $p_i$ . But let us remind that, as the sender  $p_i$  is correct, it is impossible for  $p_y$  to mbrb-deliver anything different from  $(v, sn, i)$ . Therefore,  $p_y$  has already mbrb-delivered  $(v, sn, i)$  at line 13.
- If  $p_y$  passes the condition at lines 5 and 6 after processing the last `BUNDLE` $(v, sn, i, \{sig_i, sig_x\})$  message of the quorum from a process  $p_x$ , then  $p_y$  saves the signature  $sig_x$  at line 7, and after it passes the condition at line 11 (as it has saved strictly more than  $\frac{n+t}{2}$  signatures) and finally mbrb-delivers  $(v, sn, i)$  at line 13.

Therefore, all processes of  $A$ , which are at least  $a_{\min} = c - d - \left\lfloor \frac{d \lfloor \frac{n+t}{2} \rfloor}{c - d - \lfloor \frac{n+t}{2} \rfloor} \right\rfloor$ , mbrb-deliver  $(v, sn, i)$  at line 13 at most two rounds after  $p_i$  mbrb-broadcast  $(v, sn)$ .  $\square$

**Analysis of subsequent communication rounds** In the following, we use another approach to find the conditions on the message adversary power,  $d$ , for which Algorithm 3 terminates in 3, 4, or 5 rounds at the latest. Namely, we focus on the set of correct processes that receive the sender's signature by the end of round 2, and we analyze the average number of distinct signatures received by this set of processes by the end of rounds 2, 3, and 4. Finally, we rely on the pigeonhole principle to show that, if the average of some round  $x$  is greater than the quorum threshold  $\frac{n+t}{2}$ , then at least one correct process has passed this threshold, and can therefore forward this quorum of signatures to at least  $\ell_{MBRB} = c - d$  correct processes, which will then mbrb-deliver the value in round  $x + 1$ . Interestingly, we show that, if the MBRB sender is correct, Algorithm 3 terminates in at most 5 rounds, even in the worst possible case.

Let  $L_1, L_2, L_3$ , and  $L_4$  denote the set of correct processes that receive the sender's signature for the first time at the latest by the end of rounds 1, 2, 3 and 4, respectively, and therefore broadcast their own signature at the start of the following round. We have  $L_1 \subseteq L_2 \subseteq L_3 \subseteq L_4$ .

To show that  $\ell_{MBRB}$  correct processes deliver the value at the latest during some communication round  $x \in \{3, 4, 5\}$ , let us remark that it is sufficient to prove that there exists some correct process  $p_j$  which observed a quorum of signatures at the latest during the previous round  $x - 1$ , as  $p_j$  then disseminates this quorum of signatures to at least  $\ell_{MBRB} = c - d$  other correct

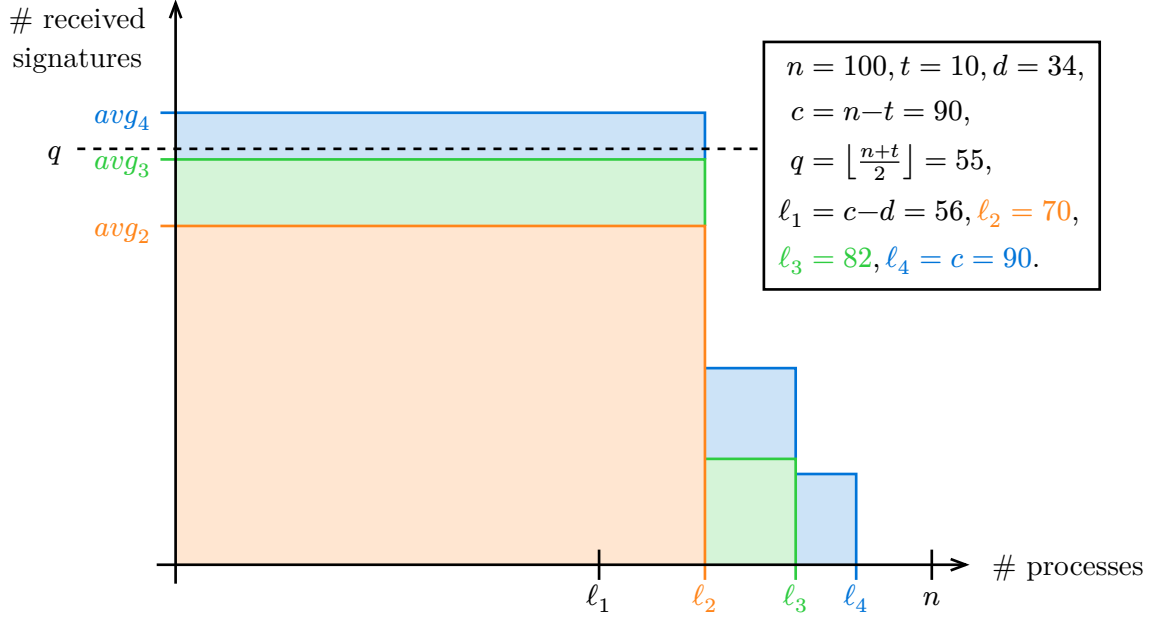


Figure 7: Worst-case distribution of signatures received by correct processes ( $n = 100$ ), after a `mbrb_broadcast()` execution by a correct process, at the end of rounds 2 (in orange), 3 (in green), and 4 (in blue)

processes at line 12. Hence, in the following developments, we analyze the conditions on  $d$  under which such a correct process exists in  $L_2$ .

The variables  $avg_2$ ,  $avg_3$ , and  $avg_4$  denote the average number of signatures received by processes of  $L_2$  by the end of rounds 2, 3 and 4, respectively. Let  $l_1 = |L_1|$ ,  $l_2 = |L_2|$ ,  $l_3 = |L_3|$  and  $l_4 = |L_4|$ . Assuming that a correct process  $p_i$  `mbrb-broadcasts`  $(v, sn)$  and that  $n = 100$ , Figure 7 represents the worst-case distribution of signatures for  $(v, sn)$  that are received by correct processes by the end of rounds 2, 3, and 4. We have:

$$c - d \leq l_1 \leq l_2 \leq l_3 \leq l_4 \leq c. \quad (5)$$

Figure 7 illustrates the special case  $n = 100$ ,  $t = 10$ ,  $d = 34$ ,  $c = n - t$ ,  $l_1 = c - d$ ,  $l_2 = 70$ ,  $l_3 = 82$ , and  $l_4 = c$ . Moreover, the orange, green, and blue areas represent the sets of signatures sent and received during round 2, 3, and 4, respectively. As before, we note  $q = \lfloor \frac{n+t}{2} \rfloor$  the (exclusive) quorum threshold. We can understand from this illustration how to find lower bounds for the values of  $avg_2$ ,  $avg_3$ , and  $avg_4$ .

**$avg_2$  lower bound** First,  $avg_2$  is the average number of signatures sent by processes of  $L_1$  that are received by the processes of  $L_2$ . The set of all signatures sent and received during round 2 corresponds to the orange area of Figure 7. By the definition of the MA, we have the following lower bound:

$$avg_2 \geq \frac{\ell_1(c-d)}{\ell_2}. \quad (6)$$

**$avg_3$  lower bound** Similarly,  $avg_3$  equals the average number of signatures received by processes of  $L_2$  during round 2 (*i.e.*,  $avg_2$ ) plus the average number of signatures sent by processes of  $L_2 \setminus L_1$  (which received the sender’s signature for the first time during round 2) that are received by processes of  $L_2$ . In the worst case, the adversary uses the processes of  $L_3 \setminus L_2$  to “absorb” the maximum amount of signatures sent during this round (*i.e.*, each process of  $L_3 \setminus L_2$  receives the signature of every process of  $L_2 \setminus L_1$ ). However, the remaining signatures must still be distributed among the processes of  $L_2$ , hence  $avg_3 \geq avg_2$ . The set of all signatures sent and received during round 3 is shown as a green area in Figure 7. Combining all the constraints just discussed on this set of signatures yields the following lower bound on  $avg_3$ :

$$avg_3 \geq avg_2 + \frac{(\ell_2 - \ell_1)(c-d) - (\ell_3 - \ell_2)(\ell_2 - \ell_1)}{\ell_2}. \quad (7)$$

**$avg_4$  lower bound** Finally,  $avg_4$  equals  $avg_3$  plus the average number of signatures sent by processes of  $L_3 \setminus L_2$  that are received by processes of  $L_2$ . Similarly, in the worst case, the adversary uses all processes of  $L_4 \setminus L_2$  to “absorb” the maximum amount of signatures sent during round 3 (*i.e.*, each process of  $L_4 \setminus L_2$  receives the signature of every process of  $L_3 \setminus L_2$ ). The set of all signatures sent and received during round 4 corresponds to the blue area of Figure 7. The previous constraints yield the following lower bound:

$$avg_4 \geq avg_3 + \frac{(\ell_3 - \ell_2)(c-d) - (\ell_4 - \ell_2)(\ell_3 - \ell_2)}{\ell_2}. \quad (8)$$

We now analyze in Lemma 9 to Lemma 13 the sufficient conditions on  $d$  for which  $avg_2$ ,  $avg_3$ , and  $avg_4$  are strictly above the quorum threshold  $q = \frac{n+t}{2}$ . These sufficient conditions on  $d$  can then be reused to ensure delivery in the round that follows: informally, Lemma 14 shows that, if  $avg_x > q$  where  $x \in \{2, 3, 4\}$ , then Algorithm 3 terminates at the latest in round  $x + 1$ .

**Lemma 9.**  $avg_2 \geq \frac{(c-d)^2}{c}$ .

**Proof.** We have the following:

$$\begin{aligned} avg_2 &\geq \frac{\ell_1(c-d)}{\ell_2} && \text{(by (6))} \\ &\geq \frac{(c-d)(c-d)}{\ell_2} && \text{(by (5), and as } c-d \geq 0) \end{aligned}$$



$$= \frac{(c-d)^2}{\ell_2}.$$

Moreover, we can see that  $\frac{(c-d)^2}{\ell_2}$  is minimal when  $\ell_2$  is maximal, *i.e.*, when  $\ell_2 = c$ . Therefore,  $avg_2 \geq \frac{(c-d)^2}{c}$ .  $\square$

**Lemma 10.** *If  $d < c - \sqrt{c \times \frac{n+t}{2}}$ , then  $avg_2 > \frac{n+t}{2}$ .*

**Proof.** Let us assume  $d < c - \sqrt{c \times \frac{n+t}{2}}$ . Using simple algebraic transformations, we can see that  $d < c - \sqrt{c \times \frac{n+t}{2}} \iff \frac{(c-d)^2}{c} > \frac{n+t}{2}$ . As  $avg_2 \geq \frac{(c-d)^2}{c}$  by Lemma 9, we can conclude that  $avg_2 > \frac{n+t}{2}$ .  $\square$

**Lemma 11.**  $avg_3 \geq \ell_2 - \ell_3 + \frac{\ell_3(c-d)}{\ell_2}$ .

**Proof.** We have the following:

$$\begin{aligned} avg_3 &\geq avg_2 + \frac{(\ell_2 - \ell_1)(c-d) - (\ell_3 - \ell_2)(\ell_2 - \ell_1)}{\ell_2} && \text{(by (7))} \\ &= avg_2 + \frac{(\ell_2 - \ell_1)(c-d - \ell_3 + \ell_2)}{\ell_2} \\ &= c - d - \ell_3 + \ell_2 + avg_2 + \frac{\ell_1(d - c + \ell_3 - \ell_2)}{\ell_2}, \\ &\geq c - d - \ell_3 + \ell_2 + \frac{\ell_1(c-d) + \ell_1(d - c + \ell_3 - \ell_2)}{\ell_2} && \text{(by (6))} \\ &= c - d - \ell_3 + \ell_2 + \frac{\ell_1(\ell_3 - \ell_2)}{\ell_2} \\ &\geq c - d - \ell_3 + \ell_2 + \frac{(c-d)(\ell_3 - \ell_2)}{\ell_2} && \text{(by (5), and as } \ell_3 - \ell_2 \geq 0) \\ &= \ell_2 - \ell_3 + \frac{\ell_3(c-d)}{\ell_2}. \end{aligned} \quad \square$$

**Lemma 12.** *If  $d < c - \frac{(n+t+2c)^2}{16c}$ , then  $avg_3 > \frac{n+t}{2}$ .*

**Proof.** Let us assume  $d < c - \frac{(n+t+2c)^2}{16c}$ . By Lemma 11, we have  $avg_3 \geq \ell_2 - \ell_3 + \frac{\ell_3(c-d)}{\ell_2}$ . To show that the lemma assumption implies  $avg_3 > \frac{n+t}{2}$ , we have to find the minimum of  $avg_3$  depending on the unknown variables  $\ell_2$  and  $\ell_3$ , to prove that even in the worst case this minimum is greater than  $\frac{n+t}{2}$ .

Let us note by  $f$  the lower bound of  $avg_3$ , such that  $f = \ell_2 - \ell_3 + \frac{\ell_3(c-d)}{\ell_2}$ . We first find the partial derivative of  $f$  on  $\ell_3$ :

$$\frac{\partial f}{\partial \ell_3} = -1 + \frac{c-d}{\ell_2} \leq -1 + \frac{c-d}{c-d} \leq 0.$$

Therefore, for a fixed  $\ell_2$ ,  $f$  monotonically decreases when  $\ell_3$  increases, hence  $f$  is minimal when  $\ell_3$  is maximal, *i.e.*, when  $\ell_3 = c$ . Let us note  $g$  the new formula obtained by substituting  $c$  for  $\ell_3$  in  $f$ :

$$avg_3 \geq f \geq g = \ell_2 - c + \frac{c(c-d)}{\ell_2}.$$

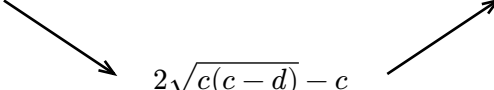
We now find the partial derivative of  $g$  on  $\ell_2$ :

$$\frac{\partial g}{\partial \ell_2} = 1 - \frac{c(c-d)}{\ell_2^2}.$$

Let us analyze the sign of this derivative:

$$\begin{aligned} 1 - \frac{c(c-d)}{\ell_2^2} &\geq 0 \\ \Leftrightarrow 1 &\geq \frac{c(c-d)}{\ell_2^2} \\ \Leftrightarrow \ell_2^2 &\geq (c(c-d)) && \text{(as } \ell_2^2 \geq 0) \\ \Leftrightarrow \ell_2 &\geq \sqrt{c(c-d)}. && \text{(as the square root function is strictly increasing)} \end{aligned}$$

By substituting  $\sqrt{c(c-d)}$  for  $\ell_2$  in  $g$  we get  $\sqrt{c(c-d)} - c + \frac{c(c-d)}{\sqrt{c(c-d)}} = 2\sqrt{c(c-d)} - c$ . We therefore have the following variations.

|                                      |   |                 |       |
|--------------------------------------|---|-----------------|-------|
| $\ell_2$                             | $c$   | $\sqrt{c(c-d)}$ | $c-d$ |
| $\frac{\partial g}{\partial \ell_2}$ | -   | 0               | +     |
| $g$                                  |  $2\sqrt{c(c-d)} - c$ |                 |       |

Hence, we have  $avg_3 \geq g \geq 2\sqrt{c(c-d)} - c$ . Moreover, simple algebraic transformations let us see that  $d < c - \frac{(n+t+2c)^2}{16c} \iff 2\sqrt{c(c-d)} - c > \frac{n+t}{2}$ . Therefore, if the lemma assumption is satisfied, then  $avg_3 > \frac{n+t}{2}$ .  $\square$

**Lemma 13.**  $avg_4 > \frac{n+t}{2}$ .

**Proof.** We have the following:

$$\begin{aligned}
 avg_4 &\geq avg_3 + \frac{(\ell_3 - \ell_2)(c-d) - (\ell_4 - \ell_2)(\ell_3 - \ell_2)}{\ell_2} && \text{(by (8))} \\
 &= avg_3 + \frac{(\ell_3 - \ell_2)(c-d - \ell_4 + \ell_2)}{\ell_2} \\
 &\geq avg_3 + \frac{(\ell_3 - \ell_2)(c-d - c + \ell_2)}{\ell_2} && \text{(by (5), and as } \ell_3 - \ell_2 \geq 0) \\
 &= avg_3 + \frac{(\ell_3 - \ell_2)(\ell_2 - d)}{\ell_2} = \ell_3 - \ell_2 + d + avg_3 - \frac{d\ell_3}{\ell_2} \\
 &\geq d + \frac{\ell_3(c-d) - d\ell_3}{\ell_2} && \text{(by Lemma 11)} \\
 &= d + \frac{\ell_3(c-2d)}{\ell_2}.
 \end{aligned}$$

Let us remark that the minimum of  $d + \frac{\ell_3(c-2d)}{\ell_2}$  can be found when  $\ell_2$  is maximal and  $\ell_3$  is minimal. By (5), as we have  $\ell_2 \leq \ell_3$ , this minimum is reached when  $\ell_2 = \ell_3$ . Hence,  $avg_4 \geq d + \frac{\ell_3(c-2d)}{\ell_2} \geq d + \frac{\ell_2(c-2d)}{\ell_2} = c-d$ , which is strictly greater than  $\frac{n+t}{2}$  by Lemma 7.  $\square$

**Lemma 14.** If a correct process  $p_i$  mbrb-broadcasts  $(v, sn)$  and  $avg_x > \frac{n+t}{2}$  for  $x \in \{2, 3, 4\}$ , then at least  $c-d$  correct processes mbrb-deliver  $(v, sn, i)$  at most  $x+1$  rounds later.

**Proof.** Let us assume that a correct process  $p_i$  mbrb-broadcasts  $(v, sn)$  and that  $avg_x > \frac{n+t}{2}$  for some  $x \in \{2, 3, 4\}$ . If  $avg_x$  is strictly greater than the quorum threshold  $\frac{n+t}{2}$ , then, by the pigeonhole principle, some correct process  $p_j \in L_2$  receives a **BUNDLE** $(v, sn, i, sigs)$  message at line 4 by the end of round  $x$ , where  $sigs$  contains at least a quorum of signatures. We prove that, in any case,  $p_j$  must broadcast a quorum of signatures at line 12.

- Case 1:  $p_j$  already mbrb-delivered some value  $v'$  with sequence number  $sn$  from  $p_i$ . By MBRB-No-duplcity, we have  $v' = v$ . Therefore, before mbrb-delivering  $v$  with  $sn$  from  $p_i$  at line 13,  $p_j$  must have broadcast a **BUNDLE** $(v, sn, i, sigs')$  message where  $sigs$  contains at least a quorum of signatures at line 12.
- Case 2:  $p_j$  did not already mbrb-deliver any value with sequence number  $sn$  from  $p_i$ . Then, after receiving the **BUNDLE** $(v, sn, i, sigs)$  message,  $p_j$  saves all valid signatures of  $sigs$  at line 7, passes the condition at line 11 (as  $|sigs| > \frac{n+t}{2}$ ), and forward this quorum of signatures to the network in a **BUNDLE** message at line 12.

Therefore,  $p_i$  necessarily broadcasts a **BUNDLE** message containing a quorum of signatures at line 12. By the definition of the message adversary, this **BUNDLE** message is received by at least  $c - d$  correct processes at line 4, by the end of round  $x + 1$ . All these  $c - d$  correct processes then save all signatures in  $sigs$  at line 7, pass the condition at line 11 and mbrb-deliver  $(v, sn, i)$  at line 13 (as MBRB-No-duplcity ensures no other value  $v' \neq v$  is mbrb-delivered for this sender  $p_i$  and sequence number  $sn$ ).  $\square$

**Lemma 15** (MBRB-Time-cost). *In Algorithm 3, if a correct process  $p_i$  mbrb-broadcasts a value  $v$  with sequence number  $sn$ , then  $\ell_{MBRB} = c - d$  correct processes mbrb-deliver  $v$  from  $p_i$  with sequence number  $sn$  at most  $\delta$  communication rounds later, where:*

$$\delta = \left\{ \begin{array}{l} 2 \text{ if } d = 0, \text{ else} \\ 3 \text{ if } d < c - \sqrt{c \times \frac{n+t}{2}}, \text{ else} \\ 4 \text{ if } d < c - \frac{(n+t+2c)^2}{16c}, \text{ else} \\ 5 \text{ otherwise} \end{array} \right\}.$$

**Proof.** Let us consider a correct process  $p_i$  that mbrb-broadcasts  $(v, sn)$ . By exhaustion:

- Case where  $d = 0$ .

By Lemma 8, at least  $c - d - \left\lfloor \frac{d \lfloor \frac{n+t}{2} \rfloor}{c - d - \lfloor \frac{n+t}{2} \rfloor} \right\rfloor$  correct processes mbrb-deliver  $(v, sn, i)$  two rounds after  $p_i$  has mbrb-broadcast  $(v, sn)$ . By replacing  $d$  by 0, in this formula, we obtain that

all  $c$  correct processes mbrb-deliver  $(v, sn, i)$  at most 2 rounds after  $p_i$  has mbrb-broadcast  $(v, sn)$ .

- Case where  $d < c - \sqrt{c \times \frac{n+t}{2}}$ .

If  $d < c - \sqrt{c \times \frac{n+t}{2}}$ , Lemma 10 applies and thus  $avg_2 > \frac{n+t}{2}$ . Then, Lemma 14 states that at least  $\ell_{MBRB} = c - d$  correct processes mbrb-deliver  $(v, sn, i)$  at most 3 rounds after  $p_i$  has mbrb-broadcast  $(v, sn)$ .

- Case where  $d < c - \frac{(n+t+2c)^2}{16c}$ .

If  $d < c - \frac{(n+t+2c)^2}{16c}$ , Lemma 12 applies and thus  $avg_3 > \frac{n+t}{2}$ . Then, Lemma 14 states that at least  $\ell_{MBRB} = c - d$  correct processes mbrb-deliver  $(v, sn, i)$  at most 4 rounds after  $p_i$  has mbrb-broadcast  $(v, sn)$ .

- Remaining case.

Lemma 13 always applies and thus  $avg_4 > \frac{n+t}{2}$ . Then, Lemma 14 states that at least  $\ell_{MBRB} = c - d$  correct processes mbrb-deliver  $(v, sn, i)$  at most 5 rounds after  $p_i$  has mbrb-broadcast  $(v, sn)$ .  $\square$

**Remark** Lemma 15 shows that the previous constraints on  $d$  are sufficient conditions for Algorithm 3 to terminate (*i.e.*,  $\ell_{MBRB}$  correct processes mbrb-deliver the value mbrb-broadcast by a correct sender) in at most 2, 3, 4, or 5 rounds, respectively. We conjecture that these conditions are also necessary for Algorithm 3, as the upper bounds on  $d$  that we have obtained were constructed by considering the worst-case strategy of the message adversary, where the received signatures are evenly distributed among the correct processes of  $L_2$  to delay the delivery by a correct process for as long as possible. One angle of attack for proving this conjecture could be to show that the first correct process that observes a quorum of signatures (and thus mbrb-delivers the sender's value) is necessarily a process of  $L_2$  (*i.e.*, a process that has seen the sender's signature at the latest during round 2), which seems to be verified according to numerical tests we have performed.

**Visualization of the upper bounds of  $d$**  In Figure 8, we illustrate the *inclusive* upper bounds of  $d$  depending on  $t$  for obtaining a termination of Algorithm 3 in at most 2, 3, 4, or 5 rounds (Lemma 15).<sup>31</sup> We consider the case  $n = 100$  and  $c = n - t$ , hence, by sb-MBRB-Assumption, we have  $0 \leq t \leq 33$  and  $0 \leq d \leq 49$ . The values of these upper bounds are taken directly from Lemma 15<sup>32</sup>:

31. Recall that, by “Algorithm 3 terminates in at most  $x$  rounds” where  $x \in \{2, 3, 4, 5\}$ , we mean that, if a correct sender  $p_i$  mbrb-broadcasts some  $(v, sn)$ , then at least  $\ell_{MBRB} = c - d$  correct processes mbrb-deliver  $(v, sn, i)$  at most  $x$  rounds later.

32. Given the *exclusive* upper bounds on  $d$  of Lemma 15, we find the corresponding *inclusive* upper bounds by observing that  $\forall d \in \mathbb{N}, x \in \mathbb{R} : d < x \Leftrightarrow d \leq \lceil x \rceil - 1$ .

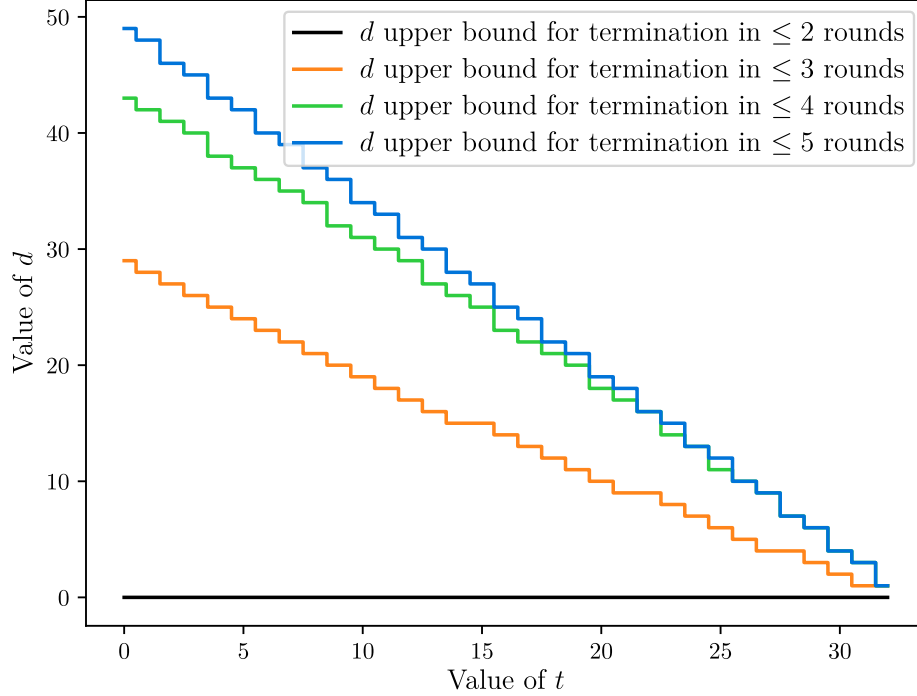


Figure 8: Maximum values of  $d$  depending on  $t$  for a termination in at most 2 rounds (in black), 3 rounds (in orange), 4 rounds (in green), or 5 rounds (in blue), assuming  $n = 100$  and  $c = n - t$

- the black line represents the fact that Algorithm 3 terminates in at most 2 rounds if  $d = 0$ ;
- the orange steps represent the fact that Algorithm 3 terminates in at most 3 rounds if  $0 < d \leq \left\lceil c - \sqrt{c \times \frac{n+t}{2}} \right\rceil - 1$ ;
- the green steps represent the fact that Algorithm 3 terminates in at most 4 rounds if  $\left\lceil c - \sqrt{c \times \frac{n+t}{2}} \right\rceil - 1 < d \leq \left\lceil c - \frac{(n+t+2c)^2}{16c} \right\rceil - 1$ ;
- the blue steps represent the fact that Algorithm 3 terminates in at most 5 rounds in the worst case, *i.e.*, if  $\left\lceil c - \frac{(n+t+2c)^2}{16c} \right\rceil - 1 < d \leq \left\lceil \frac{n-3t}{2} \right\rceil - 1$ . The *exclusive* upper-bound of  $d < \frac{n-3t}{2}$  is derived from sb-MBRB-Assumption, which states that  $n > 3t + 2d$ .

**Lemma 16** (MBRB-Message-cost). *In Algorithm 3, the mbrb-broadcast of a value  $v$  by a correct process  $p_i$  entails the sending of at most  $\mu = 2n^2$  messages by correct processes overall.*

**Proof.** The broadcast of a message by a correct process at line 3 entails its forwarding by at most  $n - 1$  other correct processes at line 10. As each broadcast by correct processes corresponds to the sending of  $n$  messages, then at most  $n^2$  messages are sent in a first step.

In a second step, at least one correct process reaches a quorum of signatures and passes the condition at line 11, and then broadcasts this quorum of signatures at line 12. Upon receiving this quorum, every correct process also passes the condition at line 11 (if it has not done it already) and broadcasts the message containing the quorum at line 12. Hence, at most  $n^2$  messages are also sent in this second step, which amounts to a maximum of  $\mu = 2n^2$  messages sent in total.  $\square$

**Lemma 17** (MBRB-Communication-cost). *In Algorithm 3, the mbrb-broadcast of a value  $v$  by a correct process  $p_i$  entails the sending of at most  $\kappa = O(n^2|v| + n^3\lambda)$  bits by correct processes overall.*

**Proof.** Let us first characterize the size of the different data structures included in the BUNDLE messages of Algorithm 3. Firstly,  $|v|$  denotes the size of the applicative value  $v$ . We assume the size of the sequence number  $sn$  to be constant, hence it is negligible in the following analysis. The identity  $i$  of the sender is encoded in  $O(\log n)$  bits. Finally, the set of signatures  $sigs$  contains at most a quorum of  $O(n)$  signature (of  $O(\lambda)$  bits), implicitly accompanied by the signer's identity (of  $O(\log n)$  bits). This amounts to  $O(n(\lambda + \log n))$  bits for a set  $sigs$ . Hence, a BUNDLE message has  $O(|v| + \log n + n(\lambda + \log n))$  bits in total, which we can simplify to  $O(|v| + n\lambda)$ , since we have  $\lambda = \Omega(\log n)$  by assumption (see Section 4.1).

Since correct processes communicate  $O(n^2)$  messages overall (see Lemma 16), it means that they send  $\kappa = O(n^2|v| + n^3\lambda)$  bits overall.  $\square$

## 4.5 Conclusion

This chapter has presented an asynchronous algorithm implementing the Message-Adversary-tolerant Byzantine reliable broadcast (MBRB) abstraction, introduced in Chapter 3 to capture the problem of reliable broadcast in the context of the hybrid failures. More precisely, this algorithm works in spite of Byzantine failures and message losses caused by an adversary. To do so, this MBRB algorithm exploits cryptographic signatures, and assumes  $n > 3t + 2d$  (where  $n$  is the number of processes,  $t$  is the maximum number of Byzantine processes, and  $d$  is an upper bound on the power of the message adversary), that we have shown to be a necessary and sufficient condition in Section 3.3 (hence, the bound is tight). Moreover, when there is no message adversary, this algorithm is optimal both in terms of Byzantine resilience and the number of communication rounds.

# $k2\ell$ -CAST: A MODULAR ABSTRACTION FOR SIGNATURE-FREE MBRB

---

C'est dans les vieux pots qu'on fait les meilleures confitures.

---

*Proverbe français*

The previous chapter (Chapter 4) demonstrated how the problem of designing an MA-tolerant BRB (MBRB) could be solved by leveraging digital signatures within a monolithic algorithm. However, the use of cryptographic signatures introduces important assumptions about the adversary's computing power (*e.g.*, the intractability of forging bogus signatures), which weakens the algorithm's deterministic properties. This is why the study of cryptography-free (also called *error-free*) distributed algorithms remains an essential area of research. In this context, this chapter shows that the MBRB problem can also be solved in an error-free context using a modular strategy. In particular, we present the following results in this chapter.

- We first introduce a new modular abstraction,  $k2\ell$ -cast, which appears to be a foundational building block for implementing BRB abstractions (with or without the presence of an MA). This communication abstraction systematically dissociates the predicate used to forward (network) messages from the predicate that triggers the delivery of a value, and lies at the heart of the work presented in the chapter. When proving the  $k2\ell$ -cast communication abstraction, the chapter presents an in-depth analysis of the power of an adversary that controls at most  $t$  Byzantine processes and an MA of power  $d$ .
- Using the  $k2\ell$ -cast abstraction, we then deconstruct two signature-free BRB algorithms (Bracha's [32] and Imbs and Raynal's [87] algorithms) and reconstruct new versions that tolerate *both* Byzantine processes and an MA. Interestingly, when considering Byzantine failures only, these deconstructed versions use smaller quorum sizes and are thus more efficient than their initial counterparts.



| Acronyms      | Meaning  |
|---------------|--|
| MA            | Message adversary  |
| BRB           | Byzantine reliable broadcast   |
| MBRB          | MA-tolerant Byzantine reliable broadcast                                       |
| Notations     | Meaning  |
| $n$           | number of processes in the system  |
| $t$           | upper bound on the number of Byzantine processes                               |
| $d$           | power of the message adversary   |
| $c$           | effective number of correct processes in a run ( $n - t \leq c \leq n$ )       |
| $\star$       | unspecified value  |
| $\ell_{MBRB}$ | minimal number of correct processes that mbrb-deliver a value                  |
| $k$           | minimal nb of correct processes that $k2\ell$ -cast a value                    |
| $\ell$        | minimal nb of correct processes that $k2\ell$ -deliver a value                 |
| $k'$          | minimal nb of correct $k2\ell$ -casts if there is a correct $k2\ell$ -delivery |
| $\delta$      | <b>true</b> if no-duplicity is guaranteed, <b>false</b> otherwise              |
| $q_d$         | size of the $k2\ell$ -delivery quorum  |
| $q_f$         | size of the forwarding quorum  |
| <i>single</i> | <b>true</b> iff only a single value can be endorsed, <b>false</b> otherwise    |

Table 4: Acronyms and notations used in Chapter 5

To limit our working assumptions as much as possible, we further assume that the adversary’s computability power is unbounded (except for the cryptography-based algorithm presented in Section 5.4), which precludes the use of signatures. (However, we assume that each point-to-point communication channel is authenticated.) Table 4 summarizes the acronyms and notations of this chapter.

### 5.1 The $k2\ell$ -cast abstraction: Definition

Signature-free BRB algorithms [22,32,87] often rely on successive waves of internal messages (*e.g.*, the ECHO or READY messages of Bracha’s algorithm [32]) to provide safety and liveness. Each wave is characterized by a threshold-based predicate that triggers the algorithm’s next phase when fulfilled (*e.g.*, enough ECHO messages for the same value  $v$ ). In this section, we introduce a new modular abstraction, called  $k2\ell$ -cast, that encapsulates a wave/thresholding mechanism that is both Byzantine- and MA-tolerant.

**Specification**  $k2\ell$ -cast (for  $k$ -to- $\ell$ -cast) is a many-to-many communication abstraction<sup>33</sup>. Intuitively, it relates the number  $k$  of correct processes that disseminate a value  $v$  (we say that these processes  $k2\ell$ -cast  $v$ ) with the number  $\ell$  of correct processes that deliver  $v$  (we say that they  $k2\ell$ -deliver  $v$ ). Both  $k$  and  $\ell$  are subject to thresholding constraints: enough correct processes must  $k2\ell$ -cast a value for it to be  $k2\ell$ -delivered at least once; and as soon as one (correct)  $k2\ell$ -delivery occurs, some minimal number of correct processes are guaranteed to  $k2\ell$ -deliver as well.

More formally,  $k2\ell$ -cast is a multi-shot abstraction, *i.e.*, each value  $v$  that is  $k2\ell$ -cast or  $k2\ell$ -delivered is associated with an identity  $id$ . (Typically, such an identity is a pair consisting of a process identity and a sequence number.) It provides two operations,  $k2\ell\_cast$  and  $k2\ell\_deliver$ , whose behavior is defined by the values of four parameters: three integers  $k'$ ,  $k$ ,  $\ell$ , and a Boolean  $\delta$ . This behavior is captured by the following six properties:

- **Safety.**
  - **$k2\ell$ -Validity.** If a correct process  $p_i$   $k2\ell$ -delivers a value  $v$  with identity  $id$ , then at least  $k'$  correct processes  $k2\ell$ -cast  $v$  with identity  $id$ .
  - **$k2\ell$ -Non-duplication.** A correct process  $k2\ell$ -delivers at most one value  $v$  with identity  $id$ .
  - **$k2\ell$ -Non-duplicity.** If the Boolean  $\delta$  is **true**, then no two different correct processes  $k2\ell$ -deliver different values with the same identity  $id$ .
- **Liveness.**<sup>34</sup>
  - **$k2\ell$ -Local-delivery.** If at least  $k$  correct processes  $k2\ell$ -cast a value  $v$  with identity  $id$  and no correct process  $k2\ell$ -casts a value  $v' \neq v$  with identity  $id$ , then at least one correct process  $k2\ell$ -delivers the value  $v$  with identity  $id$ .
  - **$k2\ell$ -Weak-Global-delivery.** If a correct process  $k2\ell$ -delivers a value  $v$  with identity  $id$ , then at least  $\ell$  correct processes  $k2\ell$ -deliver a value  $v'$  with identity  $id$  (each of them possibly different from  $v$ ).
  - **$k2\ell$ -Strong-Global-delivery.** If a correct process  $k2\ell$ -delivers a value  $v$  with identity  $id$ , and no correct process  $k2\ell$ -casts a value  $v' \neq v$  with identity  $id$ , then at least  $\ell$  correct processes  $k2\ell$ -deliver the value  $v$  with identity  $id$ .

---

33. An example of this family is the binary reliable broadcast introduced in [104], which is defined by specific delivery properties—not including MA-tolerance—allowing binary consensus to be solved efficiently with the help of a common coin.

34. The liveness properties comprise a *local* delivery property that provides a necessary condition for the  $k2\ell$ -delivery of a value by at least *one* correct process, and two *global* delivery properties that consider the collective behavior of correct processes.

This specification is *parameterized* in the sense that each tuple  $(k', k, \ell, \delta)$  defines a specific communication abstraction with different guarantees. This versatility explains why the  $k2\ell$ -cast abstraction can be used to produce highly compact reconstructions of existing BRB algorithms, rendering them MA-tolerant in the process (using four and three lines of pseudo-code respectively, see Section 5.3). Despite this versatility, however, we will see in Section 5.2 that  $k2\ell$ -cast can be implemented using a single (parameterized) algorithm, underscoring the fundamental commonalities of MA-tolerant BRB algorithms.

Intuitively, the parameters  $k'$ ,  $k$ , and  $\ell$  hobble the disruption power of the Byzantine/MA adversary by setting limits on the number of correct processes that are either required or guaranteed to be involved in one communication “wave” (corresponding to one identity  $id$ ).  $k'$  sets the minimal number of correct processes that must  $k2\ell$ -cast for any  $k2\ell$ -delivery to occur: it thus limits the ability of the Byzantine/MA adversary to trigger spurious  $k2\ell$ -deliveries. The role of  $k$  is symmetrical. It guarantees that some  $k2\ell$ -delivery will necessarily occur if  $k$  correct processes  $k2\ell$ -cast some message. It thus prevents the adversary from silencing correct processes as soon as some critical mass of them participates. Finally,  $\ell$  captures a “quite-a-few-or-nothing” guarantee that mirrors the traditional “all-or-nothing” delivery guarantee of traditional BRB. As soon as one correct  $k2\ell$ -delivery occurs (for some identity  $id$ ), then  $\ell$  correct processes must also  $k2\ell$ -deliver (with the same identity).

The fourth parameter,  $\delta$ , is a flag that, when **true**, enforces agreement between  $k2\ell$ -deliveries. When  $\delta = \mathbf{true}$ , the  $k2\ell$ -No-duplicity property implies that all the values  $v'$  involved in the  $k2\ell$ -Weak-Global-delivery property are equal to  $v$ .

## 5.2 Signature-free $k2\ell$ -cast

Among the many possible ways of implementing  $k2\ell$ -cast, this section presents a quorum-based<sup>35</sup> signature-free implementation<sup>36</sup> of the abstraction. To overcome the disruption caused by Byzantine processes and message losses from the MA, our algorithm uses the broadcast primitive (cf. our communication model in Section 3.1) to accumulate and forward **ENDORSE** messages before deciding whether to deliver. Forwarding and delivery are triggered by *two thresholds* (a pattern also found, for instance, in Bracha’s BRB algorithm [32]):

- a first threshold,  $q_d$ , triggers the delivery of a value  $v$  when enough **ENDORSE** messages supporting  $v$  have been received;

35. In this chapter, a quorum is a set of processes that (at the implementation level) broadcast the same message. This definition takes quorums in their ordinary sense. In a deliberative assembly, a quorum is the minimum number of members that must vote the same way for an irrevocable decision to be taken. Let us notice that this definition does not require quorum intersection. However, if quorums have a size greater than  $\frac{n+t}{2}$ , the intersection of any two quorums contains, despite Byzantine processes, at least one correct process [32,116].

36. Another  $k2\ell$ -cast implementation, which uses digital signatures and allows to reach optimal values for  $k$  and  $\ell$ , is presented in Section 5.4.

- a second threshold,  $q_f$ , which is lower than  $q_d$ , controls how **ENDORSE** messages are forwarded during the algorithm’s execution.

Forwarding, which is controlled by  $q_f$ , amplifies how correct processes react to **ENDORSE** messages, and is instrumental to ensure the algorithm’s liveness. As soon as some critical “mass” of agreeing **ENDORSE** messages accumulates within the system, forwarding triggers a chain reaction which guarantees that a minimum number of correct processes eventually  $k2\ell$ -deliver the corresponding value.

```

1 object SigFreeK2LCast( $q_d, q_f, single$ ) is
2   operation  $k2\ell\_cast(v, id)$  is
3     if ENDORSE( $\star, id$ ) not already broadcast then
4       broadcast ENDORSE( $v, id$ ).
5   when ENDORSE( $v, id$ ) is received do
6     % forwarding step %
7     if ENDORSE( $v, id$ ) received from at least  $q_f$  processes and
8       ( $\neg single$  or ENDORSE( $v, id$ ) not broadcast yet) then
9         broadcast ENDORSE( $v, id$ );
10    % delivery step %
11    if ENDORSE( $v, id$ ) received from at least  $q_d$  processes and
12      no ( $\star, id$ )  $k2\ell$ -delivered yet then
13      broadcast  $k2\ell\_deliver(v, id)$ .
    
```

Algorithm 4: Signature-free  $k2\ell$ -cast (code for  $p_i$ )

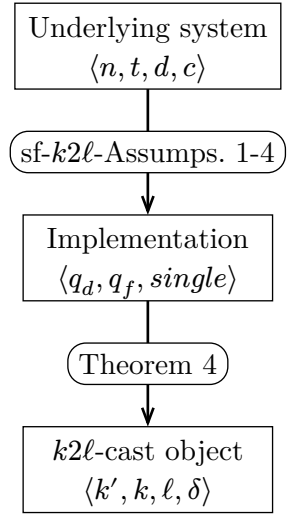


Figure 9: From the system parameters to a  $k2\ell$ -cast implementation

More concretely, our algorithm provides an object (**SigFreeK2LCast**, Algorithm 4), instantiated using the function **SigFreeK2LCast**( $q_d, q_f, single$ ), using three input parameters:

- $q_d$ : the number of matching **ENDORSE** messages that must be received from distinct processes in order to  $k2\ell$ -deliver a value;
- $q_f$ : the number of matching **ENDORSE** messages that must be received from distinct processes for the local  $p_i$  to endorse the corresponding value (if it has not yet);
- $single$ : a Boolean that controls whether a given correct process can endorse different values for the same identity  $id$  ( $single = \mathbf{false}$ ), or not ( $single = \mathbf{true}$ ).

The algorithm provides the operations  $k2\ell\_cast$  and  $k2\ell\_deliver$ . Given a value  $v$  with identity  $id$ , the operation  $k2\ell\_cast(v, id)$  broadcasts **ENDORSE**( $v, id$ ) provided  $p_i$  has not yet endorsed any different value for the same identity  $id$  (lines 3-4). When  $p_i$  receives a message **ENDORSE**( $v, id$ ), its executes two steps. If the forwarding quorum  $q_f$  has been reached,  $p_i$  first retransmits

ENDORSE( $v, id$ ) (Forwarding step, lines 6-8). Then, if the  $k2\ell$ -delivery quorum  $q_d$  is attained,  $p_i$   $k2\ell$ -delivers  $v$  (Delivery step, lines 9-11).

For brevity, we define  $\alpha = n + q_f - t - d - 1$ . Given an execution defined by the system parameters  $n, t, d$ , and  $c$ , Algorithm 4 requires the following assumptions to hold for the input parameters  $q_f$  and  $q_d$  of a  $k2\ell$ -cast instance (a global picture linking all parameters is presented in Figure 9). The prefix “sf” stands for “signature-free.”

**sf- $k2\ell$ -Assumption 1.**  $c - d \geq q_d \geq q_f + t \geq 2t + 1$ .

**sf- $k2\ell$ -Assumption 2.**  $\alpha^2 - 4(q_f - 1)(n - t) \geq 0$ .

**sf- $k2\ell$ -Assumption 3.**  $\alpha(q_d - 1) - (q_f - 1)(n - t) - (q_d - 1)^2 > 0$ .

**sf- $k2\ell$ -Assumption 4.**  $\alpha(q_d - 1 - t) - (q_f - 1)(n - t) - (q_d - 1 - t)^2 \geq 0$ .

In particular, the safety of Algorithm 4 algorithm relies solely on sf- $k2\ell$ -Assumption 1, while its liveness relies on all four of them. sf- $k2\ell$ -Assumptions 2 through 4 constrain the solutions of a second-degree inequality resulting from the combined action of the MA, the Byzantine processes, and the message-forwarding behavior of Algorithm 4. We show in Sections 5.3.1 and 5.3.2 that, in practical cases, these assumptions can be satisfied by a bound of the form  $n > \lambda t + \xi d + f(t, d)$ , where  $\lambda, \xi \in \mathbb{N}$  and  $f(t, 0) = f(0, d) = 0$ . Together, the assumptions allow Algorithm 4 to provide a  $k2\ell$ -cast abstraction (with values of the parameters  $k', k, \ell$ , and  $\delta$  defining a specific  $k2\ell$ -cast instance) as stated by the following theorem.

**Theorem 4** (Algorithm 4 correctness). *If sf- $k2\ell$ -Assumptions 1 to 4 are verified, Algorithm 4 implements  $k2\ell$ -cast with the following guarantees:*

- $k2\ell$ -Validity with  $k' = q_f - n + c$ ,
- $k2\ell$ -Non-duplication,
- $k2\ell$ -Non-duplicity with  $\delta = (q_f > \frac{n+t}{2}) \vee (single \wedge q_d > \frac{n+t}{2})$ ,
- $k2\ell$ -Local-delivery with  $k = \left\lfloor \frac{c(q_f-1)}{c-d-q_d+q_f} \right\rfloor + 1$ ,
- $\left\{ \begin{array}{l} \text{if } single = \text{false, } k2\ell\text{-Weak-Global-delivery} \\ \text{if } single = \text{true, } k2\ell\text{-Strong-Global-delivery} \end{array} \right\}$  with  $\ell = \left\lceil c \left( 1 - \frac{d}{c-q_d+1} \right) \right\rceil$ .

### 5.2.1 Proof intuition of Algorithm 4

The proofs of the  $k2\ell$ -cast safety properties stated in Theorem 4 ( $k2\ell$ -Validity,  $k2\ell$ -No-duplication, and  $k2\ell$ -No-duplicity) are fairly straightforward. For the sake of the presentation, these proofs (Lemmas 37- 40) appear in Section B.1.

Compared to safety, the proofs of the  $k2\ell$ -cast liveness properties stated in Theorem 4 ( $k2\ell$ -Local-delivery,  $k2\ell$ -Weak-Global-delivery,  $k2\ell$ -Strong-Global-Delivery) are more involved, and are informally sketched below (Lemmas 18- 26). Their full development can be found in Section B.2.

When seeking to violate the liveness properties of  $k2\ell$ -cast, the attacker can use the MA to control in part how many ENDORSE messages are received by each correct process, thus interfering with the quorum mechanisms defined by  $q_d$  and  $q_f$ . To analyze the joint effect of this interference with Byzantine faults, our proofs consider seven well-chosen subsets of correct processes ( $A$ ,  $B$ ,  $C$ ,  $U$ ,  $F$ ,  $NF$ , and  $NB$ , depicted in Figure 10a).

These subsets are defined for an execution of Algorithm 4 in which  $k_I$  correct processes  $k2\ell$ -cast  $(v, id)$  (the  $I$  in  $k_I$  is for “Initial”), and  $\ell_e$  correct processes receive at least  $q_d$  messages ENDORSE $(v, id)$ . The first three subsets,  $A$ ,  $B$ , and  $C$ , partition correct processes based on the number of ENDORSE $(v, id)$  messages they receive.

- $A$  contains the  $\ell_e$  correct processes that receive at least  $q_d$  ENDORSE $(v, id)$  messages (be it from correct or from Byzantine processes), and thus  $k2\ell$ -deliver some message.<sup>37</sup>
- $B$  contains the correct processes that receive at least  $q_f$  but less than  $q_d$  ENDORSE $(v, id)$  messages and thus do not  $k2\ell$ -deliver  $(v, id)$ .
- $C$  contains the remaining correct processes that receive less than  $q_f$  ENDORSE $(v, id)$  messages. They neither forward nor deliver any message for identity  $id$  (since  $q_f \leq q_d$ ).

In our proofs, we count the number of messages  $\text{ENDORSE}(v, id)$  broadcast by correct processes that are received by the processes of  $A$  (resp.  $B$  and  $C$ ). We note these quantities  $w_A^c$ ,  $w_B^c$ , and  $w_C^c$ , and use them to bootstrap our proofs using bounds on messages (see below).

The last four subsets intersect with  $A$ ,  $B$ , and  $C$ , and distinguish correct processes based on the broadcast operations they perform.

- $U$  consists of the correct processes that broadcast  $\text{ENDORSE}(v, id)$  at line 4.
- $F$  denotes the correct processes of  $A \cup B$  that broadcast  $\text{ENDORSE}(v, id)$  at line 8 (*i.e.*, they perform a forwarding).
- $NF$  denotes the correct processes of  $A \cup B$  that broadcast  $\text{ENDORSE}(v, id)$  at line 4.
- $NB$  denotes the correct processes of  $A \cup B$  that never broadcast  $\text{ENDORSE}(v, id)$ , be it at line 4 or at line 8. These processes have received at least  $q_f$  messages  $\text{ENDORSE}(v, id)$ , but do not forward  $\text{ENDORSE}(v, id)$ , because they have already broadcast  $\text{ENDORSE}(v', id)$  at line 4 or at line 8 for a value  $v' \neq v$ .

**Proof strategy** We note  $k_U = |U|$ ,  $k_F = |F|$ ,  $k_{NF} = |NF|$ ,  $k_{NB} = |NB|$ . Observe that  $k_U \leq k_I$  and  $k_{NF} \leq k_U$ , since all (correct) processes in  $U$  and  $NF$  invoke  $k2\ell\_cast$ . Also,  $(k_U + k_F)$  represents the total number of correct processes that broadcast a message  $\text{ENDORSE}(v, id)$ . Figure 10b illustrates how these quantities constrain the distribution of  $\text{ENDORSE}$  messages across  $A$ ,  $B$  and  $C$ . Our core proof strategy lies in bounding the areas shown in Figure 10b. (For instance, observe that  $w_A^c \leq |A| \times (k_U + k_F)$ , since each of the  $\ell_e$  correct processes in  $A$  can receive at most one  $\text{ENDORSE}$  message from each of the  $(k_U + k_F)$  correct processes that send them.) This reasoning on bounds yields a polynomial involving  $\ell_e = |A|$ ,  $k_I$ , and  $k_U$ , whose roots can then be constrained to yield the liveness guarantees required by the  $k2\ell$ -cast specification.

**Observation** In the same way we have bounded  $w_A^c$ , we can also bound  $w_B^c$  by observing that there are  $(k_{NF} + k_{NB} + k_F - \ell_e)$  processes in  $B$  and that each can receive at most  $q_d - 1$   $\text{ENDORSE}$  messages. Similarly, we can bound  $w_C^c$  by observing that the  $(c - k_{NF} - k_{NB} - k_F)$  processes of  $C$  can receive at most  $q_f - 1$   $\text{ENDORSE}$  messages. Thus, we have:

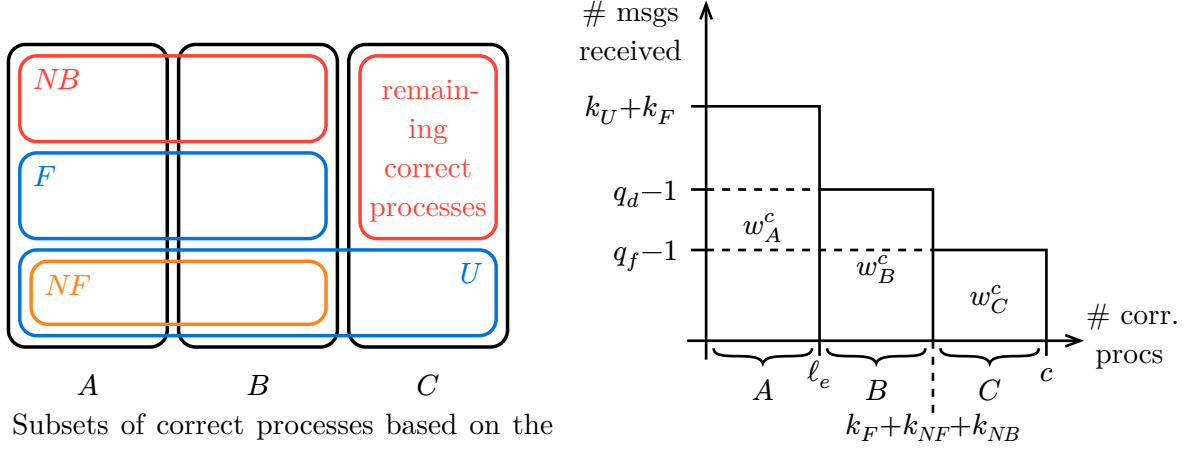
$$w_A^c \leq (k_U + k_F)\ell_e, \quad (9)$$

$$w_B^c \leq (q_d - 1)(k_{NF} + k_{NB} + k_F - \ell_e), \quad (10)$$

$$w_C^c \leq (q_f - 1)(c - k_{NF} - k_{NB} - k_F). \quad (11)$$

---

37. Because of the condition at line 10, these processes do not necessarily  $k2\ell$ -deliver  $(v, id)$ , but all do  $k2\ell$ -deliver a value for identity  $id$ .



(a) Subsets of correct processes based on the number of received ENDORSE messages ( $A$ ,  $B$ , and  $C$ ) and their broadcast actions ( $U$ ,  $F$ ,  $NF$ , and  $NB$ )

(b) Distribution of ENDORSE messages among correct processes of  $A$ ,  $B$ , and  $C$ , sorted by decreasing number of ENDORSE messages received

Figure 10: Subsets of correct processes and distribution of ENDORSE messages among them

Moreover, the MA cannot suppress more than  $d$  copies of each individual ENDORSE message broadcast to the  $c$  correct processes. Thus, the total number of ENDORSE messages received by correct processes ( $w_A^c + w_B^c + w_C^c$ ) is such that:

$$w_A^c + w_B^c + w_C^c \geq (k_U + k_F)(c - d). \quad (12)$$

**Lemma 18.**  $\ell_e \times (k_U + k_F - q_d + 1) \geq (k_U + k_F)(c - d - q_d + q_f) - c(q_f - 1) - k_{NB}(q_d - q_f)$ .

**Proof sketch.** We get this result by combining (9), (10), (11) and (12), and using sf- $k2\ell$ -Assumption 1 with the fact that  $k_{NF} \leq k_U$ . (Full derivations in Section B.2.)  $\square$

**Lemma 19.** *If no correct process  $k2\ell$ -casts  $(v', id)$  with  $v' \neq v$ , then no correct process forwards  $\text{ENDORSE}(v', id)$  at line 6 (and then  $k_{NB} = 0$ ). (Proof in Section B.2.)*



**Lemma 20** (*k2ℓ-Local-delivery*). *If at least  $k = \left\lfloor \frac{c(q_f-1)}{c-d-q_d+q_f} \right\rfloor + 1$  correct processes  $k2\ell$ -cast a value  $v$  with identity  $id$  and no correct process  $k2\ell$ -casts any value  $v'$  with identity  $id$  such that  $v \neq v'$ , then at least one correct process  $p_i$   $k2\ell$ -delivers  $v$  with identity  $id$ .*

**Proof sketch.** From the hypotheses, Lemma 19 helps us determine that  $k_{NF} = 0$ . Then, the property is proved by contraposition, by assuming that no correct process  $k2\ell$ -delivers  $(m, id)$ , which leads us to  $\ell_e = 0$ . Using prior information and sf- $k2\ell$ -Assumption 1, we can rewrite the inequality of Lemma 18 to get the threshold of  $k2\ell$ -casts above which there is at least one  $k2\ell$ -delivery. (Full derivations in Section B.2.)  $\square$

**Lemma 21.**  $(single = \mathbf{false}) \implies (k_{NB} = 0)$ . (Proof in Section B.2.)

**Lemma 22.** *If at least one correct process  $k2\ell$ -delivers  $(v, id)$  and  $x = k_U + k_F$  (the number of correct processes that broadcast  $\text{ENDORSE}(v, id)$  at line 4 or 8), then  $x \geq q_d - t$  and  $x^2 - x(c - d + q_f - 1 - k_{NB}) \geq -(c - k_{NB})(q_f - 1)$ .*

**Proof sketch.** We prove this lemma by counting the total number of messages (sent by Byzantine or correct processes) that are received by the processes of  $A$ , and by using (9), (11) (12), and sf- $k2\ell$ -Assumption 1. (Full derivations in Section B.2.)  $\square$

**Lemma 23.** *If  $k_{NB} = 0$ , and at least one correct process  $k2\ell$ -delivers  $(v, id)$ , then  $k_U + k_F \geq q_d$ .*

**Proof sketch.** Given that  $k_{NB} = 0$ , we can rewrite the inequality of Lemma 22, which gives us a second-degree polynomial (where  $x = k_U + k_F$  is the unknown variable). We compute its roots and show that the smaller one contradicts Lemma 22, and that the larger one is greater than or equal to  $q_d$ . The fact that  $x$  must be greater than or equal to the larger root to satisfy Lemma 22 proves the lemma. (Full derivations in Section B.2.)  $\square$

**Lemma 24.** *If  $k_{NB} = 0$  and  $k_U + k_F \geq q_d$ , then at least  $\left\lceil c \left(1 - \frac{d}{c - q_d + 1}\right) \right\rceil$  correct processes  $k2l$ -deliver some value with identity  $id$  (not necessarily  $v$ ).*

**Proof sketch.** From the hypotheses, we can rewrite the inequality of Lemma 18 to get a lower bound on  $\ell_e$ . Using sf- $k2l$ -Assumption 3, we can determine that this lower bound is decreasing with the number of broadcasts by correct processes ( $x = k_u + k_f$ ). Hence, this lower bound is minimum when  $x$  is maximum, that is, when  $x = c$ . This gives us the minimum number of correct processes that  $k2l$ -deliver under the given hypotheses. (Full derivations in Section B.2.)  $\square$

**Lemma 25** ( $k2l$ -Weak-Global-delivery). *If  $single = \mathbf{false}$ , and a correct process  $k2l$ -delivers a value  $v$  with identity  $id$ , then at least  $\ell = \left\lceil c \left(1 - \frac{d}{c - q_d + 1}\right) \right\rceil$  correct processes  $k2l$ -deliver a value  $v'$  with identity  $id$  (each possibly different from  $v$ ).*

**Proof sketch.** As  $single = \mathbf{false}$  and one correct process  $k2l$ -delivers  $(m, id)$ , Lemmas 21 and 23 apply, and we have  $k_{NF} = 0$  and  $k_U + k_F \geq q_d$ . This provides the prerequisites for Lemma 24, which concludes the proof. (Full derivations in Section B.2.)  $\square$

**Lemma 26** ( $k2l$ -Strong-Global-delivery). *If  $single = \mathbf{true}$ , and a correct process  $k2l$ -delivers a value  $v$  with identity  $id$ , and no correct process  $k2l$ -casts a value  $v' \neq v$  with identity  $id$ , then at least  $\ell = \left\lceil c \left(1 - \frac{d}{c - q_d + 1}\right) \right\rceil$  correct processes  $k2l$ -deliver  $v$  with identity  $id$ .*

**Proof sketch.** As  $single = \mathbf{true}$ , Lemma 19 holds and implies that  $k_{NB} = 0$ . As above, Lemma 23 and Lemma 24 hold, yielding the lemma. (Full derivations in Section B.2.)  $\square$

### 5.3 $k2l$ -cast in action: From classical BRB to MBRB algorithms

As previously announced, this section uses  $k2l$ -cast to reconstruct two signature-free BRB algorithms initially introduced in a pure Byzantine context (*i.e.*, without any MA): Bracha's BRB [32] (Algorithm 1) and Imbs-Raynal's BRB [87] (Algorithm 2). These reconstructions produce Byzantine-MA-tolerant versions of the initial algorithms that implement the MBRB specification of Section 3.2. Moreover, when  $d = 0$ , our two reconstructed BRB algorithms are strictly more efficient than the original algorithms that gave rise to them: they terminate earlier.

More precisely, the original and reconstructed versions of Bracha’s BRB are identical in terms of communication cost, time complexity, and  $t$ -resilience (when  $d = 0$ ). The same comparison applies to the original and reconstructed versions of Imbs and Raynal’s BRB. However, both reconstructed BRB algorithms use smaller quorums than their original versions, requiring fewer messages to progress. In an actual network, this means a lower latency in practice, as practical networks typically exhibit a long-tail distribution of latencies (a phenomenon well-studied by system and networking researchers [47,59,138]).

To help readers familiar with the initial algorithms, we use the same message types (INIT, ECHO, READY, WITNESS) as in the original publications. Recall that the MBRB problem can be solved if and only if  $n > 3t + 2d$  (Theorem 1, page 35).

### 5.3.1 Bracha’s BRB algorithm (1987) reconstructed

**Reconstructed version** Bracha’s BRB algorithm comprises three phases. When a process invokes `brb_broadcast( $v, sn$ )`, it disseminates the value  $v$  in an INIT message (first phase). The reception of this message by a correct process triggers its participation in a second phase implemented by the exchange of messages tagged ECHO. Finally, when a process has received ECHO messages from “enough” processes, it enters the third phase, in which READY messages are exchanged, at the end of which it brb-delivers the value  $v$ . Algorithm 5 is a reconstructed version of the Bracha’s BRB, which assumes  $n > 3t + 2d + 2\sqrt{td}$ .

```

1 init:  $obj_E \leftarrow \text{SigFreeK2LCast}(q_d = \lfloor \frac{n+t}{2} \rfloor + 1, q_f = t + 1, \text{single} = \text{true});$ 
    $obj_R \leftarrow \text{SigFreeK2LCast}(q_d = 2t + d + 1, q_f = t + 1, \text{single} = \text{true}).$ 
2 operation mbrb_broadcast( $v, sn$ ) is broadcast INIT( $v, sn$ ).
3 when INIT( $v, sn$ ) is received from  $p_j$  do  $obj_E.k2\ell\_cast(\text{ECHO}(v), (sn, j)).$ 
4 when (ECHO( $v$ ), ( $sn, j$ )) is  $obj_E.k2\ell\_delivered$  do  $obj_R.k2\ell\_cast(\text{READY}(v), (sn, j)).$ 
5 when (READY( $v$ ), ( $sn, j$ )) is  $obj_R.k2\ell\_delivered$  do mbrb_deliver( $v, sn, j$ ).
    
```

Algorithm 5: Multi-shot  $k2\ell$ -cast-based reconstruction of Bracha’s BRB algorithm (code for  $p_i$ )

The algorithm requires two instances of  $k2\ell$ -cast, denoted  $obj_E$  and  $obj_R$ , associated with the ECHO messages and the READY messages, respectively. For both these objects, the Boolean *single* is set to `true`. For the quorums, we have the following:

- $obj_E$ :  $q_f = t + 1$  and  $q_d = \lfloor \frac{n+t}{2} \rfloor + 1$ ,
- $obj_R$ :  $q_f = t + 1$  and  $q_d = 2t + d + 1$ .

The integer  $sn$  is the sequence number of the value  $v$  mbrb-broadcast by  $p_i$ . The identity of  $v$  is consequently the pair  $\langle sn, i \rangle$ .

Algorithm 5 provides  $\ell_{MBRB} = \lceil c(1 - \frac{d}{c-2t-d}) \rceil$  under the following assumption (B87 stands for Bracha 1987).

**B87-Assumption.**  $n > 3t + 2d + 2\sqrt{td}$ .

The proof of correctness of Algorithm 5 can be found in Section C.1.

**Comparison (Table 5)** When  $d = 0$ , both Bracha’s algorithm and its reconstruction use the same quorum size for the **READY** phase. However, the quorums of the **ECHO** phase are different (Table 5). As the algorithm requires  $n > 3t$ , we define  $\Delta = n - 3t$  as the slack between the lower bound on  $n$  and the actual value of  $n$ . When considering the forwarding threshold  $q_f$ , we have  $\lfloor \frac{n+t}{2} \rfloor + 1 = 2t + \lfloor \frac{\Delta}{2} \rfloor + 1 > t + 1$ . As a result, the reconstruction of Bracha’s algorithm always uses a lower forwarding threshold for **ECHO** messages than the original. It, therefore, forwards messages more rapidly and reaches the delivery quorum faster.

| Threshold        | Original version (ECHO phase)       | $k2\ell$ -cast-based version ( $obj_E$ ) |
|------------------|-------------------------------------|--|
| Forwarding $q_f$ | $\lfloor \frac{n+t}{2} \rfloor + 1$ | $t + 1$                                  |
| Delivery $q_d$   | $\lfloor \frac{n+t}{2} \rfloor + 1$ | $\lfloor \frac{n+t}{2} \rfloor + 1$      |

Table 5: Bracha’s original version vs.  $k2\ell$ -cast-based reconstruction when  $d = 0$

### 5.3.2 Imbs-Raynal’s BRB algorithm (2016) reconstructed

**Reconstructed version** Imbs and Raynal’s BRB is another BRB implementation, which achieves an optimal good-case latency (only two communication steps) at the cost of a non-optimal  $t$ -resilience. Its reconstructed version requires  $n > 5t + 12d + \frac{2td}{t+2d}$ .

```

1 init:  $obj_W \leftarrow \text{SigFreeK2LCast}(q_d = \lfloor \frac{n+3t}{2} \rfloor + 1, q_f = \lfloor \frac{n+t}{2} \rfloor + 1, \text{single} = \text{false})$ .
2 operation  $\text{mbrb\_broadcast}(v, sn)$  is broadcast  $\text{INIT}(v, sn)$ .
3 when  $\text{INIT}(v, sn)$  is received from  $p_j$  do  $obj_W.k2\ell\_cast(\text{WITNESS}(v), (sn, j))$ .
4 when  $(\text{WITNESS}(v), (sn, j))$  is  $obj_W.k2\ell\_delivered$  do  $\text{mbrb\_deliver}(v, sn, j)$ .

```

Algorithm 6: Multi-shot  $k2\ell$ -cast-based reconstruction of Imbs-Raynal’s BRB algorithm (code for  $p_i$ )

The algorithm requires a single  $k2\ell$ -cast object, denoted  $obj_W$ , associated with the WITNESS message, and which is instantiated with  $q_f = \lfloor \frac{n+t}{2} \rfloor + 1$  and  $q_d = \lfloor \frac{n+3t}{2} \rfloor + 3d + 1$ , and the Boolean  $single = \mathbf{false}$ . Similarly to Bracha's reconstructed BRB, an identity of value in this algorithm is a pair  $\langle sn, i \rangle$  containing a sequence number  $sn$  and a process identity  $i$ .

Algorithm 6 provides  $\ell_{MBRB} = \left\lceil c \left( 1 - \frac{d}{c - \lfloor \frac{n+3t}{2} \rfloor - 3d} \right) \right\rceil$  under the following assumption (IR16 stands for Imbs-Raynal 2016).

**IR16-Assumption.**  $n > 5t + 12d + \frac{2td}{t+2d}$  (where  $t + d > 0$ ).

The proof of correctness of Algorithm 6 can be found in Section C.2.

**Comparison (Table 6)** Table 6 compares Imbs and Raynal's original algorithm against its  $k2\ell$ -cast reconstruction for  $d = 0$ . Recall that this algorithm saves one communication step with respect to Bracha's at the cost of a weaker  $t$ -tolerance, *i.e.*, it requires  $n > 5t$ . As for Bracha, let us define the slack between  $n$  and its minimum as  $\Delta = n - 5t$ , we have  $\Delta \geq 1$ .

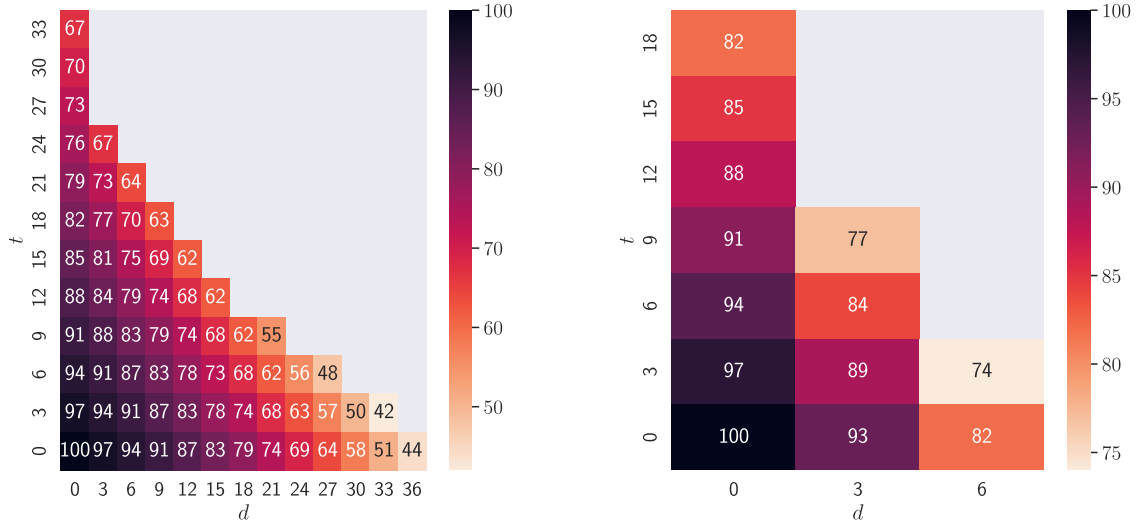
- Let us first consider the size of the forwarding quorum (first line of the table). We have  $n - 2t = 3t + \Delta$  and  $\lfloor \frac{n+t}{2} \rfloor + 1 = 3t + \lfloor \frac{\Delta}{2} \rfloor + 1$ . When  $\Delta > 2$ , we always have  $\Delta > \lfloor \frac{\Delta}{2} \rfloor + 1$ , it follows that the forwarding predicate of the reconstructed version is equal or weaker than the one of the original version.
- The same occurs for the size of the delivery quorum (second line of the table). We have  $n - t = 4t + \Delta$  and  $\lfloor \frac{n+3t}{2} \rfloor + 1 = 4t + \lfloor \frac{\Delta}{2} \rfloor + 1$ . So both reconstructed quorums are lower than those of the original version when  $\Delta > 2$ , making the reconstructed algorithm quicker as soon as  $n \geq 5t + 3$ . The two versions behave identically for  $5t + 3 \geq n \geq 5t + 2$  (where  $\Delta \in \{1, 2\}$ ).

| Threshold        | Original version (WITNESS phase) | $k2\ell$ -cast-based version ( $obj_W$ ) |
|------------------|----------------------------------|--|
| Forwarding $q_f$ | $n - 2t$                         | $\lfloor \frac{n+t}{2} \rfloor + 1$      |
| Delivery $q_d$   | $n - t$                          | $\lfloor \frac{n+3t}{2} \rfloor + 1$     |

Table 6: Imbs-Raynal's original version vs.  $k2\ell$ -cast-based reconstruction when  $d = 0$

### 5.3.3 Numerical evaluation of the MBRB algorithms

Figure 11 provides a numerical evaluation of the delivery guarantees of both  $k2\ell$ -cast-based MBRB algorithms (Algorithms 5 and 6) in the presence of Byzantine processes and an MA.



(a) Reconstructed Bracha MBRB (Algorithm 5)      (b) Reconstructed Imbs-Raynal MBRB (Algorithm 6)

Figure 11: Values of  $\ell_{MBRB}$  for the reconstructed BRB algorithms when varying  $t$  and  $d$  ( $n = 100$  and  $c = n - t$ ) within the ranges that satisfy B87-Assumption and IR16-Assumption

Results were obtained for  $n = 100$  and  $c = n - t$ , and show the values of  $\ell_{MBRB}$  for different values of  $t$  and  $d$ . For instance, Figure 11a shows that with 6 Byzantine processes and an MA suppressing up to 9 broadcast messages, Algorithm 5 ensures the MBRB-Global-delivery property with  $\ell_{MBRB} = 83$ . The figures illustrate that the reconstructed Bracha algorithm performs in a broader range of parameter values, mirroring the bounds on  $n$ ,  $t$ , and  $d$  captured by B87-Assumption and IR-Assumption. Nonetheless, both algorithms exhibit values of  $\ell_{MBRB}$  that can support real-world applications in the presence of an MA.

**Additional results** The following presents additional numerical results that complement those of Figure 11, and provides concrete lower-bound values for the  $k$  and  $\ell$  parameters of the  $k2\ell$ -cast objects used in the reconstructed Bracha MBRB algorithm (Algorithm 5). Again, results were obtained by considering a network with  $n = 100$  processes and varying values of  $t$  and  $d$ .

Figure 12 and Figure 13 present the values of  $k$  and  $\ell$  for the  $obj_E$  and  $obj_R$  of Algorithm 5. The numbers in each cell show the value of  $k$  (Figures 12a and 13a), resp.  $\ell$  (Figures 12b and 13b) that is required, resp. guaranteed, by the corresponding  $k2\ell$ -cast object. The two plots show the two roles of the two  $k2\ell$ -cast objects. The first,  $obj_E$ , needs to provide agreement among the possibly different messages sent by Byzantine processes (Figure 12). As a result, it can operate in a more limited region of the parameter space. On the other hand,  $obj_R$  would, in



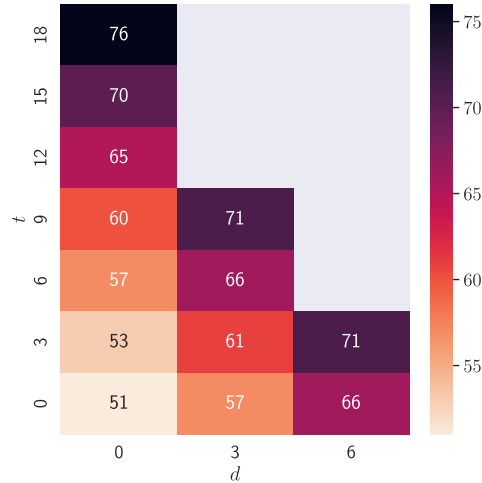


Figure 14: Required values of  $k$  for  $obj_W$  in the reconstructed Imbs-Raynal BRB algorithm with varying values of  $t$  and  $d$  within the ranges that satisfy IR16-Assumption

principle, be able to support larger values of  $t$  and  $d$ , but it needs to operate in conjunction with  $obj_E$  (Figure 13).

Figure 11b already displays the values of  $\ell$  provided by  $obj_W$  in the Imbs-Raynal algorithm. Figure 14 complements it by showing the required values of  $k$  for  $obj_W$ . The extra constraint introduced by chaining the two objects suggests that a single  $k2\ell$ -cast algorithm could achieve better performance. But this is not the case if we examine the performance of the reconstructed Imbs-Raynal algorithm depicted in Figure 11b. The reason lies in the need for higher quorum values in  $obj_W$  due to  $single = \text{false}$ . In the future, we plan to investigate if variants of this algorithm can achieve tighter bounds and explore the limits of signature-free  $k2\ell$ -cast-based broadcast in the presence of an MA and Byzantine processes.

## 5.4 Signature-based $k2\ell$ -cast

This section presents an implementation of  $k2\ell$ -cast based on digital signatures. The underlying model is the same as that of Section 5.2 (page 62, except that the computing power of the attacker is now bounded, which allows us to leverage asymmetric cryptography.

### 5.4.1 Algorithm

The signature-based algorithm is described in Algorithm 7. It uses an asymmetric cryptosystem to sign messages and verify their authenticity. Every process has a public/private key pair. Public keys are known to everyone, but private keys are only known to their owner. (Byzantine processes may exchange their private keys.) Each process also knows the mapping between process indexes and associated public keys, and each process can produce a unique, valid signature for a given message, and check if a signature is valid.



```

1 object SigBasedK2LCast( $q_d$ ) is
2   operation  $k2\ell\_cast(v, id)$  is
3     if  $(\star, id)$  not already signed by  $p_i$  then
4        $sig_i \leftarrow$  signature of  $(v, id)$  by  $p_i$ ;
5        $sigs_i \leftarrow$  {all valid signatures for  $(m, id)$  broadcast by  $p_i$ }  $\cup$  { $sig_i$ };
6       broadcast BUNDLE( $v, id, sigs_i$ );
7       check_delivery().
8   when BUNDLE( $v, id, sigs$ ) is received do
9     if  $sigs$  contains valid signatures for  $(v, id)$  not already broadcast by  $p_i$  then
10       $sigs_i \leftarrow$  {all valid signatures for  $(v, id)$  broadcast by  $p_i$ }
11         $\cup$  {all valid signatures for  $(v, id)$  in  $sigs$ };
12      broadcast BUNDLE( $v, id, sigs_i$ );
13      check_delivery().
14   internal operation check_delivery() is
15     if  $p_i$  broadcast at least  $q_d$  valid signatures for  $(v, id)$ 
16       and no  $(\star, id)$   $k2\ell$ -delivered yet then  $k2\ell\_deliver(m, id)$ .

```

Algorithm 7: Signature-based  $k2\ell$ -cast (code for  $p_i$ )

It is a simple algorithm that ensures that a value must be  $k2\ell$ -cast by at least  $k$  correct processes to be  $k2\ell$ -delivered by at least  $\ell$  correct processes. For the sake of simplicity, we say that a correct process  $p_i$  “broadcasts a set of signatures” if it broadcasts a BUNDLE( $v, id, sigs_i$ ) message in which  $sigs_i$  contains the signatures at hand. A correct process  $p_i$  broadcasts a value  $v$  with identity  $id$  at line 6 or line 11.

- If this occurs at line 6,  $p_i$  includes in the message it broadcasts all the signatures it has already received for  $(v, id)$  plus its own signature.
- If this occurs at line 11,  $p_i$  has just received a message containing a set of signatures  $sigs$  for the pair  $(v, id)$ . The process  $p_i$  then aggregates in  $sigs$  the valid signatures it just received with the ones it did know about beforehand (line 10).

This algorithm relies on the following assumption (the prefix “sb” stands for signature-based).

**sb- $k2\ell$ -Assumption 1.**  $c > 2d$ .

**sb- $k2\ell$ -Assumption 2.**  $c - d \geq q_d \geq t + 1$ .

Thanks to digital signatures, processes can relay the messages of other processes in Algorithm 7. The algorithm, however, does not use forwarding in the same way Algorithm 4 did: there is no equivalent of  $q_f$  here, that is, the only way to “endorse” a value (which, in this case, is equivalent to signing this value) is to invoke the  $k2\ell\_cast$  operation. Furthermore, only one value can be endorsed by a correct process for a given identity (which is the equivalent of  $single = \mathbf{true}$  in the signature-free version).

Although this implementation of  $k2\ell$ -cast provides better guarantees than Algorithm 4, using it to reconstruct signature-free BRB algorithms would be counter-productive. This is because signatures allow for MA-tolerant BRB algorithms that are more efficient in terms of round and message complexity than those that can be constructed using  $k2\ell$ -cast (see Chapter 4).

However, a signature-based  $k2\ell$ -cast does make sense in contexts in which many-to-many communication patterns are required [22], and, we believe, opens the path to novel ways to handle local state resynchronization resilient to Byzantine failures and message adversaries. For instance, we are using the following algorithm in our own work to design churn-tolerant money transfer systems tolerating Byzantine failures and temporary disconnections.

#### 5.4.2 Proof

In this section, we prove the following theorem.

**Theorem 5** ( *$k2\ell$ -Correctness*). *If sb- $k2\ell$ -Assumption 1 and sb- $k2\ell$ -Assumption 2 are verified, Algorithm 7 implements  $k2\ell$ -cast with the following guarantees: (i)  $k' = q_d - n + c$ , (ii)  $k = q_d$ , (iii)  $\ell = c - d$ , and (iv)  $\delta = q_d > \frac{n+t}{2}$ .*

The proof follows from the subsequent lemmas. Let us first remind that, given two sets  $A$  and  $B$ , we have  $|A \cap B| = |A| + |B| - |A \cup B|$ . Moreover, the number of correct processes  $c$  is superior or equal to  $n - t$ . Additionally, if  $A$  and  $B$  are both sets containing a majority of correct processes, we have  $|A \cup B| \leq c$ , which implies that  $|A \cap B| \geq |A| + |B| - c$ .

**Safety proof** The proofs of the safety properties are given in the following.

**Lemma 27.** *If a correct process  $p_i$   $k2\ell$ -delivers  $(v, id)$ , then at least  $q_d - n + c$  correct processes have signed  $(v, id)$  at line 4.*

**Proof.** If  $p_i$   $k2\ell$ -delivers  $(v, id)$  at line 14, then it sent  $q_d$  valid signatures for  $(v, id)$  (because of the predicate at line 9). The effective number of Byzantine processes in the system is  $n - c$ , such that  $0 \leq n - c \leq t$ . Therefore,  $p_i$  must have sent at least  $q_d - n + c$  (which, due to sb- $k2\ell$ -Assumption 2, is strictly positive because  $q_d > t \geq n - c$ ) valid distinct signatures for  $(v, id)$  that correct processes made at line 4, during a  $k2\ell\_cast(v, id)$  invocation.  $\square$

**Lemma 28** ( *$k2\ell$ -Validity*). *If a correct process  $p_i$   $k2\ell$ -delivers a value  $v$  with identity  $id$ , then at least  $k' = q_d - n + c$  correct processes  $k2\ell$ -cast  $v$  with identity  $id$ .*

**Proof.** The condition at line 3 implies that the correct processes that  $k2\ell$ -cast  $(v, id)$  constitute a superset of those that signed  $(v, id)$  at line 4. Thus, by Lemma 27, their number is at least  $k' = q_d - n + c$ .  $\square$

**Lemma 29** ( *$k2\ell$ -No-duplication*). *A correct process  $k2\ell$ -delivers at most one value  $v$  with identity  $id$ .*

**Proof.** This property derives trivially from the predicate at line 14.  $\square$

**Lemma 30** ( *$k2\ell$ -No-duplicity*). *If the Boolean  $\delta = q_d > \frac{n+t}{2}$  is **true**, then no two different correct processes  $k2\ell$ -deliver different values with the same identity  $id$ .*

**Proof.** Let  $p_i$  and  $p_j$  be two correct processes that respectively  $k2\ell$ -deliver  $(v, id)$  and  $(v', id)$ . We want to prove that, if the predicate  $(q_d > \frac{n+t}{2})$  is satisfied, then  $v = v'$ .

Thanks to the predicate at line 14, we can assert that  $p_i$  and  $p_j$  must have respectively sent at least  $q_d$  valid signatures for  $(v, id)$  and  $(v', id)$ , made by two sets of processes, that we respectively denote  $A$  and  $B$ , such that  $|A| \geq q_d > \frac{n+t}{2}$  and  $|B| \geq q_d > \frac{n+t}{2}$ . We have  $|A \cap B| > 2\frac{n+t}{2} - n = t$ . Hence, at least one correct process  $p_x$  has signed both  $(v, id)$  and  $(v', id)$ . But because of the predicate at line 3,  $p_x$  signed at most one couple  $(\star, id)$  during a  $k2\ell\_cast(v, id)$  invocation at line 4. We conclude that  $v$  is necessarily equal to  $v'$ .  $\square$

**Liveness proof** The proofs of the liveness properties are given in the following.

**Lemma 31.** *All signatures made by correct processes at line 4 are eventually received by at least  $c - d$  correct processes at line 8.*

**Proof.** Let  $\{s_1, s_2, \dots\}$  be the set of all signatures for  $(v, id)$  made by correct processes at line 4. We first show by induction that, for all  $z$ , at least  $c - d$  correct processes receive all signatures  $\{s_1, s_2, \dots, s_z\}$  at line 8.

Base case  $z = 0$ . As no correct process signed  $(v, id)$ , the proposition is trivially satisfied.

Induction. We suppose that the proposition is verified at  $z$ : signatures  $s_1, s_2, \dots, s_z$  are received by a set of at least  $c - d$  correct processes that we denote  $A$ . We now show that the proposition is verified at  $z + 1$ : at least  $c - d$  correct processes eventually receive all signatures  $s_1, s_2, \dots, s_{z+1}$ .

The correct process that makes the signature  $s_{z+1}$  broadcasts a  $\text{BUNDLE}(v, id, sigs)$  message (at line 6) where  $sigs$  contains  $s_{z+1}$ . From the definition of the MA,  $\text{BUNDLE}(v, id, sigs)$  is eventually received by a set of at least  $c - d$  correct processes that we denote  $B$ . We have  $|A \cap B| = 2(c - d) - c = c - 2d > 2d - 2t = 0$  (from sb- $k2\ell$ -Assumption 1). Hence, at least one correct process  $p_j$  eventually receives all signatures  $s_1, s_2, \dots, s_{z+1}$ , and thereafter broadcasts  $\text{BUNDLE}(v, id, sigs')$  where  $\{s_1, s_2, \dots, s_{z+1}\} \subseteq sigs'$ . Again, from the definition of the MA,  $\text{BUNDLE}(v, id, sigs')$  is eventually received by a set of at least  $c - d$  correct processes at line 8.  $\square$

**Lemma 32.** *If no correct process  $k2\ell$ -casts  $(v, id)$  at line 2, then no correct process  $k2\ell$ -delivers  $(v, id)$  at line 14.*

**Proof.** Looking for a contradiction, let us suppose that a correct process  $p_i$   $k2\ell$ -delivers  $(v, id)$  while no correct process  $k2\ell$ -cast  $(v, id)$ . Because of the condition at line 14,  $p_i$  must have broadcast at least  $q_d$  valid signatures for  $(v, id)$ , out of which at most  $t$  are made by Byzantine processes. As  $q_d > t$  (sb- $k2\ell$ -Assumption 2), we know that  $q_d - t > 0$ . Hence, at least one correct process must have  $k2\ell$ -cast  $(m, id)$ . Contradiction.  $\square$

**Lemma 33** ( $k2\ell$ -Local-delivery). *If at least  $k = q_d$  correct processes  $k2\ell$ -cast a value  $v$  with identity  $id$  and no correct process  $k2\ell$ -casts a value  $v' \neq v$  with identity  $id$ , then at least one correct process  $p_i$   $k2\ell$ -delivers the value  $v$  with identity  $id$ .*

**Proof.** As no correct process  $k2\ell$ -casts a value  $v' \neq v$  with identity  $id$ , then Lemma 32 holds, and no correct process can  $k2\ell$ -deliver  $(v', id)$  where  $v' \neq v$ . Moreover, no correct process

can sign  $(v', id)$  where  $v' \neq v$  at line 4, and thus all  $k = q_d$  correct processes that invoke  $k2\ell\_cast(v, id)$  at line 2 also pass the condition at line 3, and then sign  $(v, id)$  at line 4. From Lemma 31, we can assert that all  $q_d$  signatures are received at line 8 by a set of at least  $c - d$  correct processes, that we denote  $A$ . Let us consider  $p_j$ , one of the processes of  $A$ . There are two cases:

- If  $p_j$  passes the condition at line 9, then it sends all  $q_d$  signatures at line 11, then invokes  $check\_delivery()$  at line 12, passes the condition at line 14 (if it was not already done before) and  $k2\ell$ -delivers  $(v, id)$  at line 14;
- If  $p_j$  does not pass the condition at line 9, then it means that it has already sent all  $q_d$  signatures before, whether it be at line 6 or at line 11, but after that, it necessarily invoked  $check\_delivery()$  (at line 7 or at line 12, respectively), passed the condition at line 14 (if it was not already done before) and  $k2\ell$ -delivered  $(v, id)$  at line 14.  $\square$

**Lemma 34** ( *$k2\ell$ -Weak-Global-delivery*). *If a correct process  $k2\ell$ -delivers a value  $v$  with identity  $id$ , then at least  $\ell = c - d$  correct processes  $k2\ell$ -deliver a value  $v'$  with identity  $id$  (each of them possibly different from  $m$ ).*

**Proof.** If  $p_i$   $k2\ell$ -delivers  $(v, id)$  at line 14, then it has necessarily broadcast the  $BUNDLE(v, id, sigs)$  message containing the  $q_d$  valid signatures before, whether it be at line 6 or at line 11. From the definition of the MA, a set of at least  $c - d$  correct processes, that we denote  $A$ , eventually receives this  $BUNDLE(v, id, sigs)$  message at line 8. If some processes of  $A$  do not pass the condition at line 9 upon receiving this  $BUNDLE(v, id, sigs)$  message, it means that they already broadcast all signatures of  $sigs$ . Thus, in every scenario, all processes of  $A$  eventually broadcast all signatures of  $sigs$  at line 6 or at line 11. After that, all processes of  $A$  necessarily invoke the  $check\_delivery()$  operation at line 7 or at line 12, respectively, and then evaluate the condition at line 14. Hence, all correct processes of  $A$ , which are at least  $c - d = \ell$ ,  $k2\ell$ -deliver some value for identity  $id$  at line 14, whether it be  $v$  or any other value.  $\square$

**Lemma 35** ( *$k2\ell$ -Strong-Global-delivery*). *If a correct process  $k2\ell$ -delivers a value  $v$  with identity  $id$ , and no correct process  $k2\ell$ -casts a value  $v' \neq v$  with identity  $id$ , then at least  $\ell = c - d$  correct processes  $k2\ell$ -deliver  $v$  with identity  $id$ .*

**Proof.** If a correct process  $k2\ell$ -delivers  $(v, id)$  at line 14, then by Lemma 34, we can assert that at least  $\ell = c - d$  correct process eventually  $k2\ell$ -deliver some value (not necessarily  $v$ ) with

identity  $id$ . Moreover, as no correct process  $k2\ell$ -casts  $(v', id)$  with  $v' \neq v$ , then Lemma 32 holds, and we conclude that all  $\ell$  correct processes  $k2\ell$ -deliver  $(v, id)$ .  $\square$

## 5.5 Conclusion

This chapter discussed the implementation of message-adversary-tolerant Byzantine reliable broadcast (MBRB) in asynchronous systems in a cryptography-free (or error-free) context. To this end, it introduced a novel many-to-many communication primitive, called  $k2\ell$ -cast, which enables better quorum engineering, and enhances existing signature-free reliable broadcast algorithms (*e.g.*, Bracha [32] and Imbs-Raynal [87]) to make them not only resilient against a wider range of adversarial behaviours (namely Byzantine faults and a message adversary) but also more efficient (they terminate quicker). This approach can be applied to the design of a wide range of quorum-based distributed algorithms other than reliable broadcast. For instance, we conjecture that  $k2\ell$ -cast could benefit self-stabilizing and self-healing distributed systems [17], where a critical mass of messages from other processes is needed to re-synchronize the local state of a given process.



# A MBRB IMPLEMENTATION WITH NEAR-OPTIMAL COMMUNICATION

Information is the resolution of uncertainty.

*Claude Shannon*

As discussed in detail in Section 1.4.2, reliable broadcast plays a crucial role in key applications, including consensus algorithms, replication, event notification, and distributed file systems. These systems sometimes require broadcasting large messages or files (*e.g.*, permissioned blockchains), and thus, reducing the communication overhead to a minimum is an important aspect of achieving scalability. In that vein, this chapter aims at providing a *communication efficient* solution for the task of reliable broadcast in the presence of process and link faults (*i.e.*, MBRB, see Chapter 3).

In particular, the MBRB algorithm presented in this chapter communicates  $O(|v| + n\lambda)$  bits per process (or  $O(n|v| + n^2\lambda)$  bits overall), where  $|v|$  represents the length of the disseminated value and  $\lambda = \Omega(\log n)$  is a security parameter. This communication complexity is optimal up to the parameter  $\lambda$ .<sup>38</sup> This significantly improves upon the original MBRB solution (see Chapter 4), which incurs communication of  $O(n|v| + n^2\lambda)$  bits per process. The MBRB solution of this chapter sends at most  $4n^2$  messages overall, which is also asymptotically optimal. Reduced communication is achieved by employing coding techniques that replace the need for all processes to (re-)broadcast the entire value  $v$ . Instead, processes forward authenticated fragments of the encoding of  $v$  using an erasure-correcting code. Under the cryptographic assumptions of threshold signatures and vector commitments, and assuming  $n > 3t + 2d$ , this algorithm allows at least  $\ell_{MBRB} = n - t - (1 + \varepsilon)d$  (for any arbitrarily low  $\varepsilon > 0$ ) correct processes to reconstruct  $v$ , despite missing fragments caused by the malicious processes and the message adversary.

38. As stated in Section 2.2.2, for deterministic algorithms [57,107], the lower bound for the communication cost of BRB is of  $\Omega(|v| + n)$  bits per correct process, as every correct process must receive the entire value  $v$ , and as the reliable broadcast of a single bit necessitates at least  $\Omega(n^2)$  messages [62].



---

| Acronyms      | Meaning   |
|---------------|---|
| MA            | Message adversary   |
| MBRB          | MA-tolerant Byzantine reliable broadcast  |
| Notations     | Meaning   |
| $n$           | number of processes in the system   |
| $t$           | upper bound on the number of Byzantine processes  |
| $d$           | power of the message adversary  |
| $c$           | effective number of correct processes in a run ( $n - t \leq c \leq n$ )                        |
| $\lambda$     | security parameter of the cryptographic primitives  |
| $\star$       | unspecified value   |
| $\ell_{MBRB}$ | minimal number of correct processes that mbrb-deliver a value                                   |
| $\mu$         | message complexity of MBRB  |
| $\kappa$      | communication complexity of MBRB<br>(number of bits sent during the execution overall)          |
| $p_s$         | the designated sending process (with identity $s$ )   |
| $k$           | reconstruction threshold of the erasure code ( $k$ out of $n$ )                                 |
| $\tilde{v}_i$ | $i^{\text{th}}$ fragment of value $v$   |
| $\Sigma$      | threshold signature (TS)  |
| $\tau$        | threshold of the TS scheme (set to $\tau = \lfloor \frac{n+t}{2} \rfloor + 1$ in our algorithm) |
| $\sigma_i$    | signature share of the TS scheme by process $p_i$   |
| $sig_i, sigs$ | the pair $(\sigma_i, i)$ , set of $(\sigma_i, i)$ pairs   |
| $C$           | vector commitment (VC)  |
| $\pi_i$       | proof of inclusion of fragment $\tilde{v}_i$ in a VC  |

Table 7: Acronyms and notations used in Chapter 6

Table 7 summarizes the acronyms and notations of this chapter.

**Roadmap** Section 6.1 presents an initial overview of the MBRB solution of this chapter, and Section 6.2 introduces preliminary notions useful for understanding the algorithm. Section 6.3 showcases Algorithms 8 and 9, a coding-based MBRB implementation. For concision, the proof of the Global-delivery property of Coded-MBRB is sketched in Section 6.3.3, while the full correctness proof of the algorithm can be found in Appendix D. The communication analysis of Coded-MBRB is then given in Section 6.3.4. Section 6.4 discusses subsidiary aspect of Coded-MBRB. Namely, Section 6.4.1 explains how the reconstruction threshold of the erasure coding

scheme can be instantiated, Section 6.4.2 addresses the problem of upgrading the algorithm for it to become multi-sender and multi-shot, and Section 6.4.3 explains why alternative approaches for achieving near-optimal communication MBRB based on classic error-free BRB (such as Bracha’s [32]) would likely not sit well with a message-adversary-prone setting. Finally, Section 6.5 concludes the chapter.

## 6.1 Overview and intuition

This chapter presents an MBRB algorithm able to tolerate a hybrid adversary combining  $t$  Byzantine processes and a Message Adversary of power  $d$ , while providing optimal Byzantine resilience and near-optimal communication and power  $\ell_{MBRB}$ .

Its asymptotic communication complexity ( $O(|v| + n^2\lambda)$  bits sent per correct process) holds assuming a sufficiently long value  $v$ . Further,  $n - t - d$  is a natural upper bound on the delivery power  $\ell_{MBRB}$  of any MBRB algorithm. This bound arises from the power of the message adversary to isolate a subset of correct processes of size  $d$ , and omit all messages sent to this subset. Our solution obtains a delivery power  $\ell_{MBRB}$  that is as close to the limit as desired, at the cost of increasing communication (through the hidden constants in the asymptotic  $O(\cdot)$  term, which depends on  $\varepsilon$ ). Finally,  $n > 3t + 2d$  is a necessary condition to implement MBRB under asynchrony (see Section 3.3), thus making the solution of this chapter optimal in terms of Byzantine resilience.

The starting point of this chapter’s MBRB algorithm is the original MBRB algorithm (Chapter 4), that we call Original MBRB in the following for convenience. This algorithm achieves all the desired MBRB properties (Section 3.2), albeit with a large communication cost of at least  $n^2|v|$  bits overall. This communication cost stems from the re-emission strategy used by Original MBRB. In Original MBRB, the sender first disseminates the value  $v$  to all processes. To counter a possibly faulty sender, each process that receives  $v$  signs it and forwards it to the entire network, along with its own signature and any other signature observed so far for that value. This re-broadcasting step leads to  $n^2|v|$  bits of communication. In total, correct processes communicate  $O(n^2|v| + n^3\lambda)$  bits in Original MBRB (see Lemma 17, page 58).

In order to reduce the communication costs, we apply a coding technique, inspired by an approach by Alhaddad *et al.* [16]. Instead of communicating the value  $v$  directly, the sender first encodes the value using an error-correction code and “splits” the resulting codeword between the processes, so that each process receives one fragment of size  $O(|v|/n)$  bits. Now, each process needs to broadcast only its fragment of the value rather than the entire value. This reduced per-process communication effectively reduces the overall communication for disseminating the value itself to  $n|v|$  bits.

Some of the fragments might not arrive at their destination due to the action of the message adversary and the Byzantine processes. Error-correction codes have the property that the value  $v$

can be reconstructed from any sufficiently large subset of the fragments. But Byzantine processes can do even worse, namely, they can propagate an incorrect fragment. Correct processes cannot distinguish correct fragments from incorrect ones (at least, not until enough fragments are collected, and the value is reconstructed). Without this knowledge, correct processes might assist the Byzantine processes in propagating incorrect fragments, possibly harming the correctness and/or performance of the algorithm. To prevent this, the sender could sign each fragment that it sends. A process that receives a fragment could then verify that the fragment is correctly signed by the sender, and could ignore it otherwise. The drawback of this solution is that only the sender can generate signatures for fragments.

In MBRB algorithm of this chapter, that we call Coded MBRB in the sequel for simplicity, we rely on correct processes that have already reconstructed the correct value to disseminate its fragments to the processes that have not received any (say, due to the message adversary). In principle, when a process reconstructs the correct value, it can generate the codeword and obtain all the fragments, even if it did not receive some of them beforehand. However, the process cannot generate the sender's signature for the fragments it generated by itself. Because of this, the process cannot relay these fragments to the other processes, potentially leading to a reduced delivery power  $\ell_{MBRB}$ .

We avert this issue by exploiting vector commitments [39]. This cryptographic primitive generates a unique short digest  $C$  for any input vector of elements  $V$ . Additionally, it generates succinct proofs of inclusion for each element in  $V$ . In our system, the fragments of the (coded) value  $v$  form the vector  $V$ , and the inclusion proofs replace the need to sign each fragment separately. In more detail, every fragment of the codeword communicated by some process is accompanied by two pieces of information: the commitment  $C$  for the vector  $V$  containing all fragments of  $v$ , and a proof of inclusion showing that the specific fragment indeed belongs to  $V$  (see Section 6.2.2 for a formal definition of these properties). The sender signs only the commitment  $C$ . This means that Byzantine processes cannot generate an incorrect fragment and a proof that will pass the verification, since they cannot forge the sender's signature on  $C$ . Yet, given value  $v$ , generating a proof of inclusion for any specific fragment can be done by any process. The vector commitment on value  $v$  creates the same commitment  $C$  and the same proofs of inclusion generated by the sender. These could then be propagated to any other process along with the sender's signature on  $C$ .

To complete the description of the Coded MBRB algorithm, we mention that, similar to Original MBRB, the algorithm tries to form a quorum of signatures on some specific vector commitment  $C$ . In parallel, processes collect fragments they verify as part of the value whose vector commitment is  $C$ . Once a process collects enough signatures (for some  $C$ ) and at the same time obtains enough value fragments that are proven to belong to the same  $C$ , the process can reconstruct  $v$  and deliver (accept) it. At this point, the process also disseminates the quorum

of signatures (compacted into a threshold signature, see Section 6.2.2) along with (some of) the fragments. This allows other correct processes to reconstruct the value and verify that a quorum has been reached. In fact, the dissemination of fragments, including fragments that this process did not have before reconstructing the value, is a crucial step in amplifying the number of processes that deliver  $v$  to our stated level of  $\ell_{\text{MBRB}} = n - t - (1 + \varepsilon)d$ . See the full description of Coded MBRB in Section 6.3.1.

Although the Coded MBRB algorithm builds quorums on commitments, it departs substantially from the BRB algorithm proposed by Das, Xiang, and Ren [57], which avoids signatures and relies on hashes only. Their solution provides an overall communication complexity in  $O(n|v| + n^2\lambda)$  that is optimal up to the  $\lambda$  parameter. Following the sender’s initial dissemination of value  $v$ , their proposal runs Bracha’s algorithm on the hash of the broadcast value to ensure agreement. Unfortunately, when used with a message adversary, Bracha’s algorithm loses the sub-optimal Byzantine resilience  $n > 3t + 2d$  that the Original MBRB and Coded MBRB algorithms provide, which is why the solution presented in this chapter avoids it. (See Section 6.4.3 for a more detailed discussion of why this is so.)

## 6.2 Preliminaries

As the other algorithms presented in Chapters 4 and 5, the MBRB algorithm described in this chapter builds upon the underlying system model defined in Section 3.1. In contrast to the previously presented algorithms, the MBRB algorithm of this chapter heavily relies on the **comm**( $m_1, \dots, m_n$ ) operation, which sends (potentially different) messages  $m_j$  to every process  $p_j$  ( $j \in [1..n]$ ). Like for the **broadcast** operation, the message adversary can suppress up to  $d$  messages sent by the **comm** operation. The rest of this section describes additional features of the model that are specific to this chapter.

**General notations and conventions** For a positive integer  $n$ , let  $[n]$  denote the set  $\{1, 2, \dots, n\}$ . A sequence of elements  $(x_1, \dots, x_n)$  is shorthand as  $(x_i)_{i \in [n]}$ . All logarithms are base 2.

### 6.2.1 Error correction codes (ECC)

A central tool used in our algorithm is an error-correction code (ECC) [123]. Intuitively speaking, an ECC takes a message as input and adds redundancy to create a codeword from which the original message can be recovered even when parts of the codeword are corrupted. In this work, we focus on *erasures*, a corruption that replaces a symbol of the codeword with a special erasure mark  $\perp$ . We further denote by  $k$  the reconstruction threshold of the ECC scheme we

use: at least  $k$  out of the  $n$  fragments are sufficient to reconstruct the initial value (where  $n$  is the total number of fragments).<sup>39</sup>

Let  $\mathbb{F}$  denote a finite field whose size we set later, and let  $\perp$  be a special symbol ( $\perp \notin \mathbb{F}$ ). Given two strings of the same length,  $x, y \in \mathbb{F}^n$ , their *Hamming distance* is the number of indices where they differ,  $\Delta(x, y) = |\{i \in [n] \mid x_i \neq y_i\}|$ . Given a subset  $I \subseteq [n]$ , we denote by  $x_I \in \mathbb{F}^{|I|}$  the string  $x$  restricted to the indices in  $I$ .

To avoid confusion with global parameters, we denote the ECC-specific parameters by using a bar (e.g.,  $\bar{x}$ ). An *error-correction code* is a function  $\text{ECC} : \mathbb{F}^{\bar{k}} \rightarrow \mathbb{F}^{\bar{n}}$ , with *rate*  $\bar{r} = \frac{\bar{k}}{\bar{n}}$ , and *distance*  $\bar{d} = \min_{x, y \in \mathbb{F}^{\bar{k}}, x \neq y} \Delta(\text{ECC}(x), \text{ECC}(y))$ . The Singleton bound determines that  $\bar{d} \leq \bar{n} - \bar{k} + 1$ , and when the equality holds, the code is said to be maximum distance separable (MDS). A prominent example of MDS codes is Reed-Solomon (RS) codes [121], which exist for any  $\bar{k}, \bar{n}$ , and  $|\mathbb{F}| \geq \bar{n}$ . Such codes can be efficiently encoded and decoded [123].

**Fact 1** (Erasure Correction Capability). *Any error-correction code of distance  $\bar{d}$  can recover up to  $\bar{d} - 1$  erasures. That is, for any  $y \in (\mathbb{F} \cup \{\perp\})^{\bar{n}}$ , let  $E = \{i \mid y_i = \perp\}$  the set of erased indices. Then, if  $|E| < \bar{d}$ , there is at most a single  $x \in \mathbb{F}^{\bar{k}}$  such that  $y_{[\bar{n}] \setminus E} = \text{ECC}(x)_{[\bar{n}] \setminus E}$ .*

For convenience, we introduce the additional functions  $\text{ecc\_split}(v) \rightarrow (\tilde{v}_1, \dots, \tilde{v}_n)$  and  $\text{ecc\_reconstruct}(\tilde{v}_1, \dots, \tilde{v}_n) \rightarrow v$ . The  $\text{ecc\_split}$  function takes a value  $v$ , obtains its codeword using the  $\text{ECC}(v)$  function, and splits it into  $n$  fragments  $(\tilde{v}_1, \dots, \tilde{v}_n)$  of the same size. Conversely, the  $\text{ecc\_reconstruct}$  function takes  $n$  value fragments  $(\tilde{v}_1, \dots, \tilde{v}_n)$  (where at most  $n - k$  may be equal to the  $\perp$  sentinel value), and applies the inverse function of  $\text{ECC}(\cdot)$  to obtain the original value  $v$ , even in spite of the presence of at most  $n - k$  erasures.

### 6.2.2 Cryptographic primitives

Our algorithm relies on cryptographic assumptions. We assume that the Byzantine processes are computationally bounded with respect to the security parameter, denoted by  $\lambda$ . That is, all cryptographic algorithms are polynomially bounded in the input  $1^\lambda$ . We assume that  $\lambda = \Omega(\log n)$ . We further assume the availability of Public Key Infrastructure (PKI), which is the setting that assumes each process is assigned a pair of public/private keys, generated according to some common key-generation algorithm. Further, at the start of the computation, each process holds its own private key and the public key of all other parties. This setting implies

<sup>39</sup>. Following the dedicated literature, we use the letter  $k$  to denote the ECC reconstruction threshold, however it should not be confused with the parameter  $k$  of the  $k2l$ -cast abstraction presented in Chapter 5.

private and authenticated channels. In particular, each process has public and private keys to support the following cryptographic primitives.

**Threshold signatures** In a  $(\tau, n)$  *threshold signature* scheme [130], at least  $\tau$  out of all  $n$  processes (the threshold) produce individual *signature shares*  $\sigma$  for the same value  $v$ , which are then aggregated into a fixed-size *threshold signature*  $\Sigma$ . Verifying  $\Sigma$  does not require the public keys of the signers; one just needs to use a single system-wide public key, the same for all threshold signatures produced by the scheme. This system public key, known to everyone, is generated during the setup phase of the system, and is also distributed through the PKI.

Formally, we define a  $(\tau, n)$  threshold signature scheme as a tuple of (possibly randomized) algorithms (`ts_sign_share`, `ts_verify_share`, `ts_combine`, `ts_verify`). The signing algorithm executed by process  $p_i$  (denoted, `ts_sign_sharei`) takes a value  $v$  (and implicitly a private key) and produces a signature  $\sigma = \text{ts\_sign\_share}_i(v)$ . The share verification algorithm takes a value  $v$ , a signature share  $\sigma_i$ , and the identity  $i$  of its signer  $p_i$  (and implicitly  $p_i$ 's public key), and outputs a single bit,  $b = \text{ts\_verify\_share}(v, \sigma_i, i) \in \{\mathbf{true}, \mathbf{false}\}$ , which indicates whether the signature share is valid or not. The combination algorithm takes a set *sigs* of  $\tau$  valid signature shares produced by  $\tau$  out of  $n$  processes and the associated value  $v$  (and implicitly the system public key) and outputs a threshold signature  $\Sigma = \text{ts\_combine}(sigs)$ . The threshold signature verification algorithm takes a value  $v$  and a threshold signature  $\Sigma$  (and implicitly the system public key) and outputs a single bit  $b = \text{ts\_verify}(v, \Sigma) \in \{\mathbf{true}, \mathbf{false}\}$ , indicating if the threshold signature is valid or not.

We require the conventional robustness and unforgeability properties for threshold signatures. This scheme is parameterized by the security parameter  $\lambda$ , and the size of signature shares and threshold signatures,  $|\sigma| = |\Sigma| = O(\lambda)$  bits, is independent of the size of the signed value,  $v$ . In our algorithm, we take  $\tau = \lfloor \frac{n+t}{2} \rfloor + 1$  (*i.e.*, the integer right above  $\frac{n+t}{2}$ ).

**Vector commitments (VC)** A *vector commitment* (VC) is a short digest  $C$  for a vector of elements  $V$ , upon which a user can then generate a *proof of inclusion*  $\pi$  (sometimes called *partial opening*) of some element in  $V$  without disclosing the other elements of  $V$  to the verifier: the verifier only needs  $C$ ,  $\pi$ , the element, and its index in the vector to verify its inclusion in  $V$ . A Merkle tree [101] is a notable example of vector commitment, although with several sub-optimal properties. For example, for a hash size of  $\lambda$ , a Merkle proof of inclusion is of  $O(\lambda \log|V|)$  bits, which is significantly larger than modern schemes such as Catalano-Fiore vector commitments [39], which produce proofs of inclusion with an optimal size of  $O(\lambda)$  bits. In our construction, we use these optimal VC schemes (such as Catalano-Fiore's), which provide commitments and proofs in  $O(\lambda)$  bits. The VC scheme provides two operations, parameterized by the security parameter  $\lambda$ : `vc_commit( $\cdot$ )` and `vc_verify( $\cdot$ )`, that work as follows. For any vector

of strings  $V = (x_1, \dots, x_n) = (x_i)_{i \in [n]}$ , the function  $\text{vc\_commit}(V) \rightarrow (C, \pi_1, \dots, \pi_n)$  returns  $C$ , the commitment, and every  $\pi_i$ , the proof of inclusion for  $x_i$ . The following hold.

- **Proof of inclusion (Correctness).** Let  $(C, (\pi_i)_{i \in [n]}) = \text{vc\_commit}((x_1, \dots, x_n))$ . Then for any  $i \in [n]$ , it holds that  $\text{vc\_verify}(C, \pi_i, x_i, i) = \text{true}$ .
- **Collision-resistance (Binding).** For any  $j \in [n]$  and any randomized algorithm  $A$  taking  $(x_i)_{i \in [n]}$  and  $(C, (\pi_i)_{i \in [n]}) = \text{vc\_commit}(x_1, \dots, x_n)$  as input,  $\Pr(A \text{ outputs } (x_{j'}, \pi_{j'}, j) \wedge \text{vc\_verify}(C, \pi_{j'}, x_{j'}, j) = \text{true}) < 2^{-\Omega(\lambda)}$ . Namely, it is difficult to generate  $x_{j'} \neq x_j$  and a valid proof  $\pi_{j'}$  for the same commitment  $C$ .

We omit the traditional Hiding property of VC schemes (a commitment does not leak information on the vector [4]) as it is unneeded in our algorithm. We also implicitly assume that the  $\text{vc\_commit}$  operation is deterministic: it always returns the same commitment  $C$  given the same input vector  $V$ , no matter the calling process  $p_i$ . This is the case for Catalano-Fiore's scheme [39], which does not use random salt (a.k.a *blinding factor*).

### 6.3 The Coded MBRB algorithm

The proposed solution, named Coded MBRB (Algorithms 8-9), allows a distinguished sender  $p_s$  (known to everyone) to disseminate one specific value  $v$ . In Section 6.4.2, we discuss how to extend this algorithm so that it implements a general MBRB algorithm, allowing any process to be the sender, as well as allowing multiple instances of the MBRB, either with the same or different senders, to run concurrently. In the algorithm, we instantiate the threshold signature scheme with the threshold value set to  $\tau = \lfloor \frac{n+t}{2} \rfloor + 1$  (see Section 6.2.2).

Algorithm 8 describes the  $\text{mbrb\_broadcast}(v)$  operation as well as helper functions used in Coded MBRB, and Algorithm 9 describes the phases of Coded MBRB. The  $\text{mbrb\_broadcast}(v)$  operation takes value  $v$  and disseminates it reliably to a minimum bound of correct processes, denoted  $\ell_{\text{MBRB}}$ . That is, after executing Algorithm 9, and assuming a correct sender, at least  $\ell_{\text{MBRB}}$  correct processes will have invoked the  $\text{mbrb\_deliver}(v)$  callback, while no correct process will have invoked  $\text{mbrb\_deliver}(v' \neq v)$ .

The Coded MBRB algorithm (Algorithms 8-9) relies on the following assumption (the ‘‘c’’ prefix stands for ‘‘coded’’).

**c-MBRB-Assumption.**  $n > 3t + 2d$  and  $k \leq n - t - 2d$ .

As in the Original MBRB algorithm (Algorithm 3, page 42), Coded MBRB assumes the necessary and sufficient condition  $n > 3t + 2d$  for implementing MBRB (Theorem 1). Furthermore, while we have some flexibility in selecting the reconstruction threshold  $k$  of the erasure-correct-

ing code (as discussed in Section 6.4.1), which affects the parameters of the ECC and thus the communication complexity, Coded MBRB requires that  $k$  will not be “too large,” *i.e.*,  $k \leq n - t - 2d$ .

### 6.3.1 Algorithm description

The operation `mbrb_broadcast( $v$ )` allows the sender  $p_s$  to start disseminating value  $v$  (line 16). The sender (line 17) starts by invoking `compute_frag_vec_commit( $v$ )` (Algorithm 8). This function encodes the value  $v$  using an error-correction code, divides it into  $n$  fragments and constructs a vector commitment with an inclusion proof for each fragment. The function returns several essential values: the commitment  $C$ , and the fragment details  $(\tilde{v}_j, \pi_j, j)$ , which contain the fragment data itself  $\tilde{v}_j$  (the  $j$ -th part of the `ecc_split( $v$ )`; see below for detail), a proof of inclusion  $\pi_j$  for that part, and the respective index  $j$  of each fragment. For ease of reference, let `Commitment( $v$ )` represent the commitment  $C$  obtained from `compute_frag_vec_commit( $v$ )`. This commitment serves as a compact representation of the entire value  $v$ .

The sender process  $p_s$  is responsible for signing the computed commitment  $C$  and generating a signature share denoted  $sig_s$  (line 18). Notably, this signature share includes  $p_s$ ’s identifier. The sender then initiates  $v$ ’s propagation by employing the `comm` operation (line 19), which sends to each process  $p_j$  an individual message  $m_j$ . The message  $m_j$  is of type `SEND` and includes several components: the commitment  $C$ , the  $j$ -th fragment details  $(\tilde{v}_j, \pi_j, j)$ , and the signature share  $sig_s$  (line 18) for  $C$ .

The rest of the algorithm progresses in two phases, which we describe in turn. The first phase is responsible for value dissemination, which forwards value fragments received by the sender. The other role of this phase is reaching a quorum of processes that vouch for the same value. A process vouches for a single value by signing its commitment. processes collect and save signature shares until it is evident that sufficiently many processes agree on the same value. The subsequent phase focuses on disseminating the quorum of signature shares so that it is observed by at least  $\ell_{MBRB}$  correct processes, and on successfully terminating while ensuring the delivery of the reconstructed value.

**Validating message integrity** The validity of the signatures and inclusion proofs are checked each time a new message is received (at lines 20, 26, or 44) using the function `is_valid` (Algorithm 8). All message types (`SEND`, `FORWARD`, and `BUNDLE`) carry a vector commitment ( $C$  or  $C'$ ) and up to two value fragments with their inclusion proofs. Moreover, the `SEND` and `FORWARD` types contain up to two signature shares for the provided commitment, and the `BUNDLE` type contains a threshold signature for the provided commitment. The validation hinges on the following three key criteria.

1. Every enclosed signature share or threshold signature must be valid and correspond to the accompanying commitment.



2. For `SEND` or `FORWARD` messages, the signature share from the designated sending process  $p_s$  must be present.
3. All value fragments must contain valid inclusion proofs for the provided commitment.

Note that  $\pi_i$ , the proof of inclusion of  $\tilde{v}_i$ , does not need to be signed by  $p_s$ , as the commitment already is.

**Phase I: Message dissemination** This phase facilitates the widespread distribution of the value fragments  $\tilde{v}_j$ . Recall that the sender has sent to each process a different (encoded) fragment of the value  $v$ , however, no process currently holds enough information to retrieve the value  $v$ . The phase also sets the ground for forming a quorum on the value  $v$ .

When a process receives a `SEND` message from the sender, it begins by validating the fragment's authenticity (line 21). Process  $p_i$  then determines whether it had previously broadcast a

```

1 function compute_frag_vec_commit( $v$ ) is
2    $(\tilde{v}_1, \dots, \tilde{v}_n) \leftarrow \text{ecc\_split}(v)$ ;
3    $(C, \pi_1, \dots, \pi_n) \leftarrow \text{vc\_commit}(\tilde{v}_1, \dots, \tilde{v}_n)$ ;
4   return  $(C, (\tilde{v}_j, \pi_j, j)_{j \in [n]})$ .

5 function is_valid( $C, \text{fragtuples}, \text{sigs}, \text{isThreshSig}$ ) is
6   if  $\neg \text{isThreshSig}$  then
7     if  $\exists (\sigma_x, x) \in \text{sigs} : \neg \text{ts\_verify\_share}(C, \sigma_x, x)$  then return false;
8     if  $(\star, s) \notin \text{sigs}$  then return false;
9   else if  $\neg \text{ts\_verify}(C, \text{sigs})$  then return false;
10   $\text{fragtuples} \leftarrow \text{fragtuples} \setminus \{\perp\}$ ;
11  if  $\exists (\tilde{v}_x, \pi_x, x) \in \text{fragtuples} : \neg \text{vc\_verify}(C, \pi_x, \tilde{v}_x, x)$  then return false;
12  return true.

13 function get_thresh_sig( $C$ ) is
14   $\text{sigs}_C \leftarrow \{\text{all saved signature shares for } C\}$ ;
15  return  $\Sigma_C \leftarrow \begin{cases} \text{the threshold signature saved for } C & \text{if it exists} \\ \text{else, ts\_combine}(\text{sigs}_C) & \text{if } |\text{sigs}_C| > \frac{n+t}{2} \\ \perp & \text{otherwise} \end{cases}$ .

16 operation mbrb_broadcast( $v$ ) is  $\triangleright$  only executed by the sender  $p_s$ 
17   $(C, (\tilde{v}_j, \pi_j, j)_j) \leftarrow \text{compute\_frag\_vec\_commit}(v)$ ;
18   $\text{sig}_s \leftarrow (\text{ts\_sign\_share}_s(C), s)$ ;
19  comm( $m_1, \dots, m_n$ ) where  $m_j = \text{SEND}(C, (\tilde{v}_j, \pi_j), \text{sig}_s)$ .

```

Algorithm 8: Helper functions and `mbrb_broadcast` operation of the Coded MBRB algorithm (code for  $p_i$ )

Phase I: Value dissemination

20 **when** SEND( $C'$ ,  $(\tilde{v}_j, \pi_j)$ ,  $sig_s$ ) **is received from**  $p_s$  **do**  
21     **if**  $\neg$  is\_valid( $C'$ ,  $\{(\tilde{v}_i, \pi_i, i)\}$ ,  $\{sig_s\}$ ,  $isThreshSig = \text{false}$ ) **then return**;  
22     **if**  $p_i$  already executed line 25 or signed some commitment  $C'' \neq C'$  **then return**;  
23      $sig_i \leftarrow$  (ts\_sign\_share $_i(C')$ ,  $i$ );  
24     save  $\tilde{v}_i$ ,  $sig_s$ , and  $sig_i$  for  $C'$ ;  
25     **broadcast** FORWARD( $C'$ ,  $(\tilde{v}_i, \pi_i, i)$ ,  $\{sig_s, sig_i\}$ ).

26 **when** FORWARD( $C'$ ,  $fragtuple_j$ ,  $sigs_j = \{sig_s, sig_i\}$ ) **is received from**  $p_j$  **do**  
27     **if**  $\neg$  is\_valid( $C'$ ,  $\{fragtuple_j\}$ ,  $sigs_j$ ,  $isThreshSig = \text{false}$ ) **then return**;  
28     **if**  $p_i$  already signed some commitment  $C'' \neq C'$  **then return**;  
29     save  $sigs_j$  for  $C'$ ;  
30     **if**  $fragtuple_j \neq \perp$  **then**  
31          $(\tilde{v}_j, \pi_j, j) \leftarrow fragtuple_j$ ;  
32         save  $\tilde{v}_j$  for  $C'$ ;  
33     **if** no FORWARD message sent yet **then**  
34          $sig_i \leftarrow$  (ts\_sign\_share $_i(C')$ ,  $i$ );  
35         save  $sig_i$  for  $C'$ ;  
36         **broadcast** FORWARD( $C'$ ,  $\perp$ ,  $\{sig_s, sig_i\}$ ).

Phase II: Reaching quorum and termination

37 **when**  $\left\{ \begin{array}{l} \exists C: \text{get\_thresh\_sig}(C) \neq \perp \wedge |\{\text{saved } \tilde{v}_j \text{ for } C\}| \geq k \\ \wedge \text{no value mbrb-delivered yet} \end{array} \right\}$  **do**  
38      $v \leftarrow$  ecc\_reconstruct( $\tilde{v}_1, \dots, \tilde{v}_n$ ),  $\left\{ \begin{array}{l} \text{where } \tilde{v}_j \text{ are taken from line 37} \\ \text{when a fragment is missing use } \perp \end{array} \right\}$ ;  
39      $(C', (\tilde{v}'_j, \pi'_j, j)) \leftarrow$  compute\_frag\_vec\_commit( $v$ );  
40     **if**  $C \neq C'$  **then return**;  
41      $\Sigma_C \leftarrow$  get\_thresh\_sig( $C$ );  
42     **comm**( $m_1, \dots, m_n$ ) **where**  $m_j =$  BUNDLE( $C$ ,  $(\tilde{v}'_i, \pi'_i, i)$ ,  $(\tilde{v}'_j, \pi'_j, j)$ ,  $\Sigma_C$ );  
43     mbrb\_deliver( $v$ ).

44 **when** BUNDLE( $C'$ ,  $(\tilde{v}'_j, \pi'_j, j)$ ,  $fragtuple'_i, \Sigma$ ) **received from**  $p_j$  **do**  
45     **if**  $\neg$  is\_valid( $C'$ ,  $\{(\tilde{v}'_j, \pi'_j, j), fragtuple'_i\}$ ,  $\Sigma$ ,  $isThreshSig = \text{true}$ ) **then return**;  
46     save  $\tilde{v}'_j$  and  $\Sigma$  for  $C'$ ;  
47     **if** no BUNDLE message sent yet **and**  $fragtuple'_i \neq \perp$  **then**  
48          $(\tilde{v}'_i, \pi'_i, i) \leftarrow fragtuple'_i$ ;  
49         save  $\tilde{v}'_i$  for  $C'$ ;  
50         **broadcast** BUNDLE( $C'$ ,  $(\tilde{v}'_i, \pi'_i, i)$ ,  $\perp, \Sigma$ ).

Algorithm 9: Phases of the Coded MBRB algorithm (code for  $p_i$ , single-shot, single-sender,  $n > 3t + 2d$ ,  $k \leq n - t - 2d$ , threshold for the TS scheme  $\tau = \lfloor \frac{n+t}{2} \rfloor + 1$ )

FORWARD message at line 25 or signed a commitment  $C''$  from  $p_s$  distinct from the currently received  $C'$ , in which case the incoming message is discarded (line 22). Otherwise,  $p_i$  generates its own signature share  $sig_i$  for the commitment  $C'$  (line 23). Subsequently,  $p_i$  proceeds to save its signature share and the received information (line 24), encompassing the fragment  $\tilde{v}_i$  and the associated signature share  $sig_s$ , linked to the specific commitment  $C'$ . We clarify that  $p_i$  never saves multiple copies of the same information, *i.e.*, all save operations are to be read as adding an item to a set. Process  $p_i$  then disseminates all the relevant information, by broadcasting the message `FORWARD( $C'$ ,  $(\tilde{v}_i, \pi_i, i)$ ,  $\{sig_s, sig_i\}$ )` (line 25). The broadcast of a FORWARD message is instrumental in disseminating information for several reasons. First, up to  $d$  processes might not receive the sender's SEND message. Second, this is the process's way to disseminate its own fragment and signature share for that specific  $C'$ .

Upon the arrival of a `FORWARD( $C'$ ,  $fragtuple_j$ ,  $sigs_j$ )` message from  $p_j$  (line 26), the recipient  $p_i$  validates the incoming message using the `is_valid` function (Algorithm 8), discarding invalid messages (line 27). As for SEND messages,  $p_i$  checks if it already signed a commitment  $C''$  from  $p_s$ , in which case it discards the message (line 28). Subsequently,  $p_i$  saves the set of signature shares  $sigs_j$  linked to the specific commitment  $C'$  (line 29) and fragment contained in this message, if any (line 32). Also,  $p_i$  assesses whether a FORWARD message has been previously dispatched (line 33). If it has already done so, there is no reason to re-send it, and the processing ends here. Otherwise, similar to above,  $p_i$  generates and saves its own signature share  $sig_i$  for the commitment  $C'$ , and broadcasts the message `FORWARD( $C'$ ,  $\perp$ ,  $\{sig_s, sig_i\}$ )` (lines 34-36). Note that, in this case,  $p_i$  is unaware of his own fragment (*i.e.*, it has not received a SEND message, or otherwise it would have already sent a FORWARD message at line 25); therefore it sends the sentinel value  $\perp$  instead.

**Phase II: Reaching quorum and termination** This phase relies on the `get_thresh_sig` function described in Algorithm 8, which, given a commitment  $C$ , either returns a threshold signature for  $C$  (received beforehand or aggregating  $\tau = \lfloor \frac{n+t}{2} \rfloor + 1$  signature shares saved for  $C$ ) if it exists, or  $\perp$  otherwise. This phase focuses on ensuring that, once a Byzantine quorum (represented by the threshold signature returned by `get_thresh_sig`) and enough value fragments for reconstructing the original value  $v$  are gathered, at least  $\ell_{MBRB}$  correct processes deliver  $v$  and terminate. Process  $p_i$  enters Phase II only when there is a commitment  $C'$  for which `get_thresh_sig` returns a valid threshold signature, and  $p_i$  saves at least  $k$  value fragments. As long as no value from  $p_s$  was delivered (line 37),  $p_i$  reconstructs value  $v_i$  (line 38) using the saved value fragments, and use this value as an input to `compute_frag_vec_commit` (line 39), which outputs its commitment  $C = \text{Commitment}(m_i)$  along with coded value fragments and proofs of inclusion,  $(\tilde{v}'_j, \pi'_j, j)$ . Process  $p_i$  then ensures that the computed commitment  $C$  matches the saved commitment  $C'$  (line 40). If this condition holds true, then  $v_i = v$  is the value sent by the sender, and in particular,  $p_i$  now holds *all* the possible fragments for  $v$  along with their valid

proof of inclusion, including fragments it has never received before! Process  $p_i$  then retrieves the threshold signature  $\Sigma_C$  of  $C$  using the `get_thresh_sig` function (line 41), and disseminates it along with the value fragments to the rest of the network. In particular, to each  $p_j$  in the network,  $p_i$  sends a `BUNDLE` message (line 42) that includes the commitment  $C$ , fragment details  $(\tilde{v}'_i, \pi'_i, i)$  and  $(\tilde{v}'_j, \pi'_j, j)$ , and the associated threshold signature  $\Sigma_C$ . After these transmissions,  $p_i$  can `mbrb-deliver` the reconstructed value  $v_i$  (line 43).

The parameter  $k$  used at line 37 is the number of (valid) fragments sufficient to reconstruct value  $v$  by the error-correction code `ECC`. This parameter should be practically selected by the desired  $\ell_{MBRB}$  given in Theorem 6. That is, one needs to set  $\varepsilon > 0$  for  $\ell_{MBRB} = n - t - (1 + \varepsilon)d$  and then choose  $k \geq 1 + \frac{\varepsilon}{1+\varepsilon}(n - t - d)$ . See details in Section 6.4.1.

Upon the arrival of a `BUNDLE`( $C', (\tilde{v}'_j, \pi'_j, j), \text{fragtuple}'_i, \Sigma$ ) message from  $p_j$  (line 44), the recipient  $p_i$  validates the received message using the `is_valid` function (with the `isThreshSig` parameter set to `true` to indicate that we verify a threshold signature) and discards invalid messages (line 45). Process  $p_i$  proceeds to save the arriving value fragment  $\tilde{v}'_j$  and threshold signature  $\Sigma$  for the specific commitment  $C'$  (line 46). In the case that no `BUNDLE` message was sent by  $p_j$  and the received `fragtuple}'_i` is nonempty (so  $p_i$  learns its fragment, which it saves at line 49, unless already known),  $p_i$  broadcasts a `BUNDLE`( $C', (\tilde{v}'_i, \pi'_i, i), \perp, \Sigma$ ) message (line 50).

The use of the  $\perp$  sentinel value appears also in `BUNDLE` messages (lines 42 and 50). A `BUNDLE` message might contain up to two fragments: the sender's fragment ( $\tilde{v}'_i$  in the pseudo-code), which is always included, and the receiver's fragment ( $\tilde{v}'_j$ ), which is included only when the sender was able to reconstruct the value  $v$  (at line 38). The sender's fragments are collected by the receivers and allow reconstruction of the value once enough `BUNDLE` messages are received. The receiver's fragment allows the receiver to send `BUNDLE` messages (with its fragment), facilitating the dissemination of both threshold signatures and fragments.

### 6.3.2 The error-correction code in use

The function `compute_frag_vec_commit` (Algorithm 8) uses an error-correction code at line 2 to encode value  $v$ , before it is split into  $n$  fragments that will be disseminated by the processes. The code uses a fixed parameter  $k$ , that can be set later. Our algorithm requires that the `ECC` will be able to decode value  $v$  from any subset of  $k$  fragments out of the  $n$  fragments generated. That is, we need an `ECC` that can deal with erasures, where the erased symbols are those contained in the  $n - k$  missing fragments. To that end, we use a Reed-Solomon code `ECC` :  $\mathbb{F}^k \rightarrow \mathbb{F}^{\bar{n}}$  with  $\bar{k} > |v| / \log|\mathbb{F}|$ . Each fragment contains  $\frac{\bar{n}}{n}$  symbols of the codeword, and to be able to recover from  $(n - k) \times \frac{\bar{n}}{n}$  erased symbols by Fact 1, we can set the code's distance to be  $\bar{d} > (n - k) \times \frac{\bar{n}}{n}$ . Since a Reed-Solomon code is MDS (see Section 6.2.1),  $\bar{d} \leq \bar{n} - \bar{k} + 1$ , and we can set  $\bar{n} > \frac{\bar{n}}{k(k-1)}$ . The code will have a constant rate, *i.e.*,  $|\text{ECC}(v)| = O(|v|)$ , as long as  $v$  is

sufficiently long, *i.e.*,  $|v| = \Omega(n \log |\mathbb{F}|)$ , which implies that  $\bar{k} = \Omega(n)$ , and as long as  $k = \Omega(n)$ . Recall also that  $|\mathbb{F}| \geq \bar{n}$  is in a Reed-Solomon code.

### 6.3.3 Intuition of Coded MBRB's Global-delivery property

The following main theorem states that the Coded MBRB algorithm (composed of Algorithm 8 and Algorithm 9) is correct.

**Theorem 6** (MBRB-Correctness). *If c-MBRB-Assumption, Algorithms 8 and 9 implement MBRB with the guarantee  $\ell_{\text{MBRB}} = c - (1 + \varepsilon)d$ , where  $\varepsilon > 0$ .*

Let us remind that Coded-MBRB relies on c-MBRB-Assumption, which states that  $n > 3t + 2d$  (necessary and sufficient condition for MBRB, see Theorem 1, page 35) and  $k \leq n - t - 2d$  (the ECC reconstruction threshold is not too high). Let us also recall that Algorithms 8 and 9 describe are *single sender* and *single shot* broadcast algorithm, therefore the proof of correctness of Coded MBRB is done on a *single sender* and *single shot* version of the MBRB specification given in Section 3.2, page 33: we only consider a single sender,  $p_s$ , that mbrb-broadcasts only one value. As explained in Section 6.4.2, the Coded MBRB algorithm (and its proofs) can easily be generalized to the multi-sender/multi-shot case by employing sender identities and sequence numbers.

For concision, the full derivations of the correctness proof of Coded MBRB are given in Appendix D. In the following, we sketch the proof of MBRB-Global-delivery property of Theorem 6 (assuming the other properties hold). The detailed proof of this property can be found in page 153.

**Lemma 36** (MBRB-Global-delivery). *If a correct process  $p_i$  mbrb-delivers a value  $v$ , then at least  $\ell_{\text{MBRB}} = c - d \left( \frac{1}{1 - \frac{k-1}{c-d}} \right)$  correct processes mbrb-deliver  $v$ .*

**Proof sketch.** Let us denote by  $C_v$  the vector commitment that `compute_frag_vec_commit(v)` returns. The proof counts the BUNDLE messages disseminated by correct processes. If a correct process disseminates a BUNDLE message both at lines 42 and 50, we only consider the one from line 42.

Let  $B_{\text{send}}$  be the set of correct processes that disseminate at least one BUNDLE message during the execution. Similarly, let  $B_{\text{recv}}$  be the set of correct processes that receive at least one valid BUNDLE message from a correct process during the execution. Let  $B_{k,\text{recv}}$  be the set of correct

processes that receive **BUNDLE** messages from at least  $k$  distinct correct processes. The following holds.

**Observation 36.1.**  $c \geq |B_{\text{recv}}| \geq c - d$  and  $c \geq |B_{\text{send}}| \geq c - d$ .

**Proof of Observation 36.1.** Since  $B_{\text{send}}$  and  $B_{\text{recv}}$  contain only correct processes, trivially  $c \geq |B_{\text{send}}|$  and  $c \geq |B_{\text{recv}}|$ . Since  $p_i$  mbrb-delivers  $v$  at line 43, it disseminated **BUNDLE** messages of the form  $\text{BUNDLE}(C_v, (\tilde{v}'_i, \pi'_i, i), (\tilde{v}'_j, \pi'_j, j), \Sigma_v)$  (line 42). The **BUNDLE** messages sent by  $p_i$  eventually reach at least  $c - d$  correct processes. The detailed proof (in Appendix D) shows that these **BUNDLE** messages are valid. Hence,  $B_{\text{recv}} \geq c - d > 0$  proves the lemma's first part.

$B_{\text{recv}}$ 's processes execute lines 44-45, and reach line 47. Because  $p_i$  has included a non- $\perp$  second fragment in all its **BUNDLE** message, any of the  $c - d$  processes of  $B_{\text{recv}}$  that receive one of  $p_i$ 's **BUNDLE** messages and has not already sent a **BUNDLE** message passes the condition at line 47. Each such process then disseminates a (valid) **BUNDLE** message at line 50. This yields  $|B_{\text{send}}| \geq c - d$ .  $\square$  Observation 36.1

**Observation 36.2.**  $|B_{k,\text{recv}}| \times |B_{\text{send}}| + (k - 1)(|B_{\text{recv}}| - |B_{k,\text{recv}}|) \geq |B_{\text{send}}|(c - d)$ .

**Proof of Observation 36.2.** Let us denote by  $\#\text{BUNDLE}$  the overall number of valid **BUNDLE** messages received by correct processes from distinct correct senders. More specifically, in the case when a correct process disseminates **BUNDLE** messages both at lines 42 and 50, we only consider the *last* **BUNDLE** message, *i.e.*, the one of line 42. We know that each  $p \in B_{\text{send}}$  sends a **BUNDLE** message, which by Lemma 58 is valid. As the message adversary may drop up to  $d$  out of the  $n$  messages of this **comm** instance, we are guaranteed that at least  $c - d$  correct processes receive  $p$ 's **BUNDLE** message. This immediately implies that

$$\#\text{BUNDLE} \geq |B_{\text{send}}|(c - d). \quad (13)$$

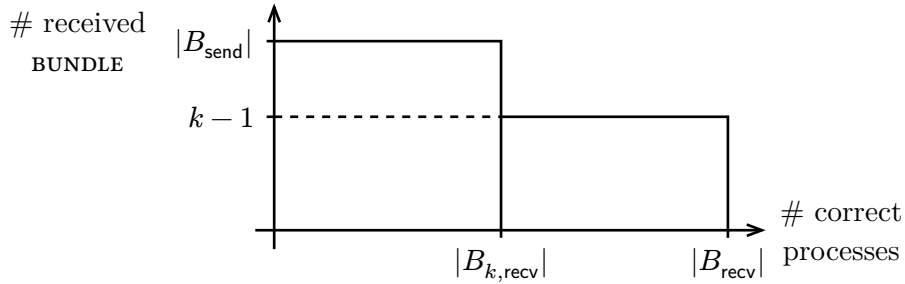


Figure 15: Distribution of distinct **BUNDLE** messages received by correct processes; combining Observation 36.1 and c-MBRB-Assumption shows that  $|B_{\text{send}}| > k - 1$

$B_{k,\text{recv}}$ 's processes may receive up to  $|B_{\text{send}}|$  BUNDLE messages (Figure 15) from distinct correct senders (one from each sender in  $B_{\text{send}}$ ), for a maximum of  $|B_{k,\text{recv}}| \times |B_{\text{send}}|$  BUNDLE messages. The remaining process of  $B_{\text{recv}} \setminus B_{k,\text{recv}}$  may each receive up to  $k-1$  valid BUNDLE messages from distinct correct senders, by definition of  $B_{k,\text{recv}}$ . As  $B_{k,\text{recv}} \subseteq B_{\text{recv}}$  by definition,  $|B_{\text{recv}} \setminus B_{k,\text{recv}}| = |B_{\text{recv}}| - |B_{k,\text{recv}}|$ , and the processes of  $B_{\text{recv}} \setminus B_{k,\text{recv}}$  account for up to  $(k-1)(|B_{\text{recv}}| - |B_{k,\text{recv}}|)$  BUNDLE messages overall. The BUNDLE messages counted by  $\#\text{BUNDLE}$  are received either by correct processes in  $B_{k,\text{recv}}$  or in  $B_{k,\text{recv}} \setminus B_{\text{recv}}$ . This implies  $|B_{k,\text{recv}}| \times |B_{\text{send}}| + (k-1)(|B_{\text{recv}}| - |B_{k,\text{recv}}|) \geq \#\text{BUNDLE}$ . Combining the latter and  $\#\text{BUNDLE} \geq |B_{\text{send}}|(c-d)$  yields the desired inequality.  $\square$  Observation 36.2

**Observation 36.3.**  $|B_{k,\text{recv}}| \geq c - d \left( \frac{1}{1 - \frac{k-1}{c-d}} \right)$ .

**Proof of Observation 36.3.** By Observation 36.2,  $|B_{\text{recv}}| \leq c$ , and  $k \geq 1$ ,

$$|B_{k,\text{recv}}| \times (|B_{\text{send}}| - k + 1) \geq |B_{\text{send}}|(c-d) - |B_{\text{recv}}|(k-1) \geq |B_{\text{send}}|(c-d) - c(k-1).$$

By Observation 36.1 and c-MBRB-Assumption,  $|B_{\text{send}}| \geq c-d \geq c-2d \geq k$ . Thus,  $|B_{\text{send}}| - k + 1 > 0$  and the previous equation can be written to

$$|B_{k,\text{recv}}| \geq \frac{|B_{\text{send}}|(c-d) - c(k-1)}{|B_{\text{send}}| - k + 1}. \quad (14)$$

Equation (14)'s right-hand side monotonically increases in  $|B_{\text{send}}|$  when  $|B_{\text{send}}| > k-1$ , as its derivative,  $\frac{d(k+1)}{(|B_{\text{send}}|-k+1)^2}$ , is positive. By Observation 36.1,  $|B_{\text{send}}| \in [c-d, c] \subseteq [k, c]$ . The minimum of the right-hand side of (14) is therefore obtained for  $B_{\text{send}} = c-d$ , yielding

$$|B_{k,\text{recv}}| \geq \frac{(c-d)^2 - c(k-1)}{(c-d) - k + 1} = c - d \left( \frac{1}{1 - \frac{k-1}{c-d}} \right),$$

which concludes the proof of the observation.  $\square$  Observation 36.3

**Observation 36.4.** All processes in  $B_{k,\text{recv}}$  *mbrb-deliver*  $v$ .

**Proof of Observation 36.4.** Let  $p_u \in B_{k,\text{recv}}$ , *i.e.*,  $p_u$  receives  $k$  valid BUNDLE messages from  $k$  distinct correct processes. Denote  $\text{BUNDLE}(C_x, (\tilde{v}_x, \pi_x, x), \star, \Sigma_x)$  these  $k$  messages with  $x \in [k]$ . The detailed proof in Appendix D (page 153) shows that for all  $x \in [k]$ ,  $C_x = C_v$ . In addition,  $p_u$  saves each received threshold signature  $\Sigma_x$ , which is valid for  $C_v$ .

Because the messages are valid, so are the proofs of inclusion  $\pi_x$ , and as we have assumed that vector commitments are collision-resistant,  $C_x = C_v$  implies that the received fragments  $\tilde{v}_x$  all belong to the set of fragments computed by  $p_s$  (line 17) for  $v$ . As the **BUNDLE** messages were received from  $k$  distinct correct processes,  $p_u$  receives at least  $k$  distinct valid fragments for  $v$  during its execution. If  $p_u$  has not mbrb-delivered any value yet, the condition at line 37 eventually becomes true for  $C_v$ , and  $p_u$  reconstructs  $v$  (line 38), since it has at least  $k$  (correct) fragments, which are sufficient for the correct recovery of  $v$ . Then,  $p_u$  mbrb-delivers  $v$  (line 43). If  $p_u$  has already mbrb-delivered some value  $v'$ , MBRB-No-duplicity implies  $v' = v$ , since  $p_i$  is known to have mbrb-delivered  $v$ . Therefore,  $p_u$  mbrb-delivers  $v$  in all cases.  $\square$  Observation 36.4

Finally, the lemma is implied by Observations 36.3 and 36.4, and the fact that all processes in  $B_{k, \text{rcv}}$  are correct.  $\square$

### 6.3.4 Communication analysis of Coded MBRB

In this section, we analyze the communication of Coded MBRB (Algorithms 8-9) and prove Theorem 7. Let us recall from Section 3.2 that the variables  $\mu$  and  $\kappa$  respectively denote the overall numbers of messages and bits communicated by correct processes during the execution of an MBRB algorithm (in this case, Coded MBRB) when the sender  $p_s$  is correct.

**Theorem 7** (MBRB-Performance). *If c-MBRB-Assumption is satisfied and the sender  $p_s$  is correct, Algorithms 8 and 9 provide the following MBRB guarantees:*

- $\mu = 4n^2$  messages sent overall,
- $\kappa = O(n|v| + n^2\lambda)$  bits sent overall.

**Proof.** Let us count the messages communicated by counting **comm** and **broadcast** invocations. The sender  $p_s$  sends **SEND** messages at line 19. In Phase I, each correct process that has received a **SEND** message broadcasts a **FORWARD** message once (lines 22 and 25). However, if it receives a **FORWARD** before the **SEND** arrives, it performs one additional **FORWARD** broadcast (line 36). This yields at most 2 **comm** and **broadcast** invocations per correct process until the end of Phase I. We can safely assume that a correct sender always sends a single **FORWARD** (*i.e.*, it immediately and internally receives the **SEND** message sent to self). Thus,  $p_s$  is also limited to at most 2 invocations up to this point. In Phase II, each correct process that mbrb-delivers a value at line 43 transmits **BUNDLE** messages at line 42. This can only happen once due to the condition at line 37. Additionally, it may transmit **BUNDLE** messages also at line 50, upon the reception of a **BUNDLE**. However, this second **BUNDLE** transmission can happen at most once, due to the if-statement at line 47. This leads to at most 2 additional **comm** and **broadcast** invocations per correct process. Thus, as the number of correct processes is bounded by  $n$ , the two phases



incur in total at most  $4n$  invocations of **comm** and **broadcast** overall. Since each invocation communicates exactly  $n$  messages, the total message cost for correct processes when executing one instance of Coded MBRB (Algorithms 8-9) is upper bounded by  $4n^2$ . Note that the above analysis holds for correct processes also in the presence of Byzantine participants, including when  $p_s$  is dishonest.

We now bound the number of bits communicated by correct processes throughout a single instance of Coded MBRB (Algorithms 8-9). Let  $v$  be a specific value of size  $|v|$ . For every codeword  $\tilde{v}$  produced from  $v$ , we have  $|\tilde{v}| = O(|v|)$  since we use a code with a constant rate (Section 6.2.1). Thus, any specific value fragment  $\tilde{v}_i$  has length  $|\tilde{v}_i| = O(|v|/n)$ . Recall that the sizes of a signature share digest  $\sigma$ , a threshold signature  $\Sigma$ , a commitment  $C$ , and an inclusion proof  $\pi$  all have  $O(\lambda)$  bits (Section 6.2.2). In a signature share (pair)  $sig = (\sigma, i)$ , the identifier  $i$  of the signing process is included, which takes additional  $O(\log n)$  bits. However, since  $\lambda = \Omega(\log n)$ , the inclusion of this field does not affect asymptotic communication costs.

We now trace all the **comm** and **broadcast** instances in Algorithms 8-9 and analyze the number of bits communicated in each. The **SEND comm** (line 19) communicates  $n$  messages, where each message includes a fragment of  $v$  ( $O(|v|/n)$  bits) with its proof of inclusion ( $O(\lambda)$  bits), a commitment ( $O(\lambda)$  bits), and a signature share ( $O(\lambda)$  bits). Thus, this operation allows the sender to communicate at most  $O(|v| + n\lambda)$  bits. Each **FORWARD broadcast** at lines 25 and 36 sends  $n$  copies of a message containing a commitment ( $O(\lambda)$  bits), at most one value fragment with its proof of inclusion ( $O(|v|/n + \lambda)$  bits), and two signature shares ( $O(\lambda)$  bits). Hence, each one of lines 25 and 36 communicates a total of  $O(|v| + \lambda n)$  bits. The **BUNDLE** communication (lines 42 or 50) sends  $n$  messages, where each contains a commitment ( $O(\lambda)$  bits), at most two value fragments with their proof of inclusion ( $O(|v|/n + \lambda)$  bits), and one threshold signature ( $O(\lambda)$  bits). Hence, each line communicates at most  $O(|v| + n\lambda)$  bits. As analyzed above, the sending process ( $p_s$ , when correct) performs at most one **comm** of **SEND** messages, while each correct process performs at most two **broadcast** of **FORWARD** messages, and at most two **comm/broadcast** of **BUNDLE** messages. Thus, each process communicates at most  $O(|v| + n\lambda)$  bits. Overall, the total bit communication by correct processes during the execution of Algorithms 8-9 is  $O(n|v| + n^2\lambda)$ . As mentioned above, the analysis holds in the presence of Byzantine processes, even if  $p_s$  is dishonest.  $\square$

## 6.4 Discussion

This section addresses subsidiary aspects of our Coded MBRB solution, namely how to choose the reconstruction threshold  $k$  of the underlying erasure-correcting code (Section 6.4.1), how to make our solution multi-sender and multi-shot (Section 6.4.2), and whether using Bracha's BRB on hashes of the disseminated value could help to obtain the same communication cost under a message adversary (Section 6.4.3).

### 6.4.1 Selection of $k$

In the previous analysis, we set  $k$  to be a parameter that controls the number of fragments that allow decoding the ECC. To obtain the communication depicted in Theorem 7, we assumed  $k = \Omega(n)$ . Furthermore, this parameter affects the delivery power of the MBRB algorithm, as seen in Lemma 64, namely  $\ell_{MBRB} = c - d \left( \frac{1}{1 - \frac{k-1}{c-d}} \right)$ .

Let us assume that we wish to design an MBRB algorithm with a specified delivery power of  $\ell_{MBRB} = c - (1 + \varepsilon)d$ , for some  $\varepsilon > 0$ . Plugging in Lemma 64, we need the delivery power  $\ell_{MBRB}$  provided by Algorithms 8-9 to surpass  $c - (1 + \varepsilon)d$ , thus

$$c - (1 + \varepsilon)d \leq c - d \left( \frac{1}{1 - \frac{k-1}{c-d}} \right)$$

leading to  $k \leq \frac{\varepsilon}{1+\varepsilon}(c-d) + 1$ . That is, choosing any integer  $k \leq \frac{\varepsilon}{1+\varepsilon}(n-t-d) + 1$  satisfies the above. Recall that the blowup of the ECC is given by  $\bar{n}/\bar{k} \approx \frac{n}{k}$  (Section 6.3.2), which implies that for any value  $v$ , we have  $|\text{ECC}(v)| \approx \frac{n}{k}|v| = \frac{1+\varepsilon}{\varepsilon} \times \frac{n}{n-t-d}|v|$ .

Together with c-MBRB-Assumption, we conclude that the constraints on  $k$  that support delivery power of  $\ell_{MBRB} \geq n - t - (1 + \varepsilon)d$ , are

$$k \leq \min \left( n - t - 2d, \frac{\varepsilon}{1 + \varepsilon}(n - t - d) + 1 \right).$$

### 6.4.2 Supporting multiple instances and multiple senders

The above analysis fits the single-shot broadcast algorithm with a fixed sender. As mentioned above, a multi-shot multi-sender algorithm can be achieved by communicating the sender's identity and a sequence number along with any information communicated or processed during this algorithm. This added information uniquely identifies any piece of information with the respective instance. Additionally, signature shares, threshold signatures, commitments, and inclusion proofs should be performed on the value  $v$  augmented with the sender's identity and the sequence number. This will prevent Byzantine processes from using valid signature shares/threshold signatures from one instance in a different instance. As a result, an additive factor of  $O(\log n)$  bits has to be added to each communicated message, which yields additive communication of  $O(n^2 \log n)$  and has no effect on the asymptotic communication, as we explained in the proof of Theorem 7. Other changes, such as augmenting the value  $v$  with the sender's identity and sequence number do not affect the length of signature shares, threshold signatures, commitments, and inclusion proofs.

### 6.4.3 Using Bracha's BRB on hash values under a message adversary

Das, Xiang, and Ren [57] have proposed a communication optimal (up to the parameter  $\lambda$ ) BRB algorithm that does not use signatures and relies on Bracha instead to reliably broadcast a hash value of the initial sender's message. One might legitimately ask whether this approach could not be easily adapted to withstand a message adversary, possibly resulting in an MBRB algorithm exhibiting optimal communication complexity (up to the size of hashes  $\lambda$ ), optimal Byzantine resilience ( $n > 3t + 2d$ ), and optimal delivery power  $n - t - d$  (or at least some close-to-optimal delivery power  $\ell_{MBRB}$ , up to some factor  $\varepsilon$ ), while avoiding signatures altogether.

Unfortunately, under a message adversary, Bracha's BRB leads to a sub-optimal Byzantine resilience, and degraded delivery power  $\ell_{MBRB}$ . Indeed, as shown in the previous chapter (Chapter 5), Bracha can be used to implement an MBRB algorithm, but with a sub-optimal resilience bound ( $n > 3t + 2d + 2\sqrt{td}$ ) and a reduced delivery power  $\ell_{BRB} = \lceil n - t - (\frac{n-t}{n-3t-d})d \rceil$ . Disappointingly, these less-than-optimal properties would in turn be passed on to any MBRB algorithm using Bracha's BRB along the lines of Das, Xiang, and Ren's solution.<sup>40</sup> By contrast, the algorithm we propose is optimal in terms of communication cost (up to  $\lambda$ ) and Byzantine resilience, and close to optimal in terms of delivery power (up to some parameter  $\varepsilon$  that can be chosen arbitrarily small).

To provide a hint of why Bracha's BRB leads to degraded resilience and delivery power when confronted with a message adversary (MA), consider the classical ECHO phase of Bracha's BRB [32]. At least one correct process must receive strictly more than  $\frac{n+t}{2}$  ECHO messages to ensure the first READY message by a correct process can be emitted. To ensure Local-delivery, the threshold  $\frac{n+t}{2}$  must remain lower than the worst number of ECHO messages a correct process can expect to receive when the sender is correct. Without an MA this constraint leads to  $\frac{n+t}{2} < n - t$ , which is verified by assuming  $n > 3t$ . With an MA, the analysis is more complex. Applying a similar argument to that of the proof of MBRB-Global-delivery of Coded MBRB (see Section 6.3.3), one can show that in the worst case the adversary can ensure that no correct process receives more than  $\frac{(n-t-d)^2}{n-t}$  ECHO messages. Ensuring that at least one correct process reaches the Byzantine quorum threshold  $\frac{n+t}{2}$  therefore requires that we have:

$$\frac{n+t}{2} < \frac{(n-t-d)^2}{n-t}.$$

This leads to a quadratic inequality involving  $n$ ,  $t$  and  $d$ , which results in the following constraint on  $n$ :

40. Taking into account the initial dissemination of value  $v$  by the sender, which is also impacted by the message adversary, such an algorithm could in fact at most reach a delivery power of  $\max(0, (n-t-d) + \ell_{BRB} - (n-t)) = \max(0, \ell_{BRB} - d) = \max(0, \lceil n - t - (\frac{n-t}{n-3t-d} + 1)d \rceil)$ .

$$n > 2t + 2d + \sqrt{(d+t)^2 + d^2} \geq 3t + 3d.$$

In Chapter 5, we improve on this resilience bound by systematically optimizing the various retransmission and phase thresholds used by Bracha’s BRB algorithm, but the resulting solution still falls short of the optimal lower bound  $n > 3t + 2d$ , which the solution presented in this chapter provides.

## 6.5 Conclusion

This chapter introduced a Coded MBRB algorithm that significantly improves the communication cost of the original MBRB solution presented in Chapter 4. Namely, it achieves optimal communication (up to the size of the cryptographic structures  $\lambda$ ) while maintaining a high delivery power, *i.e.*, it ensures that messages are delivered by at least  $\ell_{MBRB} = n - t - (1 + \varepsilon)d$  correct processes, where  $\varepsilon > 0$  is a tunable parameter that can be made arbitrarily close to 0, albeit with a marginal increase in communication costs. The proposed solution is deterministic up to its use of cryptography (threshold signatures and vector commitments). Each correct process sends no more than  $4n$  messages and communicates at most  $O(|v| + n\lambda)$  bits, where  $|v|$  represents the length of the input value and  $\lambda$  is a security parameter. We note that the algorithm’s communication efficiency holds for sufficiently long messages and approaches the natural upper bound on delivery power  $n - t - d$ , which accounts for the message adversary’s (MA) ability to isolate a subset of correct processes. This work offers a practical solution for robust communication in asynchronous message-passing systems with malicious processes and message adversaries, while delivering a communication complexity that is comparable with that of the most recent existing MA-free BRB algorithms [16,57]. One intriguing question is whether it is possible to devise an (M)BRB algorithm that does not exhibit the  $\lambda$  parameter in its communication complexity or the  $\varepsilon$  parameter in its delivery power  $\ell_{MBRB}$ , for instance by leveraging randomization [1] or error-freedom [16] (*i.e.*, avoiding cryptography).



# CONCLUSION

---

Those are my principles, and if you don't like them... well, I have others!

---

*Groucho Marx*

This thesis has explored the realm of fault-tolerant distributed systems, focusing on the implementation of reliable broadcast in asynchronous environments prone to hybrid failures. This work has been motivated by the increasing prevalence of large-scale distributed systems and the need for robust communication primitives that can withstand both process and network failures. The main contributions of this dissertation span theoretical modeling, algorithm design, and performance optimization in the context of *Message-Adversary-tolerant Byzantine Reliable Broadcast (MBRB)*. In the following, we review the contributions of this thesis and their implications in the field of fault-tolerant distributed systems (Section 7.1), and we then deliberate upon the novel research issues raised by our findings (Section 7.2).

## 7.1 Summary of contributions

**A new computing model and distributed problem: MBRB** Our first contribution laid the groundwork for the entire thesis by introducing a novel computing model for designing distributed systems. This model defines an asynchronous message-passing network of  $n$  processes, where up to  $t$  processes may exhibit Byzantine behavior, and a message adversary (MA) can remove up to  $d$  copies of a message disseminated by a correct process. Moreover, the parameter  $c$  denotes the effective number of correct (*i.e.*, non-faulty) processes in the system. By allowing Byzantine processes to collude with the MA, we created a hybrid fault model that captures a wider range of failure scenarios than traditional models.

Building on this model, we introduced the MBRB abstraction, which generalizes the standard definition of Byzantine Reliable Broadcast (BRB) to accommodate hybrid failures. Particularly, MBRB accounts for the fact that, in the presence of an MA, it is impossible to guarantee that all correct processes of the system deliver every broadcast value. To consider this array of adversarial behaviors, MBRB introduces a new parameter  $\ell_{MBRB}$ , denoting the minimum

number of correct processes that deliver the value broadcast during each MBRB execution. One can observe that the best possible value for  $\ell_{MBRB}$  is  $c - d$ , as the MA can always completely input-disconnect at most  $d$  correct processes in the system. We can also remark that, when  $\ell_{MBRB} = c$  (which can only be achieved when  $d = 0$ , *i.e.*, when there is no MA), MBRB boils down to the traditional definition of BRB.

A key theoretical result of our work is the optimality theorem for asynchronous MBRB, which states that implementation is possible if and only if  $n > 3t + 2d$ . This condition establishes clear boundaries for what is achievable in terms of fault tolerance in this family of distributed systems, and serves as a guideline for designing robust distributed algorithms. Besides laying essential theoretical foundations, this result also provides a practical benchmark against which future solutions can be measured.

**A simple signature-based MBRB implementation** Our second contribution presented a concrete implementation of MBRB using cryptographic signatures. This algorithm achieves optimal Byzantine resilience and number of communication rounds when there is no message adversary. The algorithm’s reliance on the condition  $n > 3t + 2d$  aligns with our theoretical optimality theorem, demonstrating a tight connection between theory and practice. Furthermore, to show the algorithm’s practicality, we have comprehensively studied its time, message, and communication costs. This supports the idea that our solution can efficiently operate under the challenging conditions of real-world networks.

**$k2\ell$ -cast: a modular approach to construct signature-free MBRB algorithms** The third contribution of this thesis addressed the challenge of implementing MBRB in a cryptography-free context. We introduced a novel communication primitive called  $k2\ell$ -cast, which enables more efficient quorum engineering. This abstraction makes it possible to reconstruct existing signature-free reliable broadcast algorithms (such as Bracha’s [32] and Imbs-Raynal’s [87]), to make them tolerant against Byzantine failures and message losses, therefore yielding working MBRB implementations that do not rely on digital signatures. Interestingly, when there is no message adversary, the reconstructed MBRB algorithms are also more efficient than the original counterparts, as they use smaller quorums and thus need fewer messages to progress. However, the trade-off for these signature-free MBRB implementations is that they are sub-optimal in terms of delivery power and resilience against Byzantine faults and message adversaries.

Our work on  $k2\ell$ -cast opens up new possibilities for designing hybrid-fault-tolerant systems without relying on cryptographic primitives. This is particularly valuable in scenarios where computational resources are limited or where the use of cryptography is undesirable due to regulatory or performance constraints. Moreover, the potential applications of  $k2\ell$ -cast extend beyond reliable broadcast only, with possible benefits for self-stabilizing and self-healing distributed systems.

**A coding-based MBRB implementation with near-optimal communication** Our final contribution focused on optimizing the communication cost of MBRB. We introduced the Coded MBRB algorithm that achieves near-optimal communication complexity while preserving an optimal resilience of  $n > 3t + 2d$ . More specifically, when the sender is correct, Coded MBRB features a communication cost of  $O(|v| + n\lambda)$  bits sent per correct process, where  $|v|$  is the size of the broadcast value, and  $\lambda$  is the security parameter of the underlying cryptographic primitives. This significantly improves upon the initial signature-based MBRB algorithm that we presented previously, and which possesses a communication complexity of  $O(n|v| + n^2\lambda)$  bits sent per correct process. Importantly, Coded MBRB nearly reaches the theoretical lower bound of  $\Omega(|v| + n)$  bits communication by each correct process, as our solution exhibits only one additional factor  $\lambda$  in its asymptote. In addition, Coded MBRB maintains a high delivery power of  $\ell_{MBRB} = c - (1 + \varepsilon)d$ , where  $\varepsilon$  is a tunable parameter that can be made arbitrarily close to 0.

Coded MBRB offers these high-performance guarantees by leveraging several tools and concepts inspired by the coding and cryptography literature, namely error-correcting codes (ECC), threshold signatures, and vector commitments. This combination of techniques yields an MBRB algorithm that strikes a balance between theoretical guarantees and practical efficiency, making it suitable for real-world deployment. This algorithm represents a significant advancement in Byzantine Reliable Broadcast under a Message Adversary (MBRB), demonstrating that it is possible to achieve near-optimal communication complexity without sacrificing resilience. This contribution is especially significant for distributed systems where bandwidth is a precious resource.

## 7.2 Future directions

While we hope our work will prove to be of interest to the wider research community, it also opens up several avenues for future research. The following section discusses several promising axes of research that warrant additional scrutiny.

**Enhancing the efficiency and resilience of our algorithms** While we have shown the good theoretical guarantees of our proposed algorithms regarding performance and fault-tolerance, there is still potential to optimize them further.

For instance, the  $k2\ell$ -cast primitive of Chapter 5 demonstrates that signature-free MBRB is feasible, but the resulting implementations exhibit sub-optimal resilience and delivery power. Investigating whether these metrics could be optimized while staying in a cryptography-free context demands further exploration.

Moreover, the signature-based  $k2\ell$ -cast implementation presented in Section 5.4 suffers from a prohibitive message cost of  $O(n^2)$  messages sent per correct process  $p_i$  during an execution, as  $p_i$  broadcasts its set of known signatures every time it receives a new signature. This results in an



overall cubic message complexity of  $O(n^3)$ , which also inevitably affects the bit-communication complexity of the algorithm. Decreasing this message complexity could be done by integrating a more refined mechanism for forwarding signatures in the algorithm.

Another intriguing question raised by our work is whether it is possible to devise an MBRB algorithm that offers *optimal* communication cost of  $\kappa = O(n|v| + n^2)$  (unlike Coded MBRB, which is only *near-optimal* as it has an additional  $\lambda$  factor in its asymptote) while maintaining *optimal* delivery power of  $\ell_{MBRB} = c - d$  (i.e., without the  $\varepsilon$  parameter of the Coded MBRB algorithm). Such a solution could possibly be attained by leveraging randomization or error-freedom techniques [16].

**Experimental evaluations and practical applications** While our algorithms are designed with practicality in mind, future work should focus on implementing these solutions in real-world systems and conducting comprehensive performance evaluations under various network conditions. In particular, we have started prototyping some of our algorithms in the Rust programming language and performing preliminary latency measurements on them in an experimental setting.

To further illustrate the usability of our solutions, we can also delve into their potential applications in other distributed algorithms and systems. Given the relevance of reliable broadcast to decentralized payment systems [21,23,55,79], exploring how our MBRB solutions could be applied to improve the scalability and resilience of cryptocurrencies and blockchain platforms is a promising direction.

Additionally, our conjecture about the potential benefits of  $k2\ell$ -cast for self-stabilizing and self-healing systems warrants further investigation. Developing concrete algorithms in this domain could lead to more robust and adaptive distributed systems.

**Developing the theory of hybrid fault models** Our current computing model considers an asynchronous message-passing system prone to Byzantine faults and message losses caused by a Message Adversary (see Section 3.1). Future work could focus on exploring the computability limits of this model and how to enrich it with additional types of failures, synchrony assumptions, and interactions between processes to increase its expressiveness.

For instance, message corruption or spurious message creation could also be considered in an extended version of our model. From an initial examination, it seems that an asynchronous model where the Message Adversary can arbitrarily corrupt some messages broadcast by correct processes might be overly restrictive, as the asynchrony can delay non-altered messages, so that altered and Byzantine messages are received first by correct processes. As any correct process may appear Byzantine to other correct processes, existing algorithms would require significant changes to thwart these message corruptions caused by the adversary. On the process fault side, if we want our solutions to be applicable to open large-scale peer-to-peer systems, we

have to take into account Sybil attacks [66], where the attacker creates many fake identities to surpass the prescribed maximum number of Byzantine processes tolerated in the system. Common strategies to attain Sybil-resistance rely on a scarce resource that the adversary cannot control in too large quantities, such as computing power (in the case of *Proof-of-Work* [106]) or cryptocurrency (in the case of *Proof-of-Stake* [74]).

Our current solutions are primarily asynchronous and deterministic. Investigating whether (partial) synchrony or randomization could help to circumvent our model’s impossibilities or improve our algorithms’ efficiency and resilience is an interesting lead.

While this thesis has concentrated on distributed *message-passing* systems, it is worthwhile to consider whether our findings can be extended to the other principal paradigm of distributed computing: *shared memory*. It is a known fact that a distributed memory can be emulated on top of a message-passing system [20]. In particular, reliable broadcast enables the construction of a memory where each process of the system has its dedicated *single-writer multi-reader* register. To do that, all processes maintain their local view of the entire memory, in which they can immediately read the value of any register. To write a value in its own register, a process  $p_i$  simply has to reliably broadcast this value to the system, and upon its delivery, every receiving process directly updates  $p_i$ ’s register in its local view.

This begs a natural question: Can MBRB, which generalizes reliable broadcast to hybrid failures, implement a distributed memory tolerant to both Byzantine faults and a Message Adversary? As the Message Adversary can hamper the delivery of messages, it appears that the classical guarantees of shared registers (*e.g.*, every written value can eventually be read) have to be relaxed. The thorough study of the consistency levels and possible constructions that can be achieved in this hybrid-fault-tolerant distributed memory constitutes a promising line of work.

### 7.3 Final words

In conclusion, this thesis has made several strides in the field of fault-tolerant distributed systems by introducing new models, abstractions, and algorithms for reliable broadcast in the presence of hybrid failures. Our work bridges the gap between theoretical bounds and practical implementations, thus aiming to provide a solid foundation for building more resilient and efficient distributed systems. This increased resilience is essential for applications in critical infrastructure, financial systems, and other domains where robustness is paramount. Indeed, as the scale and complexity of distributed systems continue to grow, the need for communication solutions that can withstand diverse failure scenarios becomes increasingly crucial. We believe that the concepts and techniques presented in this thesis can play a role in shaping the next generation of fault-tolerant distributed systems, enabling more reliable, scalable, and secure applications.



# CIRCUMVENTING THE FLP IMPOSSIBILITY

---

In the following, we review some of the most notable solutions to circumvent the FLP impossibility theorem.

Some of these solutions involve adding synchrony assumptions. In a synchronous setting, consensus can be implemented, not only with one crash, but with any number of Byzantine faults [93]<sup>41</sup>, demonstrating the sheer difference in computability power between synchrony and asynchrony. Resilient consensus can also be implemented under partial synchrony, as illustrated by the Paxos [96] and PBFT [38] algorithms. Failure detectors also belong to this category: processes have access to a (possibly imperfect) oracle that provides information on other processes' failures and allows the system to solve consensus [42]. Failure detectors thus bridge the gap between full asynchrony, where crashes can never be detected, and full synchrony, where they can always be detected.

Randomization is another technique to implement consensus in asynchronous and fault-prone systems [25,32,104]. For instance, the algorithms presented in [25,32] rely on a random coin tossed by all system processes at each round, and there is a non-null chance that all processes get the same coin value during one round. These solutions weaken consensus liveness: It is possible to have an execution that never terminates, but the probability becomes asymptotically null as the number of rounds increases.

Another approach involves restricting the occurrence of failures in the system. For instance, it is not widely known that the original FLP paper also presented an asynchronous consensus algorithm that tolerates less than half of faulty processes, only if all these faults happen at the start of the execution [70]. Another example comes from the shared-memory world: mutual exclusion (which requires the same computability power as consensus) can tolerate process failures as long as they do not happen inside the critical section [94].

Other solutions circumvent the impossibility either by restricting the set of possible inputs (proposals) or expanding the set of possible outputs (decisions). An example of the former

---

41. Two algorithms were presented in [93]: one that tolerates any number of Byzantines (as long as this number is known by all processes) but requires digital signatures (the “signed messages” model), and the other that tolerates less than one-third of Byzantines in the system but is signature-free (the “oral messages” model). This shows that signatures also significantly enrich the underlying computing model.

---

category is given in [105], where the authors fully characterize the set of favorable input vectors (*i.e.*, proposals for each process) for which consensus can be solved. Conversely, an example of the latter category is approximate agreement [64], which relaxes the guarantee of strong agreement of consensus, and permits a small divergence among processes on their decision values. The smaller we want this divergence to be, the longer it takes for the approximate agreement algorithm to terminate under asynchrony and process faults.

Another example from this last category is  $k$ -set agreement [45], which is a natural generalization of consensus: processes can decide at most  $k$  different values out of the proposed ones (so consensus is 1-set agreement). Later, it was shown that the FLP impossibility can also be naturally generalized: asynchronous  $k$ -set agreement cannot be implemented in the presence of  $k$  faults [5,18,27,30,81,125]. Intuitively, this impossibility can be understood in 2 steps: (1) in an asynchronous wait-free system of  $n$  processes (*i.e.*, where at most  $n-1$  processes may fail),  $(n-1)$ -set agreement is impossible, as, due to asynchrony, each process must make a decision “in isolation” (*i.e.*, without communicating), which can trivially lead to safety violations (more than  $n-1$  different values decided); and (2) the *BG simulation* technique generalizes this result from wait-free systems to non-wait-free systems, by showing that adding more non-faulty processes to a system does not increase its computability power [30,118,124].

# PROOF OF THE SIGNATURE-FREE $k2\ell$ -CAST IMPLEMENTATION (ALGORITHM 4)

---

## B.1 Safety of Algorithm 4

**Lemma 37.** *If a correct process  $p_i$   $k2\ell$ -delivers  $(v, id)$ , then at least  $(q_f - n + c)$  correct processes have broadcast  $\text{ENDORSE}(v, id)$  at line 4.*

**Proof.** If  $p_i$   $k2\ell$ -delivers  $(v, id)$  at line 11, then it received  $q_d$  copies of  $\text{ENDORSE}(v, id)$  (because of the predicate at line 10). The effective number of Byzantine processes in the system is  $n - c$ , such that  $0 \leq n - c \leq t$ . Therefore,  $p_i$  must have received at least  $q_d - n + c$  (which is strictly positive because  $q_d \geq q_f > t \geq n - c$  by sf- $k2\ell$ -Assumption 1) messages  $\text{ENDORSE}(v, id)$  that correct processes broadcast, either during a  $k2\ell\_cast(v, id)$  invocation at line 4, or during a forwarding step at line 8. There are two cases.

- If no correct process has forwarded  $\text{ENDORSE}(v, id)$  at line 8, then at least  $q_d - n + c \geq q_f - n + c$  (as  $q_d \geq q_f$  by sf- $k2\ell$ -Assumption 1) correct processes have broadcast  $\text{ENDORSE}(v, id)$  at line 4.
- If at least one correct process forwarded  $\text{ENDORSE}(v, id)$ , then let us consider  $p_j$ , the first correct process that forwards  $\text{ENDORSE}(v, id)$ . Because of the predicate at line 7,  $p_j$  must have received at least  $q_f$  distinct copies of the  $\text{ENDORSE}(v, id)$  message, out of which at most  $n - c$  have been broadcast by Byzantine processes, and at least  $q_f - n + c$  (which is strictly positive because  $q_f > t \geq n - c$  by sf- $k2\ell$ -Assumption 1) have been sent by correct processes. Moreover, as  $p_j$  is the first correct process that forwards  $\text{ENDORSE}(v, id)$ , all of the  $q_f - n + c$   $\text{ENDORSE}$  messages it receives from correct processes must have been sent at line 4.  $\square$

**Lemma 38** (*k2ℓ-Validity*). *If a correct process  $p_i$  k2ℓ-delivers a value  $v$  with identity  $id$ , then at least  $k' = q_f - n + c$  correct processes have k2ℓ-cast  $v$  with  $id$ .*

**Proof.** The condition at line 3 implies that the correct processes that broadcast  $\text{ENDORSE}(v, id)$  at line 4 constitute a subset of those that k2ℓ-cast  $(v, id)$ . Thus, by Lemma 37, their number is at least  $k' = q_f - n + c$ .  $\square$

**Lemma 39** (*k2ℓ-No-duplication*). *A correct process  $p_i$  k2ℓ-delivers a value  $v$  with identity  $id$  at most once.*

**Proof.** This property derives trivially from the predicate at line 10.  $\square$

**Lemma 40** (*k2ℓ-Conditional-no-duplicity*). *If the Boolean  $\delta = ((q_f > \frac{n+t}{2}) \vee (\text{single} \wedge q_d > \frac{n+t}{2}))$  is **true**, then no two different correct processes k2ℓ-deliver different values with the same identity  $id$ .*

**Proof.** Let  $p_i$  and  $p_j$  be two correct processes that respectively k2ℓ-deliver  $(v, id)$  and  $(v', id)$ . We want to prove that, if the predicate  $((q_f > \frac{n+t}{2}) \vee (\text{single} \wedge q_d > \frac{n+t}{2}))$  is satisfied, then  $v = v'$ . There are two cases.

- Case  $(q_f > \frac{n+t}{2})$ .

We denote by  $A$  and  $B$  the sets of correct processes that have respectively broadcast  $\text{ENDORSE}(v, id)$  and  $\text{ENDORSE}(v', id)$  at line 4. By Lemma 37, we know that  $|A| \geq q_f - n + c > \frac{n+t}{2} - n + c$  and  $|B| \geq q_f - n + c > \frac{n+t}{2} - n + c$ . As  $A$  and  $B$  contain only correct processes, we have  $|A \cap B| > 2(\frac{n+t}{2} - n + c) - c = t - n + c \geq t - t = 0$ . Hence, at least one correct process  $p_x$  has broadcast both  $\text{ENDORSE}(v, id)$  and  $\text{ENDORSE}(v', id)$  at line 4. But because of the predicate at line 3,  $p_x$  broadcasts at most one message  $\text{ENDORSE}(\star, id)$  at line 4. We conclude that  $v$  is necessarily equal to  $v'$ .

- Case  $(\text{single} \wedge q_d > \frac{n+t}{2})$ .

Thanks to the predicate at line 10, we can assert that  $p_i$  and  $p_j$  must have respectively received at least  $q_d$  distinct copies of  $\text{ENDORSE}(v, id)$  and  $\text{ENDORSE}(v', id)$ , from two sets of processes, that we respectively denote  $A$  and  $B$ , such that  $|A| \geq q_d > \frac{n+t}{2}$  and  $|B| \geq q_d > \frac{n+t}{2}$ . We have  $|A \cap B| > 2\frac{n+t}{2} - n = t$ . Hence, at least one correct process  $p_x$  has broadcast both  $\text{ENDORSE}(v, id)$  and  $\text{ENDORSE}(v', id)$ . But because of the predicates at lines 3 and 7,

and as  $single = \mathbf{true}$ ,  $p_x$  broadcasts at most one message  $\text{ENDORSE}(\star, id)$ , either during a  $k2\ell\text{-cast}(v, id)$  invocation at line 4 or during a forwarding step at line 8. We conclude that  $v$  is necessarily equal to  $v'$ .  $\square$

## B.2 Liveness of Algorithm 4

**Lemma 18.**  $\ell_e \times (k_U + k_F - q_d + 1) \geq (k_U + k_F)(c - d - q_d + q_f) - c(q_f - 1) - k_{NB}(q_d - q_f)$ .

**Proof.** Combining (9), (10), (11) and (12) yields:

$$\begin{aligned} (k_U + k_F)\ell_e + (q_d - 1)(k_{NF} + k_{NB} + k_F - \ell_e) + \\ (q_f - 1)(c - k_{NF} - k_{NB} - k_F) &\geq (k_U + k_F)(c - d), \\ \ell_e \times (k_U + k_F - q_d + 1) &\geq (k_U + k_F)(c - d) - (q_d - 1)(k_{NF} + k_{NB} + k_F) - \\ &\quad (q_f - 1)(c - k_{NF} - k_{NB} - k_F), \\ &\geq (k_U + k_F)(c - d) - (q_d - q_f)(k_{NF} + k_{NB} + k_F) - c(q_f - 1). \end{aligned}$$

Using sf- $k2\ell$ -Assumption 1, we have  $q_d - q_f \geq 0$ . By definition, we also have  $k_{NF} \leq k_U$ , which yields:

$$\begin{aligned} \ell_e \times (k_U + k_F - q_d + 1) &\geq (k_U + k_F)(c - d) - (q_d - q_f)(k_U + k_F + k_{NB}) - c(q_f - 1), \\ &\geq (k_U + k_F)(c - d - q_d + q_f) - c(q_f - 1) - k_{NB}(q_d - q_f). \quad \square \end{aligned}$$

**Lemma 19.** *If no correct process  $k2\ell$ -casts  $(v', id)$  with  $v' \neq v$ , then no correct process forwards  $\text{ENDORSE}(v', id)$  at line 6 (and then  $k_{NB} = 0$ ).*

**Proof.** Assume there is a correct process that broadcasts  $\text{ENDORSE}(v', id)$  at line 8} with  $v' \neq v$ . Let us consider the first such process  $p_i$ . To execute line 8,  $p_i$  must first receive  $q_f$  messages  $\text{ENDORSE}(v', id)$  from distinct processes. Since  $q_f > t$  (sf- $k2\ell$ -Assumption 1), at least one of these processes,  $p_j$ , is correct. Since  $p_i$  is the first correct process to forward  $\text{ENDORSE}(v', id)$  at line 8, the  $\text{ENDORSE}(v', id)$  message of  $p_j$  must come from line 4}, and  $p_j$  must have  $k2\ell\text{-cast}(v', id)$ . We have assumed that no correct process  $k2\ell\text{-cast } v' \neq v$ , therefore  $v' = v$ . Contradiction.

We conclude that, under these assumptions, no correct process broadcasts  $\text{ENDORSE}(v', id)$  with  $v' \neq v$ , be it at line 4 (by assumption) or at line 8 (shown by this proof). As a result,  $k_{NB} = 0$ .  $\square$



**Lemma 20** (*k2l-Local-delivery*). *If at least  $k = \left\lfloor \frac{c(q_f-1)}{c-d-q_d+q_f} \right\rfloor + 1$  correct processes k2l-cast a value  $v$  with identity  $id$  and no correct process k2l-casts any value  $v'$  with identity  $id$  such that  $v \neq v'$ , then at least one correct process  $p_i$  k2l-delivers  $v$  with identity  $id$ .*

**Proof.** Let us assume that no correct process k2l-casts  $(v', id)$  with  $v' \neq v$ . No correct process therefore broadcasts  $\text{ENDORSE}(v', id)$  with  $v' \neq v$  at line 4. Lemma 19 also applies and no correct process forwards  $\text{ENDORSE}(v', id)$  with  $v' \neq v$  at line 8 either, so  $k_{NB} = 0$ . Because no correct process broadcasts  $\text{ENDORSE}(v', id)$  with  $v' \neq v$  whether at lines 4 or 8, a correct process receives at most  $t$  messages  $\text{ENDORSE}(v', id)$  (all coming from Byzantine processes). As by sf-k2l-Assumption 1,  $t < q_d$ , no correct process k2l-delivers  $(v', id)$  with  $v' \neq v$  at line 11.

We now prove the contraposition of the Lemma. Let us assume no correct process k2l-delivers  $(v, id)$ . Since, by our earlier observations, no correct process k2l-delivers  $(v', id)$  with  $v' \neq v$  either, the condition at line 10 implies that no correct process ever receives at least  $q_d$   $\text{ENDORSE}(v, id)$ , and therefore  $\ell_e = 0$ . By Lemma 18 we have  $c(q_f - 1) \geq (k_U + k_F)(c - d - q_d + q_f)$ . sf-k2l-Assumption 1 implies that  $c - d - q_d \geq 0 \iff c - d - q_d + q_f > 0$  (as  $q_f \geq t + 1 \geq 1$ ), leading to  $k_U + k_F \leq \frac{c(q_f-1)}{c-d-q_d+q_f}$ . Because of the condition at line 3, a correct process  $p_j$  that has k2l-cast  $(m, id)$  but has not broadcast  $\text{ENDORSE}(v, id)$  at line 4 has necessarily broadcast  $\text{ENDORSE}(v, id)$  at line 8. We therefore have  $k_I \leq k_U + k_F$ , which gives  $k_I \leq \frac{c(q_f-1)}{c-d-q_d+q_f}$ . By contraposition, if  $k_I > \frac{c(q_f-1)}{c-d-q_d+q_f}$ , then at least one correct process must k2l-deliver  $(v, id)$ . Hence, we have  $k = \left\lfloor \frac{c(q_f-1)}{c-d-q_d+q_f} \right\rfloor + 1$ .  $\square$

**Lemma 21.**  $(\text{single} = \text{false}) \implies (k_{NB} = 0)$ .

**Proof.** Let us consider a correct process  $p_i \in A \cup B$ . If we assume  $p_i \notin F$ ,  $p_i$  never executes line 8 by definition. Because  $p_i \in A \cup B$ ,  $p_i$  has received at least  $q_f$  messages  $\text{ENDORSE}(v, id)$ , and therefore did not fulfill the condition at line 7 when it received its  $q_f^{\text{th}}$  message  $\text{ENDORSE}(v, id)$ . As  $\text{single} = \text{false}$  by Lemma assumption, to falsify this condition,  $p_i$  must have had already broadcast  $\text{ENDORSE}(v, id)$  when this happened. Because  $p_i$  never executes line 8, this implies that  $p_i$  broadcasts  $\text{ENDORSE}(v, id)$  at line 4, and therefore  $p_i \in NF$ . This reasoning proves that  $A \cup B \setminus F \subseteq NF$ . As the sets  $F$ ,  $NF$  and  $NB$  partition  $A \cup B$ , this shows that  $NB = \emptyset$ , and  $k_{NB} = |\emptyset| = 0$ .  $\square$

**Lemma 22.** *If at least one correct process  $k2\ell$ -delivers  $(v, id)$  and  $x = k_U + k_F$  (the number of correct processes that broadcast  $\text{ENDORSE}(v, id)$  at line 4 or 8), then  $x \geq q_d - t$  and  $x^2 - x(c - d + q_f - 1 - k_{NB}) \geq -(c - k_{NB})(q_f - 1)$ .*

**Proof.** Let us write  $w_A^b$  the total number of  $\text{ENDORSE}(v, id)$  messages from Byzantine processes received by the processes of  $A$ , and  $w_A = w_A^c + w_A^b$  the total of number  $\text{ENDORSE}(v, id)$  messages received by the processes of  $A$ , whether these  $\text{ENDORSE}$  messages originated from correct or Byzantine senders. By definition,  $w_A^b \leq t\ell_e$  and  $w_A \geq q_d\ell_e$ . By combining these two inequalities with (9) on  $w_A^c$  we obtain:

$$\begin{aligned} q_d\ell_e \leq w_A &= w_A^c + w_A^b \leq (k_U + k_F)\ell_e + t\ell_e = (t + k_U + k_F)\ell_e, \\ q_d &\leq t + k_U + k_F, \quad (\text{as } \ell_e > 0) \\ q_d - t &\leq k_U + k_F = x. \end{aligned} \tag{15}$$

This proves the first inequality of the lemma. The processes in  $A \cup B$  each receive at most  $k_U + k_F$  distinct  $\text{ENDORSE}(v, id)$  messages from correct processes, so we have  $w_A^c + w_B^c \leq (k_{NF} + k_F + k_{NB})(k_U + k_F)$ . Combined with the inequalities (11) on  $w_C^c$  and (12) on  $w_A^c + w_B^c + w_C^c$  that remain valid in this case, we now have:

$$\begin{aligned} (k_{NF} + k_F + k_{NB})(k_U + k_F) + (q_f - 1)(c - k_{NF} - k_{NB} - k_F) &\geq (k_U + k_F)(c - d), \\ (k_{NF} + k_F + k_{NB})(k_U + k_F - q_f + 1) &\geq (k_U + k_F)(c - d) - c(q_f - 1). \end{aligned} \tag{16}$$

Let us determine the sign of  $(k_U + k_F - q_f + 1)$ . We derive from (15):

$$\begin{aligned} k_U + k_F - q_f + 1 &\geq q_d - t - q_f + 1 \\ &\geq 1 > 0. \quad (\text{as } q_d - q_f \geq t \text{ by sf-}k2\ell\text{-Assumption 1}) \end{aligned}$$

As  $(k_U + k_F - q_f + 1)$  is positive and we have  $k_U \geq k_{NF}$  by definition, we can transform (16) into:

$$\begin{aligned} (k_U + k_F + k_{NB})(k_U + k_F - q_f + 1) &\geq (k_U + k_F)(c - tm) - c(q_f - 1), \\ (x + k_{NB})(x - q_f + 1) &\geq x(c - d) - c(q_f - 1), \quad (\text{as } x = k_U + k_F) \\ x^2 - x(c - d + q_f - 1 - k_{NB}) &\geq -(c - k_{NB})(q_f - 1). \quad \square \end{aligned}$$

**Lemma 23.** *If  $k_{NB} = 0$ , and at least one correct process  $k2\ell$ -delivers  $(v, id)$ , then  $k_U + k_F \geq q_d$ .*

**Proof.** By Lemma 22 we have:

$$x^2 - x(c - d + q_f - 1 - k_{NB}) \geq -(c - k_{NB})(q_f - 1). \quad (17)$$

As (17) holds for all, values of  $c \in [n - t, n]$ , we can in particular consider  $c = n - t$ . Moreover, as by hypothesis,  $k_{NB} = 0$ , we have.

$$\begin{aligned} x^2 - x(n - t - d + q_f - 1) + (q_f - 1)(n - t) &\geq 0, \\ x^2 - \alpha x + (q_f - 1)(n - t) &\geq 0. \end{aligned} \quad (\text{by definition of } \alpha) \quad (18)$$

Let us first observe that the discriminant of the second-degree polynomial in (18) is nonnegative, *i.e.*,  $\alpha^2 - 4(q_f - 1)(n - t) \geq 0$  by sf- $k2\ell$ -Assumption 2. This allows us to compute the two real-valued roots as follows:

$$r_0 = \frac{\alpha}{2} - \frac{\sqrt{\alpha^2 - 4(q_f - 1)(n - t)}}{2} \quad \text{and} \quad r_1 = \frac{\alpha}{2} + \frac{\sqrt{\alpha^2 - 4(q_f - 1)(n - t)}}{2}.$$

Thus (18) is satisfied if and only if  $x \leq r_0 \vee x \geq r_1$ .

- Let us prove  $r_0 \leq q_d - 1 - t$ . We need to show that:

$$\begin{aligned} \frac{\alpha}{2} - \frac{\sqrt{\alpha^2 - 4(q_f - 1)(n - t)}}{2} &\leq q_d - 1 - t \\ \frac{\alpha}{2} - (q_d - 1) + t &\leq \frac{\sqrt{\alpha^2 - 4(q_f - 1)(n - t)}}{2} \\ \frac{\sqrt{\alpha^2 - 4(q_f - 1)(n - t)}}{2} &\geq \frac{\alpha}{2} - (q_d - 1) + t \\ \sqrt{\alpha^2 - 4(q_f - 1)(n - t)} &\geq \alpha - 2(q_d - 1) + 2t. \end{aligned}$$

The inequality is trivially satisfied if  $\alpha - 2(q_d - 1) + 2t < 0$ . For all other cases, we need to verify that:

$$\begin{aligned} \alpha^2 - 4(q_f - 1)(n - t) &\geq (\alpha - 2(q_d - 1) + 2t)^2, \\ \alpha^2 - 4(q_f - 1)(n - t) &\geq \alpha^2 + 4(q_d - 1)^2 + 4t^2 - 4\alpha(q_d - 1) + 4\alpha t - 8t(q_d - 1), \end{aligned}$$

$$\begin{aligned}
 -4(q_f - 1)(n - t) &\geq 4(q_d - 1)^2 + 4t^2 - 4\alpha(q_d - 1) + 4\alpha t - 8t(q_d - 1), \\
 -(q_f - 1)(n - t) &\geq (q_d - 1)^2 + t^2 - \alpha(q_d - 1) + \alpha t - 2t(q_d - 1), \\
 -(q_f - 1)(n - t) &\geq (q_d - 1 - t)^2 - \alpha(q_d - 1 - t),
 \end{aligned}$$

and thus  $\alpha(q_d - 1 - t) - (q_f - 1)(n - t) - (q_d - 1 - t)^2 \geq 0$ , which is true by sf- $k2\ell$ -Assumption 4.

- Let us prove  $r_1 > q_d - 1$ . We want to show that:

$$\frac{\alpha}{2} + \frac{\sqrt{\alpha^2 - 4(q_f - 1)(n - t)}}{2} > q_d - 1.$$

Let us rewrite the inequality as follows:

$$\begin{aligned}
 \alpha + \sqrt{\alpha^2 - 4(q_f - 1)(n - t)} &> 2(q_d - 1) \\
 \sqrt{\alpha^2 - 4(q_f - 1)(n - t)} &> 2(q_d - 1) - \alpha.
 \end{aligned}$$

The inequality is trivially satisfied if  $2(q_d - 1) - \alpha < 0$ . For all other cases, we can take the squares as follows:

$$\begin{aligned}
 \alpha^2 - 4(q_f - 1)(n - t) &> (2(q_d - 1) - \alpha)^2, \\
 \alpha^{\{2\}} - 4(q_f - 1)(n - t) &> 4(q_d - 1)^2 + \alpha^2 - 4\alpha(q_d - 1), \\
 -4(q_f - 1)(n - t) &> 4(q_d - 1)^2 - 4\alpha(q_d - 1), \\
 4\alpha(q_d - 1) - 4(q_f - 1)(n - t) - 4(q_d - 1)^2 &> 0, \\
 \alpha(q_d - 1) - (q_f - 1)(n - t) - (q_d - 1)^2 &> 0,
 \end{aligned}$$

which is true by sf- $k2\ell$ -Assumption 3.

We now know that  $r_0 \leq q_d - 1 - t$  and that  $r_1 > q_d - 1$ . In addition, as  $x \leq r_0 \vee x \geq r_1$ , we have  $x \leq q_d - t - 1 \vee x > q_d - 1$ . But Lemma 22 states that  $x \geq q_d - t$ , which is incompatible with  $x \leq q_d - t - 1$ . So we are left with  $x > q_d - 1$ , which implies, as  $q_d$  and  $x$  are integers that  $x \geq q_d$ , thus proving the lemma for  $c = n - t$ .

Let us now consider the set  $E_0$  of all executions in which  $t$  processes are Byzantine, and therefore  $c = n - t$ , and a set  $E_c$  of executions in which there are fewer Byzantine processes, and thus  $c > n - t$  correct processes. We show that  $E_c \subseteq E_0$  in that a Byzantine process can always simulate the behavior of a correct process. In particular, if the simulated correct process is not subject to the message adversary, the simulating Byzantine process simply operates like a

correct process. If, on the other hand, the simulated correct process misses some messages as a result of the message adversary, the Byzantine process can also simulate missing such messages. As a result, the executions that can happen when  $c > n - t$  can also happen when  $c = n - t$ . Thus, our result proven for  $c = n - t$  can be extended to all possible values of  $c$ .  $\square$

**Lemma 24.** *If  $k_{NB} = 0$  and  $k_U + k_F \geq q_d$ , then at least  $\left\lceil c \left(1 - \frac{d}{c - q_d + 1}\right) \right\rceil$  correct processes  $k_{2\ell}$ -deliver some value with identity  $id$  (not necessarily  $v$ ).*

**Proof.** As  $k_{NB} = 0$  and  $k_U + k_F \geq q_d$ , we can rewrite the inequality of Lemma 18 into:

$$\ell_e \times (k_U + k_F - q_d + 1) \geq (k_U + k_F)(c - d - q_d + q_f) - c(q_f - 1).$$

From  $k_U + k_F \geq q_d$  we derive  $k_U + k_F - q_d + 1 > 0$ , and we transform the above inequality into:

$$\ell_e \geq \frac{(k_U + k_F)(c - d - q_d + q_f) - c(q_f - 1)}{k_U + k_F - q_d + 1}.$$

Let us now focus on the case in which  $c = n - t$ , we obtain:

$$\ell_e \geq \frac{(k_U + k_F)(n - t - d - q_d + q_f) - (n - t)(q_f - 1)}{k_U + k_F - q_d + 1}.$$

The right side of the inequality is of the form:

$$\ell_e \geq \frac{\varphi x - \beta}{x - \gamma} = \varphi + \frac{\varphi\gamma - \beta}{x - \gamma} \tag{19}$$

with:

$$\begin{aligned} x &= k_U + k_F, \\ \gamma &= q_d - 1, \\ \alpha &= n - t - d + q_f - 1, \\ \varphi &= n - t - d - q_d + q_f, \\ \beta &= c(q_f - 1). \end{aligned}$$

Since, by hypothesis,  $x = k_U + k_F \geq q_d$ , we have:

$$x - \gamma = k_U + k_F - q_d + 1 > 0. \tag{20}$$

We also have:

$$\begin{aligned}
 \varphi\gamma - \beta &= (\alpha - \gamma)\gamma - c(q_f - 1) = \alpha\gamma - \gamma^2 - c(q_f - 1), \\
 &= \alpha(q_d - 1) - (q_d - 1)^2 - (n - t)(q_f - 1) > 0, \quad (\text{by sf-}k2l\text{-Assumption 3}) \\
 \varphi\gamma - \beta &> 0.
 \end{aligned} \tag{21}$$

Injecting (20) and (21) into (19), we conclude that  $\varphi + \frac{\varphi\gamma - \beta}{x - \gamma}$  is a *decreasing hyperbole* defined over  $x \in ]\gamma, \infty]$  with *asymptotic value*  $\varphi$  when  $x \rightarrow \infty$ . As  $x$  is a number of correct processes,  $x \leq c$ . The decreasing nature of the right-hand side of (19) leads us to:  $\ell_e \geq \varphi + \frac{\varphi\gamma - \beta}{c - \gamma} = \frac{\varphi c - \beta}{c - \gamma} \geq \frac{c(c - d - q_d + q_f) - c(q_f - 1)}{c - q_d + 1} \geq c \times \frac{c - d - q_d + 1}{c - q_d + 1} = c \left(1 - \frac{d}{c - q_d + 1}\right)$ .

Since  $\ell_e$  is a positive integer, we conclude that at least  $\ell_{\min} = \left\lceil c \left(1 - \frac{d}{c - q_d + 1}\right) \right\rceil$  correct processes receive at least  $q_d$  message `ENDORSE`( $v, id$ ) at line 10. As each of these processes either  $k2l$ -delivers ( $v, id$ ) when this first happens, or has already  $k2l$ -delivered another value  $v' \neq v$  with identity  $id$ , we conclude that at least  $\ell_{\min}$  correct processes  $k2l$ -deliver some value (whether it be  $v$  or  $v' \neq v$ ) with identity  $id$  when  $c = n - t$ . The reasoning for extending this result to any value of  $c \in [n - t, n]$  is identical to the one at the end of the proof of Lemma 23 just above.  $\square$

**Lemma 25** ( *$k2l$ -Weak-Global-delivery*). *If  $single = \mathbf{false}$ , and a correct process  $k2l$ -delivers a value  $v$  with identity  $id$ , then at least  $\ell = \left\lceil c \left(1 - \frac{d}{c - q_d + 1}\right) \right\rceil$  correct processes  $k2l$ -deliver a value  $v'$  with identity  $id$  (each possibly different from  $v$ ).*

**Proof.** Let us assume  $single = \mathbf{false}$ , and one correct process  $k2l$ -delivers ( $v, id$ ). By Lemma 21,  $k_{NB} = 0$ . The prerequisites for Lemma 23 are verified, and therefore  $k_U + k_F \geq q_d$ . This provides the prerequisites for Lemma 24, from which we conclude that at least  $\ell = \left\lceil c \left(1 - \frac{d}{c - q_d + 1}\right) \right\rceil$  correct processes  $k2l$ -deliver a value  $v'$  with identity  $id$ , which concludes the proof of the lemma.  $\square$

**Lemma 26** ( *$k2l$ -Strong-Global-delivery*). *If  $single = \mathbf{true}$ , and a correct process  $k2l$ -delivers a value  $v$  with identity  $id$ , and no correct process  $k2l$ -casts a value  $v' \neq v$  with identity  $id$ , then at least  $\ell = \left\lceil c \left(1 - \frac{d}{c - q_d + 1}\right) \right\rceil$  correct processes  $k2l$ -deliver  $v$  with identity  $id$ .*

**Proof.** Let us assume that (i)  $single = \mathbf{true}$ , (ii) no correct process  $k2l$ -casts ( $v', id$ ) with  $v' \neq v$ , and (iii) one correct process  $k2l$ -delivers ( $v, id$ ). Lemma 19 holds and implies that  $k_{NB} = 0$ . From there, as above, Lemmas 23 and 24 hold, and at least  $\ell = \left\lceil c \left(1 - \frac{d}{c - q_d + 1}\right) \right\rceil$  correct processes  $k2l$ -deliver a value for identity  $id$ .

By hypothesis, no correct process broadcasts  $\text{ENDORSE}(v', id)$  at line 4 with  $v' \neq v$ . Similarly, because of Lemma 19, no correct process broadcasts  $\text{ENDORSE}(v', id)$  at line 8 with  $v' \neq v$ . As a result, a correct process can receive at most receive  $t$  messages  $\text{ENDORSE}(v', id)$  at line 10 (all from Byzantine processes). As  $q_d > t$  (by  $\text{sf-}k2\ell$ -Assumption 1), the condition of line 10 never becomes true for  $v' \neq v$ , and as result no correct process delivers a value  $v' \neq v$  with identity  $id$ . All processes that  $k2\ell$ -deliver a value with identity  $id$ , therefore,  $k2\ell$ -deliver  $v$ , which concludes the lemma.  $\square$

# PROOF OF THE SIGNATURE-FREE MBRB IMPLEMENTATIONS

The proofs of correctness that follow use integer arithmetic. Given a real number  $x$  and an integer  $i$ , let us recall that  $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$ ,  $\lfloor x + i \rfloor = \lfloor x \rfloor + i$ ,  $\lceil x + i \rceil = \lceil x \rceil + i$ ,  $\lfloor -x \rfloor = -\lceil x \rceil$ ,  $(i > x) \iff (i \geq \lfloor x \rfloor + 1)$ ,  $(i < x) \iff (i \leq \lceil x \rceil - 1)$ .

## C.1 Proof of Bracha's reconstructed MBRB (Algorithm 5)

### C.1.1 Instantiating the parameters of the $k2\ell$ -cast objects

In Algorithm 5 (page 70), we instantiate the  $k2\ell$ -cast objects  $obj_E$  and  $obj_R$  using the signature-free implementation presented in Section 5.2. Let us mention that, given that  $obj_E.single = obj_R.single = \mathbf{true}$ , then we use the strong variant of the global-delivery property of  $k2\ell$ -cast ( $k2\ell$ -Strong-Global-delivery) for both objects  $obj_E$  and  $obj_R$ . Moreover, according to the definitions of  $k'$ ,  $k$ ,  $\ell$  and  $\delta$  and their values stated in Theorem 4, we have:

- $obj_E.k' = obj_E.q_f - n + c = t + 1 - n + c \geq t + 1 - t = 1$ ,
- $obj_E.k = \left\lfloor \frac{c(obj_E.q_f - 1)}{c - d - obj_E.q_d + obj_E.q_f} \right\rfloor + 1 = \left\lfloor \frac{c(t+1-1)}{c - d - \lfloor \frac{n+t}{2} \rfloor - 1 + t + 1} \right\rfloor + 1$   
 $= \left\lfloor \frac{ct}{c - d - \lfloor \frac{n+t}{2} \rfloor} \right\rfloor + 1$ ,
- $obj_E.\ell = \left\lceil c \left( 1 - \frac{d}{c - obj_E.q_d + 1} \right) \right\rceil = \left\lceil c \left( 1 - \frac{d}{c - \lfloor \frac{n+t}{2} \rfloor} - 1 + 1 \right) \right\rceil$   
 $= \left\lceil c \left( 1 - \frac{d}{c - \lfloor \frac{n+t}{2} \rfloor} \right) \right\rceil$ ,
- $obj_E.\delta = ((obj_E.q_f > \frac{n+t}{2}) \vee (obj_E.single \wedge obj_E.q_d > \frac{n+t}{2}))$   
 $= ((t + 1 > \frac{n+t}{2}) \vee (\mathbf{true} \wedge \lfloor \frac{n+t}{2} \rfloor + 1 > \frac{n+t}{2}))$   
 $= (\mathbf{false} \vee (\mathbf{true} \wedge \mathbf{true})) = \mathbf{true}$ ,
- $obj_R.k' = obj_R.q_f - n + c = t + 1 - n + c \geq t + 1 - t = 1$ ,



- $obj_R.k = \left\lfloor \frac{c(obj_R.q_f - 1)}{c - d - obj_R.q_d + obj_R.q_f} \right\rfloor + 1 = \left\lfloor \frac{c(t+1-1)}{c-d-2t-d-1+t+1} \right\rfloor + 1$   
 $= \left\lfloor \frac{ct}{c-2d-t} \right\rfloor + 1,$
- $obj_R.l = \left\lceil c \left( 1 - \frac{d}{c - obj_R.q_d + 1} \right) \right\rceil = \left\lceil c \left( 1 - \frac{d}{c - 2t - d - 1 + 1} \right) \right\rceil$   
 $= \left\lceil c \left( 1 - \frac{d}{c - 2t - d} \right) \right\rceil,$
- $obj_R.\delta = ((obj_R.q_f > \frac{n+t}{2}) \vee (obj_R.single \wedge obj_R.q_d > \frac{n+t}{2}))$   
 $= ((t+1 > \frac{n+t}{2}) \vee (\mathbf{true} \wedge 2t + d + 1 > \frac{n+t}{2})) \in \{\mathbf{true}, \mathbf{false}\}.$

We recall that parameter  $\delta$  controls the conditional no-duplicity property. The value for  $obj_E.\delta$  is **true**, but that of value for  $obj_R.\delta$  may be either **true** or **false** depending on the values of  $n$ ,  $t$ , and  $d$ . This is fine because, in Bracha's reconstructed algorithm (Algorithm 5), it is the first round ( $obj_E$ ) that ensures no-duplicity. Once this has happened, the second round ( $obj_R$ ) does not need to provide no-duplicity but only needs to guarantee the termination properties of local and global delivery. This observation allows  $obj_R$  to operate with lower values of  $q_d$  and  $q_f$ .

Finally, we observe that for Algorithm 5, sf- $k2\ell$ -Assumptions 1 through 4 are all satisfied by B87-Assumption  $n > 3t + 2d + 2\sqrt{td}$ . In the following, we prove that  $3t + 2d + 2\sqrt{td} \geq 2t + d + \sqrt{t^2 + 6td + d^2} \geq 3t + 2d$ .

**Observation 40.1.** For  $d, t \in \mathbb{N}_0$  non-negative integers, we have:

$$3t + 2d + 2\sqrt{td} \geq 2t + d + \sqrt{t^2 + 6td + d^2} \geq 3t + 2d.$$

**Proof.** Let us start by proving the first inequality.

$$\begin{aligned} t^2 + 6td + d^2 + 4\sqrt{td}(t+d) &\geq t^2 + 6td + d^2, \\ t^2 + d^2 + 4td + 4t\sqrt{td} + 4d\sqrt{td} + 2td &\geq t^2 + 6td + d^2, \\ (t+d+2\sqrt{td})^2 &\geq t^2 + 6td + d^2, \\ t+d+2\sqrt{td} &\geq \sqrt{t^2 + 6td + d^2}, \\ 3t+2d+2\sqrt{td} &\geq 2t+d+\sqrt{t^2 + 6td + d^2}. \end{aligned}$$

Let us then prove the second inequality:

$$t^2 + 6td + d^2 \geq t^2 + 2td + d^2 = (t+d)^2,$$

$$\begin{aligned}\sqrt{t^2 + 6td + d^2} &\geq t + d, \\ 2t + d + \sqrt{t^2 + 6td + d^2} &\geq 3t + 2d.\end{aligned}\quad \square$$

### C.1.2 Proof of satisfaction of the assumptions of Algorithm 4

In the following, we prove that all the assumptions of the signature-free  $k2\ell$ -cast implementation presented in Algorithm 4 (page 63) are well respected for the two  $k2\ell$ -cast instances used in Algorithm 5 ( $obj_E$  and  $obj_R$ ).

**Lemma 41.** Algorithm 4's *sf-k2ℓ-Assumptions* are well respected for  $obj_E$ .

**Proof.** Let us recall that  $q_f = t + 1$  and  $q_d = \lfloor \frac{n+t}{2} \rfloor + 1$  for  $obj_E$ .

- *Proof of satisfaction of sf-k2ℓ-Assumption 1* ( $c - d \geq obj_E \cdot q_d \geq obj_E \cdot q_f + t \geq 2t + 1$ ):

By B87-Assumption and Observation 40.1, we have the following:

$$\begin{aligned}c - d &\geq n - t - d = \frac{2n - 2t - 2d}{2}, && \text{(by definition of } c) \\ &> \frac{n + 3t + 2d - 2t - 2d}{2} = \frac{n + t}{2}, && \text{(as } n > 3t + 2d) \\ &\geq \left\lfloor \frac{n + t}{2} \right\rfloor + 1.\end{aligned}\quad (22)$$

We also have:

$$\begin{aligned}\left\lfloor \frac{n + t}{2} \right\rfloor + 1 &\geq \left\lfloor \frac{3t + 2d + 1 + t}{2} \right\rfloor + 1, && \text{(as } n > 3t + 2d) \\ &\geq \left\lfloor 2t + d + \frac{1}{2} \right\rfloor + 1 = 2t + d + 1 \geq 2t + 1.\end{aligned}\quad (23)$$

By combining ((22)) and ((23)), we get:

$$\begin{aligned}c - d &\geq \left\lfloor \frac{n + t}{2} \right\rfloor + 1 \geq 2t + 1 \geq 2t + 1, \\ c - d &\geq obj_E \cdot q_d \geq obj_E \cdot q_f + t \geq 2t + 1. && \text{(sf-k2ℓ-Assumption 1)}\end{aligned}$$

- *Proof of satisfaction of sf-k2ℓ-Assumption 2* ( $\alpha^2 - 4(obj_E \cdot q_f - 1)(n - t) \geq 0$ ):

Let us recall that, for object  $obj_E$ , we have  $q_f = t + 1$  and  $q_d = \lfloor \frac{n+t}{2} \rfloor + 1$ . We therefore have  $\alpha = n + q_f - t - d - 1 = n - d$ . Let us now consider the quantity:

$$\begin{aligned}\Delta &= \alpha^2 - 4(q_f - 1)(n - t) = (n - d)^2 - 4t(n - t) \\ &= 4t^2 + d^2 + n^2 + n(-4t - 2d).\end{aligned}$$

The inequality is satisfied if  $n > 2\sqrt{td} + 2t + d$ , which is clearly the case as  $n > 3t + 2d + 2\sqrt{td}$ . This proves sf- $k2\ell$ -Assumption 2.

- *Proof of satisfaction of sf- $k2\ell$ -Assumption 3* ( $\alpha(\text{obj}_E \cdot q_d - 1) - (\text{obj}_E \cdot q_f - 1)(n - t) - (\text{obj}_E \cdot q_d - 1)^2 > 0$ ):

Let us consider the quantity on the left-hand side of sf- $k2\ell$ -Assumption 3 and substitute  $q_f = t + 1$ ,  $q_d = \lfloor \frac{n+t}{2} \rfloor + 1$ :

$$\begin{aligned}& \alpha(q_d - 1) - (q_f - 1)(n - t) - (q_d - 1)^2, \\ &= (n + q_f - t - d - 1)(q_d - 1) - (q_f - 1)(n - t) - (q_d - 1)^2, \\ &= (n - d) \left( \left\lfloor \frac{n+t}{2} \right\rfloor \right) - t(n - t) - \left( \left\lfloor \frac{n+t}{2} \right\rfloor \right)^2.\end{aligned}\tag{24}$$

We now observe that  $\left( \left\lfloor \frac{n+t}{2} \right\rfloor \right) = \left( \frac{n+t-\varepsilon}{2} \right)$  with  $\varepsilon = 0$  if  $n + t = 2k$  is even, and  $\varepsilon = 1$  if  $n + t = 2k + 1$  is odd. We thus rewrite (24) as follows:

$$\begin{aligned}& (n - d) \left( \frac{n+t-\varepsilon}{2} \right) - t(n - t) - \left( \frac{n+t-\varepsilon}{2} \right)^2, \\ &= \frac{n+t-\varepsilon}{2} \times \frac{2n-2d-n-t+\varepsilon}{2} - t(n - t), \\ &= \frac{(n+t-\varepsilon)(n-2d-t+\varepsilon) - 4t(n-t)}{4}, \\ &= \frac{n^2 - t^2 - 2td + 2t\varepsilon - 2nd + 2d\varepsilon - \varepsilon^2 - 4nt + 4t^2}{4}, \\ &= \frac{n^2 + 3t^2 - 2td - 2n(d+2t) + \varepsilon(2t+2d-\varepsilon)}{4}.\end{aligned}$$

As we want to show that the above quantity is positive, the result will not change if we multiply it by 4:

$$n^2 + 3t^2 - 2td - 2n(d + 2t) + \varepsilon(2t + 2d - \varepsilon) > 0.\tag{25}$$

We now solve the inequality to obtain:

$$n > 2t + d + \sqrt{t^2 + 6td + d^2 - \varepsilon(2t + 2d - \varepsilon)}.$$

We observe that, for  $t + d \geq 1$ , the quantity  $-\varepsilon(2t + 2d - \varepsilon)$  is strictly negative if  $\varepsilon = 1$ , therefore if  $\varepsilon = 1 \vee t + d \geq 1$ :

$$\begin{aligned} n &> 3t + 2d + 2\sqrt{td}, \\ &\geq t + d + \sqrt{t^2 + 6td + d^2}, \quad (\text{by Observation 40.1}) \\ &\geq 2t + d + \sqrt{t^2 + 6td + d^2 - \varepsilon(2t + 2d - \varepsilon)}. \end{aligned}$$

This leaves out the case  $(t = d = 0) \wedge (n = 2k + 1 \text{ is odd})$ , for which we can show that (25) is positive or null for  $n \geq 1$ :

$$\begin{aligned} (25) : n^2 + 3t^2 - 2td - 2n(d + 2t) + \varepsilon(2t + 2d - \varepsilon), \\ = n^2 - 1 \geq 0 \text{ for } n \geq 1. \end{aligned}$$

This completes the proof of sf- $k2\ell$ -Assumption 3.

- *Proof of satisfaction of sf- $k2\ell$ -Assumption 4* ( $\alpha(obj_E.q_d - 1 - t) - (obj_E.q_f - 1)(n - t) - (obj_E.q_d - 1 - t)^2 \geq 0$ ):

Let us consider the quantity on the left-hand side of sf- $k2\ell$ -Assumption 4 and substitute  $q_f = t + 1$ ,  $q_d = \lfloor \frac{n+t}{2} \rfloor + 1$ :

$$\begin{aligned} &\alpha(q_d - 1 - t) - (q_f - 1)(n - t) - (q_d - 1 - t)^2, \\ &= (n + q_f - t - d - 1)(q_d - 1 - t) - (q_f - 1)(n - t) - (q_d - 1 - t)^2, \\ &= (n - d) \left( \left\lfloor \frac{n+t}{2} \right\rfloor - t \right) - t(n - t) - \left( \left\lfloor \frac{n+t}{2} \right\rfloor - t \right)^2. \end{aligned} \quad (26)$$

Like before, we observe that  $\lfloor \frac{n+t}{2} \rfloor = \frac{n+t-\varepsilon}{2}$  with  $\varepsilon = 0$  if  $n + t = 2k$  is even, and  $\varepsilon = 1$  if  $n + t = 2k + 1$  is odd. We thus rewrite (26) as follows:

$$\begin{aligned} &(n - d) \left( \frac{n+t-\varepsilon}{2} - t \right) - t(n - t) - \left( \frac{n+t-\varepsilon}{2} - t \right)^2, \\ &= (n - d) \times \frac{n-t-\varepsilon}{2} - t(n - t) - \left( \frac{n-t-\varepsilon}{2} \right)^2, \\ &= \frac{n-t-\varepsilon}{2} \times \frac{2n-2d-n+t+\varepsilon}{2} - t(n - t), \\ &= \frac{(n-t-\varepsilon)(n-2d+t+\varepsilon) - 4nt + 4t^2}{4}, \\ &= \frac{-t^2 + 2td - 2t\varepsilon + 2d\varepsilon - 2dn - \varepsilon^2 + n^2}{4}. \end{aligned}$$

As we want to show that the above quantity is non-negative, the result will not change if we multiply it by 4:

$$-t^2 + 2td - 2t\varepsilon + 2d\varepsilon - \varepsilon^2 - 2dn + n^2.$$

We then solve the inequality to obtain:  $n \geq \max(t + \varepsilon, -t + 2d - \varepsilon)$ , which is clearly satisfied as  $n \geq 3t + 2d + 2\sqrt{td} + 1$ . This proves all previous inequality and thus sf- $k2\ell$ -Assumption 4.  $\square$

**Lemma 42.** Algorithm 4's sf- $k2\ell$ -Assumptions are well respected for  $obj_R$ .

**Proof.** Let us recall that  $q_f = t + 1$  and  $q_d = 2t + d + 1$  for  $obj_R$ . Let us observe that we have then  $q_d - q_f - t - d = 0$ .

- *Proof of satisfaction of sf- $k2\ell$ -Assumption 1* ( $c - d \geq obj_R.q_d \geq obj_R.q_f + t \geq 2t + 1$ ):

From Observation 40.1, we have:

$$\begin{aligned} c - d &\geq n - t - d \geq 3t + 2d + 1 - t - d \geq 2t + d + 1, & (\text{as } n > 3t + 2d) \\ c - d &\geq 2t + d + 1 \geq 2t + 1 \geq 2t + 1, \\ c - d &\geq obj_R.q_d \geq obj_R.q_f + t \geq 2t + 1. & (\text{sf-}k2\ell\text{-Assumption 1}) \end{aligned}$$

- *Proof of satisfaction of sf- $k2\ell$ -Assumption 2* ( $\alpha^2 - 4(obj_R.q_f - 1)(n - t) \geq 0$ ):

Let us recall that, for object  $obj_R$ , we have  $q_f = t + 1$  and  $q_d = 2t + d + 1$ . As sf- $k2\ell$ -Assumption 2 depends on  $q_d$  but not on  $q_f$ , and since  $obj_E.q_f = obj_R.q_f$ , we refer the reader to the proof we gave in Lemma 41 for  $obj_E$ .

- *Proof of satisfaction of sf- $k2\ell$ -Assumption 3* ( $\alpha(obj_R.q_d - 1) - (obj_R.q_f - 1)(n - t) - (obj_R.q_d - 1)^2 > 0$ ):

Let us consider the quantity on the left-hand side of sf- $k2\ell$ -Assumption 3:

$$\begin{aligned} &\alpha(q_d - 1) - (q_f - 1)(n - t) - (q_d - 1)^2, \\ &= (n + q_f - t - d - 1)(q_d - 1) - (q_f - 1)(n - t) - (q_d - 1)^2, \\ &= (n - d)(2t + d) - t(n - t) - (2t + d)^2, \\ &= 2nt + nd - 2td - d^2 - nt + t^2 - 4t^2 - d^2 - 4td, \\ &= n(t + d) - 6td - 2d^2 - 3t^2, \\ &= n(t + d) - (6td + 2d^2 + 3t^2). \end{aligned} \tag{27}$$

Then, we observe that we can lower bound the quantity on the left side of (27) by substituting B87-Assumption, *i.e.*,  $n > 3t + 2d + 2\sqrt{td} \geq 2t + d + \sqrt{t^2 + 6td + d^2}$ . For convenience, in the following, we write  $\rho = t^2 + 6td + d^2$ , thus  $n > 2t + d + \sqrt{\rho}$ . We get:

$$\begin{aligned} & n(t + d) - (3t^2 + 6td + 2d^2), \\ & > (2t + d + \sqrt{\rho})(t + d) - (3t^2 + 6td + 2d^2), \\ & = \sqrt{\rho}(t + d) - d^2 - t^2 - 3td. \end{aligned}$$

We now want to show that the above quantity is positive or null, *i.e.*:

$$\sqrt{\rho}(t + d) - d^2 - t^2 - 3td \geq 0. \quad (28)$$

We now rewrite (28) as follows:

$$\begin{aligned} \sqrt{\rho}(t + d) &\geq d^2 + t^2 + 2td + td, \\ \sqrt{\rho}(t + d) &\geq (d + t)^2 + td, \\ (t^2 + 6td + d^2)(t + d)^2 &\geq ((d + t)^2 + td)^2, \quad (\text{as } (d + t)^2 + td \geq 0) \\ ((t + d)^2 + 4td)(t + d)^2 &\geq ((d + t)^2 + td)^2, \\ (t + d)^4 + 4td(t + d)^2 &\geq (d + t)^4 + (td)^2 + 2td(t + d)^2, \\ 2td(t + d)^2 &\geq (td)^2, \\ 2td(t^2 + d^2 + 2td) &\geq (td)^2, \\ 2td(t^2 + d^2) + 4(td)^2 &\geq (td)^2, \\ 2td(t^2 + d^2) + 3(td)^2 &\geq 0. \end{aligned}$$

This proves (28) and all previous inequalities and ultimately sf- $k2\ell$ -Assumption 3.

- *Proof of satisfaction of sf- $k2\ell$ -Assumption 4* ( $\alpha(\text{obj}_R \cdot q_d - 1 - t) - (\text{obj}_R \cdot q_f - 1)(n - t) - (\text{obj}_R \cdot q_d - 1 - t)^2 \geq 0$ ):

Let us consider the quantity on the left-hand side of sf- $k2\ell$ -Assumption 4:

$$\begin{aligned} & \alpha(q_d - 1 - t) - (q_f - 1)(n - t) - (q_d - 1 - t)^2, \\ & = (n + q_f - t - d - 1)(q_d - 1 - t) - (q_f - 1)(n - t) - (q_d - 1 - t)^2, \\ & = (n - d)(t + d) - t(n - t) - (t + d)^2, \\ & = (t + d)(n - 2d - t) - t(n - t), \\ & = nt + nd - 2td - 2d^2 - t^2 - td - nt + t^2, \end{aligned} \quad (29)$$

$$\begin{aligned}
 &= nd - 3td - 2d^2, \\
 &= d(n - 3t - 2d).
 \end{aligned} \tag{30}$$

Like before, we observe that we can lower bound the quantity on the left side of (30) by substituting B87-Assumption, *i.e.*,  $n > 3t + 2d + 2\sqrt{td} \geq 3t + 2d$ , so we have:

$$\begin{aligned}
 (30) : d(n - 3t - 2d) \\
 &> d(3t + 2d - 2d - 3t) = 0.
 \end{aligned} \tag{31}$$

which recursively proves that (29) is positive or zero, and thus sf- $k2\ell$ -Assumption 4.  $\square$

### C.1.3 MBRB proof of Algorithm 5

We prove in the sequel the following theorem.

**Theorem 8** (MBRB-Correctness). *If B87-Assumption is verified, then Algorithm 5 implements MBRB with the guarantee  $\ell_{MBRB} = \left\lceil c \left(1 - \frac{d}{c-2t-d}\right) \right\rceil$ .*

The proof follows from the next lemmas.

**Lemma 43.**  $c - d \geq \text{obj}_E.k$ .

**Proof.** We want to show that:

$$c - d \geq \left\lfloor \frac{ct}{c - d - \lfloor \frac{n-t}{2} \rfloor} \right\rfloor + 1 = \text{obj}_E.k. \tag{32}$$

As the left-hand side is also an integer, we can rewrite (32) as follows:

$$\begin{aligned}
 c - d &> \frac{ct}{c - d - \lfloor \frac{n-t}{2} \rfloor}, \\
 (c - d) \left( c - d - \left\lfloor \frac{n-t}{2} \right\rfloor \right) &> ct. \quad (\text{as } (c - d - \lfloor \frac{n-t}{2} \rfloor) > 0)
 \end{aligned}$$

We now observe that  $(\lfloor \frac{n+t}{2} \rfloor) = (\frac{n+t-\varepsilon}{2})$  with  $\varepsilon = 0$  if  $n + t = 2k$  is even, and  $\varepsilon = 1$  if  $n + t = 2k + 1$  is odd, which leads us to:

$$(c - d) \left( c - d - \frac{n - t - \varepsilon}{2} \right) > ct,$$

$$(c - d)(2c - 2d - n + t + \varepsilon) > 2ct,$$

$$(c - d)(2c - 2d - n + t + \varepsilon) - 2ct > 0.$$

Like for the proofs of Lemma 23 and Lemma 24, we leverage the fact that the executions that can happen when  $c > n - t$  can also occur when  $c = n - t$ . We thus rewrite our inequality for  $c = n - t$ :

$$(n - t - d)(n - t - 2d + \varepsilon) - 2(n - t)t > 0,$$

$$(n - t)(n - t - 2d + \varepsilon - 2t) - d(n - t - 2d + \varepsilon) > 0,$$

$$(n - t)^2 + (n - t)(-2d + \varepsilon - 2t) - d(n - t - 2d + \varepsilon) > 0,$$

$$n^2 + t^2 - 2nt - 2nd + n\varepsilon - 2nt + 2td - t\varepsilon + 2t^2 - nd + td + 2t^2 - \varepsilon d > 0,$$

$$n^2 + 3t^2 - 4nt - 3nd + n\varepsilon + 3td - t\varepsilon + 2d^2 - \varepsilon d > 0,$$

$$n^2 - n(4t + 3d - \varepsilon) + 3t^2 + 3td + 2d^2 - \varepsilon(t + d) > 0.$$

We now solve the second-degree inequality with respect to  $n$ . It is easy to see that the discriminant is non-negative for non-negative values of  $t$  and  $d$ . So we obtain:

$$n > 2t + \frac{3d}{2} - \frac{\varepsilon}{2} + \frac{\sqrt{4t^2 + 12td - 4t\varepsilon + d^2 - 2d\varepsilon + \varepsilon^2}}{2},$$

$$-4t - 3d + \varepsilon + 2n - \sqrt{4t^2 + 12td - 4t\varepsilon + d^2 - 2d\varepsilon + \varepsilon^2} > 0,$$

which is implied by the following as  $n \geq 3t + 2d + 2\sqrt{td} + 1$ :

$$4\sqrt{td} + 2t + d + 2 + \varepsilon - \sqrt{4t^2 + 12td - 4t\varepsilon + d^2 - 2d\varepsilon + \varepsilon^2} > 0,$$

$$4\sqrt{td} + 2t + d + 2 + \varepsilon > \sqrt{4t^2 + 12td - 4t\varepsilon + d^2 - 2d\varepsilon + \varepsilon^2}.$$

Taking the squares as both the argument of the square root and the left-hand side are non-negative leads to:

$$(4\sqrt{td} + 2t + d + \varepsilon + 2)^2 > 4t^2 + 12td - 4t\varepsilon + d^2 - 2d\varepsilon + \varepsilon^2,$$

$$16t^{\frac{3}{2}}\sqrt{d} + 8\sqrt{td}^{\frac{3}{2}} + 8\sqrt{t}\sqrt{d}\varepsilon + 16\sqrt{t}\sqrt{d} + 4t^2 + 20td + 4t\varepsilon + 8t + d^2 + 2d\varepsilon$$

$$+ 4d + \varepsilon^2 + 4\varepsilon + 4 > 4t^2 + 12td - 4t\varepsilon + d^2 - 2d\varepsilon + \varepsilon^2,$$

which simplifies to:

$$16t^{\frac{3}{2}}\sqrt{d} + 8\sqrt{td}^{\frac{3}{2}} + 8\sqrt{t}\sqrt{d}\varepsilon + 16\sqrt{t}\sqrt{d} + 8td + 8t\varepsilon + 8t + 4d\varepsilon$$

$$+ 4d + 4\varepsilon + 4 > 0. \tag{33}$$



We can then easily observe that the left-hand side of (33) is strictly positive, thereby proving all previous inequalities and thus the lemma.  $\square$

**Lemma 44.**  $obj_E.l \geq obj_R.k$ .

**Proof.** We need to prove:

$$obj_E.l = \left\lceil c \left( 1 - \frac{d}{c - \lfloor \frac{n+t}{2} \rfloor} \right) \right\rceil \geq \left\lfloor \frac{ct}{c - 2d - t} \right\rfloor + 1 = obj_R.k. \quad (34)$$

For two real numbers  $x$  and  $m$ , we observe that  $x \geq \lfloor m \rfloor + 1$  if and only if  $x > m$ , and that  $m \geq \lfloor m \rfloor$ . Therefore (34) is implied by the following:

$$\begin{aligned} c \left( 1 - \frac{d}{c - \frac{n+t-\varepsilon}{2}} \right) &\geq \frac{ct}{c - t - 2d}, \\ c - \frac{2dc}{2c - t - n + \varepsilon} &> \frac{ct}{c - t - 2d}. \end{aligned}$$

As both denominators are positive, we can solve:

$$\begin{aligned} -t(-t + 2c + \varepsilon - n) - 2d(-t - 2d + c) + (-t - 2d + c)(-t + 2c + \varepsilon - n) &> 0, \\ -t(-t + 2c + \varepsilon - n) + (-t - 2d + c)(-t - 2d + 2c + \varepsilon - n) &> 0, \\ -t(-2t + c + \varepsilon) + (-t - 2d + c)(-2t - 2d + c + \varepsilon) &> 0, \quad (\text{as } c \geq n - t) \\ -t(-2t + c + \varepsilon) + (-t - 2d + c)(-2t - 2d + c + \varepsilon) &> 0, \\ -t(-t + 2c - n) + \varepsilon(-3t - 2d + c) + (-t - 2d + c)^2 &> 0, \\ t^2 - 2tc + dn + \varepsilon(-3t - 2d + c) + (-t - 2d + c)^2 &> 0, \\ t^2 - 2tc + t(2\sqrt{t}\sqrt{d} + 3t + 2d + 1) + \varepsilon(-3t - 2d + c) + (-t - 2d + c)^2 &> 0, \\ (\text{as } n \geq 3t + 2d + 2\sqrt{td}) & \\ 2t^{\frac{3}{2}}\sqrt{d} + 4t^2 + 2td - 2tc + t + \varepsilon(-3t - 2d + c) + (-t - 2d + c)^2 &> 0, \\ 2t^{\frac{3}{2}}\sqrt{d} + 5t^2 + 6td - 4tc + t + 4d^2 - 4dc + c^2 + \varepsilon(-3t - 2d + c) &> 0. \end{aligned}$$

We now consider the two possible values of  $\varepsilon$ :

- $\varepsilon = 0$ :

$$2t^{\frac{3}{2}}\sqrt{d} + 5t^2 + 6td - 4tc + t + 4d^2 - 4dc + c^2 > 0. \quad (35)$$

We solve the inequality with respect to  $c$  to obtain (when the discriminant is positive):

$$c > 2t + 2d + \sqrt{-2t^{\frac{3}{2}}\sqrt{d} - t^2 + 2td - t}$$

which we prove by observing that  $c \geq n - t \geq 2t + 2d + 2\sqrt{td} + 1$  and that:

$$2t + 2d + 2\sqrt{td} + 1 > 2t + 2d + \sqrt{-2t^{\frac{3}{2}}\sqrt{d} - t^2 + 2td - t},$$

as all terms except  $2td$  inside the square root are negative. When the discriminant is negative (e.g., for  $d = 0$ ), inequality (35) is satisfied for all values of  $c$ .

- $\varepsilon = 1$ :

In this case, we obtain:

$$2t^{\frac{3}{2}}\sqrt{d} + 5t^2 + 6td - 4tc - 2t + 4d^2 - 4dc - 2d + c^2 + c > 0,$$

which is implied by a negative discriminant or by:

$$c > 2t + 2d + \sqrt{-2t^{\frac{3}{2}}\sqrt{d} - t^2 + 2td + \frac{1}{4} - \frac{1}{2}}.$$

Like before, we simply observe that:

$$\begin{aligned} 2\sqrt{td} &\geq \sqrt{2td} + \frac{1}{2} - \frac{1}{2}, \\ &\geq \sqrt{2td + \frac{1}{4} - \frac{1}{2}}, \\ &\geq \sqrt{-2t^{\frac{3}{2}}\sqrt{d} - t^2 + 2td + \frac{1}{4} - \frac{1}{2}}, \end{aligned}$$

thereby proving the second case and the lemma.  $\square$

**Lemma 45** (MBRB-Validity). *If a correct process  $p_i$  mbrb-delivers a value  $v$  from a correct process  $p_j$  with sequence number  $sn$ , then  $p_j$  mbrb-broadcast  $v$  with sequence number  $sn$ .*

**Proof.** If  $p_i$  mbrb-delivers  $(v, sn, j)$  at line 2, then it  $k2\ell$ -delivered  $(\text{READY}(v), (sn, j))$  using  $obj_R$ . From  $k2\ell$ -Validity, and as  $obj_R.k' = 1$ , we can assert that at least one correct process  $p_x$   $k2\ell$ -cast  $(\text{READY}(v), (sn, j))$  at line 4, after having  $k2\ell$ -delivered  $(\text{ECHO}(v), (sn, j))$  using  $obj_E$ . Again, from  $k2\ell$ -Validity, we can assert that at least  $obj_E.k' = 1$  correct process  $p_y$   $k2\ell$ -cast  $(\text{ECHO}(v), (sn, j))$  at line 3, after having received an  $\text{INIT}(v, sn)$  message from  $p_j$ . And as  $p_j$  is

correct and the network channels are authenticated, then  $p_j$  has broadcast  $\text{INIT}(v, sn)$  at line 2, during a  $\text{mbrb\_broadcast}(v, sn)$  invocation.  $\square$

**Lemma 46** (MBRB-No-duplication). *A correct process  $p_i$  mbrb-delivers at most one value from a process  $p_j$  with sequence number  $sn$ .*

**Proof.** By  $k2\ell$ -No-duplication, we know that a correct process  $p_i$  can  $k2\ell$ -deliver at most one  $\text{READY}(\star)$  with identity  $(sn, j)$ . Therefore,  $p_i$  can mbrb-deliver only one value from  $p_j$  with sequence number  $sn$ .  $\square$

**Lemma 47** (MBRB-No-duplicity). *No two different correct processes mbrb-deliver different values from a process  $p_i$  with the same sequence number  $sn$ .*

**Proof.** We proceed by contradiction. Let us consider two correct processes  $p_w$  and  $p_x$  that respectively mbrb-deliver  $(v, sn, i)$  and  $(v', sn, i)$  at line 5, such that  $v \neq v'$ . It follows that  $p_w$  and  $p_x$  respectively  $k2\ell$ -delivered  $(\text{READY}(v), (sn, i))$  and  $(\text{READY}(v'), (sn, i))$  using  $\text{obj}_R$ .

From  $k2\ell$ -Validity, and as  $\text{obj}_R.k' \geq 1$ , we can assert that two correct processes  $p_y$  and  $p_z$  respectively  $k2\ell$ -cast  $(\text{READY}(v), (sn, i))$  and  $(\text{READY}(v'), (sn, i))$  at line 4, after having respectively  $k2\ell$ -delivered  $(\text{ECHO}(v), (sn, i))$  and  $(\text{ECHO}(v'), (sn, i))$  using  $\text{obj}_E$ . But as  $\text{obj}_E.\delta = \text{true}$ , then, by  $k2\ell$ -No-duplicity, we know that  $v = v'$ . There is a contradiction.  $\square$

**Lemma 48** (MBRB-Local-delivery). *If a correct process  $p_i$  mbrb-broadcasts a value  $v$  with sequence number  $sn$ , then at least one correct process  $p_j$  eventually mbrb-delivers  $v$  from  $p_i$  with sequence number  $sn$ .*

**Proof.** If  $p_i$  mbrb-broadcasts  $(v, sn)$  at line 2, then it invokes broadcasts  $\text{INIT}(v, sn)$ . By the definition of the MA, the message  $\text{INIT}(v, sn)$  is then received by at least  $c - d$  correct processes at line 3, which then  $k2\ell$ -cast  $(\text{ECHO}(v), sn, i)$ . As  $p_i$  is correct and broadcasts only one message  $\text{INIT}(\star, sn)$ , then no correct process  $k2\ell$ -casts any different  $(\text{ECHO}(\star), sn, i)$ . Moreover, thanks to Lemma 43, we know that:

$$c - d \geq \text{obj}_E.k = \left\lfloor \frac{ct}{c - d - \lfloor \frac{n-t}{2} \rfloor} \right\rfloor + 1.$$

Hence, from  $k2\ell$ -Local-delivery and  $k2\ell$ -Strong-Global-delivery, at least  $obj_E.\ell = \left\lceil c \left( 1 - \frac{d}{c - \lfloor \frac{n+t}{2} \rfloor} \right) \right\rceil$  correct processes eventually  $k2\ell$ -deliver  $(\text{ECHO}(v), (sn, i))$  using  $obj_E$  and then  $k2\ell$ -cast  $(\text{READY}(v), (sn, i))$  using  $obj_R$  at line 4. By  $k2\ell$ -Validity, and as  $obj_R.k' \geq 1$ , then no correct process can  $k2\ell$ -cast a different  $(\text{READY}(\star), (sn, i))$ , because otherwise it would mean that at least one correct process would have  $k2\ell$ -cast a different  $(\text{ECHO}(\star), (sn, i))$ , which is impossible (see before). Moreover, thanks to Lemma 44, we know that:

$$\left\lceil c \left( 1 - \frac{d}{c - \lfloor \frac{n+t}{2} \rfloor} \right) \right\rceil = obj_E.\ell \geq obj_R.k = \left\lfloor \frac{ct}{c - 2d - t} \right\rfloor + 1.$$

Therefore,  $k2\ell$ -Local-delivery applies and we know that at least one correct processes eventually  $k2\ell$ -delivers  $(\text{READY}(v), (sn, i))$  using  $obj_R$  and then  $mbrb$ -delivers  $(v, sn, i)$  at line 5.  $\square$

**Lemma 49** (MBRB-Global-delivery). *If a correct process  $p_i$   $mbrb$ -delivers a value  $v$  from a process  $p_j$  with sequence number  $sn$ , then at least  $\ell_{MBRB} = \left\lceil c \left( 1 - \frac{d}{c - 2t - d} \right) \right\rceil$  correct processes  $mbrb$ -deliver  $v$  from  $p_j$  with sequence number  $sn$ .*

**Proof.** If  $p_i$   $mbrb$ -delivers  $(v, sn, j)$  at line 5, then it has  $k2\ell$ -delivered  $(\text{READY}(v), (sn, j))$  using  $obj_R$ . From  $k2\ell$ -Validity, we know that at least  $obj_R.k' \geq 1$  correct process  $k2\ell$ -cast  $(\text{READY}(v), (sn, j))$  using  $obj_R$  at line 4 and thus  $k2\ell$ -delivered  $(\text{ECHO}(v), (sn, j))$  using  $obj_E$ . From  $k2\ell$ -No-duplicity, and as  $obj_E.\delta = \text{true}$ , we can state that no correct process  $k2\ell$ -delivers any  $(\text{ECHO}(v'), (sn, j))$  where  $v' \neq v$  using  $obj_E$ , so no correct process  $k2\ell$ -casts any  $(\text{READY}(v'), (sn, j))$  where  $v' \neq v$  using  $obj_R$  at line 4. It means that  $k2\ell$ -Strong-Global-delivery applies, and we can assert that at least  $obj_R.\ell = \left\lceil c \left( 1 - \frac{d}{c - 2t - d} \right) \right\rceil = \ell_{MBRB}$  correct processes eventually  $k2\ell$ -deliver  $(\text{READY}(v), (sn, j))$  using  $obj_R$  and thus  $mbrb$ -deliver  $(v, sn, j)$  at line 5.  $\square$

## C.2 Proof of Imbs-Raynal's reconstructed MBRB (Algorithm 6)

### C.2.1 Instantiating the parameters of the $k2\ell$ -cast object

In Algorithm 6 (page 71), we instantiate the  $k2\ell$ -cast object  $obj_W$  using the signature-free implementation presented in Section 5.2 with parameters  $q_d = \lfloor \frac{n+3t}{2} \rfloor + 3d + 1$ ,  $q_f = \lfloor \frac{n+t}{2} \rfloor + 1$ , and  $single = \text{false}$ . Based on Theorem 4} (page 64), these parameters lead to the following values for  $k'$ ,  $k$ ,  $\ell$  and  $\delta$ .

- $obj_W.k' = obj_W.q_f - n + c = \lfloor \frac{n+t}{2} \rfloor + 1 - n + c$   
 $\geq \lfloor \frac{n+t}{2} \rfloor + 1 - n + n - t = \lfloor \frac{n-t}{2} \rfloor + 1,$

- $obj_W.k = \left\lfloor \frac{c(obj_W.q_f - 1)}{c - d - obj_W.q_d + obj_W.q_f} \right\rfloor + 1,$   
 $= \frac{\lfloor c(\lfloor \frac{n+t}{2} \rfloor + 1 - 1) \rfloor}{c - d - (\lfloor \frac{n+3t}{2} \rfloor + 3d + 1) + \lfloor \frac{n+t}{2} \rfloor + 1} + 1,$   
 $= \left\lfloor \frac{c \lfloor \frac{n+t}{2} \rfloor}{c - t - 4d} \right\rfloor + 1,$
- $obj_W.l = \left\lceil c \left( 1 - \frac{d}{c - obj_W.q_d + 1} \right) \right\rceil = \left\lceil c \left( 1 - \frac{d}{c - (\lfloor \frac{n+3t}{2} \rfloor + 3d + 1) + 1} \right) \right\rceil,$   
 $= \left\lceil c \left( 1 - \frac{d}{c - \lfloor \frac{n+3t}{2} \rfloor - 3d} \right) \right\rceil,$
- $obj_W.\delta = ((obj_W.q_f > \frac{n+t}{2}) \vee (obj_W.single \wedge obj_W.q_d > \frac{n+t}{2})),$   
 $= (\mathbf{true} \vee (\mathbf{false} \wedge \mathbf{true})) = \mathbf{true}.$

Finally, we observe that for Algorithm 6, sf- $k2\ell$ -Assumptions 1 through 4 are all satisfied by IR16-Assumption ( $n > 5t + 12d + \frac{2td}{t+2d}$ ).

### C.2.2 Proof of satisfaction of the assumptions of Algorithm 4

This section proves that all the assumptions of the signature-free  $k2\ell$ -cast implementation presented in Algorithm 4} (page 63) are well respected for the  $k2\ell$ -cast instance used in Algorithm 6 ( $obj_W$ ).

**Lemma 50.** Algorithm 4's sf- $k2\ell$ -Assumptions are well respected for  $obj_W$ .

**Proof.** Let us recall that  $q_f = \lfloor \frac{n+t}{2} \rfloor + 1$  and  $q_d = \lfloor \frac{n+3t}{2} \rfloor + 3d + 1$  for object  $obj_W$ .

- *Proof of satisfaction of sf- $k2\ell$ -Assumption 1* ( $c - d \geq obj_W.q_d \geq obj_W.q_f + t \geq 2t + 1$ ):

From IR16-Assumption ( $n > 5t + 12d + \frac{2td}{t+2d}$ ), we get that  $n > 5t + 8d$ , which yields:

$$\begin{aligned}
 c - d &\geq n - t - d = \frac{2n - 2t - 2d}{2}, && \text{(by definition of } c) \\
 &> \frac{n + 5t + 8d - 2t - 2d}{2} = \frac{n + 3t}{2}, && \text{(as } n > 5t + 8d) \\
 &\geq \left\lfloor \frac{n + 3t + 6d}{2} \right\rfloor + 1 = \left\lfloor \frac{n + 3t}{2} \right\rfloor + 3d + 1. && (36)
 \end{aligned}$$

We also have:

$$\begin{aligned} \left\lfloor \frac{n+3t}{2} \right\rfloor + 1 &> \left\lfloor \frac{5t+8d+3t}{2} \right\rfloor + 1 = 4t + 4d + 1, \quad (\text{as } n > 5t + 8d) \\ &\geq 2t + 1. \end{aligned} \tag{37}$$

By combining (36) and (37), we obtain:

$$\begin{aligned} c - d &\geq \left\lfloor \frac{n+3t}{2} \right\rfloor + 3d + 1 \geq \left\lfloor \frac{n+3t}{2} \right\rfloor + 1 \geq 2t + 1, \\ c - d &\geq \text{obj}_W \cdot q_d \geq \text{obj}_W \cdot q_f + t \geq 2t + 1. \end{aligned}$$

- *Proof of satisfaction of sf-k2l-Assumption 2* ( $\alpha^2 - 4(\text{obj}_W \cdot q_f - 1)(n - t) \geq 0$ ):

Let us recall that for object  $\text{obj}_W$ , we have  $q_f = \lfloor \frac{n+t}{2} \rfloor + 1$  and  $q_d = \lfloor \frac{n+3t}{2} \rfloor + 3d + 1$ . We therefore have  $\alpha = \lfloor \frac{3n-t}{2} \rfloor - d$ . Let us now consider the following quantity:

$$\begin{aligned} \Delta &= \alpha^2 - 4(q_f - 1)(n - t), \\ &= \left( \left\lfloor \frac{3n-t}{2} \right\rfloor - d \right)^2 - 4 \left\lfloor \frac{n+t}{2} \right\rfloor (n - t). \end{aligned} \tag{38}$$

We now observe that  $\left( \lfloor \frac{m}{2} \rfloor \right) = \binom{m-\varepsilon}{2}$  with  $\varepsilon = 0$  if  $m = 2k$  is even, and  $\varepsilon = 1$  if  $m = 2k + 1$  is odd. We thus rewrite (38) as follows:

$$\begin{aligned} \Delta &= \left( \frac{3n-t-\varepsilon}{2} - d \right)^2 - 4 \frac{n+t-\varepsilon}{2} (n-t), \\ &= \left( \frac{3n-t-\varepsilon-2d}{2} \right)^2 - 4 \frac{n+t-\varepsilon}{2} (n-t), \\ &= \frac{t^2 + 4td + 2t\varepsilon - 6tn + 4d^2 + 4d\varepsilon - 12dn + \varepsilon^2 - 6\varepsilon n + 9n^2}{4} \\ &\quad + \frac{8t^2 - 8t\varepsilon + 8\varepsilon n - 8n^2}{4}, \\ &= \frac{9t^2 + 4td - 6t\varepsilon - 6tn + 4d^2 + 4d\varepsilon - 12dn + \varepsilon^2 + 2\varepsilon n + n^2}{4}, \\ &= \frac{9t^2 - 6tn + n^2 + 4td - 12dn + 4d^2 + 4d\varepsilon - 6t\varepsilon + \varepsilon^2 + 2\varepsilon n}{4}, \\ &= \frac{(n-3t)^2 + 4d(t-3n+d) + \varepsilon(4d-6t+\varepsilon+2n)}{4}. \end{aligned}$$

We now multiply by 4 and solve the inequality:

$$\begin{aligned} n^2 - 6n(t + 2d) + 9t^2 + 4td + 4d^2 + \varepsilon(-6t + 4d + \varepsilon + 2n) &\geq 0, \\ n &\geq 3t + 4\sqrt{d}\sqrt{2t + 2d} - \varepsilon + 6d - \varepsilon. \end{aligned} \quad (39)$$

By IR16-Assumption we have  $n > 5t + 12d + \frac{2td}{t+2d}$ . To prove (39), we therefore show that  $5t + 12d + \frac{2td}{t+2d} \geq 3t + 4\sqrt{d}\sqrt{2t + 2d} + 6d$ :

$$\begin{aligned} 5t + 12d + \frac{2td}{t+2d} &\geq 3t + 4\sqrt{d}\sqrt{2t + 2d} + 6d \\ \Leftrightarrow 2t + 6d + \frac{2td}{t+2d} &\geq 4\sqrt{d}\sqrt{2t + 2d} \\ \Leftrightarrow \left(2t + 6d + \frac{2td}{t+2d}\right)^2 &\geq 16d(2t + 2d) \\ \Leftrightarrow -16d(t + 2d)(2t + 2d) + (2td + 2t(t + 2d) + 6d(t + 2d))^2 &\geq 0 \\ \Leftrightarrow 4t^4 + 48t^3d + 192t^2d^2 - 32t^2d + 288td^3 - 96td^2 + 144d^4 - 64d^3 &\geq 0. \end{aligned} \quad (40)$$

We observe that (40) holds as  $144d^4 \geq 64d^3$ ,  $288td^3 \geq 96td^2$ , and  $192t^2d^2 \geq 32t^2d$ , therefore proving sf- $k2\ell$ -Assumption 2.

- *Proof of satisfaction of sf- $k2\ell$ -Assumption 3* ( $\alpha(obj_W \cdot q_d - 1) - (obj_W \cdot q_f - 1)(n - t) - (obj_W \cdot q_d - 1)^2 > 0$ ):

Let us consider the quantity on the left-hand side of sf- $k2\ell$ -Assumption 3 and substitute  $q_f = \lfloor \frac{n+t}{2} \rfloor + 1$ ,  $q_d = \lfloor \frac{n+3t}{2} \rfloor + 3d + 1$ , and  $\alpha = \lfloor \frac{3n-t}{2} \rfloor - d$ :

$$\begin{aligned} &\alpha(q_d - 1) - (q_f - 1)(n - t) - (q_d - 1)^2, \\ &= \left(\left\lfloor \frac{3n-t}{2} \right\rfloor - d\right) \left(\left\lfloor \frac{n+3t}{2} \right\rfloor + 3d\right) - \left(\left\lfloor \frac{n+t}{2} \right\rfloor\right)(n-t) \\ &\quad \left(\left\lfloor \frac{n+3t}{2} \right\rfloor + 3d\right)^2. \end{aligned}$$

We now observe that  $\lfloor \frac{m}{2} \rfloor = \left(\frac{m-\varepsilon}{2}\right)$  with  $\varepsilon = 0$  if  $m = 2k$  is even, and  $\varepsilon = 1$  if  $m = 2k + 1$  is odd, and rewrite the expression accordingly:

$$\begin{aligned} &\frac{3n-t-2d-\varepsilon}{2} \times \frac{n+3t+6d-\varepsilon}{2} - \frac{(n+t-\varepsilon)(n-t)}{2} \\ &- \left(\frac{n+3t+6d-\varepsilon}{2}\right)^2, \\ &= \frac{(n+3t+6d-\varepsilon)(3n-t-2d-\varepsilon-n-3t-6d+\varepsilon)}{4} - \frac{(n+t-\varepsilon)(n-t)}{2}, \end{aligned}$$

$$\begin{aligned}
 &= \frac{(n + 3t + 6d - \varepsilon)(2n - 4t - 8d)}{4} - \frac{(n + t - \varepsilon)(n - t)}{2}, \\
 &= \frac{-12t^2 - 48td + 4t\varepsilon + 2tn - 48d^2 + 8d\varepsilon + 4dn - 2\varepsilon n + 2n^2 + 2t^2 - 2t\varepsilon + 2\varepsilon n - 2n^2}{4}, \\
 &= \frac{-10t^2 - 48td + 2t\varepsilon + 2tn - 48d^2 + 8d\varepsilon + 4dn}{4}.
 \end{aligned}$$

As the coefficients of  $n$  are all positive, we can lower-bound the quantity using  $n > 5t + 12d + \frac{2td}{t+2d}$ :

$$\begin{aligned}
 &\frac{-10t^2 - 48td - 48d^2 + 2n(t + 2d) + 2\varepsilon(t + 8d)}{4}, \\
 &= \frac{-10t^2 - 48td - 48d^2 + 2\left(5t + 12d + \frac{2td}{t+2d}\right)(t + 2d) + 2\varepsilon(t + 8d)}{4}, \\
 &= \frac{-10t^2 - 48td - 48d^2 + 10t^2 + 44td + 48d^2 + 4td + 2\varepsilon(t + 8d)}{4}, \\
 &= \frac{\varepsilon(t + 8d)}{2} \geq 0,
 \end{aligned}$$

which proves all previous inequalities and thus *sf-k2l*-Assumption 3.

- *Proof of satisfaction of sf-k2l-Assumption 4* ( $\alpha(\text{obj}_W.q_d - 1 - t) - (\text{obj}_W.q_f - 1)(n - t) - (\text{obj}_W.q_d - 1 - t)^2 \geq 0$ ):

Let us consider the quantity on the left-hand side of *sf-k2l*-Assumption 4 and substitute  $q_f = \lfloor \frac{n+t}{2} \rfloor + 1$ ,  $q_d = \lfloor \frac{n+3t}{2} \rfloor + 3d + 1$ , and  $\alpha = \lfloor \frac{3n-t}{2} \rfloor - d$ :

$$\begin{aligned}
 &\alpha(q_d - 1 - t) - (q_f - 1)(n - t) - (q_d - 1 - t)^2, \\
 &= \left( \lfloor \frac{3n-t}{2} \rfloor - d \right) \left( \lfloor \frac{n+3t}{2} \rfloor + 3d - t \right) - \left( \lfloor \frac{n+t}{2} \rfloor \right) (n - t) \\
 &\quad - \left( \lfloor \frac{n+3t}{2} \rfloor + 3d - t \right)^2.
 \end{aligned}$$

We now observe that  $(\lfloor \frac{m}{2} \rfloor) = (\frac{m-\varepsilon}{2})$  with  $\varepsilon = 0$  if  $m = 2k$  is even, and  $\varepsilon = 1$  if  $m = 2k + 1$  is odd, and rewrite the expression accordingly:

$$\begin{aligned}
 &\left( \frac{3n-t-\varepsilon}{2} - d \right) \left( \frac{n+3t-\varepsilon}{2} + 3d - t \right) - \left( \frac{n+t-\varepsilon}{2} \right) (n - t) \\
 &\quad - \left( \frac{n+3t-\varepsilon}{2} + 3d - t \right)^2,
 \end{aligned}$$



$$\begin{aligned}
&= \left( \frac{3n - t - 2d - \varepsilon}{2} \right) \left( \frac{n + t + 6d - \varepsilon}{2} \right) - \left( \frac{n + t - \varepsilon}{2} \right) (n - t) \\
&\quad - \frac{n + t + 6d - \varepsilon}{2} \Big)^2, \\
&= \frac{(n + t + 6d - \varepsilon)(3n - t - 2d - \varepsilon - n - t - 6d + \varepsilon)}{4} - \left( \frac{(n + t - \varepsilon)(n - t)}{2} \right), \\
&= \frac{(n + t + 6d - \varepsilon)(2n - 2t - 8d)}{4} - \left( \frac{(n + t - \varepsilon)(n - t)}{2} \right), \\
&= \frac{(n + t + 6d - \varepsilon)(n - t - 4d) - (n + t - \varepsilon)(n - t)}{2}, \\
&= \frac{-10td - 24d^2 + 4d\varepsilon + 2dn}{2}.
\end{aligned}$$

As the coefficients of  $n$  are all positive, we can get a lower bound using  $n > 5t + 12d + \frac{2td}{t+2d} > 5t + 12d$  to obtain:

$$\begin{aligned}
&\frac{-10td - 24d^2 + 4d\varepsilon + 2d(5t + 12d)}{2}, \\
&= \frac{-10td - 24d^2 + 4d\varepsilon + 10td + 24d^2}{2}, \\
&= 2d\varepsilon \geq 0,
\end{aligned}$$

which proves sf- $k2\ell$ -Assumption 4. □

### C.2.3 MBRB proof of Algorithm 6

We prove in the sequel the following theorem.

**Theorem 9** (MBRB-Correctness). *If IR16-Assumption is verified, then Algorithm 2 implements MBRB with the guarantee  $\ell_{MBRB} = \left\lceil c \left( 1 - \frac{d}{c - \lfloor \frac{n+3t}{2} \rfloor - 3d} \right) \right\rceil$ .*

The proof follows from the next lemmas.

**Lemma 51.**  $c - d \geq \text{obj}_W \cdot k$ .

**Proof.** This proof is presented in reverse order: we start with the result we want to prove and finish with a proposition we know to be true. In this manner, given two consecutive proposi-

tions, we only need the latter to imply the former and not necessarily the converse. We want to show that:

$$c - d \geq \left\lfloor \frac{c \lfloor \frac{n+t}{2} \rfloor}{c - t - 4d} \right\rfloor + 1 = \text{obj}_W.k,$$

$$c - d > \frac{c \lfloor \frac{n+t}{2} \rfloor}{c - t - 4d}, \quad (\text{as } x \geq \lfloor y \rfloor + 1 \iff x > y)$$

$$c - d > \frac{c \frac{n+t}{2}}{c - t - 4d},$$

$$c - d > \frac{c(n+t)}{2(c-t-4d)},$$

$$c - d > \frac{c(n+t)}{2c-2t-8d},$$

$$(c-d)(2c-2t-8d) > c(n+t), \quad (\text{as } 2c-2t-8d > 0 \text{ by IR16-Assumption})$$

$$(c-d)(2c-2t-8d) > c(c-2t) \geq c(n+t), \quad (\text{as } n \leq c+t)$$

$$(c-d)(2c-2t-8d) - c(c-2t) > 0,$$

$$c^2 + 2td - 4tc + 8d^2 - 10dc > 0,$$

$$2td + 8d^2 + c^2 + c(-4t - 10d) > 0.$$

The left-hand side of the above inequality is a second-degree polynomial, whose roots we can solve:

$$\left[ 2t + 5d - \sqrt{4t^2 + 18td + 17d^2}, 2t + 5d + \sqrt{4t^2 + 18td + 17d^2} \right].$$

We now need to show that:

$$c > 2t + 5d + \sqrt{4t^2 + 18td + 17d^2}.$$

By IR16-Assumption, we know that:

$$n \geq 5t + 12d + \frac{2td}{t+2d} + 1,$$

and thus that:

$$n \geq 5t + 12d + 1,$$

$$c \geq 4t + 12d + 1.$$

So we want to show that:

$$4t + 12d + 1 > 2t + 5d + \sqrt{4t^2 + 18td + 17d^2},$$

$$2t + 7d + 1 > \sqrt{4t^2 + 18td + 17d^2}.$$

It is easy to see that the right-hand side of the above inequality is non-negative, so we get:

$$(2t + 7d + 1)^2 > 4t^2 + 18td + 17d^2,$$

$$4t^2 + 28td + 4t + 49d^2 + 14d + 1 > 4t^2 + 18td + 17d^2,$$

$$10td + 4t + 32d^2 + 14d + 1 > 0.$$

This concludes the proof.  $\square$

**Lemma 52** (MBRB-Validity). *If a correct process  $p_i$  mbrb-delivers a value  $v$  from a correct process  $p_j$  with sequence number  $sn$ , then  $p_j$  mbrb-broadcast  $v$  with sequence number  $sn$ .*

**Proof.** If  $p_i$  mbrb-delivers  $(v, sn, j)$  at line 2, then it  $k2\ell$ -delivered  $(\text{WITNESS}(v), (sn, j))$  using  $obj_W$ . From  $k2\ell$ -Validity, and as  $obj_W.k' \geq 1$ , we can assert that at least one correct process  $p_{i'}$   $k2\ell$ -cast  $(\text{WITNESS}(v), (sn, j))$  at line 3, after having received an  $\text{INIT}(v, sn)$  message from  $p_j$ . And as  $p_j$  is correct and the network channels are authenticated, then  $p_j$  has broadcast  $\text{INIT}(v, sn)$  at line 2, during a  $\text{mbrb\_broadcast}(v, sn)$  invocation.  $\square$

**Lemma 53** (MBRB-No-duplication). *A correct process  $p_i$  mbrb-delivers at most one value from a process  $p_j$  with sequence number  $sn$ .*

**Proof.** By  $k2\ell$ -No-duplication, we know that a correct process  $p_i$  can  $k2\ell$ -deliver at most one  $\text{WITNESS}(\star)$  with identity  $(sn, j)$ . Therefore,  $p_i$  can mbrb-deliver only one value from  $p_j$  with sequence number  $sn$ .  $\square$

**Lemma 54** (MBRB-No-duplicity). *No two distinct correct processes mbrb-deliver different values from a process  $p_i$  with the same sequence number  $sn$ .*

**Proof.** As  $obj_W.\delta = \text{true}$ , then, by  $k2\ell$ -No-duplicity, we know that no two correct processes can  $k2\ell$ -deliver two different values with the same identity using  $obj_W$  at line 4. Hence, no two correct processes mbrb-deliver different values for a given sequence number  $sn$  and sender  $p_i$ .  $\square$

**Lemma 55** (MBRB-Local-delivery). *If a correct process  $p_i$  mbrb-broadcasts an value  $v$  with sequence number  $sn$ , then at least one correct process  $p_j$  eventually mbrb-delivers  $v$  from  $p_i$  with sequence number  $sn$ .*

**Proof.** If  $p_i$  mbrb-broadcasts  $(v, sn)$  at line 2, then it broadcasts  $\text{INIT}(v, sn)$ . By the definition of the MA, the message  $\text{INIT}(v, sn)$  is then received by at least  $c - d$  correct processes at line 3, which then  $k2\ell$ -cast  $(\text{WITNESS}(v), (sn, i))$ . But thanks to Lemma 51}, we know that:

$$c - d \geq \text{obj}_W.k = \left\lfloor \frac{c \lfloor \frac{n+t}{2} \rfloor}{c - t - 4d} \right\rfloor + 1.$$

As  $p_i$  is correct and broadcasts only one message  $\text{INIT}(\star, sn)$ , then no correct process  $k2\ell$ -casts any different  $(\text{WITNESS}(\star), (sn, i))$ ,  $k2\ell$ -Local-delivery applies and at least one correct processes eventually  $k2\ell$ -delivers  $(\text{WITNESS}(v), (sn, i))$  using  $\text{obj}_W$  and thus mbrb-delivers  $(v, sn, i)$  at line 4.  $\square$

**Lemma 56** (MBRB-Global-delivery). *If a correct process  $p_i$  mbrb-delivers a value  $v$  from a process  $p_j$  with sequence number  $sn$ , then at least  $\ell_{\text{MBRB}} = \left\lceil c \left( 1 - \frac{d}{c - \frac{\lfloor n+3t \rfloor}{2} - 3d} \right) \right\rceil$  correct processes mbrb-deliver  $m$  from  $p_j$  with sequence number  $sn$ .*

**Proof.** If  $p_i$  mbrb-delivers  $(v, sn, j)$  at line 4, then it has  $k2\ell$ -delivered  $(\text{WITNESS}(v), (sn, j))$  using  $\text{obj}_W$ . As  $\text{obj}_W.\delta = \text{true}$ , we can assert from  $k2\ell$ -Weak-Global-delivery and  $k2\ell$ -No-duplication that at least  $\text{obj}_W.\ell = \left\lceil c \left( 1 - \frac{d}{c - q_d + 1} \right) \right\rceil$  correct processes eventually  $k2\ell$ -deliver  $(\text{WITNESS}(v), (sn, j))$  using  $\text{obj}_W$  and thus mbrb-deliver  $(v, sn, j)$  at line 4. By substituting the values of  $q_f$  and  $q_d$ , we obtain  $\text{obj}_W.\ell = \left\lceil c \left( 1 - \frac{d}{c - \frac{\lfloor n+3t \rfloor}{2} - 3d} \right) \right\rceil = \ell_{\text{MBRB}}$ , thus proving the lemma.  $\square$



# CORRECTNESS PROOF OF CODED MBRB (ALGORITHMS 8-9)

---

In this appendix, we prove that the Coded MBRB algorithm of Chapter 6 (composed of Algorithm 8 and Algorithm 9) satisfies the single sender/single shot version of the MBRB abstraction (defined in Section 3.2). More specifically, we prove the following theorem.

**Theorem 6.** *If  $c$ -MBRB-Assumption, Algorithms 8 and 9 implement MBRB with the guarantee  $\ell_{\text{MBRB}} = c - (1 + \varepsilon)d$ , where  $\varepsilon > 0$ .*

The safety proofs of Coded MBRB are given in Lemma 60 (MBRB-Validity), Lemma 61 (MBRB-No-duplication), and Lemma 62 (MBRB-No-duplicity), while the liveness proofs of Coded MBRB are given in Lemma 63 (MBRB-Local-delivery) and Lemma 64 (MBRB-Global-delivery). Let us recall that the Coded MBRB algorithm relies on the following assumption (the “ $c$ ” prefix stands for “coded”).

**$c$ -MBRB-Assumption.**  $n > 3t + 2d$  and  $k \leq n - t - 2d$ .

We begin with a few technical lemmas.

**Lemma 57.** *If a correct process  $p_u$  saves a value fragment  $\tilde{v}_j$  associated to a proof of inclusion  $\pi_j$  for some commitment  $C'$  and process identity  $j$ , then  $\pi_j$  is valid with respect to  $C'$ , that is  $\text{vc\_verify}(C', \pi_j, \tilde{v}_j, j) = \text{true}$ .*

**Proof.** A correct process saves fragments for a commitment  $C'$  at lines 24, 32, 46, or 49, when receiving SEND, FORWARD, or BUNDLE messages, respectively. The fragments saved at these lines and their proof have been received through the corresponding message, whose content is verified

---

by a call to `is_valid` (at lines 21, 27, or 45). The `is_valid` function (described in Algorithm 8) checks that inclusion proofs are valid for the corresponding commitment.  $\square$

The following notion of *valid messages* will be used throughout the analysis to indicate messages containing only valid information, as the algorithm dictates.

**Definition 2** (Valid messages). *We say that a message of type `SEND`, `FORWARD`, or `BUNDLE` is valid if and only if `is_valid` returns `true` at line 21, line 27, or line 45, respectively, upon the receipt of that message.*

Operatively, valid messages satisfy the following, which is immediate from the definition of the `is_valid` function (Algorithm 8).

**Corollary 1.** *To be valid, a message must meet the following criteria: (i) all the signatures shares or threshold signatures it contains must be valid and correspond to the commitment included in the message; (ii) if it is of type `SEND` or `FORWARD`, it must contain a signature by the designated sending process  $p_s$ ; and (iii) all inclusion proofs must be valid with respect to the commitment included in the message.*

We now show that the correct parties always send valid messages. However, they might receive invalid messages sent by Byzantine processes.

**Lemma 58.** *All `SEND`, `FORWARD`, or `BUNDLE` messages sent by a correct process  $p_u$ , are valid.*

**Proof.** The only correct process that sends `SEND` messages is  $p_s$  at line 19. Indeed, when  $p_s$  is correct, this message will contain a valid signature share by  $p_s$  produced at line 18, and one of the valid proofs of inclusion produced at line 17.

Now, consider a `FORWARD` message sent either at line 25 or line 36. To reach there,  $p_u$  must have passed line 21 or line 27, which guarantee that  $p_u$  received a valid signature for  $C'$  made by  $p_s$  (where  $C'$  is the commitment in the received message triggering this code). Then, at line 24 or at line 29,  $p_u$  saves a signature of  $p_s$  for this  $C'$ , and at line 23 and line 34,  $p_u$  signs the same  $C'$ . Thus, conditions (i) and (ii) of Corollary 1 hold, and if the `FORWARD` is sent at line 36, then condition (iii) vacuously holds as well. If the `FORWARD` message is sent at line 25, it contains a fragment that was saved by  $p_u$  for the same  $C'$ , and by Lemma 57, its associated proof of inclusion is valid; thus condition (iii) holds in this case as well.

Finally, consider a `BUNDLE` message. First off, condition (ii) of Corollary 1 does not concern this type of message. For the transmission at line 42, condition (i) follows from the construction of the threshold signature  $\Sigma_C$  at line 41.  $\Sigma_C$  is guaranteed to be non- $\perp$  by the condition at line 37 of Algorithm 9, and is provided by the helper function `get_thresh_sig( $\cdot$ )` (line 15 of Algorithm 8). When executing `get_thresh_sig( $\cdot$ )`, the first possibility is that  $\Sigma_C$  is already known by  $p_u$  because it was received by  $p_u$  at line 44 and saved at line 46. In this case, the validity of  $\Sigma_C$  is ensured by the check at line 45. The second possibility is that  $\Sigma_C$  aggregates  $\tau = \lfloor \frac{n+t}{2} \rfloor + 1$  signature shares received by  $p_u$  at line 20 or line 26, and saved at line 24 or line 29, respectively. In this case, the validity of all these signature shares is ensured by the checks at line 21 and line 27, respectively, and thus the aggregated threshold signature  $\Sigma_C$  is also valid. Condition (iii) follows since the proofs of inclusion were computed at line 39 by  $p_u$  and match the same commitment  $C'$  used in that `BUNDLE` message, as enforced by line 40. For the broadcast at line 50, conditions (i) and (iii) follow since the threshold signature  $\Sigma$  and the fragment tuple  $(\tilde{v}'_j, \pi'_j, j)$  come from the incoming `BUNDLE` message at line 44, whose validity (w.r.t.  $C'$ ) has been verified at line 45.  $\square$

**Lemma 59.** *A correct process  $p$  signs a most one commitment  $C$ .*

**Proof.** Some process  $p$  signs a commitment either at line 18 (for  $p_s$ ), line 23, or line 34. We consider two cases, depending on whether  $p$  is  $p_s$  or not.

- *Case 1:  $p \neq p_s$ .* Process  $p$  can sign some commitment only at line 23 or line 34. By the conditions at lines 22 and 28, lines 23 or and 34 are executed only if either  $p$  has not signed any commitment yet, or has already signed the exact same commitment  $C'$ .
- *Case 2:  $p = p_s$ .* Because valid messages must contain  $p_s$ 's signature share (due to calls to `is_valid()` at lines 21 and 27), and because we have assumed that signatures cannot be forged, line 18 is always executed before lines 22 and 28. By the same reasoning as Case 1,  $p_s$  therefore never signs a different commitment at line 23 or line 34.  $\square$

We recall that the above lemmas, and as a consequence, the upcoming theorems, hold with high probability, assuming a computationally-bounded adversary that forges signature shares/threshold signatures or finds commitment collisions with only negligible probability. We can now prove the properties required for an MBRB algorithm.

**Lemma 60** (MBRB-Validity). *If  $p_s$  is correct and a correct process  $p_i$  mbrb-delivers a value  $v$ , then  $p_s$  has previously mbrb-broadcast  $v$ .*



---

**Proof.** Suppose  $p_i$  mbrb-delivers  $v$  at line 43. Consider  $C'$  the commitment that satisfies the condition at line 37, and  $C$  the commitment computed at line 39. It holds that  $C' = C$  by line 40, or otherwise  $p_i$  could not have reached line 43.

Consider the threshold signature  $\Sigma_C$  returned by the `get_thresh_sig` function at line 41. Using the same reasoning as in the proof of Lemma 58,  $\Sigma_C$  is valid, and must, therefore, aggregate at least  $\tau = \lfloor \frac{n+t}{2} \rfloor + 1$  valid signature shares for  $C$ . Let us remark that, out of all these valid signature shares, at least  $\lfloor \frac{n+t}{2} \rfloor + 1 - t = \lfloor \frac{n-t}{2} \rfloor + 1 \geq 1$  are generated by correct processes<sup>42</sup>. Thus, at least one correct process  $p_j$  must have produced a signature share for  $C$ , whether it be at line 23 or line 34 if  $p_j \neq p_s$ , or at line 18 if  $p_j = p_s$ . However, in all these cases, the sender  $p_s$  must have necessarily produced a signature share for  $C$ : the case  $p_j = p_s$  is trivial, and in the case  $p_j \neq p_s$ ,  $p_j$  must have verified the presence of a valid signature share from  $p_s$  in the message it received, at line 21 or line 27, respectively.

Under the assumption that the adversary cannot forge signature shares/threshold signatures (Section 6.2.2), and recalling that  $p_s$  is correct, the only way in which  $p_s$  could have signed  $C'$  is by executing line 18 when mbrb-broadcasting some value  $v'$  at line 16; see also the proof of Lemma 59. Furthermore, recall that the commitment is collision-resistant (or binding, see Section 6.2.2), meaning that except with negligible probability, the value  $v'$  that  $p_s$  uses in line 16 satisfies  $v' = v$ , since it holds that  $C' = \text{Commitment}(v') = \text{Commitment}(v) = C$ .  $\square$

**Lemma 61** (MBRB-No-duplication). *A correct process  $p_i$  mbrb-delivers at most one value  $v$ .*

**Proof.** The condition at line 37 directly implies the proof.  $\square$

**Lemma 62** (MBRB-No-duplicity). *No two different correct processes mbrb-deliver different values.*

**Proof.** Suppose, towards a contradiction, that  $p_i$  mbrb-delivers  $v$  and  $p_j$  mbrb-delivers  $v' \neq v$ , where  $p_i$  and  $p_j$  are both correct processes. Let us denote by  $C$ , resp.  $C'$ , the commitment returned by `compute_frag_vec_commit()` for  $v$ , resp. for  $v'$ . As commitments are assumed to be collision-resistant (Section 6.2.2),  $v \neq v'$  implies  $C \neq C'$ .

By the condition at line 37,  $p_i$  gets a threshold signature  $\Sigma_i \neq \perp$  from the `get_thresh_sig` function that aggregates a set  $Q_i$  containing  $\tau = \lfloor \frac{n+t}{2} \rfloor + 1$  valid signature shares for  $C$ . Similarly,  $p_j$  gets

---

42. Remind that,  $\forall x \in \mathbb{R}, i \in \mathbb{Z} : \lfloor x \rfloor + i = \lfloor x + i \rfloor$ .

a threshold signature  $\Sigma_j$  aggregating a set  $Q_j$  of signature shares for  $C'$ . We know that  $|Q_i \cup Q_j| = |Q_i| + |Q_j| - |Q_i \cap Q_j|$ . Moreover, we know that,  $\forall x \in \mathbb{R}, k \in \mathbb{Z} : k = \lfloor x \rfloor + 1 \Rightarrow k > x$ , and hence we have  $Q_i > \frac{n+t}{2} < Q_j$ . Thus,  $|Q_i \cup Q_j| \geq |Q_i| + |Q_j| - n > 2\frac{n+t}{2} - n = t$ . In other words,  $Q_i$  and  $Q_j$  have at least one *correct* process,  $p_u$ , in common that has signed both  $C$  and  $C'$ . Line 59, and the fact that  $p_u$  has signed both  $C$  and  $C'$  leads the proof to the needed contradiction. Thus,  $v = v'$ , and the lemma holds.  $\square$

**Lemma 63** (MBRB-Local-delivery). *If  $p_s$  is correct and mbrb-broadcasts  $v$ , then at least one correct process  $p_j$  mbrb-delivers  $v$ .*

**Proof.** Let us denote by  $C_v$  the commitment computed at line 3 when executing `compute_frag_vec_commit(v)`. The proof of the lemma will follow from Observations 63.1 to 63.8 stated and proven below.

**Observation 63.1.** *All valid SEND, FORWARD, or BUNDLE messages received by some correct process  $p_u$  contain  $C_v$ .*

**Proof of Observation 63.1.** Recall that  $p_s$  mbrb-broadcasts  $v$ , thus we know that  $p_s$  has included its own signature share,  $sig_s = (\text{ts\_sign\_share}_s(C_v), s)$ , when it propagates `SEND( $C_v, (\tilde{v}_j, \pi_j, j), sig_s$ )` (lines 17-19). Consider a correct process  $p_u$  that receives a valid SEND, FORWARD, or BUNDLE message containing a commitment  $C_u$  at lines 20, line 26, or 44. If the message is of type SEND or FORWARD, then, as it is valid, it must contain  $p_s$ 's signature on  $C_u$ . If the message is of type BUNDLE, then, similarly to Lemma 60, its valid threshold signature for  $C_u$  aggregates a set of valid signature shares for  $C_u$  that contains at least one share produced by a correct process  $p_x$ . But for  $p_x$  to produce this share,  $p_s$  must also have produced a valid signature share for  $C_u$ , either because  $p_x$  must have checked its existence at line 21 or line 27 (before signing, at line 23 or line 34, respectively), or because  $p_x$  is the sender. Hence, in any case,  $p_s$  produces a signature share for  $C_u$ . Since  $p_s$  is correct, by Lemma 59, it does not sign another commitment  $C' \neq C_v$ . Under the assumption that signatures cannot be forged, the above implies that  $C_u = C_v$ .  $\square$ Observation 63.1

**Observation 63.2.** *A correct process  $p_u$  only signs valid signature shares for  $C_v$ .*

**Proof of Observation 63.2.** If  $p_u = p_s$ , it mbrb-broadcasts a single value and executes line 18 only once, signing  $C_v$ . Besides line 18, a correct process  $p_u$  only signs signature shares after receiving a valid SEND or FORWARD message (at line 23 or 34), and when it does,  $p_u$  only ever

---

signs the commitment received in the message. By Observation 63.1, this implies  $p_u$  never signs any  $C' \neq C_v$ . □<sub>Observation 63.2</sub>

**Observation 63.3.** *If a correct process  $p_u$  broadcasts a FORWARD message, this message is of the form  $\text{FORWARD}(C_v, \star, \{sig_s, sig_u\})$ , where  $sig_s, sig_u$  are  $p_s$ 's and  $p_u$ 's valid signature shares for  $C_v$ .*

**Proof of Observation 63.3.** Consider a correct process  $p_u$  that broadcasts a message  $\text{FORWARD}(C', \star, sigs)$  either at lines 25 or 36. By Observation 63.1,  $C' = C_v$ . The observation then follows from Lemma 58. □<sub>Observation 63.3</sub>

We now define  $F_{\text{recv}}$  to be the set of correct processes that receive a valid message  $\text{FORWARD}(C_v, \star, sigs)$  at line 26, where  $sigs$  contains  $p_s$ 's valid signature for  $C_v$ . We analyze its size and the behavior of such processes in the following observations.

**Observation 63.4.**  $F_{\text{recv}}$  contains at least one correct process, i.e.,  $F_{\text{recv}} \neq \emptyset$ .

**Proof of Observation 63.4.** If  $p_s$  is correct and mbrb-broadcasts  $v$ , it executes line 19 and disseminates messages of the form  $\text{SEND}(C_v, (\tilde{v}_j, \pi_j), sig_s)$  to all processes, where  $sig_s$  is  $p_s$ 's signature share of  $C_v$ . By definition of the message adversary, these SEND messages are received by at least  $c - d$  correct processes.

By c-MBRB-Assumption,  $n > 3t + 2d$ , and therefore  $c - d \geq n - t - d > 0$ . At least one correct process  $p_x$ , therefore, receives one of the SEND messages disseminated by  $p_s$  at line 19. As  $p_s$  is correct, by Lemma 58, this message is valid, and is handled by  $p_x$  at lines 20-25.

By Observation 63.2,  $p_x$  only signs signature shares for  $C_v$ , and thus passes the test at line 22, and reaches line 25, where it disseminates a FORWARD message. By Observation 63.3, this message is of the form  $\text{FORWARD}(C_v, \star, \{sig_s, sig_x\})$ , and is valid. As above, by definition of the message adversary, this FORWARD message is received by at least  $c - d > 0$  correct processes. By definition these processes belong to  $F_{\text{recv}}$ , which yield  $|F_{\text{recv}}| > 0$  and  $F_{\text{recv}} \neq \emptyset$ . □<sub>Observation 63.4</sub>

**Observation 63.5.** *Any  $p_u \in F_{\text{recv}}$  broadcasts a  $\text{FORWARD}(C_v, \star, \{sig_s, sig_u\})$  message, where  $sig_s$  and  $sig_u$  are  $p_s$  and  $p_u$ 's valid signature shares for  $C_v$ , respectively.*

**Proof of Observation 63.5.** Let  $p_u \in F_{\text{recv}}$  upon receiving a valid  $\text{FORWARD}(C_v, \star, sigs)$  message at line 26. By the condition of line 33,  $p_u$  has either previously sent a FORWARD message at line 25 or it will send such a message at line 36. In both cases, Observation 63.3 applies and

guarantees that this message contains  $C_v$  and both  $p_u$ 's and  $p_s$ 's valid signature shares.

□<sub>Observation 63.5</sub>

Note that  $F_{\text{recv}}$  is defined over an entire execution of Algorithm 9. Observation 63.5 therefore states that any correct process  $p_u$  that receives a valid FORWARD message *at some point of its execution* also broadcasts a matching FORWARD message *at some point of its execution*. The two events (receiving and sending a FORWARD message) might, however, occur in any order. For instance,  $p_u$  might first receive a SEND message from  $p_s$  at line 20, disseminate a FORWARD message as a result at line 25, and later on possibly receive a FORWARD message from some other process at line 26. Alternatively,  $p_u$  might first receive a FORWARD message at line 26, and disseminate its own FORWARD message at line 36 as a result. In this second case,  $p_u$  might also eventually receive a SEND message from  $p_s$  (at line 20). If this happens,  $p_u$  will disseminate a second FORWARD message at line 25. A correct process, however, never disseminates more than two FORWARD messages (at lines 25 and 36).

**Observation 63.6.** *Any broadcast of  $\text{FORWARD}(C_v, \text{fragtuple}, \{\text{sig}_s, \text{sig}_u\})$  by a correct process  $p_u \in F_{\text{recv}}$  arrives to at least  $c - d$  correct processes that are each, eventually, in  $F_{\text{recv}}$ .*

**Proof of Observation 63.6.** At least  $c - d$  correct processes eventually receive each broadcast of a FORWARD message by a correct process  $p_u$  by definition of the message adversary. By Observation 63.3 the FORWARD message contains  $C_v$ , by Lemma 58 it is valid. Thus, each of its correct recipients, which are at least  $c - d$ , belong to  $F_{\text{recv}}$ , by definition. □<sub>Observation 63.6</sub>

Because FORWARD messages are disseminated at line 36, the reception and sending of FORWARD messages by correct processes will induce a “chain reaction” until a correct process is reached that has already disseminated a FORWARD message. This “chain reaction” mechanism is the intuitive reason why some correct process will eventually receive enough distinct FORWARD messages to trigger an mbrb-delivery, as captured by the following observation.

**Observation 63.7.** *There exists a correct process  $p_j$  that receives  $\text{FORWARD}(C_v, \star, \text{sig}_s = \{\text{sig}_s, \text{sig}_u\})$  messages from at least  $c - d$  distinct correct processes  $p_u$ , where  $\text{sig}_s = (\text{ts\_sign\_share}_s(C_v), s)$  and  $\text{sig}_u = (\text{ts\_sign\_share}_u(C_v), u)$  are  $p_s$  and  $p_u$ 's valid signature shares for  $C_v$ , respectively, and the FORWARD message is the last message sent by  $p_u$ .*

**Proof of Observation 63.7.** By Observation 63.5, any processes  $p_u \in F_{\text{recv}}$  broadcasts at least one message  $\text{FORWARD}(C_v, \star, \text{sig}_s = \{\text{sig}_s, \text{sig}_u\})$ , that includes  $p_u$ 's valid signature share for  $C_v$ ,  $\text{sig}_u = (\text{ts\_sign\_share}_u(C_v), u)$ . Consider all the FORWARD messages sent by processes in  $F_{\text{recv}}$  during the *last time* they perform such a broadcast. By Observation 63.6, there are  $|F_{\text{recv}}|$

---

senders,  $p_u \in F_{\text{recv}}$ , such that each of  $p_u$ 's last broadcast of a **FORWARD** message is guaranteed to be delivered to at least  $c - d$  correct processes  $p_x$ , such that eventually  $p_x \in F_{\text{recv}}$ . Thus, at least  $|F_{\text{recv}}|(c - d)$  such messages are received by processes in  $F_{\text{recv}}$ , overall. By Observation 63.4,  $F_{\text{recv}}$  contains at least one process. We can, therefore, apply the pigeonhole principle, where  $F_{\text{recv}}$  are the holes and the above  $|F_{\text{recv}}|(c - d)$  messages are the pigeons, and observe that there exists a process  $p_j \in F_{\text{recv}}$  that will receive at least  $|F_{\text{recv}}|(c - d)/|F_{\text{recv}}|$  such messages. Since we limit the discussion to a *single*, *i.e.*, the last, broadcast performed by each process in  $F_{\text{recv}}$ , no process in  $F_{\text{recv}}$  receives two of the above messages that were originated by the same process in  $F_{\text{recv}}$ . Therefore, we deduce that  $p_j$  has received messages of the form  $\text{FORWARD}(C_v, \star, \text{sig}_u)$  from at least  $c - d$  *distinct* correct processes  $p_u$  and the **FORWARD** message is the *last* message sent by  $p_u$ . □<sub>Observation 63.7</sub>

**Observation 63.8.** *At least one correct process mbrb-delivers  $v$  from  $p_s$ .*

**Proof of Observation 63.8.** By Observation 63.7, there is a correct process  $p_j$  that receives messages of the form  $\text{FORWARD}(C_v, \star, \text{sig}_u)$  from at least  $c - d$  distinct correct processes  $p_u$ , such that these **FORWARD** messages are the *last* message sent by each  $p_u$ . Let us denote by  $U$  the set of such processes  $p_u$ , hence,  $|U| \geq c - d$ .

Still by Observation 63.7,  $p_j$  receives a valid signature share  $\text{sig}_u = (\text{ts\_sign\_share}_u(C_v), u)$  from each process  $p_u \in U$ . It thus receives at least  $c - d$  *distinct* signature shares for  $C_v$ . c-MBRB-Assumption says  $3t + 2d < n$ , and thus,  $n + 3t + 2d < 2n$  and  $n + t < 2n - 2t - 2d$ . Since  $n - t \leq c$ , we have  $\frac{n+t}{2} < n - t - d \leq c - d$ . Thus,  $p_j$  receives more than  $\frac{n+t}{2}$  valid distinct signature shares for  $C_v$ .

Let us now consider the set of correct processes  $S$  that receive the initial **SEND** messages disseminated by  $p_s$  at line 19. Any process  $p_x \in S$  receives through the **SEND** message its allocated fragment  $(\tilde{v}_x, \pi_x, x)$  from  $p_s$ . By definition of the message adversary, the **SEND** messages disseminated at line 19 are received by at least  $c - d$  correct processes, therefore  $|S| \geq c - d$ . Furthermore, all processes in  $S$  broadcast a **FORWARD** message at line 25, and this will be their *last* **FORWARD** message, due to the condition at line 33. By the above reasoning, this **FORWARD** message will contain their value fragment, that is, it will be of the form  $\text{FORWARD}(C_v, (\tilde{v}_x, \pi_x, x), \text{sig}_u)$ . By Lemma 58, they are all valid.

By definition of  $S$  and  $U$ , both these sets contain only correct processes, thus,  $|S \cup U| \leq c$ . As a result,  $|S \cap U| = |S| + |U| - |S \cup U| \geq 2(c - d) - c = c - 2d$ . The last **FORWARD** messages broadcast by processes in  $S \cap U$  are received by  $p_j$  by the definition of  $U$ . As argued above about processes in  $S$  (and thus applying to processes in  $S \cap U$ ), **FORWARD** messages sent by a process in  $S \cap U$  contain their valid value fragment and proof of inclusion  $(\tilde{v}_x, \pi_x, x)$ . It follows

that  $p_j$  accumulates at least  $c - 2d$  distinct such value fragments with their (valid) proof of inclusion. By c-MBRB-Assumption,  $c - 2d \geq k$ .

To conclude the proof, note that we have shown that  $p_j$  eventually receives more than  $\frac{n+t}{2}$  valid distinct signature shares for  $C_v$ , and additionally, that  $p_j$  accumulates at least  $k$  valid value fragments with their proof of inclusion. At this point, the condition of line 37 becomes true for  $p_j$ . Because the commitment is collision-resistant (Section 6.2.2), once  $C_v$  is fixed, we can assume that, except with negligible probability, all the value fragments that  $p_j$  has received correspond to the fragments computed by  $p_s$  at line 17. By the parameters of the ECC we use, it can recover the value  $v$  from any  $k$  or more (correct) fragments generated by  $p_s$ , where missing fragments are considered as erasures. Therefore, the value  $v_j$  reconstructed at line 38 by  $p_j$  is the value initially mbrb-broadcast by  $p_s$ . As a result  $v_j = v$ , and  $p_j$  mbrb-delivers  $v$  at line 43. □<sub>Observation 63.8</sub>

□

**Lemma 64** (MBRB-Global-delivery). *If a correct process  $p_i$  mbrb-delivers a value  $v$ , then at least  $\ell_{MBRB} = c - d \left( \frac{1}{1 - \frac{k-1}{c-d}} \right)$  correct processes mbrb-deliver  $v$ .*

**Proof.** Suppose a correct process  $p_i$  mbrb-delivers  $v$  (line 43). Let us denote by  $C_v$  the commitment returned by `compute_frag_vec_commit(v)`. The proof follows a counting argument on the BUNDLE messages disseminated by correct processes at lines 42 and 50. In the following, if a correct process disseminates a BUNDLE message both at line 42 and line 50, we only consider the one from line 42.

**Observation 64.1.** *All valid BUNDLE messages exchanged during the execution of Algorithm 9 contain  $C_v$ , the vector commitment of value  $v$ , where  $v$  is the message mbrb-delivered by  $p_i$ .*

**Proof of Observation 64.1.** Consider a valid message  $\text{BUNDLE}(C', \text{fragtuple}'_j, \text{fragtuple}'_i, \Sigma')$ . By definition of a valid BUNDLE message,  $\Sigma'$  aggregates a set  $\text{sigs}'$  of  $\tau = \lfloor \frac{n+t}{2} \rfloor + 1$  valid signature shares for  $C'$ . Similarly, when  $p_i$  mbrb-delivers  $v$  at line 43, it has a threshold signature  $\Sigma_v$  which aggregates a set  $\text{sigs}_v$  of  $\tau = \lfloor \frac{n+t}{2} \rfloor + 1$  valid signature shares for  $C_v$ . By a reasoning identical to that of Lemma 62, these two inequalities imply that  $\text{sigs}' \cap \text{sigs}_v$  contains the signature shares from at least one common correct process,  $p_u$ . As signature shares cannot be forged,  $p_u$  has issued signature shares for both  $C'$  and  $C_v$ , and by Lemma 59,  $C' = C_v$ . To complete the proof, note that by the definition of a valid BUNDLE message, the threshold

---

signature it contains is valid with respect to the commitment it carries. Hence, all valid **BUNDLE** messages must contain the commitment  $C_v$  of the value  $v$  that matches the threshold signature  $\Sigma'$  they contain. □<sub>Observation 64.1</sub>

Let  $B_{\text{send}}$  be the set of correct processes that disseminate at least one **BUNDLE** message during their execution. Similarly, let  $B_{\text{recv}}$  be the set of correct processes that receive at least one valid **BUNDLE** message from a correct process during their execution. The following holds.

**Observation 64.2.**  $c \geq |B_{\text{recv}}| \geq c - d$  and  $c \geq |B_{\text{send}}| \geq c - d$ .

**Proof of Observation 64.2.** Since  $B_{\text{send}}$  and  $B_{\text{recv}}$  contain only correct processes, trivially  $c \geq |B_{\text{send}}|$  and  $c \geq |B_{\text{recv}}|$ . Since  $p_i$  *mbrb-delivers*  $v$  at line 43, it must have disseminated **BUNDLE**( $C_v, (\tilde{v}'_i, \pi'_i, i), (\tilde{v}'_j, \pi'_j, j), \Sigma_C$ ) messages at line 42. The **BUNDLE** messages sent by  $p_i$  eventually reach at least  $c - d$  correct processes, as the message adversary can remove at most  $d$  of these **BUNDLE** messages. By Lemma 58, these **BUNDLE** messages are valid. Hence,  $|B_{\text{recv}}| \geq c - d > 0$  proves the lemma's first part.

The processes in  $B_{\text{recv}}$  (which are correct) execute lines 44 and 45, and reach line 47. Because  $p_i$  has included a non- $\perp$  second fragment in all its **BUNDLE** message, any of the  $(c - d)$  processes of  $B_{\text{recv}}$  that receive one of  $p_i$ 's **BUNDLE** messages and has not already sent a **BUNDLE** message passes the condition at line 47. Each such process then disseminates a (valid) **BUNDLE** message at line 50. This behavior yields  $|B_{\text{send}}| \geq c - d$ . □<sub>Observation 64.2</sub>

Let  $B_{k,\text{recv}}$  be the set of correct processes that receive **BUNDLE** messages from at least  $k$  distinct correct processes.

**Observation 64.3.**  $|B_{k,\text{recv}}| \times |B_{\text{send}}| + (k - 1)(|B_{\text{recv}}| - |B_{k,\text{recv}}|) \geq |B_{\text{send}}|(c - d)$ .

**Proof of Observation 64.3.** Let us denote by  $\#\text{BUNDLE}$  the overall number of valid **BUNDLE** messages received by correct processes from distinct correct senders. More specifically, in the case when a correct process disseminates **BUNDLE** messages both at line 42 and line 50, we only consider the *last* **BUNDLE** message, *i.e.*, the one of line 42. We know that each  $p \in B_{\text{send}}$  sends a **BUNDLE** message, which is valid by Lemma 58. As the message adversary may drop up to  $d$  out of the  $n$  messages of this **comm**, we are guaranteed that at least  $c - d$  correct processes receive  $p$ 's **BUNDLE** message. This immediately implies that

$$\#\text{BUNDLE} \geq |B_{\text{send}}|(c - d). \quad (41)$$

As illustrated in Figure 15 (page 97), the processes in  $B_{k,\text{recv}}$  may receive up to  $|B_{\text{send}}|$  valid **BUNDLE** messages from distinct correct senders (one from each sender in  $B_{\text{send}}$ ), for a maximum

of  $|B_{k,\text{recv}}| \times |B_{\text{send}}|$  BUNDLE messages overall. The remaining processes of  $B_{\text{recv}} \setminus B_{k,\text{recv}}$  may each receive up to  $k - 1$  valid BUNDLE messages from distinct correct senders, by definition of  $B_{k,\text{recv}}$ . As  $B_{k,\text{recv}} \subseteq B_{\text{recv}}$  by definition,  $|B_{\text{recv}} \setminus B_{k,\text{recv}}| = |B_{\text{recv}}| - |B_{k,\text{recv}}|$ , and the processes of  $B_{\text{recv}} \setminus B_{k,\text{recv}}$  accounts for up to  $(k - 1)(|B_{\text{recv}}| - |B_{k,\text{recv}}|)$  valid BUNDLE messages overall. As the BUNDLE messages counted by  $\#\text{BUNDLE}$  are received either by correct processes in  $B_{k,\text{recv}}$  or in  $B_{k,\text{recv}} \setminus B_{\text{recv}}$ , these observations lead to

$$|B_{k,\text{recv}}| \times |B_{\text{send}}| + (k - 1)(|B_{\text{recv}}| - |B_{k,\text{recv}}|) \geq \#\text{BUNDLE}. \quad (42)$$

Combining (41) and (42) yields the desired inequality.  $\square$  Observation 64.3

**Observation 64.4.**  $|B_{k,\text{recv}}| \geq c - d \left( \frac{1}{1 - \frac{k-1}{c-d}} \right)$ .

**Proof of Observation 64.4.** Rearranging the terms of Observation 64.3, and recalling that  $|B_{\text{recv}}| \geq c$  and  $k \geq 1$ , we get

$$|B_{k,\text{recv}}| \times (|B_{\text{send}}| - k + 1) \geq |B_{\text{send}}|(c - d) - |B_{\text{recv}}|(k - 1) \geq |B_{\text{send}}|(c - d) - c(k - 1).$$

By Observation 64.2 and c-MBRB-Assumption,  $|B_{\text{send}}| \geq c - d \geq c - 2d \geq k$ , therefore  $|B_{\text{send}}| - k + 1 > 0$ , and the previous equation can be transformed into

$$|B_{k,\text{recv}}| \geq \frac{|B_{\text{send}}|(c - d) - c(k - 1)}{|B_{\text{send}}| - k + 1}. \quad (43)$$

Note that the right-hand side of (43) is a monotone increasing function in  $|B_{\text{send}}|$  when  $|B_{\text{send}}| > k - 1$ , as its derivative,  $\frac{d(k+1)}{(|B_{\text{send}}| - k + 1)^2}$ , is positive. By Observation 64.2,  $|B_{\text{send}}| \in [c - d, c] \subseteq [k, c]$ . The minimum of the right-hand side of (43) is therefore obtained for  $B_{\text{send}} = c - d$ , yielding  $|B_{k,\text{recv}}| \geq \frac{(c-d)^2 - c(k-1)}{(c-d) - k + 1} = c - d \left( \frac{1}{1 - \frac{k-1}{c-d}} \right)$ .  $\square$  Observation 64.4

**Observation 64.5.** All processes in  $B_{k,\text{recv}}$  mbrb-deliver  $v$ .

**Proof of Observation 64.5.** Consider  $p_u \in B_{k,\text{recv}}$ . By the definition of  $B_{k,\text{recv}}$ , the process  $p_u$  receives  $k$  valid BUNDLE messages from  $k$  distinct correct processes. Let us denote by  $\text{BUNDLE}(C_x, (\tilde{v}_x, \pi_x, x), \star, \Sigma_x)$  these  $k$  messages with  $x \in [k]$ . By Observation 64.1, for all  $x \in [k]$ ,  $C_x = C_v$ . In addition,  $p_u$  saves each received threshold signature  $\Sigma_x$ , which is valid for  $C_v$ . Because the messages are valid, so are the proofs of inclusions  $\pi_x$ , and as we have assumed that the commitments are collision-resistant,  $C_x = C_v$  implies that the received fragments  $\tilde{v}_x$  all belong to the set of fragments computed by  $p_s$  at line 17 for  $v$ . As the BUNDLE messages



---

were received from  $k$  distinct correct processes, the process  $p_u$  receives at least  $k$  distinct valid fragments for  $v$  during its execution. If  $p_u$  has not mbrb-delivered any value yet, the condition at line 37 eventually becomes true for  $C_v$ , and  $p_u$  reconstructs  $v$  at line 38, since it possesses at least  $k$  (correct) value fragments, which are sufficient for the correct recovery of  $v$  by our choice of ECC. Then,  $p_u$  mbrb-delivers  $v$  at line 43. On the other hand, if  $p_u$  has already mbrb-delivered some value  $v'$ , then Lemma 62 (MBRB-No-duplication) implies  $v' = v$ , since  $p_i$  is known to have mbrb-delivered  $v$ . Therefore, in all possible cases,  $p_u$  mbrb-delivers  $v$ .  $\square$  Observation 64.5

Lemma 64 (MBRB-Global-delivery) follows from Observations 64.4 and 64.5, and the fact that all processes in  $B_{k, \text{recv}}$  are correct.  $\square$

# BIBLIOGRAPHY

---

- [1] Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. 2023. Communication complexity of Byzantine agreement, revisited. *Distributed Comput.* 36, 1 (2023), 3–28. <https://doi.org/10.1007/S00446-022-00428-8>
- [2] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. 2021. Good-case latency of Byzantine broadcast: a complete categorization. In *Proc. 40th ACM Symposium on Principles of Distributed Computing (PODC'21)*, 2021. ACM, 331–341. <https://doi.org/10.1145/3465084.3467899>
- [3] Yehuda Afek and Eli Gafni. 2013. Asynchrony from synchrony. In *Proc. 14th Int'l Conference on Distributed Computing and Networking (ICDCN'13) (Lecture Notes in Computer Science)*, 2013. Springer, 225–239. [https://doi.org/10.1007/978-3-642-35668-1\\_16](https://doi.org/10.1007/978-3-642-35668-1_16)
- [4] Timothé Albouy, Emmanuelle Anceaume, Davide Frey, Mathieu Gestin, Arthur Rauch, Michel Raynal, and François Taïani. 2024. Asynchronous BFT asset transfer: quasi-anonymous, light, and consensus-free. *CoRR* (2024). <https://doi.org/10.48550/ARXIV.2405.18072>
- [5] Timothé Albouy, Antonio Fernández Anta, Chryssis Georgiou, Mathieu Gestin, Nicolas Nicolaou, and Junlang Wang. 2024. AMECOS: A modular event-based framework for concurrent object specification. In *Proc. 28th Int'l Conference on Principles of Distributed Systems (OPODIS'24) (LIPIcs)*, 2024. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, ??????. <https://doi.org/??????>
- [6] Timothé Albouy, Antonio Fernández Anta, Chryssis Georgiou, Mathieu Gestin, Nicolas Nicolaou, and Junlang Wang. 2024. AMECOS: A modular event-based framework for concurrent object specification. *CoRR* (2024). <https://doi.org/10.48550/ARXIV.2405.10057>
- [7] Timothé Albouy, Davide Frey, Ran Gelles, Carmit Hazay, Michel Raynal, Elad Michael Schiller, François Taïani, and Vassilis Zikas. 2024. Near-optimal communication Byzantine reliable broadcast under a message adversary. In *Proc. 28th Int'l Conference on Principles of Distributed Systems (OPODIS'24) (LIPIcs)*, 2024. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, ??????. <https://doi.org/??????>
- [8] Timothé Albouy, Davide Frey, Ran Gelles, Carmit Hazay, Michel Raynal, Elad Michael Schiller, François Taïani, and Vassilis Zikas. 2024. Brief announcement: Towards optimal communication Byzantine reliable broadcast under a message adversary. In *Proc. 38th*

- 
- Int'l Symposium on Distributed Computing (DISC'24) (LIPICs)*, 2024. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 1–7. <https://doi.org/10.4230/LIPICs.DISC.2024.41>
- [9] Timothé Albouy, Davide Frey, Ran Gelles, Carmit Hazay, Michel Raynal, Elad Michael Schiller, François Taïani, and Vassilis Zikas. 2024. Near-optimal communication Byzantine reliable broadcast under a message adversary. *CoRR* (2024). <https://doi.org/10.48550/ARXIV.2312.16253>
- [10] Timothé Albouy, Davide Frey, Mathieu Gestin, Michel Raynal, and François Taïani. 2023. Context adaptive cooperation. *CoRR* (2023). <https://doi.org/10.48550/ARXIV.2311.08776>
- [11] Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani. 2021. Byzantine-tolerant reliable broadcast in the presence of silent churn. In *Proc. 23rd Int'l Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'21) (Lecture Notes in Computer Science)*, 2021. Springer, 21–33. [https://doi.org/10.1007/978-3-030-91081-5\\_2](https://doi.org/10.1007/978-3-030-91081-5_2)
- [12] Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani. 2022. A modular approach to construct signature-free BRB algorithms under a message adversary. In *Proc. 26th Int'l Conference on Principles of Distributed Systems (OPODIS'22) (LIPICs)*, 2022. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 1–23. <https://doi.org/10.4230/LIPICs.OPODIS.2022.26>
- [13] Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani. 2022. A modular approach to construct signature-free BRB algorithms under a message adversary. 2022. Retrieved from <https://arxiv.org/abs/2204.13388>
- [14] Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani. 2023. Asynchronous Byzantine reliable broadcast with a message adversary. *Theor. Comput. Sci.* 978, (2023). <https://doi.org/10.1016/J.TCS.2023.114110>
- [15] Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani. 2024. Good-case early-stopping latency of synchronous Byzantine reliable broadcast: the deterministic case. *Distributed Computing* 37, (2024), 1–23. <https://doi.org/10.1007/s00446-024-00464-6>
- [16] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. 2022. Balanced Byzantine reliable broadcast with near-optimal communication and improved computation. In *Proc. 41st ACM Symposium on Principles of Distributed Computing (PODC'22)*, 2022. ACM, 399–417. <https://doi.org/10.1145/3519270.3538475>

- [17] Karine Altisen, Stéphane Devismes, Swan Dubois, and Franck Petit. 2019. *Introduction to Distributed Self-Stabilizing Algorithms*. Morgan & Claypool Publishers. <https://doi.org/10.2200/S00908ED1V01Y201903DCT015>
- [18] Hagit Attiya and Armando Castañeda. 2013. A non-topological proof for the impossibility of  $k$ -set agreement. *Theor. Comput. Sci.* 512, (2013), 41–48. <https://doi.org/10.1016/J.TCS.2012.09.012>
- [19] Hagit Attiya and Faith Ellen. 2014. *Impossibility Results for Distributed Computing*. Morgan & Claypool Publishers. <https://doi.org/10.2200/S00551ED1V01Y201311DCT012>
- [20] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. 1995. Sharing memory robustly in message-passing systems. *J. ACM* 42, 1 (1995), 124–142. <https://doi.org/10.1145/200836.200869>
- [21] Alex Auvolat, Davide Frey, Michel Raynal, and François Taïani. 2020. Money transfer made simple: a specification, a generic algorithm, and its proof. *Bull. EATCS* 132, (2020).
- [22] Alex Auvolat, Michel Raynal, and François Taïani. 2019. Byzantine-tolerant set-constrained delivery broadcast. In *Proc. 23rd Int'l Conference on Principles of Distributed Systems (OPODIS'19) (LIPIcs)*, 2019. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 1–23. <https://doi.org/10.4230/LIPICS.OPODIS.2019.6>
- [23] Mathieu Baudet, George Danezis, and Alberto Sonnino. 2020. FastPay: High-performance Byzantine fault tolerant settlement. In *Proc. 2nd ACM Conference on Advances in Financial Technologies (AFT'20)*, 2020. ACM, 163–177. <https://doi.org/10.1145/3419614.3423249>
- [24] Mathieu Baudet, Alberto Sonnino, Mahimna Kelkar, and George Danezis. 2023. Zef: Low-latency, scalable, private payments. In *Proc. 22nd Workshop on Privacy in the Electronic Society (WPES'23)*, 2023. ACM, 1–16. <https://doi.org/10.1145/3603216.3624952>
- [25] Michael Ben-Or. 1983. Another advantage of free choice: Completely asynchronous agreement Protocols (Extended Abstract). In *Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC'83)*, 1983. ACM, 27–30. <https://doi.org/10.1145/800221.806707>
- [26] Piotr Berman, Krzysztof Diks, and Andrzej Pelc. 1997. Reliable broadcasting in logarithmic time with Byzantine link failures. *J. Algorithms* 22, 2 (1997), 199–211. <https://doi.org/10.1006/JAGM.1996.0810>
- [27] Martin Biely, Peter Robinson, and Ulrich Schmid. 2011. Easy impossibility proofs for  $k$ -set agreement in message passing systems. In *Proc. 15th Int'l Conference on Principles of*

- 
- Distributed Systems (OPODIS'11) (Lecture Notes in Computer Science)*, 2011. Springer, 299–312. [https://doi.org/10.1007/978-3-642-25873-2\\_21](https://doi.org/10.1007/978-3-642-25873-2_21)
- [28] Martin Biely, Ulrich Schmid, and Bettina Weiss. 2011. Synchronous consensus under hybrid process and link failures. *Theor. Comput. Sci.* 412, 40 (2011), 5602–5630. <https://doi.org/10.1016/J.TCS.2010.09.032>
- [29] Silvia Bonomi, Jérémie Decouchant, Giovanni Farina, Vincent Rahli, and Sébastien Tixeuil. 2021. Practical Byzantine reliable broadcast on partially connected networks. In *Proc. 41st IEEE Int'l Conference on Distributed Computing Systems (ICDCS'21)*, 2021. IEEE, 506–516. <https://doi.org/10.1109/ICDCS51616.2021.00055>
- [30] Elizabeth Borowsky and Eli Gafni. 1993. Generalized FLP impossibility result for  $t$ -resilient asynchronous computations. In *Proc. 25th ACM Symposium on Theory of Computing (STOC'93)*, 1993. ACM, 91–100. <https://doi.org/10.1145/167088.167119>
- [31] Gabriel Bracha and Sam Toueg. 1985. Asynchronous consensus and broadcast protocols. *J. ACM* 32, 4 (1985), 824–840. <https://doi.org/10.1145/4221.214134>
- [32] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. *Inf. Comput.* 75, 2 (1987), 130–143. [https://doi.org/10.1016/0890-5401\(87\)90054-X](https://doi.org/10.1016/0890-5401(87)90054-X)
- [33] Christian Cachin and Jonathan A. Poritz. 2002. Secure intrusion-tolerant replication on the Internet. In *Proc. 3rd Int'l Conference on Dependable Systems and Networks (DSN'2002)*, 2002. IEEE Computer Society, 167–176. <https://doi.org/10.1109/DSN.2002.1028897>
- [34] Christian Cachin and Stefano Tessaro. 2005. Asynchronous verifiable information dispersal. In *Proc. 19th Int'l Conference on Distributed Computing (DISC'05) (Lecture Notes in Computer Science)*, 2005. Springer, 503–504. [https://doi.org/10.1007/11561927\\_42](https://doi.org/10.1007/11561927_42)
- [35] Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. 2011. *Introduction to Reliable and Secure Distributed Programming* (2nd ed.). Springer. <https://doi.org/10.1007/978-3-642-15260-3>
- [36] Ran Canetti and Tal Rabin. 1993. Fast asynchronous Byzantine agreement with optimal resilience. In *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC'93)*, 1993. ACM, 42–51. <https://doi.org/10.1145/167088.167105>
- [37] Armando Castañeda, Sergio Rajsbaum, and Michel Raynal. 2023. A linearizability-based hierarchy for concurrent specifications. *Commun. ACM* 66, 1 (2023), 86–97. <https://doi.org/10.1145/3546826>

- [38] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine fault tolerance. In *Proc. 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI'99)*, 1999. USENIX Association, 173–186.
- [39] Dario Catalano and Dario Fiore. 2013. Vector commitments and their applications. In *Proc. 16th Int'l Conference on Practice and Theory in Public-Key Cryptography (PKC'13) (Lecture Notes in Computer Science)*, 2013. Springer, 55–72. Retrieved from [https://doi.org/10.1007/978-3-642-36362-7\\_5](https://doi.org/10.1007/978-3-642-36362-7_5)
- [40] Keren Censor-Hillel, Shir Cohen, Ran Gelles, and Gal Sela. 2023. Distributed computations in fully-defective networks. *Distributed Comput.* 36, 4 (2023), 501–528. <https://doi.org/10.1007/S00446-023-00452-2>
- [41] Keren Censor-Hillel, Ran Gelles, and Bernhard Haeupler. 2019. Making asynchronous distributed computations robust to noise. *Distributed Comput.* 32, 5 (2019), 405–421. <https://doi.org/10.1007/S00446-018-0343-5>
- [42] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. 1992. The weakest failure detector for solving consensus. In *Proc. 11th Annual ACM Symposium on Principles of Distributed Computing (PODC'92)*, 1992. ACM, 147–158. <https://doi.org/10.1145/135419.135451>
- [43] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. 1996. The Weakest Failure Detector for Solving Consensus. *J. ACM* 43, 4 (1996), 685–722. <https://doi.org/10.1145/234533.234549>
- [44] Bernadette Charron-Bost and André Schiper. 2009. The heard-of model: computing in distributed systems with benign faults. *Distributed Comput.* 22, 1 (2009), 49–71. <https://doi.org/10.1007/S00446-009-0084-6>
- [45] Soma Chaudhuri. 1990. Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. In *Proc. 9th ACM Symposium on Principles of Distributed Computing (PODC'90)*, 1990. ACM, 311–324. <https://doi.org/10.1145/93385.93431>
- [46] Anamika Chauhan, Om Prakash Malviya, Madhav Verma, and Tejinder Singh Mor. 2018. Blockchain and scalability. In *Proc. 4th IEEE Int'l Conference on Software Quality, Reliability and Security Companion (QRS'18)*, 2018. IEEE, 122–128. <https://doi.org/10.1109/QRS-C.2018.00034>
- [47] Xusheng Chen, Haoze Song, Jianyu Jiang, Chaoyi Ruan, Cheng Li, Sen Wang, Gong Zhang, Reynold Cheng, and Heming Cui. 2021. Achieving low tail-latency and high scalability for serializable transactions in edge computing. In *Proc. 16th European Confer-*

- 
- ence on Computer Systems (EuroSys'21), 2021. ACM, 210–227. <https://doi.org/10.1145/3447786.3456238>
- [48] Noam Chomsky and Marcel-Paul Schützenberger. 1963. The algebraic theory of context-free languages. *Computer programming and formal systems* 26, (1963), 118–161.
- [49] Noam Chomsky. 1956. Three models for the description of language. *IRE Trans. Inf. Theory* 2, 3 (1956), 113–124. <https://doi.org/10.1109/TIT.1956.1056813>
- [50] Noam Chomsky. 1959. On certain formal properties of grammars. *Inf. Control.* 2, 2 (1959), 137–167. [https://doi.org/10.1016/S0019-9958\(59\)90362-6](https://doi.org/10.1016/S0019-9958(59)90362-6)
- [51] Alonzo Church. 1936. An unsolvable problem of elementary number theory. *American Journal of Mathematics* 58, 2 (1936), 345–363.
- [52] Alonzo Church. 1936. A note on the Entscheidungsproblem. *J. Symb. Log.* 1, 1 (1936), 40–41. <https://doi.org/10.2307/2269326>
- [53] Bram Cohen. 2002. BitTorrent protocol 1.0. Retrieved from [https://www.bittorrent.org/beps/bep\\_0003.html](https://www.bittorrent.org/beps/bep_0003.html)
- [54] Shir Cohen and Idit Keidar. 2021. Tame the wild with Byzantine linearizability: reliable broadcast, snapshots, and asset transfer. In *Proc. 35th Int'l Symposium on Distributed Computing (DISC'21) (LIPICs)*, 2021. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 1–18. <https://doi.org/10.4230/LIPICs.DISC.2021.18>
- [55] Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, and Athanasios Xygkis. 2020. Online payments by merely broadcasting messages. In *Proc. 50th Annual IEEE/IFIP Int'l Conference on Dependable Systems and Networks (DSN'20)*, 2020. IEEE, 26–38. <https://doi.org/10.1109/DSN48063.2020.00023>
- [56] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *Proc 17th European Conference on Computer Systems (EuroSys'22)*, 2022. ACM, 34–50. <https://doi.org/10.1145/3492321.3519594>
- [57] Sourav Das, Zhuolun Xiang, and Ling Ren. 2021. Asynchronous data dissemination and its applications. In *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS'21)*, 2021. ACM, 2705–2721. <https://doi.org/10.1145/3460120.3484808>
- [58] Pallab Dasgupta. 1998. Agreement under faulty interfaces. *Inf. Process. Lett.* 65, 3 (1998), 125–129. [https://doi.org/10.1016/S0020-0190\(97\)00202-0](https://doi.org/10.1016/S0020-0190(97)00202-0)

- [59] Diego Didona and Willy Zwaenepoel. 2019. Size-aware sharding for improving tail latencies in in-memory key-value stores. In *Proc. 16th USENIX Symposium on Networked Systems Design and Implementation, (NSDI'19)*, 2019. USENIX Association, 79–94. Retrieved from <https://www.usenix.org/conference/nsdi19/presentation/didona>
- [60] Edsger W. Dijkstra. 1965. Solution of a problem in concurrent programming control. *Commun. ACM* 8, 9 (1965), 569–570. <https://doi.org/10.1145/365559.365617>
- [61] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The second-generation onion router. In *Proc. 13th USENIX Security Symposium*, 2004. USENIX, 303–320.
- [62] Danny Dolev and Rüdiger Reischuk. 1985. Bounds on information exchange for Byzantine agreement. *J. ACM* 32, 1 (1985), 191–204. <https://doi.org/10.1145/2455.214112>
- [63] Danny Dolev and H. Raymond Strong. 1983. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput.* 12, 4 (1983), 656–666. <https://doi.org/10.1137/0212045>
- [64] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. 1986. Reaching approximate agreement in the presence of faults. *J. ACM* 33, 3 (1986), 499–516. <https://doi.org/10.1145/5925.5931>
- [65] Danny Dolev. 1982. The Byzantine generals strike again. *J. Algorithms* 3, 1 (1982), 14–30. [https://doi.org/10.1016/0196-6774\(82\)90004-9](https://doi.org/10.1016/0196-6774(82)90004-9)
- [66] John R. Douceur. 2002. The Sybil attack. In *Proc. 1st Int'l Workshop on Peer-to-Peer Systems (IPTPS'02) (Lecture Notes in Computer Science)*, 2002. Springer, 251–260. [https://doi.org/10.1007/3-540-45748-8\\_24](https://doi.org/10.1007/3-540-45748-8_24)
- [67] Sisi Duan, Michael K. Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT made practical. In *Proc. 25th ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*, 2018. ACM, 2028–2041. <https://doi.org/10.1145/3243734.3243812>
- [68] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. 1988. Consensus in the presence of partial synchrony. *J. ACM* 35, 2 (1988), 288–323. <https://doi.org/10.1145/42282.42283>
- [69] Patrick Th. Eugster, Rachid Guerraoui, Sidath B. Handurukande, Petr Kouznetsov, and Anne-Marie Kermarrec. 2003. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.* 21, 4 (2003), 341–374. <https://doi.org/10.1145/945506.945507>
- [70] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. 1985. Impossibility of distributed consensus with one faulty process. *J. ACM* 32, 2 (1985), 374–382. <https://doi.org/10.1145/3149.214121>



- 
- [71] Matthias Fitzi and Jesper Buus Nielsen. 2009. On the number of synchronous rounds sufficient for authenticated Byzantine agreement. In *Proc. 23rd Int'l Symposium on Distributed Computing (DISC'09) (Lecture Notes in Computer Science)*, 2009. Springer, 449–463. [https://doi.org/10.1007/978-3-642-04355-0\\_46](https://doi.org/10.1007/978-3-642-04355-0_46)
- [72] Davide Frey, Lucie Guillou, Michel Raynal, and François Taïani. 2024. Process-commutative distributed objects: From cryptocurrencies to Byzantine-Fault-Tolerant CRDTs. *Theor. Comput. Sci.* 1017, (2024), 114794–114795. <https://doi.org/10.1016/J.TCS.2024.114794>
- [73] Eli Gafni and Giuliano Losa. 2023. Invited paper: Time is not a healer, but it sure makes hindsight 20:20. In *Proc. 25th Int'l Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'23) (Lecture Notes in Computer Science)*, 2023. Springer, 62–74. [https://doi.org/10.1007/978-3-031-44274-2\\_6](https://doi.org/10.1007/978-3-031-44274-2_6)
- [74] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nikolai Zeldovich. 2017. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *Proc. 26th Symposium on Operating Systems Principles (SOSP'17)*, 2017. ACM, 51–68. <https://doi.org/10.1145/3132747.3132757>
- [75] Seth Gilbert and Nancy A. Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News* 33, 2 (2002), 51–59. <https://doi.org/10.1145/564585.564601>
- [76] Li Gong, Patrick Lincoln, and John Rushby. 1995. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults. In *Proc. 5th Conference on Dependable Computing for Critical Applications (DEPENDABLECOMP'95) (Dependable Computing and Fault Tolerant Systems)*, 1995. IEEE, 139–158.
- [77] Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, and Andrei Tonkikh. 2020. Dynamic Byzantine reliable broadcast. In *Proc. 24th Int'l Conference on Principles of Distributed Systems (OPODIS'20) (LIPIcs)*, 2020. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 1–18. <https://doi.org/10.4230/LIPICS.OPODIS.2020.23>
- [78] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. 2019. Scalable Byzantine reliable broadcast. In *Proc. 33rd Int'l Symposium on Distributed Computing (DISC'19) (LIPIcs)*, 2019. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 1–16. <https://doi.org/10.4230/LIPICS.DISC.2019.22>
- [79] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. 2022. The consensus number of a cryptocurrency. *Distributed Comput.* 35, 1 (2022), 1–15. <https://doi.org/10.1007/S00446-021-00399-2>

- [80] Saurabh Gupta. 2016. A non-consensus based decentralized financial transaction processing model with support for efficient auditing.
- [81] Maurice Herlihy and Nir Shavit. 1999. The topological structure of asynchronous computability. *J. ACM* 46, 6 (1999), 858–923. <https://doi.org/10.1145/331524.331529>
- [82] Maurice Herlihy and Nir Shavit. 2008. *The art of multiprocessor programming*. Morgan Kaufmann.
- [83] Maurice Herlihy, Sergio Rajsbaum, and Michel Raynal. 2013. Power and limits of distributed computing shared memory models. *Theor. Comput. Sci.* 509, (2013), 3–24. <https://doi.org/10.1016/J.TCS.2013.03.002>
- [84] Maurice Herlihy. 1991. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.* 13, 1 (1991), 124–149. <https://doi.org/10.1145/114005.102808>
- [85] Martin Hirt, Ard Kastrati, and Chen-Da Liu-Zhang. 2020. Multi-threshold asynchronous reliable broadcast and consensus. In *Proc. 24th Int'l Conference on Principles of Distributed Systems (OPODIS'20) (LIPICs)*, 2020. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 1–16. <https://doi.org/10.4230/LIPICs.OPODIS.2020.6>
- [86] William M. Hoza and Leonard J. Schulman. 2016. The adversarial noise threshold for distributed protocols. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'16)*, 2016. SIAM, 240–258. <https://doi.org/10.1137/1.9781611974331.CH18>
- [87] Damien Imbs and Michel Raynal. 2016. Trading off  $t$ -resilience for efficiency in asynchronous Byzantine reliable broadcast. *Parallel Process. Lett.* 26, 4 (2016), 1–8. <https://doi.org/10.1142/S0129626416500171>
- [88] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to Modern Cryptography, Second Edition*. CRC Press. Retrieved from <https://www.crcpress.com/Introduction-to-Modern-Cryptography-Second-Edition/Katz-Lindell/p/book/9781466570269>
- [89] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. 2003. Probabilistic reliable dissemination in large-scale systems. *IEEE Trans. Parallel Distributed Syst.* 14, 3 (2003), 248–258. <https://doi.org/10.1109/TPDS.2003.1189583>
- [90] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. OmniLedger: A secure, scale-Out, decentralized ledger via sharding. In *Proc. 39th IEEE Symposium on Security and Privacy (SP'18)*, 2018. IEEE Computer Society, 583–598. <https://doi.org/10.1109/SP.2018.000-5>
- [91] Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. 2010. Distributed computation in dynamic networks. In *Proc. 42nd ACM Symposium on Theory of Computing (STOC'10)*, 2010. ACM, 513–522. <https://doi.org/10.1145/1806689.1806760>

- 
- [92] Petr Kuznetsov, Yvonne-Anne Pignolet, Pavel Ponomarev, and Andrei Tonkikh. 2023. Permissionless and asynchronous asset transfer. *Distributed Comput.* 36, 3 (2023), 349–371. <https://doi.org/10.1007/S00446-023-00449-X>
- [93] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. 1982. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (1982), 382–401. <https://doi.org/10.1145/357172.357176>
- [94] Leslie Lamport. 1974. A new solution of Dijkstra's concurrent programming problem. *Commun. ACM* 17, 8 (1974), 453–455. <https://doi.org/10.1145/361082.361093>
- [95] Leslie Lamport. 1987. distribution (email). Retrieved from <https://lamport.azurewebsites.net/pubs/distributed-system.txt>
- [96] Leslie Lamport. 1998. The part-time parliament. *ACM Trans. Comput. Syst.* 16, 2 (1998), 133–169. <https://doi.org/10.1145/279227.279229>
- [97] Michael C. Loui and Hosame H. Abu-Amara. 1987. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing research* 4, (1987), 163–183.
- [98] Nancy A. Lynch. 1996. *Distributed Algorithms*. Morgan Kaufmann.
- [99] Dahlia Malkhi and Michael K. Reiter. 1998. Byzantine quorum systems. *Distributed Comput.* 11, 4 (1998), 203–213. <https://doi.org/10.1007/S004460050050>
- [100] Petar Maymounkov and David Mazières. 2002. Kademia: A peer-to-peer information system based on the XOR metric. In *Proc. 1st Int'l Workshop on Peer-to-Peer Systems (IPTPS'02) (Lecture Notes in Computer Science)*, 2002. Springer, 53–65. [https://doi.org/10.1007/3-540-45748-8\\_5](https://doi.org/10.1007/3-540-45748-8_5)
- [101] Ralph C. Merkle. 1989. A certified digital signature. In *Proc. 9th Annual Int'l Cryptology Conference (CRYPTO'89) (Lecture Notes in Computer Science)*, 1989. Springer, 218–238. [https://doi.org/10.1007/0-387-34805-0\\_21](https://doi.org/10.1007/0-387-34805-0_21)
- [102] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proc. 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*, 2016. ACM, 31–42. <https://doi.org/10.1145/2976749.2978399>
- [103] Gordon E. Moore. 1998. Cramming more components onto integrated circuits. *Proc. IEEE* 86, 1 (1998), 82–85. <https://doi.org/10.1109/JPROC.1998.658762>
- [104] Achour Mostéfaoui, Moumen Hamouma, and Michel Raynal. 2014. Signature-free asynchronous Byzantine consensus with  $t < n/3$  and  $O(n^2)$  messages. In *Proc. 33rd ACM Sym-*

- posium on Principles of Distributed Computing (PODC'14)*, 2014. ACM, 2–9. <https://doi.org/10.1145/2611462.2611468>
- [105] Achour Mostéfaoui, Sergio Rajsbaum, and Michel Raynal. 2003. Conditions on input vectors for consensus solvability in asynchronous distributed systems. *J. ACM* 50, 6 (2003), 922–954. <https://doi.org/10.1145/950620.950624>
- [106] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system.
- [107] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. 2020. Improved extension protocols for Byzantine broadcast and agreement. In *Proc. 34th Int'l Symposium on Distributed Computing (DISC'20) (LIPIcs)*, 2020. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 1–17. <https://doi.org/10.4230/LIPICS.DISC.2020.28>
- [108] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. 2006. Data consistency for P2P collaborative editing. In *Proc. 11th ACM Conference on Computer Supported Cooperative Work (CSCW'06)*, 2006. ACM, 259–268. <https://doi.org/10.1145/1180875.1180916>
- [109] David A. Patterson. 2006. Future of computer architecture. In *Proc. 3rd Berkeley EECS Annual Research Symposium (BEARS)*, 2006. 934–945.
- [110] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. 1980. Reaching agreement in the presence of faults. *J. ACM* 27, 2 (1980), 228–234. <https://doi.org/10.1145/322186.322188>
- [111] Andrzej Pelc. 1992. Reliable communication in networks with Byzantine link failures. *Networks* 22, 5 (1992), 441–459. <https://doi.org/10.1002/NET.3230220503>
- [112] Kenneth J. Perry and Sam Toueg. 1986. Distributed agreement in the presence of processor and communication faults. *IEEE Trans. Software Eng.* 12, 3 (1986), 477–482. <https://doi.org/10.1109/TSE.1986.6312888>
- [113] Michel Raynal and Julien Stainer. 2013. Synchrony weakened by message adversaries vs asynchrony restricted by failure detectors. In *Proc. 32nd ACM Symposium on Principles of Distributed Computing (PODC'13)*, 2013. ACM, 166–175. <https://doi.org/10.1145/2484239.2484249>
- [114] Michel Raynal. 2013. *Distributed Algorithms for Message-Passing Systems*. Springer. <https://doi.org/10.1007/978-3-642-38123-2>
- [115] Michel Raynal. 2016. Message adversaries. *Encyclopedia of Algorithms*, 1272–1276. [https://doi.org/10.1007/978-1-4939-2864-4\\_609](https://doi.org/10.1007/978-1-4939-2864-4_609)

- 
- [116] Michel Raynal. 2018. *Fault-Tolerant Message-Passing Distributed Systems - An Algorithmic Approach*. Springer. <https://doi.org/10.1007/978-3-319-94141-7>
- [117] Michel Raynal. 2021. On the versatility of Bracha's Byzantine reliable broadcast algorithm. *Parallel Process. Lett.* 31, 3 (2021), 1–9. <https://doi.org/10.1142/S0129626421500067>
- [118] Michel Raynal. 2022. The BG Simulation. In *Concurrent crash-prone shared memory systems: A few theoretical notions*. Springer, 71–86. [https://doi.org/10.1007/978-3-031-79213-7\\_4](https://doi.org/10.1007/978-3-031-79213-7_4)
- [119] Michel Raynal. 2023. About informatics, distributed computing, and our job: A personal view. In *Proc. 30th Int'l Colloquium on Structural Information and Communication Complexity (SIROCCO'23) (Lecture Notes in Computer Science)*, 2023. Springer, 33–45. [https://doi.org/10.1007/978-3-031-32733-9\\_3](https://doi.org/10.1007/978-3-031-32733-9_3)
- [120] Michel Raynal. 2023. Mutual exclusion vs consensus: Both sides of the same coin?. *Bull. EATCS* 140, (2023).
- [121] Irving S. Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics* 8, 2 (1960), 300–304.
- [122] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126. <https://doi.org/10.1145/359340.359342>
- [123] Ron M. Roth. 2006. *Introduction to coding theory*. Cambridge University Press.
- [124] Matthieu Roy. 2016. BG Distributed Simulation Algorithm. *Encyclopedia of Algorithms*, 199–203. [https://doi.org/10.1007/978-1-4939-2864-4\\_611](https://doi.org/10.1007/978-1-4939-2864-4_611)
- [125] Michael E. Saks and Fotios Zaharoglou. 2000. Wait-free  $k$ -set agreement is impossible: The topology of public knowledge. *SIAM J. Comput.* 29, 5 (2000), 1449–1483. <https://doi.org/10.1137/S0097539796307698>
- [126] Nicola Santoro and Peter Widmayer. 1989. Time is not a healer. In *Proc. 6th Annual Symposium on Theoretical Aspects of Computer Science (STACS'89) (Lecture Notes in Computer Science)*, 1989. Springer, 304–313. <https://doi.org/10.1007/BFB0028994>
- [127] Nicola Santoro and Peter Widmayer. 2007. Agreement in synchronous networks with ubiquitous faults. *Theor. Comput. Sci.* 384, 2–3 (2007), 232–249. <https://doi.org/10.1016/J.TCS.2007.04.036>
- [128] Ulrich Schmid and Christof Fetzer. 2003. Randomized asynchronous consensus with imperfect communications. In *Proc. 22nd Symposium on Reliable Distributed Systems*

- (*SRDS'03*), 2003. IEEE Computer Society, 361–370. <https://doi.org/10.1109/RELDIS.2003.1238089>
- [129] Marc Shapiro, Nuno M. Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-free replicated data types. In *Proc. 13th Int'l Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'11) (Lecture Notes in Computer Science)*, 2011. Springer, 386–400. [https://doi.org/10.1007/978-3-642-24550-3\\_29](https://doi.org/10.1007/978-3-642-24550-3_29)
- [130] Victor Shoup. 2000. Practical threshold signatures. In *Proc. Int'l Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'2000) (Lecture Notes in Computer Science)*, 2000. Springer, 207–220. [https://doi.org/10.1007/3-540-45539-6\\_15](https://doi.org/10.1007/3-540-45539-6_15)
- [131] Hin-Sing Siu, Yeh-Hao Chin, and Wei-Pang Yang. 1998. Byzantine agreement in the presence of mixed faults on processors and links. *IEEE Trans. Parallel Distributed Syst.* 9, 4 (1998), 335–345. <https://doi.org/10.1109/71.667895>
- [132] Robert I. Soare. 1999. The history and concept of computability. *Handbook of Computability Theory* 140, 3–36. [https://doi.org/10.1016/S0049-237X\(99\)80017-2](https://doi.org/10.1016/S0049-237X(99)80017-2)
- [133] Gadi Taubenfeld. 1991. On the nonexistence of resilient consensus protocols. *Inf. Process. Lett.* 37, 5 (1991), 285–289. [https://doi.org/10.1016/0020-0190\(91\)90221-3](https://doi.org/10.1016/0020-0190(91)90221-3)
- [134] Lewis Tseng, Qinzi Zhang, Saptarni Kumar, and Yifan Zhang. 2020. Exact consensus under global asymmetric Byzantine links. In *Proc. 40th IEEE Int'l Conference on Distributed Computing Systems (ICDCS'20)*, 2020. IEEE, 721–731. <https://doi.org/10.1109/ICDCS47774.2020.00103>
- [135] Alan M. Turing. 1937. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.* 1 (1937), 230–265. <https://doi.org/10.1112/PLMS/S2-42.1.230>
- [136] Hagen Völzer. 2004. A constructive proof for FLP. *Inf. Process. Lett.* 92, 2 (2004), 83–87. <https://doi.org/10.1016/J.IPL.2004.06.008>
- [137] Jun Wan, Hanshen Xiao, Elaine Shi, and Srinivas Devadas. 2020. Expected constant round Byzantine broadcast under dishonest majority. In *Proc. 18th Int'l Conference on Theory of Cryptography (TCC'20) (Lecture Notes in Computer Science)*, 2020. Springer, 381–411. [https://doi.org/10.1007/978-3-030-64375-1\\_14](https://doi.org/10.1007/978-3-030-64375-1_14)
- [138] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. 2022. DispersedLedger: High-throughput Byzantine consensus on variable bandwidth networks. In *Proc. 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI'22)*, 2022. USENIX Association, 493–512.









---

**Titre :** Fondations de la Coopération Fiable avec de l'Asynchronie, des Pannes Byzantines, et des Adversaires de Messages

**Mots clés :** Algorithmes distribués, Systèmes asynchrones, Diffusion fiable, Tolérance aux pannes byzantines, Adversaire de message.

**Résumé :** Cette thèse se penche sur les systèmes distribués tolérants les pannes, et s'intéresse plus particulièrement au problème de la diffusion fiable dans des environnements asynchrones sujets à des défaillances hybrides. Elle introduit un nouveau modèle de calcul combinant des défaillances byzantines de processus avec un adversaire de messages. Elle définit ensuite l'abstraction de *Diffusion Fiable Byzantine Tolérante aux Adversaires de Messages (MBRB)* et prouve sa condition de résilience optimale. Elle propose enfin trois algorithmes clés pour réaliser cette abstraction : un algorithme MBRB sim-

ple basé sur les signatures, une nouvelle primitive appelée  $k2\ell$ -cast pour des implémentations MBRB sans cryptographie, et un algorithme MBRB basé sur les codes correcteurs d'erreurs optimisant la complexité de communication. Ces contributions font progresser la compréhension des systèmes distribués tolérants les pannes, et participent aux fondations nécessaires à la conception d'algorithmes répartis résilients et efficaces, avec des applications dans les infrastructures critiques, les systèmes financiers et les technologies blockchain.

---

**Title:** Foundations of Reliable Cooperation under Asynchrony, Byzantine Faults, and Message Adversaries

**Keywords:** Distributed algorithms, Asynchronous systems, Reliable broadcast, Byzantine fault tolerance, Message adversary, Message losses.

**Abstract:** This thesis explores fault-tolerant distributed systems. It focuses more specifically on implementing reliable broadcast in asynchronous environments prone to hybrid failures. We introduce a novel computing model combining Byzantine process failures with a message adversary. We then define the *Message-Adversary-tolerant Byzantine Reliable Broadcast (MBRB)* abstraction and prove its optimal resilience condition. We present three key algorithms implement-

ing this abstraction: a simple signature-based MBRB algorithm, a new primitive called  $k2\ell$ -cast for cryptography-free MBRB implementations, and an erasure-coding-based MBRB algorithm optimizing communication complexity. These contributions advance the understanding of fault-tolerant distributed systems and provide a foundation for designing resilient and efficient distributed algorithms, with applications in critical infrastructures, financial systems, and blockchain technologies.