Probabilistic methods for collaboration systems in large-scale trustless networks

Alex Auvolat

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Thèse de doctorat de

L'UNIVERSITÉ DE RENNES 1

École Doctorale N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par

Alex AUVOLAT

Probabilistic Methods for Collaboration Systems in Large-scale Trustless Networks

Méthodes Probabilistes pour les Systèmes Collaboratifs dans les Réseaux Large-échelle Sans Confiance

Thèse présentée et soutenue à Rennes, le 2 décembre 2021
Unité de recherche : Irisa (UMR 6074)

Rapporteurs avant soutenance :

Vivien Quéma	Professeur des Universités - Grenoble INP/ENSIMAG
Sonia Ben Mokhtar	Directrice de Recherches - CNRS

Composition du Jury :

Rapporteurs :	Vivien Quéma	Professeur des Universités - Grenoble INP/ENSIMAG
	Sonia Ben Mokhtar	Directrice de Recherches - CNRS
Examinateurs :	Guillaume Pierre	Professeur des Universités - Univ Rennes, CNRS, Inria, IRISA
	Martin Kleppmann	Senior Research Associate - University of Cambridge
Dir. de thèse :	François Taïani	Professeur des Universités - Univ Rennes, CNRS, Inria, IRISA
Co-dir. de thèse :	Yérom-David Bromberg	Professeur des Universités - Univ Rennes, CNRS, Inria, IRISA

Remerciements

En premier lieu, je souhaite remercier Sonia Ben Mokhtar et Vivien Quéma qui ont accepté d'êtres les rapporteurs de ma thèse pour le temps qu'ils accordent à mes travaux : la relecture de bout en bout d'une thèse, conclusion de trois ans de travaux, reste un travail immense pour lequel vous avez mon entière gratitude. Je souhaite également remercier Martin Kleppmann et Guillaume Pierre qui ont accepté de faire partie de mon jury.

L'équipe WIDE a été un formidable lieu d'accueil durant ces trois années, qui m'a enrichi tant du point de vue personnel que professionnel. À ce titre je voudrais commencer par remercier Virginie Desroches, notre assistante d'équipe, pour son aide précieuse sur tous les sujets administratifs, qu'elle m'a toujours apporté avec beaucoup de bienveillance, ainsi que tous les personnels administratifs et techniques qui travaillent dans l'ombre mais sans qui rien ne serait possible. J'ai évidemment une pensée émue pour les camarades que j'ai rencontré lors de mon arrivée au labo, et avec qui nous sommes devenus très proches grâce à nos longues discussions lors des pauses du midi. Je pense en particulier à Quentin et Adrien, avec qui nous avons lancé le projet Deuxfleurs : je considère que c'est un honneur et une chance immense pour moi de les avoir rencontrés, et c'est avec un enthousiasme sans modération que j'envisage la poursuite de nos travaux en commun. Je pense également à Louison et Loïck que je souhaite remercier pour leur camaraderie et leur soutien durant cette longue épreuve. Je voudrais remercier également les permanents de l'équipe pour les discussions enrichissantes et les collaborations fructueuses : Davide Frey, Michel Raynal, Erwan Le Merrer, Matthieu Simonin et George Giakkoupis, ainsi que tous les stagiaires, post-docs ou autres qui sont passés à un moment dans l'équipe. Enfin et surtout, je remercie mes directeurs de thèse, François Taïani et David Bromberg, qui ont rendu tout ça possible : ils m'ont apporté au quotidien non seulement leur aide et leurs compétences sur les aspects scientifiques, mais également leur soutien sur toutes questions de type logistique, administrative, ou existentielle. Merci de m'avoir soutenu, mais aussi supporté, pendant trois ans, et de m'avoir aidé à aller jusqu'au bout.

La réussite d'une thèse ayant trait non seulement à ce qui s'y passe, mais également à ce qui se passe ailleurs, je voudrais remercier tous mes amis et toutes mes amies, des personnes formidables avec qui les contacts ont malheureusement parfois été trop rares. Il

y a, par ordre alphabétique dans chaque catégorie, les amis des études : Alexis, Clémence, Élie, Émile, Émilie, Étienne, Hélène, Hélène, Jean, Jill-Jênn, Jonathan, Julie, Juliette, Leïla, Léonard, Louise, Maëliss, Marion, Matthias, Matthieu, Maxime, Mendes, Nicolas, Nicolas, Nina, Paul, Sebastien, Théo, Thomas, Tito ; les colocs de Rennes : Clémence, Emma, Étienne, François, Nolwenn ; les amis d'avant : Brendan et Lucas ; les amis d'après : Ariane, Ella, Roméo, Valentin, mais aussi l'autre groupe qui se reconnaîtra ; les amies de la danse : Amélie, Anaïs, Bruno, Béatrice, Cécile, Cédric, Florian, Gildwen, Guillaume, Gwenaël, Hélène, Laurine, Louise, Marie, Mathilde, Milena, Nathalie, Raphaëlle, Romain, Sandrine, Servane, Thierry, Valentine ; ainsi que toutes les personnes que j'ai oubliées (je pense quand même à vous !)

Je voudrais par ailleurs élargir mes remerciements à une catégorie rarement mentionnée, mais dont vous comprendrez en lisant cette thèse qu'ils ont une importance toute particulière pour moi : ce sont les gens qui s'expriment, créent, discutent, jouent, et plus généralement existent sur Internet. En plus du divertissement ou des connaissances qu'ils nous apportent, leur force de créativité est pour moi une inspiration remarquable. Pendant cette période du COVID, ils ont pris une place encore plus importante dans ma vie qu'avant, et m'ont permis de me sentir moins seul. Ils sont bien entendus trop nombreux pour les nommer tous, mais je voudrais remercier par exemple sur Twitch : Modiiie et tous les petits camarades ratons et les petites camarades ratonnes, Antoine Daniel, Ponce, Domingo, Zerator, Redfanny, Usul, Ost, Baghera, Jean Massiet, Copainduweb, Nono, DFG, Wirtual, et tout un tas d'autres chez qui je suis passé plus ou moins régulièrement. Sur Youtube, je pense par exemple aux chaînes suivantes : Wintergatan, Hit the Road, Charles et Mélanie, Nico Mathieux, Daily Dose of Internet, CGP Grey, Thinkerview, The 8-bit Guy, Retro Recipes, Khaled Freak, Hydraulic Press Channel, Beyond The Press, Linus Tech Tips, ASMR Politics, Andreas Kling, Linguisticae, Micode, Taylor Tries, Oui d'accord, ainsi que de nombreuses autres chaînes.

Pour conclure, je voudrais remercier toutes les personnes qui œuvrent pour créer et diffuser de la connaissance scientifique et des informations d'utilité générale. Je voudrais remercier toutes les chercheuses et tous les chercheurs dont les travaux sont mentionnés dans cette thèse. Je voudrais également remercier les journalistes et les bloggueurs qui produisent aussi bien des informations factuelles que des argumentations politisées qui dans tous les cas enrichissent notre vision du monde. Enfin, je voudrais remercier toutes les personnes qui contribuent à la démocratisation de ces connaissances notamment sur Internet.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Notations

Processes/nodes in a distributed system

p, p_i	A node (also called a process) in a distributed system
Π	The set of nodes
N	Number of nodes in a network
f	Fraction of Byzantine nodes
t	Number of Byzantine nodes in a network ($t = fN$)

Processes/nodes in a distributed system

m	An application message (as opposed to an internal protocol message)
$m \rightarrow m'$	Happened-before relationship, or causality relationship, between m and m' (Lamport, 1978)
sn	Sequence number
λ	Network latency
Δ	Maximum bound on network round-trip time

Conflict-free replicated data types

$\sqcup$	The CRDT merge operator (forms a join-semilattice)
$\bot$	The bottom element, such that $\forall x, \bot \sqcup x = x$

List of Abbreviations

AS	Autonomous system
BCO	Byzantine causal order (Chapter 3)
BGP	Border gateway protocol
BR broadcast	Byzantine reliable broadcast
BFT	Byzantine-fault tolerant
CGN	Carrier-grade NAT
CIDR	Classless inter-domain routing
CRDT	Conflict-free replicated data type
DAG	Directed acyclic graph
DHT	Distributed hash table
FIFO	First in first out
IP	Internet protocol
ISP	Internet service provider
MBR broadcast	Multi-shot Byzantine reliable broadcast
MPT	Merkle prefix tree
MST	Merkle search tree (Chapter 4)
NAT	Network address translation
PoS	Proof-of-Stake
PoW	Proof-of-Work
RPS	Random peer sampling
SBO	Simultaneous block ordering (Chapter 6)
SHA2	Secure hash algorithm 2
SHA-256	SHA2 with 256-bit digests
SPS	Secure peer sampling (Jesi et al., 2010)
TLS	Transport layer security
TPS	Transactions per second

List of Publications

Publications included in this manuscript:

— Alex Auvolat and François Taïani (2019), « Merkle Search Trees: Efficient State-based CRDTs in Open Networks », *38th Symposium on Reliable Distributed Systems (SRDS)*, IEEE, pp. 221–230, DOI: 10.1109/SRDS47363.2019.00032

— Alex Auvolat (2019), « Making Federated Networks More Distributed », *38th Symposium on Reliable Distributed Systems (SRDS)*, IEEE, pp. 383–384, DOI: 10.1109/SRDS47363.2019.00058

— Alex Auvolat, Davide Frey, Michel Raynal, and François Taïani (2021b), « Byzantine-tolerant Causal Broadcast », *Theoretical Computer Science*, pp. 55–68, DOI: 10.1016/j.tcs.2021.06.021

Publications not included in this manuscript:

— Alex Auvolat, Michel Raynal, and François Taïani (2019), « Byzantine-tolerant Set-constrained Delivery Broadcast », *International Conference on Principles of Distributed Systems (OPODIS)*, ACM, 6:1–6:23, DOI: 10.4230/LIPIcs.OPODIS.2019.6

— 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

— Fabien Coulon, Alex Auvolat, Benoit Combemale, Yérom-David Bromberg, François Taïani, Olivier Barais, and Noël Plouzeau (2020), « Modular and Distributed IDE », *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*, pp. 270–282, DOI: 10.1145/3426425.3426947

Papers currently under submission:

— Alex Auvolat, Yérom-David Bromberg, Davide Frey, and François Taïani (2021a), « Basalt: A Rock-Solid Foundation for Epidemic Consensus Algorithms in Very Large, Very Open Networks », arXiv: 2102.04063

— Alex Auvolat, Yérom-David Bromberg, and François Taïani (n.d.), « SBO: Fast Epidemic Byzantine-Tolerant Consensus in Truly Open Networks »

Chapter 1

Introduction

Recent times have had the privilege to witness an exponential growth in the use of communication networks such as the Internet in our society. This breakthrough technology has caused a revolution in how we access information and communicate together: we can now communicate instantly with individuals at the other side of the globe, access vast repositories of information at our fingertips, and engage in radically new modes of socialization such as forums, tweets, videos and live streaming. The democratization of computing has also allowed us to revolutionize the way we organize our work, education and leisure, through platforms that allow computer users to easily access applications specialized for office tasks (e.g. word processing, accounting), teaching and entertainment.

The potential of these new communication platforms is maximized when two conditions are met. First, users must be free to publish information and content on their own terms in order to reach their target community. Second, users must be preserved from unwanted or dangerous content such as spam or illegal content, and must have ways to find content relevant to them in an ocean of undifferentiated data. Large centralized networks such as those of Google, Amazon, Facebook and the likes (collectively known as GAFAM) take care of these two necessities for their users: the two functions of information publishing and information selection are handled by the same entity and are often not distinguished. However conflating these two functions is not without issues: while the task of storing data and making it available upon request can be adequately reduced to a technical problem, there are intrinsic social and political choices behind practices of moderation and promotion of content, as having control over what users see can allow one to influence their behaviors. In our consumer society, the attention span of users is bid upon by advertisers aiming to promote brands and products, at the expense of time spent purposefully by the individual for themself or their community (for instance: teaching and learning, creative work, community building). While users of the Internet are seemingly acting following their free will, they are caught in a web of determination that does not lead to their emancipation, as François Bégaudeau would say (Bégaudeau, 2021). Or, in

the words of Ivan Illich (1973), the commercial Internet is not a "convivial technology": instead, it is a techno-architecture which we are trapped in, and that forces us to conform to its system (similarly to how the development of motorized transportation made it a necessity for everyone to own a car).

In the current situation of ecological crisis (IPCC, 2014; IPCC, 2018; IPCC, 2021), many consider that consumerism and growth are a peril to our survival as a society (Hansen et al., 2008; Heinberg, 2010; Heinberg, 2011; Servigne and Stevens, 2015; Barnosky et al., 2016; Mien, 2020), to which the only realistic solution seems to be to radically change our focus as a community to living simpler lives in a degrowth economy (Ariès, 2005; Latouche, 2006; Hopkins and Lipman, 2009; Parrique, 2019; Defontaine, 2020). The invasion of the Internet by commercial content, further to being a treason to its original purpose as a tool of education and collaboration, is an impediment to us tackling these challenges: as our attention is forever captured by self-branding and self-promotion through consumption, we are unable to see the big picture of the imminent dangers we face and are led instead in the opposite direction. While in the 2000's television might have been the media that was selling "available brain time" to advertisers (the famous "temps de cerveau disponible", an expression of TF1's then-CEO Patrick Le Lay), it seems that social media on the Internet have picked up this role more and more in the recent years, targeting a particularly young and vulnerable audience. The news feeds of Twitter, Facebook and Instagram are built algorithmically in ways that keep us hooked for as long as possible by maximizing "engagement", an abstract metric that favours the introduction of advertisement in the feed, often in a subtle or deceptive manner. The commercialization and the monopolization of the Internet has measurable consequences, with addiction to social networks a widely documented phenomenon (Fourcade, 2018; Papp, 2018). Targeted advertising, which uses personal data to influence consumers with arguments specifically tailored to their sensibilities, has also been wielded as a tool of political influence as revealed by the Facebook–Cambridge Analytica data scandal, leading to more and more recognition of the political and geo-political consequences of platforms that amass and make use of large quantities of user data (Falque-Pierrotin, 2018; Colin, 2018). Remarkably, even the top executives of big Internet companies like Google and Facebook are acutely aware of the dangers social media pose to our society and the youth in particular, and are setting restrictive rules in their own families to prevent harm to their children! (Rudgard, 2018)

This situation of monopoly also has consequences for the freedom of users and content producers on these platforms. Without entering in too many details, we can cite for

instance the following:

— These companies apply content moderation on their platforms with obscure and uncontrollable criteria which can be detrimental to their users. This has the effect that users can get banned (temporarily or definitively) when they arguably did not do anything wrong. Eileen Guo (E. Guo, 2021) shows how this power has been abused to stifle human right activists on YouTube. On a lighter note, a French streamer on the Twitch platform was recently banned for 24 hours from the platform for accidentally showing during a fraction of a second images that contain female nipples (in a non-sexual context). In addition to showing that users can be banned for accidental situations, i.e. in random ways that they do not fully control, the reasons why this kind of content is banned even in a non-sexual context seem to be cultural as there is a difference in perception between the French public (which did not see anything wrong with the content) and the American culture from which Twitch originates.

— Note that these moderation rules are generally enforced automatically by artificial intelligence techniques, making content creators vulnerable to false positives. The logic of reduction of costs that leads to replacing human moderators by AI also means that often content creators have no recourse to get their account back as there is no human interlocutor available to negotiate with.

— These arbitrary bans from large platforms may have financial consequences: recently, several Android application developers have had their applications pulled by Google from the Play Store for arguably bad reasons, with no possibility of recourse, in situations where these applications were paid-for and provided a source of revenues to their creator. This means that large companies such as Google can arbitrarily deprive creators of a revenue source they depend on. Also, by favoring or demoting certain types of content on platforms such as YouTube, these large platforms directly control the visibility of user-created content and thus the advertisement revenues the creators can generate.

— Similar issues have also happened to users of the Google personal office suite (Google Drive, Gmail, etc.), with situations where users get locked out of their account and lose all access to their personal data that was stored on the platform. Again, the lack of a human interlocutor at Google means that the user has no chance of getting their account back in case of a mistake or misjudgement.

These issues can all be traced back to the situation of monopoly from which large

companies such as the GAFAM benefit: the Internet platforms that they make available are immensely powerful and attract many users [1], but they remain logically controlled by a single entity that acts in the interest of its self-preservation and financial growth, and not in the interest of the users. In practice, content creators have to be on the platform in order to reach a wide audience, meaning that they basically have no other choice than to submit to the arbitrary rules these platforms impose.

In this context, it is critical that we make possible and encourage alternative practices in the way we build online communities, in order to reclaim the Internet as a means of free publication and information, of organization for democratic action and for the development of alternative ways of life, in ways such that commercial content and advertising are not the ultimate goals towards which all choices are directed. In a parallel with the class warfare between the working class and the bourgeoisie in Marxist theory, Glen Weyl conceives of producing and giving one's personal data to big companies as a form of exploitative work (Weyl, 2018), and argues that users must organize collectively to resist and reclaim their so-called "data dignity". Our goal, and the inspiration for this thesis, should be to resist collectively, in order to organize democratic governance of public platforms on the Internet, inspired by the rules proposed by Elinor Ostrom on the governance of commons (Ostrom, 1990).

While focusing on technical aspects of decentralized networks is of course not the be-all and end-all to changing the way we use online networks, this is a thesis in Computer Science, and it will thus focus specifically on technical pre-conditions that make possible a shift to a more community-driven Internet at the service of its users. Thankfully, the Internet was not originally designed as a centralized, commercial network, and it retains its original structure as a decentralized network where peers can interact directly with one another without prerequisites: any peer in the network can act as a server if it wishes to make content available [2], a property that has already been leveraged to build peer-to-peer file sharing networks such as BitTorrent. It is therefore not a fatality that the Internet end as a tool for advertisement and mass manipulation in the hand of huge international

1. Note also that due to network effects, platforms that are already big will attract more and more users, whereas platforms that are still in their infancy have a hard time attracting new users due to the limited interest of joining the platform. This issue means that transitioning to alternatives will not be an easy task and building new technical solutions is not sufficient to impulse a change; constraint by law and state control is probably necessary to get out of this monopoly situation. Cory Doctorow has an interesting discussion on the ways in which we can force (or not) platforms to be interoperable in order to neutralize monopolies (Doctorow, 2021).

2. Unfortunately less and less true with the advent of carrier-grade NAT. We can hope that IPv6 will participate in reversing this tendency.

corporations, and it is possible to build applications that leverage this structure to provide a decentralized collaboration platform controlled by the users.

1.1 From Centralized to Federated Networks to Decentralized Applications

Having observed the various failure modes of centralized platforms controlled by large multinational corporations, we enter a quest for alternative architectures and protocols that would allow us to build communication and collaboration platforms on the Internet in a decentralized fashion, i.e. platforms that are constituted by nodes owned by a variety of independent parties, using a common protocol to interoperate and allow communication between them. A first solution naturally consists in so-called *federated* networks. We will study the case of federated networks, but we will quickly dismiss them due to some important limitations that they have. These limitations will inspire us to pose new principles for trustless peer-to-peer networks in order to build what we will call *decentralized applications*.

Federated networks are not enough Federated networks such as Mastodon, Diaspora* or Matrix are often touted as technical solutions that allow users to reclaim control over the platforms they use. Similarly to how e-mail is built, a federated network is a network constituted of many servers (called *instances*), each with its domain name and its set of local users. Each server acts first as an individual communication platform between its users, but servers have in addition the ability to exchange information between them, enabling users from server A to communicate (more or less transparently) with users from server B. User accounts are intrinsically linked to the server on which they were created, and accounts are identified with identifiers in the typical `username@domain.name` form. Mastodon and Diaspora* are federated social networks inspired by the structure of Twitter and Facebook respectively, whereas Matrix is a federated communication platform that implements chat rooms similar to Discord or Slack.

While these networks have arguably been widely successful (more than 2 million users have registered an account on the Mastodon network), these solutions suffer from two main defects:

— **A tendency to centralization on very large instances.** Indeed, most of the users that choose to join the network do not know of a particular server which they

should join (and sometimes do not even understand the concept of federation). By default, they often join one of the largest instances, simply because these instances are much more visible in the network. For instance on the Mastodon network, the two largest instances have 675 000 and 560 000 users, whereas the next largest instance has only 205 000 users and most small instances have only a few users [3].

— **Increased vulnerability of the network.** Indeed, small instances are often constituted of a single machine without backups, as they are set up by administrators that have limited resources and time to dedicate to proper systems administration. This means that small to medium instances may crash and become totally unavailable at any time. In other words, federated networks depend on the continuous availability of all instance administrators at all time to fix potential issues. This means that the network as a whole is generally unreliable and cannot be trusted by the users.

In order to better understand and try to solve these issues, one has to take a critical look at the definition of decentralized and distributed computing systems. We will instead propose definitions that put the focus not on the global architecture of the system (as a collection of nodes controlled by independent parties), but on the protocol and the mechanisms it uses to know how to propagate data and how to know what data is to be trusted.

"Distributed" is not enough either Proponents of decentralization often talk of building distributed systems as the logical next step after decentralized federated networks. However, the definition of distributed systems in the computer science literature has in fact little to do with the issues of decentralization that concern us:

Definition 1.1 *A distributed computing system is a system composed of multiple autonomous computing entities called* nodes*, each with their own memory, that communicate together using message passing* [4]*.*

This definition covers a large variety of systems, so large in fact that almost all applications on the Internet are distributed in some way according to it. Indeed, on the one side

3. See `https://instances.social/list/advanced`

4. Distributed systems literature generally also considers the case of *shared memory systems* as distributed systems, however the recent advances on shared memory systems (such as issues of scheduling, NUMA, etc.) are today generally categorized as problems of *parallel computing*, another fascinating area of research which is totally outside the scope of this thesis. We will stick to "distributed systems" meaning "message-passing distributed systems" in this manuscript.

centralized platforms such as Google or Facebook internally make use of distributed software such as large scale orchestration systems (Kubernetes) and distributed data stores (DynamoDB, S3). On the other side, peer-to-peer file sharing protocols such as BitTorrent intrinsically are distributed systems. At the extreme, even a simple personal website that uses an external SQL database and is behind a CDN can be considered as a distributed system of three nodes. However, these systems are clearly very different in nature and are not targeting the same use cases. Although this thesis is definitely in the domain of distributed systems, this definition is not sufficient to cover the issues that concern us.

Identifying what makes a centralized systems The first key comes from defining, precisely, what makes a *centralized* system. This definition is compatible with the understanding of certain centralized systems (such as Google, Facebook, etc.) as being distributed. This definition puts the accent not on the structure of computing (a single machine or a cluster of nodes), but on data itself and the way in which an authoritative copy is defined:

Definition 1.2 *A centralized system is a computing system where the authoritative copy of data is provided and controlled by a single well-identified entity. In other words, to access or modify the data, a user must necessarily contact a well-identified Internet service. Once properly authenticated, the Internet service is trusted to provide valid data and to properly store inputs provided by the user.*

In practice, this consists in calling a single API endpoint (represented e.g. by a HTTP URL), which can be found using DNS and authenticated using TLS. This endpoint can be served by a single server or a distributed system such as Kubernetes cluster for example, without affecting its characterization as "centralized".

Among other things, this means that the single well-identified Internet service is a single point of failure of the system; its failure rates can however be minimized by implementing it with a distributed system.

Note that this definition covers two aspects of data and authoritative copies: providing the authoritative data, and controlling the authoritative data. In a centralized system, these two roles are not distinguished and are performed by the same entity: the data is obtained by making e.g. an HTTP request to the system, which *provides* it to us by answering the request, and we trust the answer to be authoritative as long as we are able to authenticate its source (e.g. with TLS), thus letting the centralized system *control*

what the data is. We will see that dissociating these two aspects is key to defining trustless distributed systems, the kind of systems we will be interested in here.

Federated systems in the lens of Definition 1.2 The limits of federated systems which we evoked earlier are linked directly to how they are built. While federated systems can be seen as systems where there is no single centralization point as defined above, at the technical level, a single server in a federated network is extremely similar to a centralized system: indeed, it provides and controls the authoritative copy of the data of the users of this server. The protocol is reliant on the presence and adequate configuration of individual servers to ascertain properties such as the identity of users and to provide the authoritative copy of data items. The only difference is that servers in a federated network share and replicate data from one another, making interactions between different servers possible. While there is no longer a single entity that provides and controls the data for the whole network, servers still act as individual centralization points, and are each single points of failure for their respective users. These systems thus have a decentralized structure, but the protocol still relies on properties of centralization: these systems better understood as a *collection of centralized servers* that communicate between them.

Redefining decentralization: trustless P2P systems In order to make explicit the requirements we pose on the kinds of decentralized systems we want to build, we will rather focus on defining them as *peer-to-peer* systems:

Definition 1.3 *A* peer-to-peer *(P2P) system introduces the crucial notion that data can be retrieved from any node in the network independently of its identity (as long as it has the data in the first place), instead of having to contact a specific Internet service owned by a certain entity. In other words, it is defined as a system where no single well-defined entity acts as a mandatory centralization point for accessing and modifying any piece of data, in the terms of Definition 1.2.*

The first step in making such a system consists in building *coordination-free replication*: data can be replicated on several independent machines run by different people in different physical locations, and thanks to eventually consistent replication methods such as CRDTs, no single machine ever acts as a coordination bottleneck or single point of failure in the network. At the extreme, nodes need not be specifically designated server machines but can be any personal machine in the network, such as is the case with collaborative CRDT editing systems such as Crate (Nédelec et al., 2016).

This kind of system however requires all of the nodes to trust one another, since CRDT techniques do not incorporate verification of inputs and operations that occur at one node are replicated at other nodes without any filtering (see Section 2.3.3). In real world systems however, trusting everyone is not without issues, and being able to securely authenticate the origin of data, restrict participation to a set of users or ensure a distributed computation is performed correctly is a necessity. In the terms of distributed computing, this means that some participants may be Byzantine: they do not respect the rules that were commonly enacted to make the system work and may instead have arbitrary behavior.

To generalize these kinds of systems to cases where nodes canot always be trusted, one has to introduce the notion of a *trustless* peer-to-peer system:

Definition 1.4 *A trustless peer-to-peer system is a peer-to-peer system where nodes can retrieve data from any other node in the system independently of its identity, but this node is not necessarily trusted to provide valid data. Nodes returning a piece of information must also be able to provide a proof of its validity in an agreed-upon verification scheme. For example, this proof can take the form of a digital signature that links the piece of information to its author, or a set of digital signatures attesting that this information is a consensus in the overall network.*

This definition totally removes the authoritative role of any given node of the network, and replace it with a decentralized verification scheme based on cryptographic primitives: in order to accept a piece of information as valid data in the network, a node must be able to verify that its accompanying proof verifies a set of predefined conditions. This proof can be provided by any other node in the network and its validity is not dependant on who it was obtained from.

Note that this definition does not say anything on the architecture of the network, which can retain some properties of the client/server model, even though it is called "peer-to-peer". Indeed, servers do not have to disappear altogether: machines with copious computing power and storage space still have a role to play as they allow information to be accessed more easily, for instance from mobile devices that must preserve their battery life and thus cannot participate as full peer-to-peer nodes [5]. This definition does however imply that a network node providing a piece of information must also be able upon request

5. This is not a new concepts: many peer-to-peer networks have introduced the notion of a *supernode*, a node that has a privileged role in the network, for instance as an indexing server.

to prove the validity of this piece of information. We will call *full nodes* or equivalently *servers*, nodes that are able to provide data and associated proof of validity to other nodes, or that are able to contribute in building network-wide consensus on what data is valid (we will see later that these two ways of verifying information are necessary and complementary).

This definition also does not necessarily imply that servers are never trusted by anyone: mobile devices can decide to trust a given server to provide them with an accurate representation of the data, trusting the server to validate information exchanged with other network nodes. Again this is a requirement for mobile devices which cannot be expected to validate all proofs of authenticity by themselves. Similarly, if the load on the network is too important for a single server node to verify everything, guilds of server nodes that trust one another can be built so that the whole dataset is verified by the guild collectively. Note however that these trust relationships can be revoked at any time, and reported transparently to other servers if necessary. The fundamental property is that a network user can always access the network and participate even without trusting anyone, by running a full server node themself (under the assumption that they have enough computing power to verify everything).

Example of a trustless P2P system To give an example, which we will later explore in more details, the Bitcoin network is currently an instantiation of this principle: the blockchain is by itself the proof of its validity, thanks to the rule of selecting the longest Proof-of-Work chain, and full Bitcoin nodes verify this validity when synchronizing with the network as they do not trust one another. However, many Bitcoin exchanges provide API endpoints to explore the state of the blockchain and view account balances, which requires the API consumer to trust that node to have correctly validated the PoW consensus. Further, lightweight wallets in the form of mobile apps are made possible by trusting these API servers to provide the clients with an accurate representation of the current blockchain state and to broadcast transactions they emit to the Bitcoin network.

Advantages of trustless P2P systems Building information systems upon the design principles of trustless peer-to-peer systems has many advantages, of which the following are the most prominent:

— **Easy redundancy and fault tolerance:** since any peer may provide data as long as it is properly authenticated, any node can chose to replicate a piece of data

and serve it to the network itself, as long as it is able to prove its validity. In the case of digital signatures, proving validity is trivial and thus this kind of replication is extremely simple to set up. When a proper replication scheme is implemented, server nodes are individually dispensable and the network can continue to function perfectly normally with nodes leaving for any kind of reason such as network downtime, hardware failures, or being disconnected by an administrator.

— **Horizontal trustless read scalability:** similarly, thanks to these easy properties of replication, when more users want to access content in the network, any network nodes is qualified to answer. This means that new nodes can be set up to serve read requests to follow demand as it grows without having to modify anything in the previously running network. This is similar to how BitTorrent peers can transparently become seeders of content once they have successfully downloaded it from other peers in the network.

— **Easy migration of user profiles:** the data of a user's profile is no longer linked intrinsically to a specific server: its validity is instead attested by digital signatures that can be propagated by anyone. This means that users are not linked to a single home server like in federated networks: user profiles are intrinsically nomadic and the servers users rely on (if any) can change at any time.

Trustless systems are however not exempt of limitations. This thesis is aimed precisely at identifying and alleviating the limitations of this new kind of systems.

1.2 Decentralized Applications: Vision and Challenges

In the rest of this thesis, we will adopt the terminology of a *decentralized application*, for an application such as a social network or a collaborative database that is constructed as a trustless peer-to-peer system. While this terminology choice is unfortunate as decentralized systems are often taken to refer to federated networks, which are not the subject of this thesis, the term *decentralized application* (sometimes written DApp or dApp) is gaining traction recently, for instance with the creation of the IEEE International Conference on Decentralized Applications and Infrastructures (IEEE DAPPS) in 2019. DApps specifically target P2P applications in a trustless setting (often in the context of blockchain technology, which we will describe below). We are thus making the choice to stick with the agreed-upon designation.

1.2.1 Challenges for decentralized applications

While trustless application design seems very promising in alleviating issues of centralized and federated systems, it is not without a set of challenges that comes precisely from the fact of not being able to trust anyone. In particular, modern era social networks are huge platforms with millions of users, and open to anybody to join. Trying to replicate these properties in a trustless P2P setting causes issues of scalability in particular in a design where every full node needs to verify the validity of all network operations (Saraph and Herlihy, 2019).

Moreover, the openness of the system means that we cannot count on the existence of a fixed known set of members, a property that is used in the construction of classical Byzantine-tolerant algorithms. Indeed, in closed systems of N nodes with $t < N/3$ nodes at most that exhibit malicious behavior (called Byzantine behavior), many algorithmic primitives are available to ensure the correct functioning of a system, such as reliable broadcast (that ensures that all nodes receive the same set of messages), and consensus (that allows nodes to agree on a single value globally). While in the most simple applications cryptographic signatures on pieces of information by their authors is sufficient to build the required properties, more complex applications may need stronger forms of coordination, culminating with total order broadcast (also called consensus) which allows a system to trustlessly implement any arbitrary replicated state machine as if it were a centralized system. We show an example for this last point below.

Lastly, to allow for the wide adoption of a new protocol, the implementation of a trustless P2P system should ensure that the barrier to entry should be as low as possible to allow as many members as possible to join. Ideally, any single computer with an internet connection could contribute power to the system (to increase scalability or security for instance) without any other prerequisites.

1.2.2 Components of a decentralized application

We propose a decomposition of decentralized applications as a dataflow system composed of three steps, illustrated in Figure 1.1. This decomposition will be the backbone of our analysis and a motivation for the new techniques we propose.

First, the inputs of the system are given by users, as individual information pieces that users sign with their own private keys. For instance, in the case of a cryptocurrency, the inputs would be the raw transactions emitted by the users, where each transaction

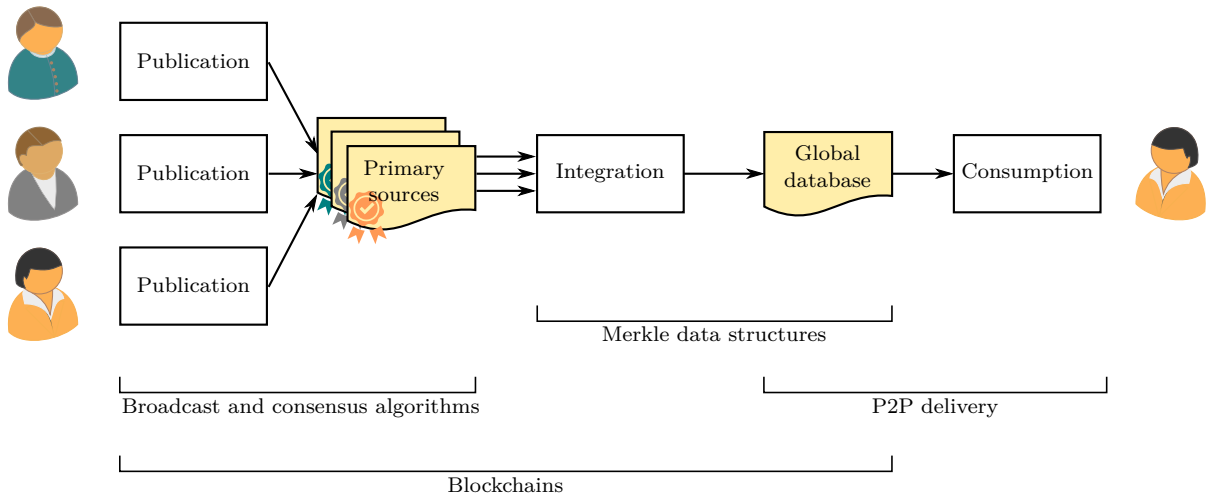


Figure 1.1 – The flow of information in distributed applications, and the technologies that intervene at each step

is signed by the holder of the expensed coins. In a social network, the inputs would be the individual messages posted by users, including references to other messages they are replying to, all of which is signed with their private key. We will refer to these inputs as *primary sources*.

Then, once all of the inputs are collected (and possibly ordered), they are integrated together in order to deduce the global system state. In a cryptocurrency, this would correspond to building the blockchain and building a Merkle tree that contains all of the account balances at each block. The root hash of this Merkle tree is then included with each block of the chain. In a social network, this would correspond for instance to collecting together the messages that appear in the same discussion thread, so that they can be found easily. Depending on the system, this integration is either done collectively and published (e.g. on the blockchain), or can be done locally by each network node. We will call the result of this integration process the *global database*, because it contains the aggregation of all inputs in the system.

Finally, users will consume content they are interested in. To do so, they will access the global database to consult the parts that are relevant to them.

This global framework leads to a few key question of how to architecture such a system in practice:

— **How are primary sources authenticated?** In a decentralized application, primary sources are authenticated by the digital signature of their author. However the schema of Figure 1.1 can also be used to analyze a centralized system, in which

case primary sources are authenticated by an API endpoint and once stored in a database, information is assumed to be valid.

— **Who does the integration process?** This can be either done by a single trusted node, done by each node when no node is trusted (the case of cryptocurrencies), or with a method that uses partial trust to alleviate load on individual nodes.

— **Is the global database shared?** The Ethereum cryptocurrency and smart contract network publishes on the blockchain the root hash of the Merkle tree that represents the system state after validating the transactions of each block. However this doesn't have to be the case, as the history of inputs encoded in the blockchain is sufficient for any new node to recalculate the blockchain state from scratch. Moreover, in less strict settings such as social networks, the integration process does not need to be strictly deterministic as it can be acceptable for users to see different views of the system, so the global database can be computed locally by each node. Sharing the database ensures that the system is consistent and allows for nodes to divide the workload of building it.

— **How is the global database authenticated?** If nodes all run the full integration process, they don't need to trust other nodes to provide them with a correct global database. If that is not the case, a verification mechanism must be used to ensure that the global database is correct with respect to the inputs.

— **How to scale the integration process?** When the load on the system is too important, a single node is not able to run the entire integration process. A mechanism must be designated to allow nodes to share the integration task between them so that all of the inputs can be processed.

In the case where the relative ordering of primary sources is not important, these sources can simply be disseminated with a gossip algorithm. In the case where ordering between independent inputs is required, consensus is needed and the dissemination method for primary sources should implement total order. We will see below that total ordering is required to implement even simple access control mechanisms, an important building block for social networks. For the sake of generality, a significant fraction of this thesis we will therefore concentrate on systems that use total order to disseminate the primary sources, storing them on a blockchain. In Chapter 3 we weaken this property by assuming that the application only needs causal ordering, and show that broadcast primitives that implement this ordering are possible in a system with Byzantine nodes. We however do not discuss the implications of using such a broadcast on the rest of the pipeline for the

integration of inputs in a global database.

Similarly, in the case where it is not a requirement in the system that different participants access the same consistent global database, the integration process can be handled locally. It can also be handled with a distributed system that does not handle consistency such as a distributed hash table (DHT) or a distributed index. However these distributed methods are hard to implement securely in a trustless setting. Moreover accessing data is usually relatively slow in a worldwide network, especially in the case of a DHT that may need $O(\log n)$ queries to find a particular data item. Combined with the lack of consistency control, this makes these methods unfit except for special cases. We will therefore exclude this situation, and assume that integration is done either locally or in a strongly consistent fashion with its result encoded in a Merkle data structure whose root hash is stored in the blockchain.

1.2.3 Weak consistency approaches and their limitations

In the general case, any application can be implemented in a distributed system if we are able to implement the primitive of a *replicated state machine* (Borg et al., 1983; Lamport, 1984; Schneider, 1990; Marandi et al., 2011): any arbitrary computation system can be expressed as a state machine, and replicating it over many nodes allows it to be tolerant to faults, and in particular to Byzantine faults if the replication method is Byzantine-tolerant. However, implementing full-fledged state machine replication (or SMR) in a P2P trustless system boils down to solving Byzantine consensus, a notoriously hard problem.

Before we tackle the issue of Byzantine consensus, let us first discuss possible weaker approaches to building decentralized applications. Peer-to-peer information dissemination using gossip-like protocols has existed for over thirty years (Demers et al., 1987), and is a well-studied basis for information dissemination in large scale networks that act independently and without coordination (Eugster et al., 2004a). In the context of the public Internet, these methods have been widely popular with file transfer applications such as Napster (1999) and BitTorrent (2001), although in their original forms these protocols still relied on a central directory of nodes [6]. In order to introduce mutability semantics in such systems, allowing for distributed modification of the data without coordination, one can use techniques such as CRDTs (Shapiro et al., 2011), that define reconciliation

6. Thanks to the BitTorrent DHT, clients in the BitTorrent network may now download files without having to contact a central server to obtain a list of peers from which to download.

semantics in order to combine modifications that may occur anywhere and at any time (a CRDT is defined by the presence of a merge operator that is associative, commutative and idempotent, which allows nodes to converge to the same state independently of the order in which they receive updates). In a trustless setting however, a CRDT by itself is not sufficient and must be supplemented by schemes for verifying the content of the data. A first approach consists in encoding the inputs to the system in a directed acyclic graph of events (an event DAG) whose edges represent causality relationships (Lamport, 1978). These edges are implemented by including in each event the cryptographic hash of its causal predecessors, which has the useful property that knowing an event of the DAG is enough to authenticate its entire causal past by recursively verifying hashes. Moreover, events are signed cryptographically by an authorized node, which allows nodes in the system to verify that events are valid. Once the validity of input events has been verified, a state resolution algorithm is used to compute the current state of the application, based on primitives similar to CRDTs. The Matrix network [7] is an example of a system based on this principle, and its formal analysis has been initiated in the academic community (Jacob et al., 2020; Jacob et al., 2021). This approach is very promising, but still has some limitations in the kinds of application that can be implemented over it, especially in the domain of the access control mechanisms it can express.

Limits of weak consistency: a counter-example

To illustrate why coordination-free mechanism are not general enough to implement certain kinds of access control mechanisms, we will use the concept of the *consensus number* (Herlihy, 1991) as a tool to study a simple example. Since we assume a message-passing architecture, a coordination-free system necessarily has consensus number 1. Our example system for access control is as follows: a group of users is defined as a set of public keys, which initially contains at least one public key. Any user whose public key is in the set may add the public key of another user to the set, and may remove any public key currently in the set. A user whose public key is not in the set may not modify the set in any way. We assume that we want sequential consistency for our implementation: indeed, if user a excludes user b from the set, and at the same time user b excludes user a from the set, only one of the two operations should be possible and either a or b should be in the final set. This means that one of the operation has to take priority, and once it is committed, the other one is disallowed by the system. We can trivially see that this

7. `https://matrix.org/`

system has consensus number at least 2: to build consensus between two nodes with this system, start with the set $\{a, b\}$ where a and b are the two nodes of the system. a and b then each request the removal of b and a, respectively. The decided value is the value proposed by a if the final set is $\{a\}$ and the value proposed by b if it is $\{b\}$. By adding any number of nodes to the initial set and making them all broadcast the removal of all other nodes, we can build a consensus primitive between any number of nodes. This simple access control mechanism is thus equivalent to a system that implements a full consensus system (consensus number ∞). Matrix is a coordination-free system that has consensus number 1, it is thus incapable of implementing such a simple mechanism. Matrix works around the issue by having "levels of power" for membership: members of a certain power level may add any other member with the same level of power as them, but may only exclude members that have strictly lower levels of power. This makes the above problem impossible as there is no situation in which two members can request operations to exclude one another.

1.2.4 Strongly consistent approaches: Blockchains

In order to be able to implement arbitrary application specifications in a trustless setting, and in particular to be able to implement access control mechanisms, we thus have no other solution than to build full-fledged consensus algorithms. Fortunately, the domain of trustless decentralized consensus algorithms is a very active area of research, with blockchain-based cryptocurrencies as the main application. The blockchain technology, introduced with Bitcoin in 2009 (Nakamoto, 2009b), allows to build a system that processes event in a totally ordered fashion (equivalent to consensus) with no central coordination. Blockchain technologies also have the desired properties that the network is fully open, in addition to being trustless, meaning that anybody can theoretically join the system. This surge of interest for cryptocurrencies has brought renewed attention to Byzantine fault-tolerant algorithms such as Byzantine consensus (Lamport et al., 1982). These algorithms are traditionally described in closed systems (also called permissioned networks) that have a fixed composition: N nodes, of which at most t are assumed to have Byzantine behavior (often with the assumption that $t < n/3$). As such, these algorithms cannot be directly applied to large-scale, trustless and open networks such as the ones we are interested in. Adapting them to this new setting has been studied in the context of cryptocurrencies, using Proof-of-Stake or Proof-of-Work to emulate the characteristics of a closed system of N nodes. In the case of Proof-of-Stake, participants control a certain

number of "virtual nodes" that is proportional to the stake they have invested in the system. In the case of Proof-of-Work, participants can create a number of "virtual nodes" proportional to their computing power. These methods necessitate backing the "power" allocated to nodes in the system (their weight in consensus votes) to a resource which is rare, such as computing power or cryptocurrency tokens. Note that the Bitcoin protocol itself can be analyzed as such, even if it is not a descendant of Byzantine-tolerant consensus primitives in closed networks: the computing power affected to compute the next block of a chain can be interpreted as a vote for this chain as the correct chain, thus the voting power of a participant is proportional to their total computing power. Unfortunately, we know of no consensus protocol that does not require backing the voting power of participants to a limited and measurable resource.

Limitations of current blockchain techniques

Blockchains, and in particular cryptocurrencies based on blockchains, unfortunately have many limitations and negative externalities, linked to the consensus protocols they use. In addition to their extremely limited throughput, Proof-of-Work blockchains are known to cause a race for computing power in order to reap the rewards of so-called "block mining", leading these systems to consume more energy than entire countries (Digiconomist, 2020). Alternative solutions such as Proof-of-Stake have been proposed, but these solutions have the unfortunate consequence of restricting participation in the network to nodes that can invest significant financial stake, creating a pyramidal structure that is strikingly close to a Ponzi scheme. We argue that these issues are mostly linked to the fact that they have been developed in the context of cryptocurrencies, using economic incentives to shape the behavior of participants. In fact, Section 2.5 of this thesis is dedicated to discussing the dangers of cryptocurrencies and the incentive structures that they create from a social, economical and political perspective.

Cryptocurrency-less blockchains

Our target, therefore, is to build consensus protocols that maintain the openness of the network as much as possible (contrarily to Proof-of-Stake), while preventing the waste of resources associated to Proof-of-Work. In particular, we will want to build consensus algorithms that are independent from cryptocurrencies, i.e. where users do not have to invest financial assets in order to participate, and where they are not necessarily financially rewarded for their participation (especially not for wasteful tasks such as Proof-of-Work

mining). This means that participating in the network should not be conditioned to expensive resources being acquired beforehand, such as specialized mining hardware for PoW mining or buying cryptocurrency tokens for staking. Further, since the kinds of systems we are trying to build are not intended to have monetary value, the security requirements can be lowered in comparison with cryptocurrency blockchains. Indeed, in a cost-benefit analysis, trying to steal cryptocurrency is potentially very profitable (but also very expensive). In comparison, taking control of a decentralized application leads to less immediate financial benefit, therefore it is less likely that an attacker will engage in important expenses to do it.

Note that in Bitcoin, there is also a role for monetary value in the necessity to limit spam transactions: indeed, Bitcoin implements a transaction fee mechanisms, where users have to choose how much fee they are willing to spend to validate a transaction, and miners will usually chose the transactions that pay the highest fees to be included in the blockchain. In a system that does not implement monetary value, it is not possible to rely on fees to block spam. This problem can be addressed by using a web-of-trust where users publicly share the trust levels they assign to other users, thus allowing anyone to obtain information on a participant's reputation. Such a mechanism is already in use in peer-to-peer networks such as Freenet [8], which uses a mechanism based on peer-to-peer captcha challenges for new users to bootstrap an initial non-negative reputation. Whereas spam protection is definitely an important topic, it is not the subject of this thesis.

The potential of epidemic consensus algorithms

A particularly interesting area of research in alleviating the issues with Proof-of-Work and Proof-of-Stake consensus algorithms consists in a new family of BFT algorithms that exploits epidemic mechanisms to provide large-scale protection against Byzantine behaviour. Epidemic algorithms allow for extremely fast dissemination of information in very large networks by means of stochastic peer-to-peer exchanges (Demers et al., 1987; Kermarrec et al., 2003). Epidemic BFT algorithms exploit this property by repeatedly sampling small sets of random peers in the network, which they then use to estimate the overall system's state, and ensure coordination and agreement between correct (i.e. non-Byzantine) nodes.

These algorithms were pioneered by Doerr et al. (2011), which provides a primitive for a single iteration of consensus using a selection rule based on median computations.

8. `https://freenetproject.org/`

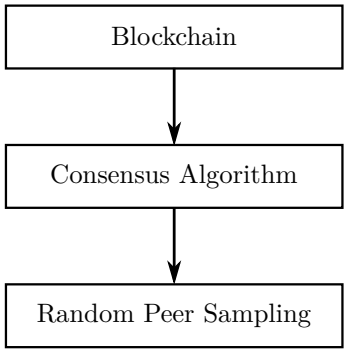


Figure 1.2 – An architecture we recommend for epidemic consensus algorithms using Blockchains

Recently, Avalanche (Team Rocket, 2018) was proposed as a new cryptocurrency that is based on an epidemic consensus mechanism where a selection is made between mutually exclusive transactions using a decision process based on metastability, i.e. the fact that a 50/50 state is not fully stable and that random perturbations will necessarily push the system in one direction or another. Once the system is sufficiently far from 50/50, the difference propagates to all of the nodes and the networks ends very rapidly in a 100/0 or 0/100 state. Other works in this domain have focused on using epidemics to implement a Byzantine reliable broadcast (Guerraoui et al., 2019b; Guerraoui et al., 2019a), a broadcast abstraction that is weaker than total order broadcast but still sufficient to implement money transfer.

The architecture we recommend is shown in Figure 1.2, and is fundamentally based on an *epidemic consensus algorithm*. Further, the consensus algorithm depends on a *random peer sampler*, which is a protocol that is able to find random nodes in the network and that is able to defend against attacks. Indeed, epidemic consensus algorithms need to exchange information with random peers to propagate it in the network, and are quite sensitive to the presence of malicious peers among the peers they exchange with. Therefore, there are two aspects to building practical epidemic consensus algorithms: the first part is ensuring that the algorithm has access to a reliable source of random peers in which malicious peers are not over-represented, and the second part is building the consensus algorithm proper.

Current techniques in this domain fall short on both aspects. Indeed, neither Doerr et al. (2011) nor Avalanche (Team Rocket, 2018) handle the random peers sampling aspect: they both assume that a perfect sampler is available that returns a malicious node with a given known probability that is reasonably small. The issue of Byzantine-tolerant

peer sampling has already been studied (Jesi et al., 2010; Bortnikov et al., 2009), but these previous approaches have limited applicability in practical scenarios. In particular, they assume that nodes in the network can only use a limited set of identifiers and that malicious actors only have access to a small number of these identifiers. In practice however, in an open system on the Internet, there is no way to prevent nodes from creating arbitrarily many node identifiers, a so-called Sybil attack, which would totally overwhelm existing peer sampling methods. A practical deployment of Avalanche in the form of the AVA networks [9] avoid this issue by using a peer sampler that is based on Proof-of-Stake. We are not satisfied by this solution as it means that AVA is necessarily linked to a cryptocurrency, which is a problem for us (among other things, Proof-of-Stake makes the system less open and leads to an oligopoly situation with few nodes controlling most of the consensus algorithm).

Similarly on the side of the consensus algorithm itself, many challenges remain to be solved. The method of Doerr et al. (2011) is not able to handle high throughput delivery and thus cannot be used in the kind of settings we are interested in. Avalanche does not provide a total ordering primitive, which means its application is limited to cryptocurrencies and it cannot handle smart contract platforms. We are not aware of other research in the domain of epidemic consensus algorithms, but we believe it to be a very promising field.

1.3 Our Contributions

Before entering considerations on total order broadcast in large-scale, trustless systems, we take a look in Chapter 3 at causally ordered broadcast in the context of closed N-node systems with Byzantine nodes. We present a formal definition of causal broadcast in Byzantine systems, which has been lacking from the existing literature, and show that it guarantees correctness and termination for the messages of correct nodes, independently of the causality relationships introduced by Byzantine nodes.

Furthering this work on causal broadcast, but also linking it with Blockchain applications, we propose in Chapter 4 a new data structure which we call a Merkle search tree. Chapter 4 is focused on showing how Merkle search trees can be used to implement causal broadcast in large-scale open networks but in the absence of Byzantine nodes. Merkle search trees can also be used in a blockchain scenario as a data structure for

9. `https://www.avalabs.org/`

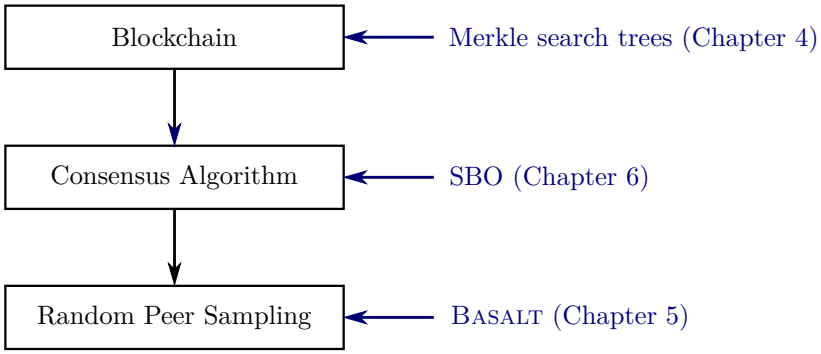


Figure 1.3 – Where the contributions of this thesis come into play

storing ordered information.

We then propose two significant improvements on the issues mentioned in the previous section, in order to build a totally-ordered blockchain-based consensus algorithm with an epidemic protocol. First, we propose Basalt (Chapter 5), a new peer sampling algorithm that tolerates Byzantine nodes and is resistant to Sybil attacks thanks to a mechanism that uses the properties of IP address space to limit the probability of sampling an adversary under realistic assumptions of their IP address distribution. Second, we propose SBO (Chapter 6), a new epidemic consensus algorithm that provides high throughput totally ordered delivery and is tolerant to Byzantine nodes. Figure 1.3 shows how these contributions align with the proposed architecture shown in Fig 1.2.

1.3.1 Accurately tracking causality: a tool for applications with weak consistency

While this thesis is primarily focused on building totally ordered delivery, studying the properties of weaker broadcast abstractions in presence of Byzantine nodes is of great interest. Indeed, new approaches which we discussed above such as Matrix, that encode events in a DAG that represents causality relationships, are a very promising alternative for the development of decentralized applications in a more lightweight fashion that does not require coordination between nodes.

Causal broadcast is a communication abstraction built on top of point-to-point send/receive networks that ensures that any two messages whose broadcasts are causally related (as captured by Lamport's "happened before" relation) are delivered in their sending order. In the tradition of Byzantine fault-tolerant literature, Chapter 3 is dedicated to studying causal broadcast in a closed system of N nodes with t Byzantine nodes. While this

situation is far from the open Internet-wide setting discussed above, studying the theoretical properties of causal broadcast in such a system is a first step to understanding its properties in a wider context.

We first give a formal definition of a causal broadcast abstraction in the presence of Byzantine processes, which we call BCO-Broadcast, in the form of two equivalent characterizations. We then present and prove correct an algorithm implementing it in an asynchronous message-passing system of N nodes with $t < N/3$ Byzantine nodes.

1.3.2 Merkle search trees: deterministic data structures for the global database that preserve order

As a continuation of our study on causal broadcast, Chapter 4 studies how a causally ordered broadcast primitive in the form of a causally consistent event store can be built in open networks subject to churn (but without Byzantine nodes). To this effect, we introduce a new data structure for anti-entropy which we call a Merkle search tree. We show that Merkle search trees can be used to improve the performances of data exchange in gossip-based anti-entropy systems that use a CDRT reconciliation rule. We compare this approach to a vector clock-based approach (Renesse et al., 2008) as well as to a Merkle tree construction that does not preserve order, and show that, in large networks and under low or moderate update rates, Merkle search trees provide the best trade-off: a 66% reduction of bandwidth usage was achieved in our simulation when compared to the vector-clock approach, as well as a 34% improvement in our consistency measure and a 32% improvement in 99th percentile delivery delay. When compared to Merkle trees without order, Merkle search trees were also better on all three metrics. In addition to being useful in the context of causal broadcast, Merkle search trees can also be used in the context of blockchains, in large-scale open networks and in presence of Byzantine nodes, as shown in Figure 1.3.

1.3.3 Basalt: Sybil-resilient random peer sampling for PoS-independent epidemic consensus

In Chapters 5 and 6, we return to the case of implementing totally ordered broadcast with blockchains, as a primitive for decentralized applications that is more powerful than causal broadcast, and to the question of how to implement it with epidemic algorithms as shown in Figure 1.2.

Epidemic BFT approaches critically depend on the availability of *good* network samples, in the sense that the proportion of Byzantine nodes in a sample should be kept as low as possible, and sampled nodes should be as varied as possible. Providing such samples is the role of a so-called *Byzantine-tolerant*, or *secure*, *random peer sampling* (RPS) service. When such a service is available, these algorithms have the potential to yield much higher throughput than PoW systems at a fraction of the cost (Team Rocket, 2018).

To ensure this security property, existing epidemic platforms for cryptocurrencies such as the AVA network [10] use a Proof-of-Stake approach to select peer samples. However, this choice limits the openness of the system as only nodes with significant stake can participate in the consensus, leading to an oligopoly situation. Moreover, this design means that only systems that are intrinsically based on a cryptocurrency can be built, a property we do not hold as desirable in the general case of collaboration and communication platforms.

In Chapter 5, we propose a radically different security design for the peer sampling service, based on the distribution of IP addresses to prevent Sybil attacks. We propose a new algorithm, Basalt, that implements our design using a stubborn chaotic search to counter attackers' attempts at becoming over-represented. We show in theory and using Monte Carlo simulations that Basalt provides samples which are extremely close to the optimal distribution even in adversarial scenarios such as tentative Eclipse attacks. Live experiments on a production cryptocurrency platform confirm that the samples obtained using Basalt are equitably distributed amongst nodes, allowing for a system which is both open and such that no single entity can gain excessive power.

1.3.4 SBO: high throughput, Byzantine tolerant, and totally ordered epidemic consensus

Epidemic consensus algorithms remain an emerging domain of research, with only a few known methods (Doerr et al., 2011; Matos et al., 2015; Team Rocket, 2018) and, to the best of our knowledge, none that provides high-throughput totally ordered delivery in the presence of Byzantine nodes.

In Chapter 6, we propose the first high-throughput Byzantine-tolerant epidemic total order consensus algorithm, which we call Simultaneous Block Ordering (SBO). SBO is based on epidemic dissemination and a novel ordering technique based on median computations that allows many messages to be processed simultaneously by the network.

10. `https://www.avalabs.org/`

Nodes repeatedly query their peers for the order of future messages, and adapt their local estimation of this order according to the information received. Nodes track the general agreement in the network about message order, and consider messages to be validated when they have stayed in the same positions for long enough, showing that the network is in a stable state of agreement on these positions.

Since SBO is a complex algorithm that is hard to analyze fully from a mathematical point of view, we focus on evaluating SBO from both simulations and a real-world implementation. Our extensive evaluation, based on simulations and a full-fledged prototype deployment, shows that SBO can scale to thousands of participants, and deliver a throughput of 5000 transactions per second and a block latency below 30 seconds with consumer-grade connectivity (100 Mbps) in a planetary deployment scenario involving 1000 nodes distributed over 20 cities.

Chapter 2

State-of-the-Art

The literature on distributed, decentralized and trustless computing is vast, and it would be impossible to summarize it exhaustively. This state-of-the-art review focuses on four particular topics that are relevant for this thesis: broadcast primitives in closed systems, gossip-based epidemic dissemination protocols, weakly consistent distributed objects, and blockchain technologies. A last section consists in an overview of socio-economical perspectives on cryptocurrencies and the issues they cause.

2.1 Broadcast Primitives in Closed Systems

This section focuses on the context of closed systems of nodes that communicate by asynchronous message passing. These systems are composed of a fixed set of N nodes of which a subset of size t is assumed to be faulty. These faults are generally assumed to be of either of two kinds: crashes only, or Byzantine faults. In a crash-only system, faulty nodes may stop their execution at any point of the algorithm, but it is assumed that until the point where they stopped, their behavior was correct: they acted accordingly to the algorithm that was implemented. In the case of nodes with Byzantine faults (called *Byzantine nodes*), these nodes may deviate arbitrarily from the specification prescribed by the algorithm (Lamport et al., 1982). In particular, Byzantine nodes might not send the same messages to other nodes than if they were correct: they may skip messages, duplicate messages, rewrite messages or introduce messages that were not prescribed by the algorithm. The internal state of a Byzantine node is not known and when proving a Byzantine-tolerant algorithm, no assumption is made on it. We reason on the state of correct nodes only, assuming that the t Byzantine nodes may send them arbitrary sets of messages. To make it possible to prove properties on Byzantine-tolerant algorithms, it is however assumed that nodes have point-to-point communication channels between them that cannot be spoofed: indeed, this allows a node to know with certainty the sender node of a message it receives.

Nodes that are subject to neither kind of faults (crashes or Byzantine faults) are said to be *correct*. In the context of trustless systems, malicious nodes may implement arbitrary behavior similarly to Byzantine nodes in a distributed system with Byzantine faults: we will then use the terms "Byzantine node" and "malicious node" interchangeably. Note however that nodes can have Byzantine behavior even in the absence of malicious intention: it can be the result of transient faults that altered the content of some local variables at some processes, thereby modifying their intended behavior in unpredictable ways.

2.1.1 Reliable broadcast

Reliable broadcast (Hadzilacos and Toueg, 1994) is a communication abstraction central to fault-tolerant asynchronous distributed systems. It allows each process to broadcast messages in the presence of process failures, with well-defined delivery properties. More precisely, it guarantees that correct processes deliver the same set of messages M, which includes at least all the messages they broadcast. This set M may also contain messages broadcast by faulty processes. The fundamental property of reliable broadcast lies in the fact that no two correct processes deliver different sets of messages. We will introduce a formal definition of this abstraction in Section 3.1.2.

Reliable broadcast in the presence of Byzantine processes (BR-broadcast)

Reliable broadcast has been studied in the context of Byzantine failures since the eighties. An elegant signature-free algorithm that implements the reliable broadcast abstraction in N-process asynchronous systems where the processes communicate by message-passing and up to $t < N/3$ of them may be Byzantine was proposed by Bracha (1987). Bracha's algorithm is optimal with respect to t-resilience, since $t < N/3$ is an upper bound on the number of Byzantine processes that can be tolerated (Bracha and Toueg, 1985; Raynal, 2018). From an operational point of view, this algorithm is based on a "double echo" mechanism of the value broadcast by the sender process. For each application message [1] it uses three types of protocol messages, requires three consecutive communication steps (one for each message type), and generates $(N - 1)(2N + 1)$ protocol messages. Another signature-free Byzantine reliable broadcast algorithm has recently been introduced

1. An *application message* is a message sent by the reliable broadcast abstraction, while a *protocol message* is a message used to implement reliable broadcast.

by Imbs and Raynal (2016). This algorithm implements the reliable broadcast of an application message with only two consecutive communication steps, two message types, and $N^2 - 1$ protocol messages. The price to pay for this gain in efficiency is a weaker t-resilience, namely $t < N/5$. Hence, these two algorithms differ in their trade-offs between t-resilience and message/time efficiency.

From single-shot to multi-shot Byzantine reliable broadcast

The original presentation of Byzantine reliable broadcast by Bracha allowed each process to send a message only once. However Byzantine reliable broadcast can be extended easily to handle the case of processes that are able to send multiple messages. This is done by replicating the Byzantine reliable broadcast algorithm in several instances identified each with a sequence number. We make use of this multi-shot BR-broadcast algorithm in Chapter 3. We introduced these multi-shot algorithms in Auvolat et al. (2021b), and they are also included in this thesis in Appendix A for completeness.

2.1.2 Causal order broadcast

Causal order broadcast is a communication abstraction that adds a first level of ordering guarantees on top of reliable broadcast. Denoted CO-broadcast (where CO stands for Causal Order), it was introduced by K. Birman and T. Joseph in a pioneering work on fault-tolerant distributed systems (Birman and Joseph, 1987). This seminal work, further discussed in Birman (1994) and Birman and Cooper (1991), considers only process crash failures.

CO-broadcast states that any two messages whose broadcasts are causally related according to Lamport's *happened before* relation (Lamport, 1978), are delivered in their causal sending order. Messages whose broadcast are not causally related can be delivered in different orders at different processes. From a causality point of view, CO-broadcast extends the FIFO property–which considers each channel separately–to system-wide causality. Like FIFO broadcast, CO-broadcast constitutes a multi-shot communication abstraction (namely, all the invocations of CO-broadcast are related by the same causality relation).

Theoretical characterizations of CO-broadcast can be found in Kshemkalyani and Singhal (1998), Murty and Garg (1997), and Raynal (2018). Algorithms implementing this communication abstraction can be found in Raynal et al. (1991) and Schwarz and

Mattern (1994) for failure-free asynchronous message-passing systems, and in Birman et al. (1991), Cachin et al. (2011), Mostefaoui et al. (2019), and Raynal (2018) for asynchronous message-passing systems where any number of processes may crash.

In Chapter 3, we will propose an extension of causal broadcast to systems with Byzantine faults.

Limit of causal broadcast In the context of Byzantine failures, it is possible for a Byzantine process that receives a protocol message carrying an application message to read the content of this application message, despite the fact that it might not yet have been BCO-delivered. This creates a kind of "insider trading" behavior that cannot be solved by BCO-broadcast alone. This problem was first addressed in Reiter and Birman (1994). Up to now, this notion of secure causality has been introduced only for total order broadcast in Reiter and Birman (1994), with further developments in Cachin et al. (2001) and Duan et al. (2017). More generally, we will see in Chapter 3 that, as for consensus, the definition of BCO-broadcast may need to be customized for the specific problem being solved.

2.2 Epidemics and Gossip Protocols

This section focuses first on methods for disseminating information in large-scale networks without a central server. We are particularly interested in very large scale systems which are open and thus prone to nodes leaving and joining frequently (a phenomenon called *churn*). In such a context, a simple broadcast implementation where the broadcasting node is responsible of sending the message to all other nodes individually through point-to-point channels is impractical due to, firstly, the cost of doing so for the sending node, and secondly, the difficulty of maintaining a precise list of recipients to which a broadcast message should be sent. These difficulties mandate the search for different dissemination protocols targeted particularly at this situation.

To this end, we first consider gossip protocols as an adaptation of broadcast protocols suited for large-scale open networks subject to churn. In the context of P2P file sharing, we also consider the case of large pieces of data which nodes may not want to access in their entirety: this poses the question of designing efficient data structures for distributed random access. For now we do not take into account aspects of coordination induced by distributed mutability of these data structures: we assume that data is produced by a

single node, and that all other nodes are simply consumers of that data.

2.2.1 Epidemic dissemination with gossip protocols

Gossip protocols, first introduced in 1987 (Demers et al., 1987), are information dissemination protocols that rely on stochastic peer-to-peer exchanges in a network. Concretely, nodes disseminate information by talking directly to one another and repeating information that they previously received, instead of depending on a single node to send the information directly to everyone. In its simplest form, when nodes receive a new information, they simply forward it to all their neighbors. This simple strategy is called a push-based flooding strategy. If the network is connected, i.e. if there exists a path between any two nodes, no node or set of nodes in the network should be isolated from the rest, and thus every node should receive the new information at least once. Moreover, the propagation time of the new information in the network is roughly proportional to the diameter of the network (the length of the longest path between two nodes). In certain random graphs such as Erdős–Rény graphs (Erdős and Rényi, 1959), the average diameter is logarithmic in the size of the network: information can thus be propagated to N nodes in $O(\log N)$ time with high probability. These properties are also true with different exchange strategies such as pull strategies, where nodes do not spontaneously send information to their neighbors but instead query their neighbors regularly to ask if they have any new information. These properties are well studied and make gossiping extremely efficient for information dissemination in large, structureless networks (Kermarrec et al., 2003; Eugster et al., 2004b; Kermarrec and Van Steen, 2007).

2.2.2 Random peer sampling

Gossip-based dissemination protocol can work on static predefined graphs, but they work best on random graphs where nodes all have a set of neighbors of about the same size. This size should be not too small so that the network doesn't become disconnected, and not too large so as not to send too much redundant information (Eugster et al., 2004a). Ideally, those neighbors are nodes selected at random in the network, thus building an Erdős–Rény random graph, so that the diameter of the network is minimized.

To build Erdős–Rény random graphs efficiently in a distributed fashion, a family of protocols called random peer sampling protocols (RPS) have been devised (Jelasity et al., 2007). These protocols are themselves gossip protocols in the sense that nodes exchange

information directly with their neighbors in the graph. However the nature of the information they exchange pertains to the structure of the network itself: two nodes exchange their lists of neighbors, so that nodes can discover the neighbors of their neighbors and thus obtain new random connections in the graph. Random peer sampling is generally dynamic, in the sense that nodes regularly close connections to their current neighbors and replace them with connections to new random nodes that were neighbors of their neighbors, so that the graph evolves and keeps a random structure over time even in the presence of churn (nodes leaving and joining the system).

The set of neighbors of a node is called the node's *view*. By selecting which nodes are part of a node's view, random peer sampling builds a so-called *overlay network*, i.e. a partial graph of connections between nodes that all have the possibility of talking to one another (because they all have Internet addresses). In comparison to other overlay network schemes such as distributed hash tables (DHTs), random peer sampling builds a structureless overlay network. On the contrary, DHTs require a precise structure to be built in the connectivity graph so that a particular node can easily be found in the network in logarithmic time. Many methods are known to implement structured overlays, including with peer exchange protocols very similar to the ones used for RPS (Voulgaris and Steen, 2013; Bouget et al., 2018).

Random peer sampling in trustless systems In the context of trustless systems, and in particularly on the public Internet, many security questions arise for random peer sampling protocols. Indeed, an attacker may be tempted to manipulate the random graph to their advantage, for instance by trying to appear more frequently in the views of honest nodes, or even to totally disconnect a target node from the network, a so-called Eclipse attack (Singh et al., 2006). This can be used to limit the propagation of information between correct nodes and replace it by false information, which enables the attacker to manipulate the victim. Such an attack can be achieved by several means, most notably by manipulating the peer sampling protocol to advertise more malicious nodes than correct nodes in order to artificially increase their connectivity. Further, random peer sampling can be a target for a Sybil attack (Douceur, 2002), i.e. an attack where an adversary creates many node identifiers (for instance by using many IP addresses), in which case the randomness property of the RPS will automatically cause them to be over-represented. In combination with an attack on the RPS protocol itself, this kind of attack can be extremely efficient in dividing the network to isolate nodes.

Random peer sampling in non-adversarial settings is a well-studied problem (Jelasity et al., 2007; Voulgaris et al., 2005). Similarly, defenses against Eclipse attacks in structured peer-to-peer overlays, such as DHTs, have been extensively researched (Singh et al., 2006; Urdaneta et al., 2011), typically by adding more structure to the selection process of overlay neighbors, or by outsourcing altogether this process to a trusted third party (McLachlan et al., 2009).

By contrast, and somewhat surprisingly, very few works have sought to develop Byzantine-tolerant RPS protocols. State-of-the art methods for Byzantine-tolerant RPS such as Brahms (Bortnikov et al., 2009) and Secure Peer Sampling (Jesi et al., 2010) are based on a classical RPS algorithm, to which is adjoined a mechanism that tries to correct for the over-representation of malicious nodes. In Brahms, the view is not updated if a peer has received more than a certain number of push messages in a given time slot, a measure meant to protect against nodes that repeatedly spam their identifier in the network. Brahms' approach can only work if we assume that malicious nodes have limited *attack force*, i.e. they may not send many more messages than correct nodes (see Section 5.3.2 for a formal definition), and must therefore target their attack on a specific victim node. Otherwise, they would be able to simply flood the whole network with many pushes and halt the peer sampling algorithm completely. In Secure Peer Sampling (SPS), nodes try to build some statistical knowledge on node behaviour; however, this mechanism is unable to cope with attacks where malicious nodes send so many messages that correct nodes do not have the time to gather sufficient statistics to block them before becoming isolated. The inherent limitations of statistical filtering techniques that can be used to correct for the biases in streams of identifiers in order to limit the over-representation of Byzantine nodes, such as those implemented in Brahms and SPS, were characterized formally by Anceaume et al. (2013a) in terms of bounds on the attack force of malicious nodes, and on the local memory available to honest nodes. This analysis however assumes a model where nodes can only use their local memory to store full node identifiers. Anceaume et al. (2013b) used count-min sketches (Cormode and Muthukrishnan, 2005) to bypass this limitation and better approximate an ideal RPS from a biased stream of identifiers. Further, the limitations proven in Anceaume et al. (2013a) only apply to a sub-model of peer sampling that does not take into account the retroaction between the identifiers selected at time t and the identifiers received at time $t' > t$.

In Chapter 5, we present Basalt, a new Byzantine-tolerant random peer sampling protocol meant to improve on this domain. We also propose a refined analysis that models

the dynamics of the network over time to take into account retroaction in time in the context of our proposed protocol, Basalt.

2.2.3 Gossip protocols in the context of blockchains

Many blockchain protocols use gossip-based epidemic dissemination to propagate new information between nodes. In the case of Bitcoin, for instance, a gossip protocol is used to propagate newly mined blocks to all nodes of the network (Heilman et al., 2015). Propagating this information fast and reliably is crucial to the network's efficiency and security, as it ensures that all nodes work on the same chain and are not losing their computing time working on alternative branches of the blockchain (which would reduce the overall efficiency of the network, as well as reduce the security as it increases the risk of one of these alternate branches of becoming the new longest chain, thus reverting part of the Bitcoin history – we discuss these points in Section 2.4.2). To ensure that the gossip protocols used by Bitcoin are safe, they implement many defenses in particular against Sybil attacks in order to limit the possibility of an attacker of disconnecting a node from the network (Heilman et al., 2015). These defense mechanisms try to make sure that nodes are connected to other nodes that are in a large variety of IP address prefixes, thus ensuring that the network is well connected, making it harder to isolate. However, some authors argue that these defenses are not enough in the face of new attacks such as BGP attacks that directly target the routing algorithm of the Internet (Maria et al., 2017), and new defense mechanisms that use alternative transmission channels have been proposed (Apostolaki et al., 2019).

To improve the dissemination efficiency of gossip algorithms in blockchain networks, i.e. to reduce the quantity of redundant information transmitted, while still ensuring authentication of the exchanged data, recent works have proposed protocols based on a combination of erasure coding and Merkle trees which enable to have both efficiency and security properties. Protocols such as RapidChain (Zamani et al., 2018) and HoneyBadger (Miller et al., 2016) make use of such methods.

2.2.4 Gossip protocols in practice: P2P file sharing

Many file sharing protocols have been devised in the last two decades that leverage direct peer-to-peer communication to ensure efficient dissemination of data among users. These protocols have had a wide success, with the BitTorrent protocol estimated to be

responsible for over 3% of the global Internet bandwidth at its peak in 2013 (Palo Alto Networks, 2013).

These protocols are based on the same mechanisms as gossip protocols, where nodes directly exchange pieces of information between one another. In order to allow for the exchange of large files, nodes rely on a mechanism called *content addressing* (Coulouris et al., 1972; Tolia et al., 2003). In content addressing, nodes that want to download a file must first obtain a compact representation of the file in the form of a hash or a list of hashes that represent the file's content. This representation must be obtained from a trusted source. Then, when nodes exchange pieces of data between them, they are able to validate that this data indeed corresponds to the file they are expecting to download by recomputing the hashes and checking that they match.

BitTorrent makes use of a simple mechanism of content addressing, where a file to be downloaded is represented by a list of hashes of the chunks of the file. The metadata composed of the file name and the chunk hashes is stored in the form of a torrent file. This file is much smaller than the eventual file to be downloaded, which enables index servers to store many torrent files and serve them directly to users that want to download content. When a node wishes to share a new file or folder to the network, it has to build the torrent file that represents the metadata of the files to be shared. Once this metadata has been produced and shared with other users, the content of the shared file or folder can no longer be modified. Note that torrent files can still be subject to scalability issue as the size of the torrent file grows linearly with the size of the downloaded file. Moreover, a torrent file can contain the hashes of many files to be downloaded, which makes it possible to share large directories of files with BitTorrent. However the size of the torrent file grows with the number of files, again limiting its use to directories that contain a reasonable number of files.

Other protocols based on distributed hash tables have allowed for extremely large pieces of data to be shared without the need to transmit full metadata out-of-band. A first approach (Tamassia and Triandopoulos, 2007) used a Merkle tree to share a key-value store in a P2P network, where tree nodes (which we will refer to here as *blocks*) are stored in a DHT organized by block hash, enabling nodes to efficiently find blocks of the tree to access the parts of the dataset they are interested in. In this first approach, the Merkle tree is built by a single node, and the root hash is signed by the author's private key. This allows the author to publish updated versions of the tree to implement write semantics. However the writes must still be done by this single publisher node; this system does not

support distributed modification.

IPFS [2] is a practical protocol that instantiates these same concepts, but instead of sharing a key-value store in the form of a Merkle tree, it stores a whole file system in the Merkle tree. In other words, IPFS allows one to share a folder of files from their system, and to continuously update this folder by updating the Merkle tree with new versions and signing the root hashes of these new versions with a version number. While this mechanism can be used to build decentralized websites that can be stored and shared by any node of the network, it also does not support distributed modification.

2.3 Introducing Distributed Mutability in Large-Scale P2P Systems

This section focuses on introducing mutability in peer-to-peer systems or gossip networks with approaches that only require weak coordination between nodes. In other words, nodes can act independently of one another when they wish to do a modification, and the data is then updated asynchronously on other nodes. This can be used to build eventually consistent systems (Vogels, 2008), but as we will see some such systems also allow for diverging points of view between nodes of the network, which is not necessarily an impediment to the system's function.

2.3.1 CRDTs and the CAP theorem

The advent of massive geo-replicated systems has prompted a growing interest in data replication techniques that are both weakly/eventually consistent and scalable. Among these, *Conflict-free Replicated Data Types*, or CRDTs (Shapiro et al., 2011), stand out by their ability to provide eventual consistency while offering a natural and modular programming paradigm to developers. CRDTs are a generic framework in which eventually consistent algorithms can be formulated. CRDTs allow replicas to perform operations concurrently and without any synchronization by providing resolution rules that allow concurrent operations to be combined a-posteriori. They can thus be used to build weakly-consistent systems with eventual consistency.

The CAP theorem (Fox and Brewer, 1999; Gilbert and Lynch, 2002) states that a distributed system may only achieve two of the following three properties: strong con-

2. `https://ipfs.io/`

sistency, availability and partition tolerance. CRDT-based systems typically forgo strong consistency in favor of availability, by allowing replicas to diverge temporarily. The particular CRDT used by a system defines a way in which replicas in divergent states can reconcile automatically to a unique shared state as soon as they are able to communicate reliably.

CRDTs are defined as a universe of possible states, with a merge operator that defines a join semi-lattice, so that two states that have been modified independently by different nodes can be reconciled at any time in a deterministic fashion, thus ensuring eventual consistency. CRDTs are generally expressed in the form of Delta-CRDTs (Almeida et al., 2014), where modifications are expressed as small elements of the same universe of states. This allows nodes to propagate updates much more efficiently than if they had to send their whole state to one another (Linde et al., 2016; Enes et al., 2019). CRDTs have been defined for many applications from simple sets and maps to collaborative text edition (Nédelec et al., 2013) to arbitrary JSON data structures (Kleppmann and Beresford, 2017), and even to email storage with IMAP access (Jungnickel et al., 2017).

2.3.2 Anti-entropy for peer-to-peer dataset synchronization

Delta-CRDTs crucially depend on the reliable delivery of all deltas to all nodes in the system. Recent delta-CRDT techniques in closed systems have focused primarily on leveraging causal broadcast primitives in order to ensure reliable causal delivery of operation deltas (Almeida et al., 2014; Linde et al., 2016). However this approach shows its limit in open networks where many nodes may join and leave, as causal broadcast primitives are not directly applicable in these kinds of networks. Causal broadcast has only recently been extended in a scalable fashion to networks with dynamic structures (Nédelec et al., 2018a) and the practicality of this approach has not yet been demonstrated.

In large-scale open distributed systems subject to churn, another option is to use a push-based gossip protocol to propagate CRDT deltas. In such a protocol, nodes that receive a new piece of information can propagate this information to random neighbors, ensuring that the information spreads the whole network. However, push-based gossip is probabilistic and deltas might not reach all nodes all the time, this scheme is thus not sufficient to implement eventual consistency. This method also does not answer the question of how a newly joining node can bootstrap its state to obtain the deltas that were propagated in the network before it joined.

A general solution to the issue of reliably updating CRDT state at all nodes in very

large systems subject to churn is therefore to build a system that allows two nodes to synchronize their states in order to detect and fix their differences, independently of which deltas they have or have not received previously and what their current state is. Such a protocol is called an *anti-entropy* protocol.

Anti-entropy protocols typically adopt an epidemic reconciliation strategy in which the nodes of a distributed system repeatedly reconcile their differences with randomly selected other nodes in order to converge to an agreed-upon state. This kind of approach was first introduced by Demers et al. (1987); in the more modern framework of CRDTs, anti-entropy consists in implementing the full CRDT state merge operator between two remote nodes that have no prior knowledge of one another's state. The question, however, is how can this merge be implemented efficiently between two remote nodes, i.e. without having to send the whole state from one node to the other in the case where only a small number of differences exist between the two states.

Overview of anti-entropy protocols Many protocols for anti-entropy, such as the one used by DynamoDB (DeCandia et al., 2007), use Merkle trees to efficiently find the data items that are different between two copies of a dataset. Other methods for remote dataset comparison, which are not based on Merkle trees, have been proposed.

Byers et al. (2002) introduced approximate reconciliation with Bloom filters (Bloom, 1970) to detect set differences between two nodes. They then applied bloom filters to tree data structures, resulting in a method that is still approximate and uses less bandwidth. Minsky et al. (2003) introduce an algebraic method of recovering missing elements in a set by using polynomials. Their method is optimal in communication costs, however it requires complex computations that scale linearly with the number of items. Another important limitation of their method is that they require to know in advance the number of items missing from one node to the other, or they use as much unnecessary bandwidth as their over-approximation margin.

Logical clocks, introduced by Lamport (1978), have been widely used for detecting differences between versions at two nodes. The algorithm of Renesse et al. (2008) detect missing updates by exchanging digests which contain the last update number that has been seen for all participating nodes. DottedDB (Goncalves et al., 2017) is a system that uses vector clocks to detect updates, making use of optimized clocks that allow for smaller representations. All these method allow precise calculation of missing updates, however they do not scale well in systems with many participants as they require keeping timing

information about all participants, past and present, and comparing all that information at each anti-entropy round.

Merkle causality DAG approaches as are used in Matrix have also been explored in this context (Sanjuan et al., 2020).

We will show in Chapter 4 that the classical Merkle tree approach is inefficient in the case where modified items are close together according to a certain database field (e.g. their timestamp), and we will build a new Merkle tree data structure based on search trees to solve this issue.

2.3.3 Limitations of CRDT-based techniques and possible solutions in trustless networks

The fundamental limitation of CRDTs is that they are not adapted to trustless settings where malicious nodes may want to interfere arbitrarily with the dataset. Indeed, the basic definition of CRDTs includes no way of verifying that updates are produced by an authorized participants, in order to select only a subset of operations to integrate. While there exist methods to ensure remote verifiability of conflict-free operations (Cachin and Ohrimenko, 2018), it seems that there is no simple way to add such a mechanism to CRDTs without severely interfering with their definition and properties. The method of Cachin and Ohrimenko (2018) also does not target the same trust structure as us: they assume a set of mutually trusting clients and a single untrusted server, whereas we assume either a set of mutually distrusting nodes, or a set of mutually distrusting servers trusted only by their local clients.

In this section we will study existing non-CRDT techniques through the lens of the schema presented in Figure 1.1. In this schema, nodes are able to identify the author of individual operations (also called primary sources) using digital signatures. Assuming that nodes are able to retrieve the inputs provided by many users, the fundamental question is thus how to integrate these inputs together in a global database that represents the overall system state, while taking into account authorization rules that may be in part defined within the system itself (such as if the system defines a list of members with authorized write access) and preventing inconsistent or conflicting operations from being selected. This can be done either by reproducing the full integration process locally at each node, or with a partial trust graph where nodes trust the results of the integration made by certain specific other nodes (we however do not know of any system that implements this

second option).

ZeroNet: a simple approach

As we saw in Section 2.2, content addressing can be used to securely disseminate information whose hash is known. This hash can be authenticated using public key cryptography, removing the need to obtain it from a trusted third party. This means that nodes may download information from any other node in the network and still be able to authenticate its origin. Further, data mutability can be implemented by adding a simple versioning scheme to the hash. This limited form of mutability only allows one user to modify the data, as the new version's hashes still have to be signed with the same key. To build a system that integrate information from many users, a first simple solution thus consists in isolating the contributions of users into individual versioned files, one for each user signed with their cryptographic key.

This is precisely what is done by the ZeroNet network [3], which targets principally social networking applications. More specifically, in ZeroNet the data files of each user consist simply of a list of database records. Moreover, data files are specific to an application, called a *zite*, which allows users to download only the data files relevant to the application they want to interact with. When a user wants to interact with an application, the node they are running downloads the files of all users of the application in the network, and stores the union of these contributions in a local database. This allows the user to have a unified view built locally that contains the contributions of all users. In the case of ZeroNet, this can be used for example to build a discussion forum: each message comes from the signed data file of its author and all contributions are merged in a single local database so that threads of discussion can be reconstituted. This method can be easily used to build open networks where everyone can contribute anywhere.

This method has many limitations, in particular in terms of scalability. Indeed, as soon as an application (a "zite") becomes too large in terms of data being stored, nodes become overwhelmed by the necessity of downloading and indexing all of the data of the application in order to interact with it. This makes it especially costly for new nodes to join the network, as they must bootstrap from scratch and download large amounts of data. This is further worsened by the fact that the network is fully open, and in many applications anyone is allowed to participate. Nodes by default replicate the contributions of everyone, which worsens these issues of full replication at every node.

3. `https://zeronet.io/`

Note also that ZeroNet is not good at establishing group-based access control, as by default everyone is free to contribute in an application. The best it can do is hide the contributions of known-malicious users, which can be done in a cooperative fashion with users sharing their block lists between one another.

Secure Scuttlebutt: almost (but not exactly) the same

Secure Scuttlebutt (SSB [4]) is another social network that works on similar principles. Instead of publishing their contributions in a versioned file, which allows users to modify and delete old contributions, SSB requires each user to publish their contributions in the form of a hash-linked chain (similar to a blockchain, but authored by a single user), where each item in the chain is signed by the private key of the user. This can be used to ensure that the set of messages emitted by a user is an append-only log: indeed, a node will refuse to handle an operation chain for a user that is in conflict with (is a fork of) the previous chain it knew of. This however does not ensure global consistency of the network in the case where a user produces two forking chains: indeed, if a user sends a chain c_1 to a certain node and a chain c_2 to another node, where c_1 and c_2 are incompatible (they make a fork), then the two nodes will cling to the two incompatible chains c_1 and c_2 respectively, and will never be able to converge back to a common state.

SSB is also subject to the issues of scalability present in ZeroNet. However it is better at managing spam and malicious behavior, as by default it makes use of a trust graph to limit propagation of information. The trust graph is defined by the "following" relationships established in the social network between users, and nodes only replicate the data of users that are at a distance 3 or less on this graph. Moreover, contributions of users at a distance strictly larger than 2 are hidden by default. This means that if a user's friends are trustworthy, and so are the friends of their friends, then they will only see content by trustworthy users and no spam or other unwanted content.

Matrix: introducing conflicts and resolving them at the same time

The Matrix protocol, which we evoked in Section 1.2.3, tries to implement more complex semantics using a directed acyclic graph of events linked by their causality relationships (an *event DAG*) and a state resolution algorithm that determines how events are applied in case of merging after a fork. The Matrix protocol uses one event DAG per

4. `https://scuttlebutt.nz`

room (per discussion channel), and users may join any number of rooms. The room is thus a small unit of decomposition of content in Matrix that allows it to scale relatively well to large networks composed of many rooms, as long as individual rooms remain small enough.

In the Matrix protocol, events can be of two types: message events, and state change events. Message events represent messages being sent by users in a room (a discussion channel), as well as message modification and deletion events, while state change events represent events that modify the control structure of the room, which is represented in the form of a key-value map. This key-value map, called the room *state*, contains the list of members in the room and their associated power level, as well as other useful information such as the name of the room, the room avatar, etc. The list of members is used for access control, to ensure that messages that are sent are indeed authorized in the room, and that state changes such as joins, leaves or power level changes, are allowed.

Events are encoded as JSON objects that contain the hashes of their immediate causal predecessor, thus forming a Merkle event DAG where edges represent causality relationships. This allows independent servers to add events to the graph without coordinating with one another; diverging branches of the DAG can be resolved later by a merge event (an event that has several immediate causal predecessors).

The state of a room (a key-value map) is defined at each event in the DAG. When a state change event has a single causal predecessor, calculating the state at the new event is easy: the state modification is simply applied to the state of the causal predecessor. In the case of a merge event, which has several causal predecessors, a state resolution algorithm is applied. This algorithm, described formally in the Matrix specification (Matrix Foundation, 2021), sorts events in the diverging branches in a deterministic order that favors operations made by highest-privileged users. In such a way, when conflicting operations are made on two branches (such as a user sending a message on one branch, and a moderator banning the user on another branch), the resolution algorithm can chose to reject one of the conflicting events (in our example, the message sent by the banned user would be rejected). In the case of conflicting state changes, this can be used to calculate the merged state on which the merge event is applied. A formal analysis of the Matrix state resolution has been initiated (Jacob et al., 2020; Jacob et al., 2021), and our study of Byzantine causal broadcast (Chapter 3) could also provide some insights on the Matrix protocol.

As we said, the events in the Matrix network are split by rooms, each having their own

event DAG and thus state resolution can be applied independently on each room. In the current version of the Matrix protocol, state resolution has to be applied locally by each homeserver (a homeserver is the server that a set of users use as an entry point to interact with the network). Users trust their homeserver to do the state resolution correctly. This system therefore implements a model where clients trust their homeserver but servers do not trust one another. The Matrix protocol has known issues of scalability, in particular in the case of very large rooms. Similarly to the ZeroNet and SSB protocols discussed above, a node joining the network must download all events in a room they want to join and must fully replicate the state resolution algorithm in order to build a consistent view of the room's state. Extremely large rooms such as the Matrix headquarters are known to cause issues with this initial process on new homeservers that join the network. Fortunately, this burden is only to be borne by servers and not by individual users such as in SSB or ZeroNet. The protocol is therefore much more manageable for end users. In terms of discoverability, Matrix rooms can link to one another, with a new kind of rooms called "communities" that are expressly designed to contain directories of other rooms to be found easily. This makes it easy to explore the Matrix network to discover new content.

2.3.4 The special case of money transfer

Among the applications that can be built in a decentralized network without consensus, money transfer (or asset transfer) is a particularly striking example, which has become widely known thanks to cryptocurrencies. Money transfer is defined as a distributed object where users have accounts and each account has a balance in a certain currency. Users may transfer money from their account to other accounts provided that the balance of the sending account stays positive after the transfer.

Since the inception of Bitcoin (Nakamoto, 2009b), it has been generally assumed that a cryptocurrency requires a total order consensus algorithm to be built, as it appeared to be the only way of preventing double-spend attacks (a double-spend attack is when a user broadcasts two transactions that spend the same funds – of which only one can be valid; this attack is made impossible by having a network-wide mechanism to chose only one of the two transactions). However, it has been shown that the consensus number of an asset transfer distributed object is in fact 1 (Guerraoui et al., 2019c) in the case where at most a single process is allowed to withdraw from any given account. Indeed, a user that wishes to withdraw from their account will never be prevented from doing so by other users withdrawing from other accounts. A user therefore does not need to

coordinate with other users when they wish to spend their money: if they have enough money, they can simply broadcast that transaction and it will be handled independently of what other users are doing. The only requirement to ensure that money transfer can be built reliably in presence of Byzantine nodes is to have a Byzantine reliable broadcast (BR-broadcast) primitive, which is used to order the transactions *of a single user*: the BR-broadcast primitive is necessary to ensure that, at a certain time step, a certain user has only broadcast one single transaction and all nodes agree on what transaction this is. This primitive can be implemented reasonably simply in a closed asynchronous message passing system of N nodes, with $t < N/3$ Byzantine nodes, as discussed above (Section 2.1.1). In a paper which we did not include in this thesis, we present a simple algorithm for money transfer in presence of Byzantine nodes relying only on BR-broadcast (Auvolat et al., 2020). Note in particular that this algorithm does not even need to encode causality relationships between transfers, as the current balance of an account is enough to know if an incoming transfer should be applied immediately or put in a queue awaiting sufficient funds to arrive.

This new discovery has led to a variety of systems that implement money transfer without requiring consensus (Collins et al., 2020; Baudet et al., 2020). Contrarily to what was thought for a long time, full-fledged blockchains might not be necessary in order to implement cryptocurrencies. However, implementing a reliable broadcast abstraction in an open setting is still an unresolved problem. The two approaches of Collins et al. (2020) and Baudet et al. (2020) assume either a closed system or a closed set of validator nodes, each time with $t < N/3$ Byzantine nodes at most. Note also that this system cannot handle accounts from which multiple users may withdraw. In the general case of a cryptocurrency, a transaction from an account may be made by anyone who possesses the private key associated to the account. In the case where several computing nodes are able to emit such a transaction, they must coordinate when doing so, otherwise they risk locking the account by sending contradictory information (this will cause the reliable broadcast to not terminate), thus losing all of the associated funds. In fact, in the case where the set of nodes authorized to withdraw from an account is of size k (for instance if only k nodes know the private key associated with an account), it has been shown that the money transfer object has consensus number k (Guerraoui et al., 2019c). Lastly, note that this property of the money transfer distributed object is not transferable to other blockchain applications such as smart contracts.

2.4 Strongly Consistent Co-authored Distributed Objects with Blockchains

There are several kinds of decentralized applications that have a consensus number larger than 1, and therefore cannot be built using weak coordination methods. In particular, smart contract platforms fall into this category as they should allow one to implement an arbitrary state machine in a strongly consistent fashion. In the context of collaborative systems, we have shown in the introduction (Section 1.2.3) that access control mechanisms where two members may mutually exclude one another from a group also cannot be implemented without consensus.

Having determined the necessity of decentralized trustless consensus in the general case, this section focuses on reviewing existing algorithms, and in particular blockchain-based algorithms for trustless consensus in large-scale open networks.

2.4.1 Architecture of a blockchain

A blockchain is originally defined as a sequence of *blocks*, produced one after the other, and linked together with hashes: the N-th block of the chain contains the cryptographic hash of the $N-1$-th block. By recursively downloading previous blocks and checking their hashes, this means that knowing the block N allows one to verify the content of all preceding blocks (N is called the *height* of the block). A block of a chain is therefore intrinsically linked to its entire history.

In addition to the hash of their predecessor in the chain, blocks contain a timestamp, a set of operations (also called transactions), and the Merkle root of a tree that represents the state of a state machine after applying the operations of the block. In addition to the rules that link blocks to their predecessors using hashes, a block's validity is conditioned to verifying that the Merkle root of the state is indeed the proper result of applying the operations of the block to the state of the previous block in the defined state machine semantics. This allows for the building of applications that use blockchains to securely implement arbitrary state machines. The first block in the chain, called the *genesis block*, must be agreed upon initially by all participants of the blockchain. Once it is agreed, any chain of blocks that links back to this block and that correctly implements the state machine semantics is valid. This structure is illustrated in Figure 2.1.

Blocks in a blockchain link to a single predecessor. However since there is no rule

to control who can create a block, there can be several blocks that link to the same predecessor: the blockchain is thus in a forked state, with two concurrent blocks existing simultaneously in the network at height N. Blockchain algorithms employ a *consensus algorithm* to decide which of the concurrent blocks to select as the valid block at height N.

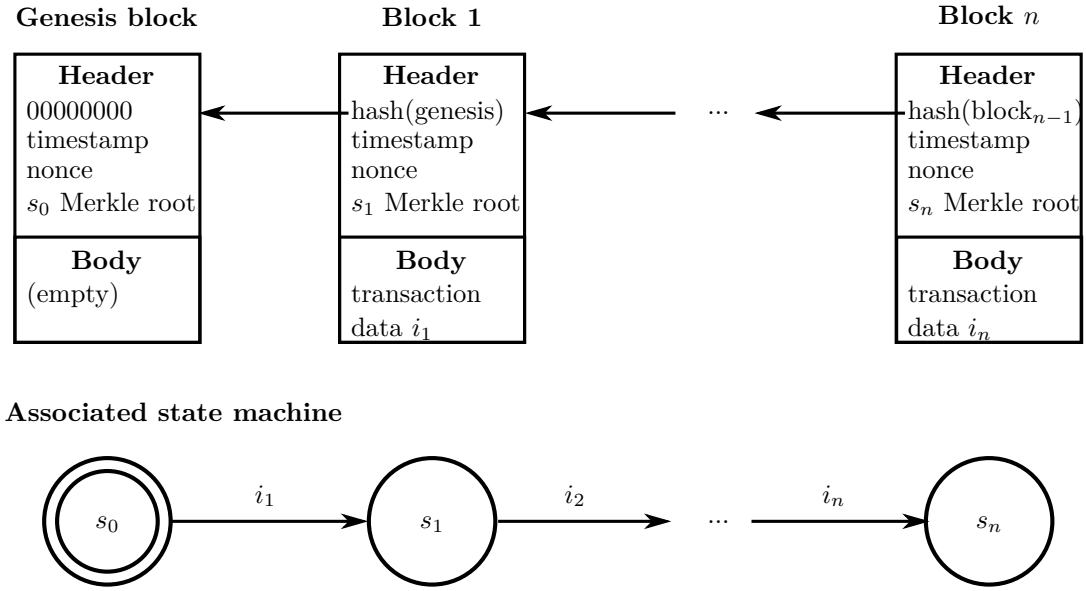


Figure 2.1 – Structure of a Blockchain implementing a state machine

2.4.2 Proof-of-Work consensus algorithms

To determine which of two forks is the correct blockchain that nodes should consider as the current valid branch, blockchain technologies use techniques based on voting of nodes. In the case of Proof-of-Work, as introduced by Bitcoin (Nakamoto, 2009b), voting for a chain is done by producing a successor to that chain: in case of conflict between two valid chains, the longest chain wins. However, Proof-of-Work adds a validation rule for blocks that makes it a hard task to build successors to a chain. Indeed, blocks now contain a special value called a *nonce*. To produce a valid block, a node must select a nonce so that the hash of the block (including the nonce) contains a large enough number of zeros in leading position (i.e. a prefix constituted only of zeros). Since there is no way to efficiently invert a hash function, the only way to find such a nonce is to try a large number of possible values. On average, to build a block whose hash has k leading zeroes (in binary representation), 2^k nonces must be tried, and the associated 2^k hashes must

be computed. This way, a valid block can only be the result of brute-forcing a sufficiently large number of nonces, and finding a nonce that provides the desired amount of leading zeros is considered as a *proof* of having expensed enough computing cycles to produce a valid successor (thus the term of *Proof-of-Work*). In Bitcoin, the number of leading zeros required to consider a block valid is determined by an auto-adaptive algorithm which updates the so-called *mining difficulty* every two weeks. The rule that was set in the Bitcoin network is that the difficulty is adapted so that the average interval between two blocks should be 10 minutes. Thanks to this rule that makes blocks hard to produce, there is true "value" in the fact of producing a block (the network cannot produce an unlimited quantity of them), and the consensus rule can be simply to select the longest of the two forks of the blockchain. Indeed, if one fork contains more blocks than another, it means that more nodes have devoted computing power to calculating the next block in this chain, and therefore that more nodes agree that that chain is the "truth" in the network.

This protocol has been subject to theoretical analyses, for example by Garay et al. (2015). It is generally agreed that the network is secure as long as no miner controls more than 50% of the hashing power. Indeed, if that were not the case, the node that controls more than 50% of the mining power could always overturn decisions ratified in the last produced blocks, by producing concurrent blocks faster. This would allow the attacker to pull off a double-spend attack: indeed, they could send a transaction sending their funds to a person A, and this transaction would appear to be validated in the network. At that point they could obtain a compensation for the funds they spend. They could then produce a concurrent block where the same funds are not sent to A but to B instead, and make it to that this concurrent chain is longer than the first one. The transaction to A would then be erased from the network, meaning that the attacker received compensation without having spent anything. It has been argued that attacks of this kind could be achieved even with less than 50% of mining power, for instance in so-called *selfish mining* attacks (Sapirshtein et al., 2016). Fruitchains (Pass and Shi, 2017a) proposed an improvement to the Bitcoin mining mechanism to improve fairness in Proof-of-Work blockchains. In practice, it is generally assumed that the original Proof-of-Work from Bitcoin is a "secure enough" mechanism for blockchains, and the two blockchains that hold the most value today (Bitcoin and Ethereum [5]) are secured by this consensus

5. According to `https://coinmarketcap.com/` (accessed 2021-04-01), the value of all Bitcoins in existence reached over 1 trillion USD in 2021.

mechanism.

In order to incentivize users to produce new blocks, Proof-of-Work is generally associated to a reward mechanism, where each node producing a block obtains a certain quantity of newly minted coins, as well as the coins that were set aside as transaction fees for the transactions embedded in the block. This has the perverse consequence of leading to a computing power arms race, where each miner tries to obtain the biggest possible share of hashing power in the network in order to reap the rewards of producing the next block as often as possible. As a consequence, the Bitcoin network is estimated to consume the same amount of electricity as a small country such as the Netherlands (Digiconomist, 2020).

In Section 2.5, we will spend more time trying to understand what is wrong with such a system, not only from the perspective of computer science, but from broader perspectives of economics, sociology and ecology. For now, we will retain the following main drawbacks to Proof-of-Work blockchains:

— **Energy consumption.** The network depends on it being hard to produce new blocks in the blockchain for its safety. However by choosing to back this "hardness" to the computing power of participants, Proof-of-Work requires nodes to expense many compute cycles doing repetitive calculations in an arguably wasteful manner.

— **Limited throughput.** The number of transactions that can be processed by a Proof-of-Work system is limited by two parameters: the average interval at which blocks are generated, and the maximum size of a block. In practice in Ethereum and Bitcoin, this means that the throughput of the system is limited to the order of 10 transactions per second. In comparison, commercial payment processors such as VISA are able to process several tens of thousands of transactions per second at peak demand.

— **Important latency.** A transaction in a Proof-of-Work system is only confirmed once a block has been mined that includes that transaction. Further, to ensure that the mined block will not later be replaced by a fork in the blockchain, it is generally recommended to wait for several blocks to be mined in the same chain after the one that contains the transaction. With a block interval of 10 minutes on average, this means that a Bitcoin transaction cannot usually be considered definitely valid before a delay of one hour.

— **Storage requirements:** A blockchain acts as a distributed ledger that records all transactions that have occurred in the network, forever. Current blockchain

technologies require keeping such records for validation purposes: a node that joins the network must be able to compute independently the current value of users' accounts without having to rely on a trusted source of information.

2.4.3 Committee-based blockchains

To solve some of the issues with Proof-of-Work blockchains, many works have sought to adapt classical Byzantine fault-tolerant (BFT) algorithms in permissioned systems to the context of permissionless networks. Indeed, solving Byzantine consensus (Lamport et al., 1982) allows one to build a total order broadcast primitive, which is the same thing that a blockchain implements. Many algorithms are known to solve this problem in permissioned networks composed of N nodes, of which at most $t < n/3$ are Byzantine (Castro, Liskov, et al., 1999; Guerraoui et al., 2010; Bessani et al., 2014). These algorithms usually have good properties in comparison with PoW blockchains, such as fast validation time only limited by the actual propagation delays in the network (a property called *responsiveness*). However, these algorithms are not suited to the requirements of modern blockchains. Indeed, in addition to comporting Byzantine nodes (nodes that do not respect the agreed-upon specification and may instead have an arbitrary behavior), modern blockchain networks are open networks in the sense that anybody can participate. This means that the assumption that the system is composed of N nodes, of which at most t are Byzantine, cannot be made. This assumption is however at the core of all of these BFT algorithms: indeed, they rely on properties such as quorum voting to ensure that nodes obtain consistent information from one another and are able to implement agreement.

To adapt these BFT consensus algorithms to the open context of blockchains, recent methods have proposed to use *committees*, i.e. small well-defined sets of nodes in the network that act during a limited period of time as the members of a permissioned system in which a classical BFT consensus algorithm is run. The selection of committee members is made with a probability distribution that is backed by a measurable property of participants. For instance, in so-called "hybrid approaches" (Andrychowicz and Dziembowski, 2015; Pass and Shi, 2017b; Zamani et al., 2018), a Proof-of-Work mechanism is used to determine the composition of committees. The transaction validation itself is then performed by the committees using a BFT algorithm such as PBFT (Castro, Liskov, et al., 1999) which has the responsiveness property. This means that these methods successfully alleviate the latency and throughput issues of simple Proof-of-Work-only blockchains.

These methods however do not solve the issue of energy consumption, as nodes are still required to expense many compute cycles in order to obtain a chance of participating in a committee.

A wide variety of committee-based algorithms has been developed (Kwon, 2014; Miller et al., 2016; Gilad et al., 2017; Pass and Shi, 2017b; Buchman et al., 2018; Zamani et al., 2018; B. Guo et al., 2020). We will not enter in the details of these methods as they are in fact quite complex, and the subtleties of the differences between them are not particularly relevant for this thesis. Further, maintaining committees and ensuring consistency when committees change (which they often do for security reasons) is a hard problem that introduces a lot of software complexity. We argue that this complexity is a threat to a proper understanding of the protocol to validate its safety, and that algorithms that do not use committees are preferable for this reason. The consensus algorithm we will introduce in Chapter 6 does not use committees; instead, all nodes are continuously part of the consensus algorithm. This algorithm is intrinsically insensitive to churn thanks to its epidemic nature, meaning it does not need to keep track of current network membership to function correctly.

2.4.4 Proof-of-Stake

To solve the energy consumption issue of Proof-of-Work, what seems to be considered the best solution to this day is to use committee-based consensus algorithms with a committee selection mechanism called *Proof-of-Stake*. In Proof-of-Stake, the probability distribution from which committee members are sampled is not defined by the computing power of nodes, but by the amount of cryptocurrency tokens that they have locked as "stake" in the network. By staking more tokens, nodes can increase their probability of being selected in a committee; nodes are able to unlock their staked tokens when they no longer participate in the selection process. Further, this staking can be used to incentivize nodes to participate correctly in the consensus protocol: indeed, a node that has had a Byzantine behavior can lose all of its staked tokens if another node is able to provide a cryptographic proof of its misbehavior.

Many different implementations of Proof-of-Stake have been proposed. In Algorand (Gilad et al., 2017), the committee creation mechanism uses Verifiable Random Functions (VRF) (Micali et al., 1999) to determine in each round which nodes should participate in this round's consensus, with a probability that is proportional to each user's stake. When successfully executed by a node p, the VRF algorithm of Algorand provides p with a proof

that p is entitled to participate in the round's committee, which it can then broadcast to the network. This has the property that the nodes that are part of a committee are not known in advance, and therefore it is impossible for an attacker to try to corrupt them in advance. This improves the resilience of the system in comparison to other implementations of Proof-of-Stake such as Ethereum 2.0 (which is yet to be launched), where committee members are selected for epochs that are constituted of precisely 32 consecutive consensus rounds, for a total duration of 6.4 minutes during which an adversary can try to corrupt validator nodes.

However, Proof-of-Stake comes with its own set of challenges and limitations. By restricting participation in the consensus protocol to nodes that have significant stake invested in the network, PoS abandons the democratic ideal of true openness that led to the creation of Bitcoin (Nakamoto, 2009b). Indeed, by mandating that new participants buy significant amounts of pre-existing assets before being allowed to enter the system and mint their own cryptocurrency tokens, PoS effectively creates a pyramidal incentive not dissimilar from that of a Ponzi scheme. We will argue in Section 2.5 that this situation is not tenable for cryptocurrencies, and that cryptocurrency platforms must be free to experiment with new economic schemes. These considerations directly lead to the investigation of consensus algorithms that *do not depend on any particular economic structure*, and in particular that do not use Proof-of-Stake, while still achieving high throughputs at a low cost. In particular, epidemic algorithms forgo the requirement of a closed, N-node system of classical distributed algorithms, which committee-based algorithms emulate, and replace it by a dependency on a random peer sampling protocol designed to withstand Byzantine behaviors and Sybil attacks, such as the Basalt protocol we introduce in Chapter 5.

2.4.5 Epidemic consensus algorithms

Epidemic algorithms promise to solve most of the issues with committee-based, Proof-of-Work and Proof-of-Stake methods described above: they are a well-known method to achieve efficient and reliable information dissemination in extremely large systems (Eugster et al., 2004a). Epidemic algorithms function with gossip protocols, where each node communicates with a small set of neighbors, letting information propagate from node to node until the whole network is covered. Adapting these protocols to build consensus algorithms is a novel and promising area of research.

One breakthrough came from the observation discussed in Section 2.3.4 that cryptocur-

rencies do not in fact require a total order broadcast (Guerraoui et al., 2019c), but instead can be implemented for instance on top of a plain Byzantine reliable broadcast (Guerraoui et al., 2019a; Auvolat et al., 2020). Guerraoui et al. (2019a) leverage this in an epidemic setting where the Byzantine reliable broadcast is implemented by a gossip algorithm, thus allowing them to profit from the high scalability of epidemic systems. Avalanche (Team Rocket, 2018) also uses an epidemic approach to build a highly scalable cryptocurrency protocol, but with a significantly different take: instead of relying on the Byzantine reliable broadcast primitive, they implement a binary consensus primitive that is used to select between conflicting transactions. By forgoing total order and letting non-conflicting transactions form a DAG, Avalanche is able to reach high throughputs, but this makes their approach unsuitable for applications that require a full-fledged BFT replicated state machine (and thus total order) which we are interested in here.

Totally ordering messages with epidemic algorithms is not a new problem, but previous approaches fall short of the requirement of modern blockchain systems. EpTO (Matos et al., 2015) for instance is able to provide high-throughput totally ordered epidemic message delivery, but is not tolerant to the presence of Byzantine nodes. Doerr et al. (2011) propose a primitive for a single instance of Byzantine-tolerant epidemic consensus, using an aggregation rule based on the median, but they do not propose an extension to several consecutive broadcasts with low latency and high throughput.

In Chapter 6, we will introduce SBO to improve on this domain. To our knowledge, SBO is the first Byzantine-tolerant high-throughput epidemic totally ordered delivery algorithm.

The epidemic algorithms cited above depend on a random peer-sampling service that has a low probability of sampling Byzantine peers. The peer-sampling must thus be immune to Sybil attacks. The AVA cryptocurrency network [6] (based on the Avalanche protocol) uses Proof-of-Stake to provide Sybil-resilient random peer sampling. In this thesis, we propose instead the Basalt algorithm (Chapter 5) as a replacement that builds a Byzantine-tolerant and Sybil-resistant RPS algorithm directly on IP networks by using the properties of the IP address space. This allows us to build consensus algorithms that do not need to be linked with a cryptocurrency to operate in the trustless, open network setting.

6. `https://www.avalabs.org/`

2.5 Cryptocurrencies are a Disaster [7]

While blockchain technology has historically been developed in the context of cryptocurrencies, we will discuss in this section the dangers associated with cryptocurrencies. This analysis goes beyond the domain of computer science: it mobilizes arguments from the fields of economics, political theory, sociology and ecology. Since those domains are outside of our field of expertise, this section should not be understood as an authoritative argument, but only as an indicative framework that explains our distrust for cryptocurrencies and our motivation to build different kinds of blockchain systems that are not intrinsically linked to cryptocurrencies (like Proof-of-Stake is).

2.5.1 The Bitcoin ideology and the ideals of crypto-anarchism

The first widely-used cryptocurrency, Bitcoin, was introduced in 2009 (Nakamoto, 2009b), following the global financial crisis of 2007-2008. The creator of Bitcoin, Satoshi Nakamoto, denounced the irresponsibility of states, banks and financial institutions that led to this crisis, and the risks caused to individuals. He denounced the fact that when one deposits one's money in a bank, the bank may make use of it in a variety of ways. In particular, banks may create new money by giving out credits, which only need to be backed by a small percentage of actual deposits in the bank. This *fractional reserve* banking system, argued Nakamoto, can lead to systemic instability and is also a risk taken by individuals on their savings. The idea of Bitcoin was that money should instead be in the direct control of its holder, and that trust in a government or bank should not be required, as they may make decisions that are against the interests of the depositors. Further, Nakamoto was worried that the over-use of monetary creation by banks (and especially central banks) would lead to inflation, meaning that the savings of individuals would lose value over time. Instead, he argued, money should be created using a fixed rule that was guaranteed algorithmically, and that the total supply of money should be capped. As Nakamoto wrote in 2009:

> *"The root problem with conventional currency is all the trust that's required to make it work. The central bank must be trusted not to debase the currency, but the history of fiat currencies is full of breaches of that trust. Banks must be trusted to hold our money and transfer it electronically, but they lend it out in waves of credit bubbles with barely a fraction in reserve."* (Nakamoto, 2009a)

7. This title was shamefully stolen from Drew DeVault who wrote a blog post titled "Cryptocurrency is an abject disaster" (DeVault, 2021). Thank you, Drew.

The proponents of Bitcoin have taken on these objectives under various justifications. As a New York Times journalist put it:

> *"Bitcoin isn't merely money; it's "a movement" – a crusade in the costume of a currency. Depending on whom you talk to, the goal is to unleash repressed economies, to take down global banking or to wage a war against the Federal Reserve."* (Feuer, 2013)

Cryptocurrency proponents are in line with the ideals of defending individual freedom, which can also be found in the free software movement: similarly to how free software idealists believe that software source code should be in the hands of everyone, and especially of the software's users, the cryptocurrency movement claims that money must be liberated from the states and from the control they exert on the private sphere. Money should instead be a tool designed by the community of its users for themselves. In the case of money, the liberation from state control necessarily implies a free market economy exempt from any kind of state regulation. Thanks to the possibility of trading them internationally without restrictions, cryptocurrencies are seen as global, neutral currencies (Lakomski-Laguerre and Desmedt, 2015).

The necessity of taking money out of state control has been theorized by the liberal economist Friedrich Hayek. In *Denationalisation of Money* (Hayek, 1990), Hayek expressed that in his opinion, to prevent inflation and to ensure that currencies do not lose their value, the process of monetary creation should be given back to the public, and not kept in the hands of governments and central banks. By letting anyone create their own currency, he argued, and by letting different currencies compete against one another for their value on a free market, a good currency could be achieved. The currencies that provide limited long-term value (i.e. that suffer from inflation) would become abandoned, and on the contrary, if a monetary system was created that was able to prevent inflation, it would become trusted by the masses and become a common means of exchange. Famously, Hayek expressed that he did not expect states or political decision to ever lead to the regime of private monetary creation that he theorized, and that the only way to achieve it was to impose it by some kind of "trick":

> *"I don't believe we shall ever have a good money again before we take the thing out of the hands of government. We can't take it violently out of the hands of government, all we can do is by some sly roundabout way introduce something that they can't stop." – Friedrich Hayek*[8]

8. We were able to find a video recording of this statement on Twitter at `https://twitter.com/`

Cryptocurrencies are the accomplishment of this dream. The means by which cryptocurrencies achieve it are distributed algorithms and cryptography. On the side of monetary creation, new bitcoins are created in decreasing quantities at each block, where the rules for block creation and coin generation are fixed in the code. On the side of exchange rules, Bitcoin enables coin holders to use them at any place and time without restriction. The Bitcoin protocol ensures that only the true holder of a coin, i.e. the person that possesses the associated private key, can spend the coin, and that a coin spent cannot be taken back. The new technology of blockchains, which relies on public-key cryptography, ensures that these properties can be implemented without having to trust any financial institution. Nakamoto made a parallel with the case of encryption, which allows one to guarantee the privacy of their data without having to rely on a system's administrator to properly implement protection schemes such as ACLs and password protection (which can always be broken by the administrator himself) (Nakamoto, 2009a).

This new technology finds its theoretical roots in the concept of *crypto-anarchism*, a new political philosophy first introduced in 1988 in the *Crypto-anarchist Manifesto* (May, 1988; Chen, 2013; Lakomski-Laguerre and Desmedt, 2015). Crypto-anarchism promotes the use of computer cryptography to defend against all forms of authority (and abuse thereof), including control by state. Indeed, crypto-anarchists believe that the state and large institutions such as banks have the power to exert abusive control over the population, and they wish to neutralize these institutions by cancelling their power by the use of cryptography. Another example of the application of the principles of crypto-anarchism is the Tor anonymity network, which uses the cryptography-based onion routing protocol to ensure that communications remain anonymous, and to prevent them from being traced and intercepted by state security agencies. In the case of this thesis, the desire to avoid control by large internet companies (GAFAM) and to build instead resilient trustless peer-to-peer systems can also be traced back to the ideals of crypto-anarchism.

In this sense, Bitcoin ideology can be interpreted as a new form of "metallism" (Lakomski-Laguerre and Desmedt, 2015): the trust in physical goods (such as gold) that have intrinsic properties of rarity that ensures their value. Although this rarity is now ensured by an algorithm, their rules are extremely similar as they respect properties that can be qualified as "physical properties" that cannot be circumvented by any kind of policy decision (which is seen as arbitrary). These properties consist in the intrinsic rarity of coins, and

`Pladizow/status/1082699459425959937` (accessed 2021-08-18) but we do not know the original context or the date of this statement.

the fact that they behave like physical items that can be traded directly from hand to hand. The rarity is guaranteed by the cap on the maximal amount of bitcoins that can ever be created (21M bitcoins). The trust in hierarchical authority is therefore replaced by a form of trust in algorithms and mathematics, which is believed to be much stronger because of their basis in formal mathematics that make them appear unbreakable. Note that this is however still a form of trust, as most users of cryptocurrencies do not understand fully all of the cryptographic and algorithmic underpinnings of the systems they use.

2.5.2 The failure of Bitcoin (and most cryptocurrencies) as a monetary system

Unfortunately, the idealistic picture painted in Section 2.5.1 does not hold under further investigation. As we will discuss in this section, many authors have criticized cryptocurrencies, explaining the reasons why they cannot be considered as a satisfactory alternative to the current monetary system.

The difference between information and currency Information and knowledge can be copied and transferred between individuals infinitely at an extremely small cost thanks to modern computers and computer networks. As such, information is a "non-rival" good: use by an individual does not limit its use by any other individuals. Therefore, by freeing information, free software activists and the hacker movement argue that the public may regain power over society and governments by better understanding technologies and social institutions. This is often illustrated by the famous aphorisms "knowledge is power" and "information wants to be free", respectively attributed to Sir Francis Bacon (1597) and Stewart Brand (1984).

However as soon as we look at "rival" goods, i.e. goods whose use (ownership or consumption) by an individual restricts its use by other individuals (examples: natural resources, land, manufactured objects, currency), things change dramatically. Indeed, these goods can no longer be made easily accessible to anyone who needs them, and unlimited accumulation in the form of private property (which is promoted by the liberal ideology) becomes an impediment to their equal enjoyment by everyone (Dupré, 2020). Cryptocurrency proponents and crypto-anarchists in general hide behind the argument of "maximizing individual freedoms" to promote such a liberal ideology, obfuscating the

fact that this has historically led to a society that may well have one of the highest rates of inequality ever observed in Human history (OECD, 2015; Inequality.org, 2021). In the context of cryptocurrencies, this might have been further amplified by the perception that "money is just information" (the knowledge of who owns what, stored in a blockchain), amplifying their endorsement by the hacker movement which is very focused on individual freedoms in the domain of information technology.

Unfortunately, currencies, and more specifically cryptocurrencies, are not just "information". They are a social institution (Davis, 2017) that organizes the means and conditions under which individuals may access physical goods, for which they may have a vital need (e.g. food, housing). Socialists such as Elinor Ostrom [9] instead conclude that rival goods should be managed democratically and not let in the hand of individuals, and that their rules of management should be determined not only by "representative democracy", but using principles of direct democracy that are appropriate to the management of common goods (Ostrom, 1990). In particular, such principles should apply to deciding the rules by which money is created and allocated in society. In other words, in opposition to the crypto-anarchist perspective, and the liberal perspective in general, that views money as a tool for achieving individual freedom, socialism argues that money is as dangerous as it is powerful and that a democratic control of its functioning at large scales is necessary for it to be truly at the service of the people. We will develop this point later in this section.

The three functions of money We will begin our discussion of the limitations of cryptocurrencies by very basic observations that show that, in practice, cryptocurrencies hardly satisfy the functions of "money", based on an argument made by Françoise Vasselin (Vasselin, 2020). We will later go into more theoretical arguments showing why this is unlikely to ever be the case.

The three functions of money are traditionally defined as follows:

— money as a measure of value;

— money as a means of exchange;

— money as a store of value.

In practice, extremely few merchants label their prices using cryptocurrency values, meaning that they are not commonly used to measure the value of goods (instead, their

9. As well as true anarchists, who by and large reject the views of crypto-anarchists and anarcho-capitalists on money and private property. See Baillargeon (2001) (pages 129–136) for a detailed discussion of this point.

purchasing power is measured through their conversion in a state-backed currency). This difficulty is amplified by the fact that cryptocurrencies typically have very volatile valuations, meaning that prices expressed in them have limited usability.

Furthermore, few users are willing to pay using cryptocurrencies or to accept currencies in exchange for goods, meaning that they also do not serve very well the function of a means of exchange. This is further amplified by the fact that cryptocurrencies require technical knowledge to be used. In the case of Bitcoin, high transaction fees are also a significant barrier to its use as a means of exchange.

Finally, several cryptocurrencies seem to work adequately as a store of value, with bitcoin hoarding becoming popular phenomenon. However even this use for cryptocurrencies is dangerous and cannot be recommended to a general public as the value of these currencies is extremely volatile, with huge drops in valuation being a common phenomenon.

State-controlled currencies and monetary policies The current status of state-controlled currencies can be seen as an instantiation of the idea that money should be managed in a manner that profits the wider public. Indeed, the possibility of implementing a *monetary policy* allows states and central banks to guarantee consumer prices by controlling inflation and deflation, and to amortize the consequences of economic shocks.

The possibility of a monetary policy directly depends on the concept of *monetary sovereignty*, i.e. the idea that states or governments should hold power over a certain number of aspects of the currencies in circulation. Monetary sovereignty can be justified by the following four aspects (Cohen, 1998; Vasselin, 2020):

— Political symbolism: the idea of a currency that is linked to a state and a territory, promoting a sense of community.
— Seigniorage: the difference between the face value of money (e.g. a 10 USD bill) and the cost to produce it, which is collected by the state that emits it. Today still, many emerging economies rely on seigniorage as a source of income that guarantees their economic independence (He, 2018).
— Macroeconomic management: managing the impacts of currencies on actual economic performances.
— Monetary isolation, as a means to limit the propagation of economic shocks from one country to another.

This monetary sovereignty induces the following 5 “rights” (C. D. Zimmermann, 2013; Vasselin, 2020):

— right to create money that has legal tender status;
— right to conduct a monetary policy;
— right to conduct an exchange rate policy;
— right to exert exchange control;
— right to organize financial regulation and supervision.

As we said, this conception of monetary sovereignty allows the state to guarantee a stable monetary regime. In particular, the possibility of monetary creation by states is a requirement for their ability to protect against the risk of structural deflation, and to enable adaptation to temporary shocks in demand for currency. States therefore can smooth the economic cycle and act as a creditor of last resort (He, 2018) [10].

All of the aspects of monetary sovereignty described above are weakened by cryptocurrencies. In the case of Bitcoin, the fact that the maximum supply is capped means that it is a deflationary currency. Deflationary economies, however, tend to lead to decreasing wages and prices over time, causing unemployment and hampering the ability of businesses to borrow money to invest in new activities. In other words, using Bitcoin as the main currency in an economy would cause a recession. Even if a cryptocurrency was created that did not have a maximum supply cap, the fact that no central entity can exert control on monetary creation means that there is no means to ensure financial stability by adapting the quantity of currency in circulation to the demand for such currency. This would induce a high volatility in prices, and an overall loss of well-being for individuals (Vasselin, 2020).

Even if they are not adopted as official currencies, the possibility of a widespread deployment of cryptocurrencies in the near future has been a subject of worries for global economic stability, as central bank monetary policies would likely lose their relevance in such a context (He, 2018). This can be related to the case of the "dollarization" of economies in emerging countries, a situation in which a large part of a local economy in fact happens using a foreign currency. In such a situation, the local currency is disconnected from the local economic reality, meaning that the economy becomes more and more disconnected from state regulation.

The lack of observability and control on cryptocurrencies, due to their anonymity and their nature as a trustless distributed system, has also been cause of worries related to their use for crime and money laundering. In practice, Bitcoin seems to be used less

10. Roughly translated from French: *"protection contre le risque de déflation structurelle, capacité à s'adapter avec souplesse aux chocs temporaires de la demande de monnaie et donc à lisser le cycle économique et capacité à faire office de prêteur en dernier ressort."* (He, 2018)

as a means of ordinary day-to-day exchange, and more as a way to finance criminal activities and as a "refuge currency" similar to gold, whose use stems from a distrust in the mainstream financial system (Chevalier and Vignolles, 2014; Messéant, 2020). It has also attracted vast interest from private speculators looking to "make a quick buck" by betting on increases in prices and causing prices to fluctuate widely with large-scale "pump-and-dump" operations (Gonzalez, 2021).

Risks associated with free markets In his 2013 book *Illusion financière* (Giraud, 2013), professor of economics and former economist in chief of the *Agence Française de Développement* Gaël Giraud offers a powerful criticism of the current free market economy and the liberal principles on which it is based. Taking as a starting point the subprime crisis of 2007, which led to a global financial crisis, Giraud criticizes the idea that free markets are an efficient way of allocating resources and risks in an economy. Indeed, the unrestricted expansion of the financial sphere, that led to the creation and massive trading of new financial products such as *collateralized debt obligations* (CDO), *credit-default swaps* (CDS), and securitization, was the main cause for the crisis of 2007-2008 in which these massive financial montages collapsed. The reason why these financial products were able to take such a prominent place in the global financial system in the first place, other than a lack of regulation, was due according to Giraud to a fundamental inefficiency of financial markets: an inability to justly price these new financial products according to the risk that they represent.

The inefficiency of financial markets, according to Giraud, is one of their fundamental features which is inevitable. This point has been widely discussed amongst economists in the academic world (Azariadis, 1981; Cass and Shell, 1983). To simplify, this is caused by the self-fulfilling nature of predictions on financial markets, a phenomenon referred to as "sunspots". Indeed, when an important economic actor announces that the value of a financial asset will increase in the near-term future, the broadcasting of this information causes many economic actors to buy the asset in order to reap profits, in turn causing the asset's valuation to indeed increase. On the contrary, an announcement of devaluation of an asset is likely to indeed cause the asset's valuation to plummet. This has also been visible many times on the cryptocurrency exchange markets, with a recent example in the following tweets by Elon Musk, that caused the price of Bitcoin to rise to 58 000 USD, then fall back down to 30 000 USD:

"You can now buy a Tesla with Bitcoin" (tweet by Elon Musk, March 24,

2021 [11])

> *"Tesla has suspended vehicle purchases using Bitcoin. [...]"* (tweet by Elon Musk, May 13th, 2021 [12])

According to Giraud, it is therefore wrong to assume that assets on a financial market have a "fundamental value" that would be reflected in their price, allowing economic actors to evaluate the associated risks in a correct manner. Instead, these kinds of financial assets always hold an intrinsic risk that the market price is unable to reflect, and their prices may fluctuate widely due to the action of speculators.

The creation of new financial objects to cover these risks, such as credit-default swaps, does not do away with these risks: they are simply transferred from one financial institution to another like a "hot potato". In the case of the 2007–2008 global financial crisis, these risks massively accumulated in the hands of large commercial banks, during an optimistic ("bullish") phase that led banks to give out massive amounts of credits with only minimal verification of solvency. When the optimistic phase of the economic cycle inevitably ended, these risks materialized, exposing the nature of these financial montages as an immense Ponzi scheme and causing many actors to go bankrupt. Unfortunately, these bankruptcies have true consequences on the real economy: between 2008 and 2012 in the United States, over three million households lost their home, and one house in 355 was estimated to be at risk of being seized due to the impossibility of their owners to pay back their mortgage. While certain large banks such as Lehman Brothers declared bankruptcy, others were bailed out by governments and central banks in unprecedented acts of monetary creation through quantitative easing. On the other side, financial actors that held the other side of the bets, i.e. that had gambled on the failure of the system (the case of hedge funds), were able to generate massive profits and came out more powerful than ever from the crisis. This double-standard of the financial system was often criticized as the "privatization of gains and socialization of losses" (Stiglitz, 2010), as the public was made to pay for the costs of financial institutions that had failed, whereas the victors were able to come out unscathed without having to contribute their fair share in resolving the ensuing calamity.

Giraud argues that the only way to avoid such catastrophes, and in general to stabilize financial markets, is to increase state regulations. In the case of cryptocurrencies, on the contrary, the underlying ideology that they implement is a push to the extreme of entirely

11. `https://twitter.com/elonmusk/status/1374617643446063105`
12. `https://twitter.com/elonmusk/status/1392602041025843203`

free markets and unrestricted circulation of financial assets. As a first example of the risks that this poses, the fact that these currencies don't have legal-tender status and are backed by no trusted third party means that their value is entirely determined by the markets: it holds only to the anticipation that they will be in high demand on exchange markets in the future, i.e. to the belief that other economical agents will use and value them in the future. However, this is a weakly anchored belief that is subject to easy manipulation (such as tweets by Elon Musk), causing high volatility for prices on exchange markets (He, 2018). Given the above arguments against free-market economy, and contrarily to their announced objectives, it seems unlikely that cryptocurrencies will ever be able to provide a stable financial system that fixes the issues with today's mainstream financial institutions.

Towards managing money as a commons Several economists have argued that, instead of relying exclusively on free markets to solve the problems of our economies, we should instead focus on managing money for the public good (Giraud, 2013; Dupré, 2020). The two fundamental aspects of money, liquidity (the power given to an individual to use their money at any time to buy goods) and credit (the ability to obtain financing for a project on the promise of future wealth), should be organized as commons [13], being governed by rules inspired by those proposed by Elinor Ostrom (Ostrom, 1990). This could allow us to avoid the risks associated with the private appropriation of liquidity such as in the global financial crisis, as well as to orient monetary creation away from financial bubbles, and back into socially useful investments such as the ecological transition. In other words, a renovated financial financial system based on commons would allow for renewed social progress in the form of reduced inequalities and stronger measures to reduce the dangers of the impending climate crisis and better protect ourselves from its consequences.

While cryptocurrencies display the intention of being democratically managed by the users for the users, Dupré et al. showed that they are in fact very far from satisfying Ostrom's criteria for commons (Dupré et al., 2015). These criteria are defined as follows:

— A definition of the community of stakeholders, and the openness of this community.
— A definition of the conditions of access, appropriation, reproduction, exclusion, distribution and extraction of the common.
— A possibility of controlling uses of the common and the incomes that it generates.
— Transparency of management.

13. *"La liquidité et le crédit devraient être organisés à la mannière de communs"* (Giraud, 2013)

— A possibility of arbitrating conflicts, and definition of a set of sanctions for misconduct that can be applied in practice.
— A democratic decision process.

Focusing on Bitcoin, Dupré et al. show that it is very far from satisfying these definitions, as the only rules that it implements are the strictly "physical" rules of circulation of money, which allow for no democratic intervention and no control by the community. The nature of cryptocurrencies and smart contract platforms as an implementation of the principle "code as law" gives a system of rules that are formal, technical, and immutable, very far from the conventional context where law and contracts have a certain dose of flexibility: the interpretation of law is to be made on a case-by-case basis, and contracts may be revoked at any time. "Code as law" implies that once a decision has been enacted, democratic discussion, and the protection of individual rights, is no longer possible. This is in direct contradiction with the required properties of commons, and carries the risk of leading to a *technocratic* regime which would have a totalitarian aspect (De Filippi, 2018). On the other side, Dupré et al. argue that Bitcoin also does not work effectively as a monetary system to manage action for the common good, due to the following limitations:

— Bitcoin does not allow economic actors to obtain the necessary liquidity for their activities through funding advances.
— Bitcoin does not offer possibilities of being employed for public action.

They conclude:

> *In no way, therefore, can Bitcoin respond to the current international monetary disorder, nor to the search for the global public good that are monetary and financial stability and the necessary contribution of liquidity to the production of useful goods and services, especially in a context of energy transition.* [14]

2.5.3 The significant negative externalities of Bitcoin (and other cryptocurrencies)

We have seen in the previous section that, from a financial perspective, cryptocurrencies (and in particular Bitcoin) cannot be considered good alternatives to the current monetary system. They provide no tangible advantage for the governance of commons, and introduce new risks to the financial system by adding instability. We will now see

14. Translated from French: *"En aucun cas, le Bitcoin ne peut donc répondre ni à l'actuel désordre monétaire international, ni à la recherche du bien public mondial que sont la stabilité monétaire et financière et le nécessaire apport de liquidités à la production de biens et services utiles, en particulier dans un contexte de transition énergétique."* (Dupré et al., 2015)

that cryptocurrencies also have important negative externalities outside of the financial sphere.

Ecological disaster The fact that most mainstream cryptocurrencies still rely on Proof-of-Work (the case of Bitcoin or Ethereum for instance) means that these systems must consume huge quantities of energy to build the blockchain (according to the rules described in Section 2.4.2). The fact that the mining difficulty adapts dynamically to the available compute power means that adding computing power to the system will always have negative overall returns; the positive benefits are only for the owner of said compute power, that might (temporarily) obtain a larger share of total mining power, meaning more opportunities to earn newly mined coins. This system is obviously not viable, as it leads to a computing power arms race where miners compete against one another, building the largest possible mining farms and using as much energy as possible in them. As a consequence, it has been estimated for instance that the Bitcoin network consumes as much electricity as a country such as the Netherlands, and has a carbon footprint similar to that of Greece (Digiconomist, 2020). As we have already said, in the context of the ecological crisis we are facing, this situation is not acceptable, especially given the fact that the energy used is not intrinsically necessary to execute Bitcoin transactions themselves, but only a requirement of this specific algorithm (for its "security", a property that can very well be achieved by other, less wasteful means – be it a centralized payment system).

Tangentially related to the energy consumption directly caused by the mining operation itself is the issue of the production of specialized mining hardware for Bitcoin mining. Indeed, the computationally costly operation in the Proof-of-Work scheme implemented by Bitcoin is a simple SHA-256 hash function which is repeated a large number of times. The probability of a miner to find the next block in the blockchain is dependent on the number of SHA-256 hashes he is able to compute per time unit. As a consequence, miners have focused on building specialized hardware that compute these SHA-256 hashes faster and with less energy than general-purpose CPUs. This hardware takes the form of "mining ASICs" (an ASIC is an Application-Specific Integrated Circuit). Obviously, this has not led to a decrease in the energy consumption of the Bitcoin network: due to Jevons' paradox, this increase in efficiency has only been translated in an increased output of the mining farms, measured in hashes computed per time unit. On the other side, the production of these new hardware units has generated the same externalities as the production of any computer hardware: the extraction and expenditure of limited

resources (rare metals and fossil energies) and the generation of additional pollution in the environment (Tu and Lee, 2009; T. Zimmermann and Gößling-Reisemann, 2013; Habib et al., 2016; Stephant, 2021). Moreover, these purpose-specific ASICs cannot be recycled as general-purpose computers once their lifetime as Bitcoin mining systems has ended, meaning that they will end up increasing the already huge quantities of e-waste (waste of electronic equipment) that litter the globe (Robinson, 2009; Premalatha et al., 2014; Akram et al., 2019).

The ecological damages caused by Bitcoin and other cryptocurrencies is often downplayed by miners and other prominent actors of the Bitcoin economy. This can easily be understood by the fact that the valuation of Bitcoin on exchange markets is not related to any intrinsic value (Giraud explained that financial assets in fact do not have a fundamental, intrinsic value), but only to the trust that economic actors globally put in it: major Bitcoin holders have private interest in preserving the image of Bitcoin as an environmentally friendly technology, without which its market valuation would be at risk of decreasing. This manipulation of the public opinion is made easier by the perception of Bitcoin as a kind of "new technology", a category which is often associated with a reduction of costs by means of automation. This perception is favored by the fact that the true ecological (and social) costs associated with technology are located far away (in mining operations, chip factories and datacenters), and are not immediately visible to the consumers. In practice however, these costs are important for all kinds of computer technologies (and not only for those associated with cryptocurrency mining), a fact that is often forgotten. This leads to a new form of "privatization of gains, socialization of losses", this time not in the financial sphere but with social and ecological consequences that are borne by populations and ecosystems at large (but especially in poor and developing countries), while the produced computer systems benefit primarily wealthy consumers, large corporations and investors in first-world countries.

A new tragedy of commons The necessity of obtaining ever more compute cycles for cryptocurrency mining has had negative consequences on the availability of these compute cycles to the general public for other activities. Indeed, while Bitcoin mining today is only profitable to mine using ASICs, it can still be mined with general-purpose hardware such as consumer CPUs or GPUs when those resources are made available for free (i.e. when somebody else bears the cost of the electricity bill). Other cryptocurrencies

that use Proof-of-Work, such as Ethereum, are even still profitable to mine using GPUs [15].

As a consequence, cryptocurrency miners have been buying massive quantities of such consumer GPUs in the recent years to build mining farms. The worldwide production capacity of computer chips is in fact quite limited, with most high-end GPU chips being produced by a single manufacturer, TSMC. The TSMC chip factories are currently running at their maximal production rate, meaning that an increase in production to satisfy the demands of cryptocurrency miners is not possible before several years (the time to open new chip factories). Further, with the COVID-19 pandemic, many chip factories were stopped or slowed down, worsening the shortage of production capacity. The high demand for GPUs for cryptocurrency mining has therefore reduced their availability to the general public: at the date of writing, to buy a high-end GPU one would look to the second-hand market, where prices have skyrocketed way higher than the MSRP [16] (Schiesser, 2021). This has in particular impacted the communities of content creators on the Internet that rely on them for their graphical computing capabilities (Needleman, 2018), but due to the fact that many manufacturers of electronic equipment are using the same TSMC production lines to manufacture their chips, this semiconductor crisis has rippled to other domains with impacts on the production of smartphones (Browne, 2021) and electric cars (Meissner and Kirschstein, 2021). Note that even if the world's chip manufacturing capacity was significantly increased, the ecological consequences of doing so, evoked above, cannot be overstated.

Consumer GPU hardware is not the only place where cryptocurrency mining has taken a foothold at the expense of other users: miners have attempted to take over all available compute cycles at the most unlikely places, such as web browsers and continuous integration services. In the case of web browsers, they are now able to block JavaScript code that uses client CPU time for cryptocurrency mining that profits the website owner. This is not the case of continuous integration services, however, whose core property is the ability to run tasks that have high CPU load. Continuous integration (CI) services allow users of source code sharing platforms such as GitHub to run scripts in the cloud, for instance to automatically check that commits added to a project are valid and do not introduce any bugs. Continuous integration services are often provided for free to open-source projects. Cryptocurrency miners have recently started using these free resources for mining, a fact that has led them to major issues for the platforms that do not have

15. See `https://whattomine.com/gpus`, accessed 2021-08-26
16. Manufacturer Suggested Retail Price

unlimited budgets to expand the resources they allocate to CI. Many such platforms have now banned cryptocurrency mining, however there is no easy way to verify whether a job that is running is legitimate or not as cryptocurrency miners always find new ways to hide their activities. This has led to a new arms race between miners and the platforms, with some platforms even removing their free tier CI offerings (DeVault, 2021).

Computing power availability is not the only limited resource whose over-exploitation by cryptocurrency miners has caused harm to other users: electricity itself is in danger of being hoarded by miners at the expense of everyone else. In certain countries where the economy is weaker and the electric grid is less developed, a large consumption of electricity by mining farms has been the cause of blackouts in cities, leading to a situation of humanitarian catastrophe. Such a situation has been reported in locations such as Abkhazia and Iran (Eckel et al., 2020; Turak, 2021)

In conclusion, cryptocurrency mining has shown us that computing power and electricity are not the unlimited resource that we fantasize, and that their appropriation by private interests at the expense of the public good is indeed possible. This is a new tragedy of commons, made possible by the new form of ultra-liberalism that cryptocurrencies represent.

2.5.4 Imagining a future for blockchains

A warning Before we enter considerations on possible uses of blockchains that do not pose all of the issues described above, let us first remind the reader that although blockchains are a technical tool, the problems in question are far from being only technical: they also carry political and ecological implications, which in a democratic society can only be settled by democratic debate. In other words, contrary to the beliefs of crypto-anarchists, technologies must never be trusted to solve large-scale problems on their own; they must always be subject to constant monitoring and reevaluation through a democratic discussion and decision process.

State-controlled digital currencies Banks and financial institutions have acknowledged the fact that cryptocurrencies answer some of the needs of users: simplicity, immediacy, ergonomics and a lower cost of management (in some cases). The governor of the *Banque de France*, Denis Beau, has emphasized the importance of preserving the innovation potential of this new technology, while considering the necessity of finding ways to avoid the issues that cryptocurrencies cause and which we have discussed above: issues of

security, stability, and sovereignty (such as their use for money laundering and financing terrorism). To this end, Beau first suggests that central banks and governments impose a new regulatory framework for cryptocurrencies. But Beau goes further, suggesting that central banks might be interested in emitting their own "digital currencies", to reap the benefit of blockchain technologies in the context of a state-controlled currency (Beau, 2020). This digital currency would act similarly to bank notes emitted by central banks, but would have many advantages such as being more convenient to use. Vasselin even argues that this would allow central banks to implement monetary policies that cannot be implemented today, such as a better enforcement of negative interest rates, especially on cash (Vasselin, 2020). Given the potential for these new kinds of state-controlled currencies, economists have observed a "race" to build such systems, with experiments for example in China, but with many central banks around the world considering the potential of such a development (Bounie, 2020).

A platform for new monetary experimentation The development of blockchains as a general platform for distributed computation allows its use in a variety of ways. It is particularly suited to cryptocurrencies, since they were the original context for their invention by Nakamoto, however this does not imply that all cryptocurrencies implemented on blockchains must implement the same monetary rules as Bitcoin. In particular, different rules for monetary creation may be experimented with blockchains, in order to implement new economic paradigms.

In line with Marxist theory, Alizart views the monetary mass in circulation as a representation of the quantity of energy available to the economy (Alizart, 2019). As a consequence, to avoid inflation or deflation, this monetary mass must be adapted to the concrete needs of the economy, instead of being created at an arbitrary first rule. Alizart argues that implementing such a scheme could allow to transition from a deflationary currency, Bitcoin, to a new cryptocurrency that is well suited to the needs of the economy.

Many cryptocurrency proponents, however, argue for more radical changes such as implementing a Universal Basic Income (UBI), a social justice measure that is seen as a possibility to directly reduce inequalities of wealth in our society. The OpenUBI discussion group is concerned with researching these possibilities, with many proposed alternatives such as Circles [17], UBIC [18] and Duniter [19]. These new cryptocurrencies form the synthesis

17. `https://joincircles.net/`
18. `https://ubic.app/`
19. `https://duniter.org/`

of the movement initiated with local currencies on the one hand, that aim at encouraging spending in local businesses in order to help develop circular economies, and cryptocurrencies on the other hand, that aim at developing new technological tools for currencies outside of the control of banks and governments. These two movements agree on the necessity for radical experimentation with new forms of money, independently of governments and their political agendas, in order to serve the population directly (Porcherot, 2019). In terms of economy, the theoretical foundations for an Universal Basic Income that preserves equity between individuals has been proposed by the so-called *Relative Theory of Money* (Laborde, 2012), which the Duniter network implements in practice in the form of a new cryptocurrency.

In conclusion: the need for blockchains independent of cryptocurrencies Experimenting with new monetary schemes using blockchains can only be done if the blockchain is itself able to function independently of the cryptocurrency that would be built upon it. Proof-of-Work blockchains satisfy this condition, as the consensus algorithm depends only on the hashing power provided by the participants, however we have seen that it is extremely wasteful and should therefore not be used. Proof-of-Stake blockchains, on the other side, do not satisfy this condition as the consensus algorithm becomes intrinsically linked to a cryptocurrency that is built on the network. It is therefore crucially important that we be able to develop new consensus algorithms for blockchain that avoid the pitfalls of these two approaches. We will propose such an alternative with the contributions we make in Chapters 5 and 6. As we have mentioned many times, these new algorithms also have many uses independent of cryptocurrencies, for instance for building large-scale collaboration systems and social networks in a trustless peer-to-peer fashion.

Chapter 3

Causally Ordered Delivery in Presence of Byzantine Nodes

In the context of large-scale systems constituted of thousand of nodes located in many distant geographical locations, both theoreticians and practitioners have observed the impossibility of building highly available systems that offer strong consistency guarantees while tolerating network partitions, a result which has been formalized as the *CAP theorem* (Fox and Brewer, 1999; Gilbert and Lynch, 2002; Gilbert and Lynch, 2012). As a consequence, a number of systems have chosen to rely instead on weaker communication abstractions that offer only causal ordering guarantees (Lesani et al., 2016; Lloyd et al., 2011; Shapiro et al., 2011). Formally, causal broadcast is a communication abstraction built on top of point-to-point send/receive networks that ensures that any two messages whose broadcasts are causally related (as captured by Lamport's "happened before" relation) are delivered in their sending order.

These causally ordered broadcast primitives are a relatively elementary mechanism that leave much work to the application in order to guarantee its desired properties. However, as we saw in our state-of-the-art review, many useful systems can be built using only such primitives. In the context of social networks, causal ordering ensures that users that see Bob's reply to Alice have also seen Alice's original message: as an example, we evoked the case of the Matrix communication network which is built by encoding causal broadcast in an event DAG. In the context of collaborative editors, causal ordering enables large numbers of users to concurrently edit documents without conflicts (Oster et al., 2006). In the domain of distributed data stores (DeCandia et al., 2007; Lloyd et al., 2011), causal ordering makes automatic conflict resolution possible in the presence of concurrent writes (DeCandia et al., 2007). We also evoked the case of the money transfer object which can be built only using a causally ordered broadcast primitive, and does not require fully-fledged consensus as implemented by blockchains (Guerraoui et al., 2019c). All of these examples motivate our study of causal broadcast, which we conduct in this

chapter and the following. We will return to more powerful, higher-level abstractions in Chapters 5 and 6.

Several causal broadcast algorithms have been designed for failure-free and crash-prone asynchronous message-passing systems. In the last few years, multiple authors have observed that causal consistency and variants of it constitute the best achievable guarantee in large-scale fault-tolerant available systems (Attiya et al., 2016; Lloyd et al., 2011; Shapiro et al., 2011). Surprisingly however, the majority of existing research work only considers and defines causal consistency only in the context of crash failures.

In this chapter, we propose a new formalisation of causally consistent broadcast, or causal broadcast, that is suited to systems with Byzantine nodes. To make this study possible, we restrict ourselves to the case of a static closed network of N nodes, and not an open network that would be subject to churn (we will return to this kind of systems in the following chapters). We first give a formal definition of a causal broadcast abstraction in the presence of Byzantine processes, which we call BCO-Broadcast, in the form of two equivalent property-based characterizations. We then present and prove correct an algorithm implementing it in a system where communication is by message-passing. This algorithm is built on top of a Byzantine reliable broadcast abstraction (Bracha, 1987). Consequently it inherits the t-resilience and the message/time complexities of the specific algorithm chosen to implement the underlying reliable broadcast. While the algorithm is simple, its proof is not, due to the very nature of Byzantine faults. Specifically, the main difficulty consists in proving that Byzantine processes can neither destroy the causality relation on the messages broadcast by the correct processes nor prevent the correct processes from delivering (according to causal order) the messages they broadcast. [1]

3.1 Computation Model

3.1.1 On the process side

Asynchronous processes The system is made of a finite set Π of $N > 1$ asynchronous sequential processes, namely $\Pi = \{p_1, \ldots, p_N\}$. *Asynchronous* means that each process proceeds at its own speed, which can vary arbitrarily with time, and always remains unknown to the other processes.

1. The work presented in this chapter has been the subject of the following publication: Alex Auvolat, Davide Frey, Michel Raynal, and François Taïani (2021b), « Byzantine-tolerant Causal Broadcast », *Theoretical Computer Science*, pp. 55–68, DOI: 10.1016/j.tcs.2021.06.021

Process failure Up to t processes can exhibit a *Byzantine* behavior (Lamport et al., 1982). A Byzantine process is a process that behaves arbitrarily: it can crash, fail to send or receive messages, send arbitrary messages, start in an arbitrary state, perform arbitrary state transitions, etc. As a simple example, a Byzantine process that is assumed to broadcast a message m to all the processes, can send a message m_1 to some processes, a different message m_2 to another subset of processes, and no message at all to the remaining processes. Moreover, Byzantine processes can collude to foil non-Byzantine processes. As is common in Byzantine-Fault-Tolerant algorithms, we assume that Byzantine processes cannot spawn, i.e. we exclude Sybil attacks. A process that exhibits a Byzantine behavior is also called *faulty*. Otherwise, it is *correct*.

Let us notice that, as each pair of processes is connected by a channel, a process can identify the sender of each message it receives. Hence, no Byzantine process can impersonate another process.

3.1.2 On the communication side

The Byzantine reliable broadcast communication abstraction The BR-broadcast abstraction is a one-shot communication abstraction that provides each process with two operations, br_broadcast() and br_deliver(). *One-shot* means that a process invokes br_broadcast() at most once, and BR-broadcast instances issued by processes are independent. As in Cachin et al. (2011), Hadzilacos and Toueg (1994), and Raynal (2018), we use the following terminology: when a process invokes br_broadcast(), we say that it "br-broadcasts a message", and when it executes br_deliver(), we say that it "br-delivers a message". BR-broadcast is defined by the following properties.

- **BR-Validity.** If a correct process br-delivers a message m from a correct process p_i, then p_i br-broadcast m.
- **BR-Integrity.** A correct process br-delivers at most one message m from a process p_i.
- **BR-Termination-1.** If a correct process br-broadcasts a message, it br-delivers it.
- **BR-Termination-2.** If a correct process br-delivers a message m from p_i (possibly faulty) then all correct processes eventually br-deliver m from p_i.

On the safety side, BR-validity relates the outputs (messages br-delivered) to the inputs (messages br-broadcast), while BR-integrity states that there is no duplication.

On the liveness side, BR-Termination-1 states that a correct process br-delivers the

messages it has br-broadcast, while BR-Termination-2 gives its name to reliable broadcast. Be the sender correct or not, every message br-delivered by a correct process is br-delivered by all correct processes. It follows from these properties that all correct processes br-deliver the same set of messages, and this set contains at least all the messages br-broadcast by the correct processes.

As indicated in Section 2.1, there are signature-free distributed algorithms that build the BR-broadcast communication abstraction on top of asynchronous message-passing systems in which processes may be Byzantine (Bracha, 1987; Imbs and Raynal, 2016; Raynal, 2018).

From one-shot to multi-shot BR-broadcast BR-broadcast is a one-shot communication abstraction: it allows a given process to invoke the operation br_broadcast() only once. Hence, the BR-broadcast instance invoked by p_i can be identified by the process index i.

A simple way to obtain a more general multi-shot BR-broadcast abstraction (MBR-broadcast) consists in associating a specific tag with each invocation of br_broadcast() by a process (Cachin et al., 2011; Raynal, 2018). Implementing the tags with sequence numbers, a process p_i now invokes br_broadcast(sn_i, m), where sn_i is the current value of the local integer variable used by p_i to generate its sequence numbers. This instance is then identified by the pair $\langle i, sn_i \rangle$.

For the MBR-broadcast abstraction the BR-Validity and BR-Integrity property become:

— **MBR-Validity.** If a correct process br-delivers a message m from a correct process p_i with sequence number sn, then p_i br-broadcast m with sequence number sn.
— **MBR-Integrity.** Given a sequence number sn, a correct process br-delivers at most one message m associated with sn from a process p_i.

The properties MBR-Termination-1 and MBR-Termination-2 are the same as their BR-broadcast counterparts, with the addition that sequence numbers are also preserved. Note that at this stage we are not assuming a relationship between sequence numbers and order of emission or of reception of messages. For completeness, multi-shot extensions of the one-shot signature-free BR-broadcast algorithms introduced in Bracha (1987) and Imbs and Raynal (2016) are presented in Appendix A.

Invocation pattern To simplify both the understanding and the presentation of the algorithm implementing the BCO-broadcast abstraction, the invocation previously written br_broadcast(sn_i, m) by process p_i is replaced by br_broadcast$(\langle i, sn_i \rangle, m)$.

3.2 Byzantine Causal Order Broadcast: Definition and a Characterization

This section first proposes a definition of Byzantine causal order broadcast (BCO-broadcast). This definition is actually an extension of the MBR-broadcast abstraction with a new property called Byzantine Causal Order, that defines a partial order on the set of application messages. Then, this section presents a theorem that characterizes BCO-broadcast with a different set of properties that correspond more closely to the algorithm. This theorem will be used in Section 3.4 to prove the algorithm described in Section 3.3.

3.2.1 Definition

BCO-broadcast is a multi-shot communication abstraction that provides processes with the operations denoted bco_broadcast() and bco_deliver() (hence a message is "bco-broadcast" and "bco-delivered"). Similarly to the case of crash-failures, BCO-Broadcast must capture causality in the entire system. This distinguishes it from FIFO-Broadcast that only requires that, for each sender process, messages from this process are delivered in their sending order. We will now give a formal characterization of this requirement.

Let BCO-Validity, BCO-Integrity, BCO-Termination-1, and BCO-Termination-2 denote the same properties as their MBR-broadcast counterparts. The definition of BCO-broadcast is based on the following partial order defined on the (application) messages.

Byzantine causal order Let M be the set of (application) messages bco-delivered by correct processes in an execution E of an algorithm $\mathcal{A}$ that respects the BCO-Validity, BCO-Integrity, BCO-Termination-1, and BCO-Termination-2 properties. A *Byzantine causal order* on M is a partial order $\rightarrow_M$ on M such that:

— **BCO-1.** For any (correct or faulty [2]) process p_i, the set of messages from p_i bco-

2. Faulty nodes are included in BCO-1 for the same reason they are in property BR-Termination-2 (Section 3.1.2). Correct processes do not know beforehand who is Byzantine. Including Byzantine processes in BCO-1 (and in BR-Termination-2) ensures all correct nodes interpret a Byzantine behavior in the same way, while circumventing the thorny problem of detecting and agreeing on who is Byzantine.

delivered by correct processes is totally ordered by $\rightarrow_M$.

— **BCO-2.** If p_i is correct and bco-delivered or bco-broadcast a message m before bco-broadcasting a message m', then $m \rightarrow_M m'$.

A Byzantine causal order $\rightarrow_M$ captures the causality relation on the set of messages bco-delivered by the correct processes. This set of messages, which includes all the messages bco-broadcast by the correct processes, may also include messages from Byzantine processes [3]. BCO-1 ensures the minimal property that all correct processes agree on a same delivery order for the messages they bco-deliver from a given (correct or Byzantine) process. BCO-2 establishes the ordering relationship between the messages bco-broadcast by a correct process and the messages it previously bco-delivered or bco-broadcast.

Byzantine causal order reliable broadcast (BCO-broadcast) The BCO-broadcast abstraction is defined by the BCO-Validity, BCO-Integrity, BCO-Termination-1, and BCO-Termination-2 properties, plus the following causality-related property.

— **BCO-Causality.** There is a Byzantine causal order $\rightarrow_M$ on the set M, such that, for any pair of messages $m, m' \in M$, if $m \rightarrow_M m'$, then no correct process bco-delivers m' before m.

We remark that, by the combination of BCO-1 and BCO-2, BCO-Causality states that a correct process must bco-deliver the messages from any other correct process in their sending order.

3.2.2 A local order property

The aim of this section is to state a theorem giving a characterization of BCO-broadcast. Following the approach introduced in Hadzilacos and Toueg (1994) for crash failures, this equivalence will be instrumental in proving the algorithm implementing the BCO-broadcast abstraction. To this end, let us consider the two following properties on message delivery.

— **BCO-Fifo-order.** If a correct process p_i bco-delivers two messages m and m' from the same process p_k in the order first m and then m', no correct process bco-delivers m' before m (BCO-Fifo-1). Moreover, if p_k is correct, it bco-broadcast m before m' (BCO-Fifo-2).

3. Let us remind that, while a Byzantine process can invoke bco_broadcast() without executing its code, it can also send (correct of fake) messages (which appear in the $\rightarrow_M$ relation and are consequently bco-delivered by the correct processes) without having invoked the bco_broadcast() operation.

— **BCO-Local-order.** If a correct process bco-delivers first a message m and later bco-broadcasts a message m', no correct process delivers m' before m. [4]

The two components of the BCO-Fifo-order property can be understood as follows: BCO-Fifo-1 states that no two correct processes disagree on the delivery order of the messages from the same process p_k (whether p_k is correct or Byzantine). BCO-Fifo-2 states that, if the sender is correct, the delivery order is the same as the sender's broadcast order.

The theorem that follows states a strong equivalence relating BCO-Causality on one side and BCO-Fifo-order plus BCO-Local-order on the other side:

Theorem 3.1 *Under the assumptions of the properties* BCO-Validity, BCO-Integrity, BCO-Termination-1*, and* BCO-Termination-2*, the property* BCO-Causality *is equivalent to* BCO-Local-order *and* BCO-Fifo-order*.*

Proof It can be easily seen that BCO-Causality (through BCO-1 and BCO-2) implies the BCO-Fifo-order property as well as the BCO-Local-order property.

In the other direction, let $\mathcal{A}$ be an algorithm that satisfies the BCO-Local-order and BCO-Fifo-order properties. Let M be the set of messages bco-delivered by correct processes during an execution of $\mathcal{A}$. We have to show that M satisfies BCO-Causality, namely, there is a partial order $\rightarrow_M$ that is included in the (total) bco-delivery order of any correct process, and that satisfies BCO-1 and BCO-2. Let us define the relation $\rightarrow_M$ as follows:

A If any correct process bco-delivered a message m before a message m', both from the same sender process, then $m \rightarrow_M m'$.

B If any correct process bco-delivered a message m before bco-broadcasting m', then $m \rightarrow_M m'$.

C If $m \rightarrow_M m'$ and $m' \rightarrow_M m''$, then $m \rightarrow_M m''$.

Let us now show that, for any correct process p_i, the relation $\rightarrow_M$ is included in the (total) bco-delivery order at p_i.

— **Case $m \rightarrow_M m'$ because of (A).** In this case, m and m' are from the same sender and there exists a correct process that bco-delivered them in the order first m, then m'. Therefore, due to the BCO-Termination-2 and the BCO-Fifo-order properties, p_i bco-delivers them in the same order.

4. This property was introduced in Hadzilacos and Toueg (1994) in the context of crash-prone asynchronous systems.

— **Case $m \rightarrow_M m'$ because of (B).** In this case, due to the BCO-Termination-1, BCO-Termination-2, and BCO-Local-order properties, p_i bco-delivered m before m'.

— **Case $m \rightarrow_M m'$ because of (C).** The local bco-delivery order at p_i is a total order, therefore transitively closed. Hence, if $m \rightarrow_M m'$ due to (C), by induction, p_i bco-delivered m before m'.

It follows from the previous items that $\rightarrow_M$ does not contain cycles, and is consequently a partial order. Let us now show that the partial order $\rightarrow_M$ satisfies the properties BCO-1 and BCO-2.

— **BCO-1.** Let M_i be the set of messages bco-delivered from a process p_i at a correct process. It follows from BCO-Termination-2 that any correct process bco-delivered them. Moreover, due to (A), any pair of messages m and m' of M_i is ordered by $\rightarrow_M$. It follows that, for any (correct or faulty) process p_i, the set of messages bco-delivered from p_i by correct processes is totally ordered by $\rightarrow_M$, which is the property BCO-1.

— **BCO-2.** First, if a correct process bco-delivers m before bco-broadcasting m', due to (B) we have $m \rightarrow_M m'$. Second, if a correct process bco-broadcast m before bco-broadcasting m', due to the BCO-Termination-1, BCO-Termination-2, and BCO-Fifo properties, all correct processes bco-deliver m before m', hence, due to (A) we have $m \rightarrow_M m'$. Combining the two cases, if a correct process p_i bco-delivered or bco-broadcast a message m before bco-broadcasting a message m', we have $m \rightarrow_M m'$, which is the statement BCO-2.

Thus $\rightarrow_M$ is a Byzantine causal order on M and all correct processes bco-deliver the messages of M in an order that contains $\rightarrow_M$, which concludes the proof of the theorem.

$\square_{Theorem\ 3.1}$

3.3 BCO-broadcast on Top of MBR-broadcast: An Algorithm

While at this point the characterization of a causal order in Byzantine systems is well understood, what remains to be shown is that an algorithm that implements such an ordering and that terminates is possible. This is done in this section and the following, in which we respectively introduce Algorithm 3.1 that implements the BCO-broadcast

abstraction and prove its correctness. This algorithm is based on the simple notion of a *causal barrier* (Hadzilacos and Toueg, 1994), which we track internally using a vector clock (Fidge, 1987; Mattern, 1989; Schmuck, 1988). While the statement of this algorithm is close to the one of existing failure-free or crash-tolerant causal broadcast algorithms, its proof (given in Section 3.4) is far from being a simple extension of the corresponding proofs. This heightened difficulty comes from

— the fact that, while in the crash-failure model, a process behaves correctly until it possibly crashes, a Byzantine process can exhibit an arbitrary behavior at any time, and

— the algorithm must ensure that correct processes (i) are never prevented from communicating between themselves, and (ii) always maintain a coherent view of message causality.

Local data structures Each process manages the following local variables.

— sn_i is an integer (initialized to 0) used by p_i to associate a sequence number with the messages it co-broadcasts.

— $co\_del_i[1..N]$ is an array of integers such that $co\_del_i[j]$ counts the number of messages that p_i has bco-delivered from p_j.

— cb_i (where cb stands for "causal barrier") is a set initialized to $\emptyset$, whose meaning is explained below.

Basic principle underlying the algorithm: capturing causal barriers Let us consider the example given in Fig. 3.1, where p_i first invoked bco_broadcast(m_1) and later invoked bco_broadcast(m_2). Moreover, between these two consecutive invocations, m, m' and m'' are the messages bco-delivered by p_i, and these messages are not causally related.

This means that, to ensure a correct bco-delivery of m_2, a correct process p_k is (i) neither allowed to bco-deliver m_2 before m_1 (because m_2 was bco-broadcast after m_1 by the same process p_i), (ii) nor allowed to bco-deliver m_2 before the messages m, m' and m'' (because their bco-deliveries at p_i causally precede the bco-broadcast of m_2). Hence, p_i stores in cb_i the identity of the messages it bco-delivers (after the bco-broadcast of m_1) that are immediate causal predecessors of the next message it will bco-broadcast in the future (here m_2).

Let us remark that, if after the bco-broadcast of m_1 and before the bco-broadcast of

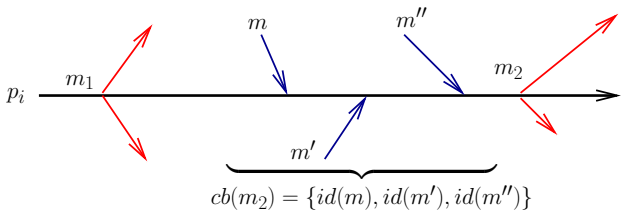


Figure 3.1 – Meaning of the set cb_i

m_2, p_i bco-delivers a message m^* that causally precedes m^+ (where m^+ is m or m' or m''), the identity of m^* is first saved in cb_i and later suppressed from it when m^+ is bco-delivered. As already said, cb_i contains the identities of the messages that are *immediate* predecessors (from a message causality point of view) of the next message that p_i will bco-broadcast.

Notations We use the following notations. Let m be an application message bco-delivered by a correct process, and $\langle i, sn \rangle$ its identity (i.e., the identity of its BCO-broadcast instance).

- $id(m)$ returns the identity of m, namely the pair $\langle i, sn \rangle$.
- $msg(i, sn)$ returns the message m identified by $\langle i, sn \rangle$.
- $cb(m)$ returns the identities of the messages that define the causal barrier of m. In Fig. 3.1, we have $cb(m_2) = \{id(m), id(m'), id(m'')\}$.

Crash failures vs Byzantine failures: validity predicate In the crash failure model, a process always executes correctly its algorithm until it terminates or crashes. Crashes are "syntactic" in the sense that they are only due to the environment in which the application executes. The situation is different in the case of Byzantine faults, which can cause fake data to be communicated by Byzantine processes. As a result, in a Byzantine context, correct processes might need to check the validity of an incoming message in terms of application logic before delivering this message. Although such validity checks are application specific, they need to be injected within the causal broadcast algorithm. This *dependency injection* (Fowler, 2004) allows correct processes to ensure that the causal past of bco-delivered messages only includes messages that are valid in terms of application semantics. This ability to ignore invalid messages while tracking causality is a natural requirement in many applications in order to curtail the disruptive power of Byzantine processes. Hence, according to the application and if needed, an appropriate predicate must allow processes to check the validity of the data contained in the messages they

Algorithm 3.1: BCO-broadcast on top of MBR-broadcast (code for p_i)

init $cb_i \leftarrow \emptyset$; $sn_i \leftarrow 0$; $co\_del_i \leftarrow [0, \ldots, 0]$.

operation bco_broadcast(m) **at** p_i **is**
(1) $sn_i \leftarrow sn_i + 1$;
(2) br_broadcast($\langle i, sn_i \rangle, \langle cb(m), m \rangle$) **where** $cb(m) = cb_i$;
(3) $cb_i \leftarrow \emptyset$;
(4) return().

when ($\langle j, sn \rangle, \langle cb(m), m \rangle$) is br_delivered **from** p_j
(5) wait($(sn = co\_del_i[j] + 1) \wedge (\forall\ \langle k, sn' \rangle \in cb(m),\ co\_del_i[k] \geq sn')$
$\wedge$(validity_pred(j, m))); % application-defined predicate
(6) $cb_i \leftarrow (cb_i \setminus cb(m)) \cup \{\langle j, sn \rangle\}$;
(7) local bco_delivery **of** m **from** p_j;
(8) $co\_del_i[j] \leftarrow co\_del_i[j] + 1$.

receive.

Following an idea introduced in Cachin et al. (2001) and used in Crain et al. (2018), we assume an application-dependent predicate to address this issue. This predicate must be provided by application developers and invoked from within the implementation of the BCO-broadcast abstraction to control or even prevent the bco-delivery of invalid messages [5]. This predicate is named validity_pred(j, m) in the BCO-broadcast algorithm that follows. (The properties this predicate should satisfy will be detailed in the proof of Theorem 3.2.) Its parameters j and m are the identity of the sender process and the application message to be bco-delivered, respectively. For the applications based on BCO-broadcast that do not need such a predicate, it is assumed that it always returns true. An example of the use of this predicate is given in Section 3.5.2.

Algorithm The algorithm is quite simple. When a process p_i invokes bco_broadcast(m), it calls the underlying br_broadcast() operation with the protocol message $\langle cb(m), m \rangle$ and sequence number sn_i, where $cb(m)$ is the current value of cb_i. Then, it resets cb_i to the empty set.

When it br-delivers the message $\langle cb(m), m \rangle$ identified $\langle j, sn \rangle$ from p_j, p_i waits until the associated delivery condition becomes satisfied. This condition states that the previous

5. From an operational point of view, this is similar to the *upcall* mechanism found in operating systems, be them centralized (Clark, 1985) or distributed (Coulouris et al., 2005).

message bco-broadcast from p_j and all the immediate causal predecessors of m have been locally bco-delivered (which is operationally captured by the predicate $\forall\ \langle k, sn' \rangle \in cb(m)$: $co\_del_i[k] \geq sn'$ (line 5)).

When this occurs, p_i updates cb_i to reflect the causality links created by the bco-delivery of m, namely, it suppresses from cb_i the message identities that are in $cb(m)$ (as these messages are no longer immediate predecessors of the next message m' that p_i will bco-broadcast), and replaces them by the identity of m (line 6). Process p_i then bco-delivers m from p_j and accordingly increases $co\_del_i[j]$.

Trivially, the computability assumption and the messages/time complexities of Algorithm 3.1 are the ones of the underlying Byzantine reliable broadcast algorithm it uses.

3.4 Proof of the Algorithm

This section first shows that Algorithm 3.1 satisfies the characterization properties introduced in Section 3.2.2. The proof that the algorithm implements BCO-broadcast then follows from Theorem 3.1.

The proof consists in showing that, while correct processes may bco-deliver messages from Byzantine processes (and consequently these messages participate in the causality relation), they cannot prevent the correct processes from bco-delivering all the messages they bco-broadcast. Hence the central parts are the proof of the Termination properties (see Theorem 3.2), and the proof of BCO-Local-order property (see Theorem 3.3).

As the predicate validity_pred() is application-dependent, the proofs of the BCO-Termination-1, BCO-Termination-2 and BCO-Causality properties assume that the predicate validity_pred() always returns true. In the general case, the validity predicate validity_pred() offers application developers a broad control of the termination properties of the BCO-broadcast algorithm. More precisely, a validity predicate conserves the termination properties of BCO-broadcast if (i) correct processes ensure that the predicate is true for the messages they broadcast, and (ii) a predicate that is true for a message m can only be invalidated by a message from the same sender as m. In this case, the FIFO-delivery of messages and the reliability of the underlying MBR-broadcast used by BCO-broadcast continue to guarantee the termination of BCO-broadcast. As to the causal delivery order of messages, the validity predicate has no influence on the BCO-Causality property of the algorithm, as it can only be used to introduce new delivery constraints, but not to cancel the causal order delivery constraints coded by the wait statement of

line 5.

Theorem 3.2 *Algorithm* 3.1 *satisfies the* BCO-Validity, BCO-Integrity, BCO-Termination-1, *and* BCO-Termination-2 *properties.*

Proof Let us remember that the underlying MBR-broadcast abstraction ensures that all correct processes br-deliver the same set of messages and this set includes at least the messages they br-broadcast.

— **Proof of the BCO-Validity property.** This property directly follows from MBR-Validity. This is because for a message m to be bco-delivered (by a correct process) from a correct process p_i (line 7), it has to be br-delivered from that same process, meaning it was br-broadcast by p_i at line 2) and consequently p_i bco-broadcast m.

— **Proof of the BCO-Integrity property.** The proof follows the fact that each message m that is bco-delivered by a correct process p_i has an identity $\langle j, sn\rangle$, and MBR-Integrity guarantees that no two messages are br-delivered with the same identity. Consequently, p_i cannot bco-deliver another message m' from p_j with the same identity.

— **Proof of the BCO-Termination-1.** It follows from the bco-delivery procedure (lines 5-8) that, for any correct process p_i and at any time, we have $\forall\ \langle j, sn\rangle \in cb_i, co\_del_i[j] \geq sn$. Therefore, when a correct process p_i bco-broadcasts a message m with sequence number sn, it br-broadcasts $(\langle i, sn\rangle, \langle cb(m), m\rangle)$ (line 2), and will br-deliver it (by MBR-termination-2), and the middle term of the wait condition (line 5) is already verified. The first term of the wait condition imposes that the process's own messages are bco-delivered in FIFO order: it becomes true for the first sequence number not yet bco-delivered. Since all sequence numbers are br-delivered, all messages from p_i preceding m (if any) and m itself are eventually bco-delivered by p_i.

— **Proof of the BCO-Termination-2.** Let p_i be a correct process, and $m_1, m_2, \ldots$ the sequence of messages bco-delivered by p_i. We show by induction the following property: if p_i bco-delivers all the messages $m_1, \ldots, m_k$, then all correct processes bco-deliver all the messages $m_1, \ldots, m_k$.

  — Base case: $k = 0$. The property is true because the set of messages bco-delivered by p_i is then empty.

  — General case. Suppose that, for a given $k > 0$, all correct processes bco-delivered $m_1, \ldots, m_k$. We need to show that all correct processes bco-deliver m_{k+1}. When

p_i bco-delivers m_{k+1} the wait condition at line 5 is verified at p_i with the following values (where $sn(m)$ and $proc(m)$ are the sequence number and the sender process of m, respectively, and $\mathsf{max}\{\} = 0$):

$$co\_del_i[\ell] = \mathsf{max}\{sn(m) \text{ such that } m \in \{m_1, \ldots, m_k\} \wedge proc(m) = p_\ell\}. \quad (3.1)$$

Let p_j be another correct process. After it bco-delivered $m_1, \ldots, m_k$, a similar property is verified for p_j:

$$co\_del_j[\ell] \geq \mathsf{max}\{sn(m) \text{ such that } m \in \{m_1, \ldots, m_k\} \wedge proc(m) = p_\ell\}. \quad (3.2)$$

In equation (3.2), "=" is replaced by "$\geq$" since p_j may have bco-delivered messages other than $m_1, \ldots, m_k$. This equation remains true in subsequent steps, as $co\_del_j[\ell]$ can only increase. Thus after p_j bco-delivered $m_1, \ldots, m_k$ and br-delivered m_{k+1}, the second term of the wait condition of line 5 is true. Let us note $p_s = proc(m_{k+1})$ the sender of m_{k+1}. When p_i bco-delivers m_{k+1}, we also have $sn(m_{k+1}) = co\_del_i[s] + 1$, which combined with (3.1) yields

$$sn(m_{k+1}) = \mathsf{max}\{sn(m) \text{ such that } m \in \{m_1, \ldots, m_k\} \wedge proc(m) = p_s\} + 1. \quad (3.3)$$

There is a moment when p_j has bco-delivered $m_1, \ldots, m_k$ and has br-delivered m_{k+1} but not yet bco-delivered it. At this moment, since by MBR-Integrity m_{k+1} is the only message with sequence number $sn(m_{k+1})$ from p_s that p_j ever br-delivers, and because the first part of line 5 imposes a FIFO order delivery, we have $co\_del_j[s] < sn(m_{k+1})$. Including the bound from (3.2), we obtain:

$$sn(m_{k+1}) > co\_del_j[s] \geq \mathsf{max}\{sn(m) \text{ such that } m \in \{m_1, \ldots, m_k\} \wedge proc(m) = p_s\}. \quad (3.4)$$

Rewriting using (3.3) we obtain:

$$sn(m_{k+1}) > co\_del_j[s] \geq sn(m_{k+1}) - 1. \quad (3.5)$$

Thus $sn(m_{k+1}) = co\_del_j[s] + 1$, which is the first condition of line 5, thus the wait statement at this moment and p_j bco-delivers m_{k+1}.

$\square_{Theorem\ 3.2}$

Theorem 3.3 *Algorithm* 3.1 *satisfies the* BCO-Fifo-order *and* BCO-Local-order *properties.*

Proof

— **Proof of the BCO-Fifo property.** Suppose a correct process p_i bco-delivers first a message m, then a message m', both from the same sender p_j. These messages were br-delivered to p_i with some sequence numbers, sn and sn'. As p_i bco-delivers message from any process in the order defined by their sequence numbers, we have $sn < sn'$. It follows then from the MBR-Termination-2 property that all correct processes bco-deliver m and m' with sequence numbers sn and sn', respectively. It follows that any correct process bco-delivers m before m'.

If the sender p_j of m and m' is correct, it associated the sequence number sn with m and the sequence number $sn' > sn$ with m'. It follows that p_j bco-broadcast m before m'.

— **Proof of the BCO-Local-order property.** Let $\rightarrow$ be the relation defined on application messages as follows:

R1 If m is a message bco-delivered by a correct process p_i, and $(\langle i, sn\rangle, \langle cb(m), m\rangle)$ is the associated protocol message, then for each $\langle j, sn'\rangle \in cb(m)$ we have $msg(j, sn') \rightarrow m$.

R2 If m and m' are any two messages successively bco-broadcast by the same correct process p_i, with no other bco-broadcast invocation between them, then we have $m \rightarrow m'$.

The condition in the wait() statement at line 5 directly encodes the relation $\rightarrow$. Therefore, if $m \rightarrow m'$, no correct process bco-delivers m' before m. Let $\rightarrow^*$ denote the reflexo-transitive closure of $\rightarrow$. The bco-delivery order at correct processes also respects $\rightarrow^*$.

Let p_i be a correct process that bco-delivers a message m before bco-broadcasting a message m', and let p_j be a correct process that bco-delivers m'. We have to show that p_j bco-delivers m before m'.

To this end, let m'' be the first message that p_i bco-broadcast after bco-delivering m. We have $m'' \rightarrow^* m'$ (either because $m'' = m'$ or by recursively applying (R2)). What remains to be shown is that $m \rightarrow^* m''$.

Let us now consider the sequence of messages $m_1, m_2, \ldots, m_z$ bco-delivered by p_i, starting at $m_1 = m$ and ending at m_z, which is the last message p_i bco-delivered before bco-broadcasting m''. Let cb_i^k be the value of cb_i just before p_i bco-delivered

m_k (lines 6-7). To prove BCO-Local Order we need to show that

$$\forall\ k \in \{1, \ldots, z\},\ \exists\ m'_k \text{ such that } id(m'_k) \in cb_i^k\ \wedge\ m \rightarrow^* m'_k.$$

For $k = 1$, due to the update of cb_i just before the bco-delivery of m_1, we have $id(m_1) \in cb_i^1$. As $m_1 \rightarrow^* m_1$, the property follows. Let us now assume that, for a given $k < z$ we have $\exists\ m'_k$ such that $id(m'_k) \in cb_i^k\ \wedge\ m \rightarrow^* m'_k$. When p_i bco-delivers m_{k+1} there are two cases.

— If $id(m'_k) \notin cb(m_{k+1})$, due to update of cb_i just before the bco-delivery of m_{k+1} at p_i (line 6), we have $id(m'_k) \in cb_i^{k+1}$. We then take $m'_{k+1} = m'_k$.

— If $id(m'_k) \in cb(m_{k+1})$, due to (R1) we have $m'_k \rightarrow m_{k+1}$. Moreover, due to the update of cb_i at line 5, we have $id(m_{k+1}) \in cb_i^{k+1}$. Hence m_{k+1} is such that $id(m_{k+1}) \in cb_i^{k+1}\ \wedge\ m \rightarrow^* m'_{k+1}$. We then take $m'_{k+1} = m_{k+1}$.

Therefore, in both cases we have $\exists\, m'_{k+1}$ such that $id(m'_{k+1}) \in cb_i^{k+1} \wedge m \rightarrow^* m'_{k+1}$. It follows by induction that $\exists\ m'_z$ such that $id(m'_z) \in cb_i^z\ \wedge\ m \rightarrow^* m'_z$.

By definition, m_z is the last message bco-delivered by p_i before bco-broadcasting m'', and thus $cb(m'') = cb_i^z$. Moreover, we have $id(m_z) \in cb(m'')$, therefore due to (R1) we have $m_z \rightarrow m''$. It follows that $m \rightarrow^* m''$, from which we conclude that p_j bco-delivers m before m', which concludes the proof of the theorem.

$\square_{Theorem\ 3.3}$

Theorem 3.4 *Algorithm* 3.1 *implements the BCO-broadcast abstraction.*

Proof By applying Theorem 3.1 to the results of Theorem 3.2 and Theorem 3.3, we obtain that Algorithm 3.1 satisfies the *BCO-Causality* property. With Theorem 3.2, this proves that it implements the BCO-broadcast abstraction. $\square_{Theorem\ 3.4}$

3.5 Byzantine Causal Broadcast in Action

3.5.1 Building the partial order on the messages

If needed by the application, each correct process p_i can easily build the partial order on the set of messages it bco-delivers. Let $poset_i$ be the message causality directed graph (initially empty) known by process p_i. A vertex is a message identity $\langle j, sn \rangle$. It is initially not marked, and becomes marked when the corresponding message is bco-delivered.

Algorithm 3.2: Extended version of Algorithm 3.1 (code for p_i)

when $(\langle j, sn\rangle, \langle cb(m), m\rangle)$ is br_delivered **from** p_j
(5) wait$((sn = co_del_i[j] + 1) \wedge (\forall\ \langle k, sn'\rangle \in cb(m),\ co_del_i[k] \geq sn'))$
$\wedge$(validity_pred(j, m))); % application-defined predicate
(6) $cb_i \leftarrow (cb_i \setminus cb(m)) \cup \{\langle j, sn\rangle\}$;
(7-M) add to the causality graph $poset_i$ the vertex $\langle j, sn\rangle$
and a directed edge pointing to it from each vertex in cb_i
and a directed edge from the vertex $\langle j, sn-1\rangle$;
(8) $co\_del_i[j] \leftarrow co\_del_i[j] + 1$.

operation bco_deliver() **at** p_i **is**
(9) wait($poset_i$ has a non-marked vertex);
(10) **let** $\langle k, sn\rangle$ be a minimal non-marked vertex;
(11) mark the vertex $\langle k, sn\rangle$;
(12) return(the message identified $\langle k, sn\rangle$).

To build the graph $poset_i$, we modify Algorithm 3.1 as described in Algorithm 3.2. Lines 1-4 are the same. But, for clarity, we use a different invocation model for bco_deliver() than in Algorithm 3.1. When the argument of the wait operation in line 5 is satisfied, Algorithm 3.2 marks the message as "ready to be delivered" by adding a new vertex and its associated directed edges to the graph $poset_i$ (line 7-M, replacing line 7). The application then chooses to deliver messages when needed by invoking bco_deliver() (lines 9-12) explicitly as opposed to using a callback like in Algorithm 3.1.

3.5.2 Money transfer based on BCO-broadcast

A money transfer algorithm Guerraoui et al. (2019b) presented an important result related to money transfer, a service that is for instance provided by cryptocurrencies (Nakamoto, 2009b). One of the main issues in such systems is to prevent *double spending attacks*, i.e. to prevent users from using the very same money to buy different goods. Guerraoui et al. (2019b) proposed two algorithms to solve the money transfer problem when, for each account, there is a single process that is allowed to transfer money from this account to other accounts. This process is called the *owner* of the account [6].

6. Note that the notion of *account ownership* here is not the same as in the case of cryptocurrencies, where ownership of an account is defined by the knowledge of a cryptographic secret key that allows one to sign money transfer operations from the account. Here, account owners must be processes in the system.

The first algorithm, which only tolerates crash failures, relies on a single writer/multi reader *snapshot object* (Afek et al., 1993), which can be implemented despite asynchrony and process failures [7]. The second algorithm is made to tolerate Byzantine failures. It eschews snapshot objects and directly implements money transfers in an asynchronous message-passing system where up to $t < N/3$ processes can be Byzantine. This algorithm, which uses an underlying secure broadcast abstraction (Malkhi and Reiter, 1997), requires processes to exchange message carrying "histories" which are data structures that grow indefinitely.

A rewriting of Guerraoui et al.'s money transfer algorithm The reader is referred to Guerraoui et al. (2019b) for a formal specification of the money transfer problem and the proof of the associated algorithms. Algorithm 3.3, described below, is a simple rewriting based on BCO-broadcast of the message-passing algorithm presented and proved in Guerraoui et al. (2019b). A process invokes the operation $\mathsf{read}(j)$ to know the balance of p_j, and the operation $\mathsf{transfer}(j)$ to transfer the amount v to p_k. Without loss of generality, we assume a correct process p_i does not transfer money to itself.

As in Guerraoui et al. (2019b), a money transfer of the quantity v by a process p_j to a process p_k is internally represented by a pair $\langle k, v\rangle$, and each process p_i manages a local multiset [8] denoted $hist_i[1..N]$ that records all the transfer operations as known by p_i. More explicitly, $\langle k, v\rangle \in hist_i[j]$ means that p_i knows that p_j transferred the quantity v to p_k. In particular, $hist_i[i]$ contains all the transfers performed by p_i. In our formulation, two transfers of the same amount v by the same process p_j to the same process p_k, appear as identical pairs ($\langle k, v\rangle$) in the multiset $hist_i[j]$ of a process p_i. This multi-set representation is equivalent to that of Guerraoui et al. (2019b), in which $hist_i[j]$ is a set that stores quadruples of the form $\langle j, sn, k, v\rangle$ instead of pairs: in the quadruple, sn is the sequence number associated by p_j with the transfer identified by $\langle k, v\rangle$. [9]

BCO-broadcast-based description of the algorithm Let us first look at the function $\mathsf{balance}(j)$ invoked by p_i. All the processes are initialized with the same constant array $init[1..N]$ where $init[k]$ is the initial value of p_k's account. The transfers from p_ℓ to

7. Namely, the consensus number of a snapshot object is 1 (Herlihy, 1991; Raynal, 2019).

8. Sometimes also called a *bag* or a *pool*, a multiset is a collection of elements in which the same element can appear several times. As an example, while $\{a, b, c\}$ and $\{a, b, a, b, b, c\}$ are the same set, they are different multisets.

9. In both algorithms described in Guerraoui et al. (2019b) and in the rewriting we present, additional bookkeeping information can be stored in $hist_i[j]$, according to application requirements.

the other processes that are known to p_i are stored in $hist_i[\ell]$, as explained just above. Hence, from p_i's local point of view, *plus* collects all the transfers to p_j, (line 1), while *minus* collects all the transfers from p_j (at line 2). When invoked by a process p_i, the operation $\mathsf{read}(j)$ returns the current value of p_j's account, as known by p_i (line 3).

When invoked by a process p_i, the operation $\mathsf{transfer}(j, v)$ transfers the quantity v from p_i to p_j. To this end, p_i first checks its own account balance (line 5). If the balance is not greater than or equal to v, the transfer aborts (line 6). Otherwise, the transfer can occur. In this case, p_i bco-broadcasts the message $\text{TRANSFER}(\langle j, v\rangle)$ (line 7) and waits until the transfer is locally committed (line 8).

When p_i bco-delivers a message $\text{TRANSFER}(\langle k, v\rangle)$ issued by a process p_j, it locally updates its view concerning p_j's transfers, namely $hist_i[j]$ (line 10). Moreover, if p_j is p_i, it switches $done_i$ to `true` (line 11), which allows it to terminate its transfer (line 8).

To prevent over-spending, the predicate $\mathsf{validity\_pred}(j, m)$ where $m = \text{TRANSFER}(\langle k, v\rangle)$ must guarantee that each correct process p_i locally sees that p_j owns enough money in its account to transfer the quantity v to p_k. Hence $\mathsf{validity\_pred}(j,\text{TRANSFER}(\langle k, v\rangle)) \stackrel{\text{def}}{=} \mathsf{balance}(j) \geq v$ [10] (line 12). Since a money transfer that has been accepted can never be undone, double-spending attacks cannot occur in this system. It is not difficult to see that the validity predicate of Algorithm 3.3 fully maintains the termination of BCO-broadcast, as: (i) a correct process p_j ensure that $\mathsf{balance}(j) \geq v$ before broadcasting $\text{TRANSFER}(\langle k, v\rangle)$, and (ii) $\mathsf{balance}(j) \geq v$ can only be invalidated by withdrawing money from p_j's account, which only p_j can do.

10. As the reader can see, $\mathsf{validity\_pred}(j,\text{TRANSFER}(\langle k, v\rangle))$ ensures that the bco-delivery of a transfer message from p_j to p_k and the check that p_j has enough money appear as being done in a single atomic step.

Algorithm 3.3: BCO-broadcast-based rewriting of Guerraoui et al. (2019b)

(code for p_i)

init $init[1..N]$: constant array where $init[k]$ is the initial value of p_k account;
$hist_i[1..N] \leftarrow [\emptyset, \cdots, \emptyset]$.

function balance(j) **is**
(1) $plus \leftarrow$ sum of the v_x such that $\langle j, v_x\rangle \in \cup_{\ell\in[1..N]} hist_i[\ell]$;
(2) $minus \leftarrow$ sum of the v_x such that $\langle -, v_x\rangle \in hist_i[j]$;
(3) return($init[j] + plus - minus$).

operation read(j) **is**
(4) return(balance(j)).

operation transfer(j, v) **is**
(5) **if** (balance(i) $< v$)
(6) **then** return(`abort`)
(7) **else** $done_i \leftarrow$ `false`; bco_broadcast TRANSFER($\langle j, v\rangle$);
(8) wait ($done_i$); return(`commit`)
(9) **end if**.

when TRANSFER($\langle k, v\rangle$) **is** bco_delivered **from** p_j **do**
% The occurrence of this event is immediately followed
% by the atomic execution by p_i of the next two lines
(10) $hist_i[j] \leftarrow hist_i[j] \cup \{\langle k, v\rangle\}$;
(11) **if** $(j = i)$ **then** $done_i \leftarrow$ `true` **end if**

predicate validity_pred(j,TRANSFER($\langle k, v\rangle$)) **is**
(12) balance(j) $\geq v$.

3.6 Conclusion

After proposing a definition of the Byzantine causal broadcast abstraction, this chapter presented an algorithm that implements this specification in asynchronous message-passing systems (a simplified version of it can be trivially obtained if one is interested in FIFO message delivery only). As far as we know, this algorithm is the first ensuring causal message delivery in the presence of Byzantine processes.

Its design relies on a modular decomposition (such as the ones presented in Hadzilacos and Toueg (1994) and Raynal (2018)), which makes algorithms easier to understand and prove. In particular, the algorithm relies on an underlying multi-shot reliable broadcast algorithm. It works with any such underlying algorithm and inherits is computability bound and time/message complexities. When instantiated with the reliable broadcast algorithm described in Bracha (1987) it requires $t < N/3$ (which is resilience optimal), 3 consecutive communication steps and $(N-1)(2N+1)$ protocol messages. When instantiated with the reliable broadcast algorithm described in Imbs and Raynal (2016) it requires $t < N/5$, 2 consecutive communication steps (which is optimal), and $N^2 - 1$ protocol messages.

Starting with the next chapter, we will return to the case of large-scale open networks in presence of churn. We will first study how to build a causal broadcast abstraction in such a context, before moving on to stronger abstractions that implement Byzantine consensus.

Chapter 4

Merkle Search Trees

Recent weakly-consistent replication techniques, and in particular techniques based on CRDTs, have focused primarily on leveraging the causal broadcast primitive in order to ensure reliable causal delivery of operations (Almeida et al., 2014; Linde et al., 2016), meaning that they never need to compare two full states directly. In the previous chapter, we saw how this primitive could be extended to the case of closed, static systems of N nodes, in presence of Byzantine nodes.

However even in the absence of Byzantine actors, these approaches show their limit in open networks where many nodes may join and leave. Causal broadcast has only recently been extended in a scalable fashion to networks with dynamic structures (Nédelec et al., 2018a) and the practicality of this approach has not yet been demonstrated. Alternative methods to ensure causal delivery exist in the form of anti-entropy protocols that uses vector clocks (Lamport, 1978) to deliver missing messages (Renesse et al., 2008). However, vector clocks require metadata that grows linearly with the number of nodes, past and present, that have participated in the network, which is an important scalability barrier even in the case of extensive optimisations (Goncalves et al., 2017).

In order to target open networks while avoiding these limitations, we will take an interest in this chapter in pure *state-based* CRDTs and to their fundamental issue: how to efficiently implement remote state merge for large states between nodes that might have had no previous interaction and might know nothing of one another's state. This is similar to the problem of anti-entropy, which was studied before the formalization of CRDTs. In this chapter, we will use the terms anti-entropy, state merge and state reconciliation as synonyms. We target more specifically CRDTs implementing sets and maps, two fundamental building blocks that can be combined with other CRDT semantics, and seek to resolve differences between two very large sets or maps with low latency and minimum bandwidth usage. As we saw in the introduction that CRDTs are generally defined and used without considering Byzantine behavior, we place this chapter in the context of networks of nodes that trust one another, i.e. we suppose that there are no

Byzantine nodes. However the new technique we develop in this chapter applies naturally to trustless networks when combined with blockchain consensus algorithms.

A common strategy for the reconciliation of sets and maps exploits Merkle trees (Merkle, 1987), a hash-based data structure that can be used to rapidly identify set-differences and is used for instance in Amazon's DynamoDB (DeCandia et al., 2007). Unfortunately, usual anti-entropy algorithms that use Merkle trees do not preserve the order of elements, making them particularly inefficient when updates are applied on sequences of close-by items. This is a very common scenario, a basic example being the case where elements are messages ordered by timestamps, the case we will study here.

In this chapter we propose to overcome this limitation thanks to a novel data structure, which we call a *Merkle search tree*, that deterministically builds a balanced search tree from a set of items and encodes it as a Merkle tree. We demonstrate how Merkle search trees can be used to build a causally consistent event store that ensures eventual delivery of all past messages, here called *events*, to all connected nodes. We compare this approach to a vector clock-based approach (Renesse et al., 2008) as well as to a Merkle tree construction that does not preserve order, and show that, in large networks and under low or moderate update rates, Merkle search trees provide the best trade-off: a 66% reduction of bandwidth usage was achieved in our simulation when compared to the vector-clock approach, as well as a 34% improvement in our consistency measure and a 32% improvement in 99th percentile delivery delay. When compared to Merkle trees without order, Merkle search trees were also better on all three metrics. In addition to being useful in the context of causal broadcast, Merkle search trees can also be used in the context of blockchains, as we will discuss in Section 4.6.3. [1]

4.1 Prerequisites

Our work draws from several well-studied domains in distributed systems as well as databases and file systems, which we present in this section.

1. The work presented in this chapter has been the subject of the following publication: Alex Auvolat and François Taïani (2019), « Merkle Search Trees: Efficient State-based CRDTs in Open Networks », *38th Symposium on Reliable Distributed Systems (SRDS)*, IEEE, pp. 221–230, DOI: 10.1109/SRDS47363.2019.00032

4.1.1 CRDTs

In this chapter we use the formulation of CRDTs as a join-semilattice on replica states, that is a CRDT is a set of possible states $\mathbb{V}$ with a symmetric join operation $\sqcup$. Operations on a state $x \in \mathbb{V}$ consist in changing x into a state x' that is strictly superior according to $\sqcup$, i.e. such that there is an item $o \in \mathbb{V}$ such that $x' = x \sqcup o$. For instance if the data type we want to implement is a grow-only set, $\mathbb{V}$ is the space of possible sets and $\sqcup$ is the set union operator. Adding an element e to a set x consists in changing x into the state $x' = x \cup \{e\}$.

The operator $\sqcup$ is also used to combine concurrent operations done at different nodes and resolve them deterministically: if a node is in state x_1 and another is in state x_2, both will resolve to state $x_1 \sqcup x_2$ (which is the same as $x_2 \sqcup x_1$). For instance, for a grow-only set CRDT, the merge operation is defined as set union, such that the resulting state is defined as the set containing all items that have been added at all nodes. More complex CRDTs can be defined to implement additional operations such as deletion or different data types such as maps or counters, the only requirements being to implement a commutative, associative and idempotent join operator $\sqcup$.

In this chapter we focus on set and map CRDTs and how to implement efficient computation of $x_1 \sqcup x_2$ when x_1 and x_2 are located on two different nodes, where efficiency is measured in terms of network round-trips and bandwidth usage.

4.1.2 Hash functions and Merkle trees

Hash functions and the checksums they produce are commonly used to identify data and check its integrity and/or authenticity. They therefore might appear as a natural choice to compare data residing on different computers, and help implement an efficient distributed reconciliation mechanism. Unfortunately, to check a very large piece of data, the hash function must be calculated on the whole data piece, a particularly costly proposition when dealing with large distributed objects, such as a typical key-value data store. To reduce this cost, the data can be chunked and each chunk hashed independently, but then a single hash is replaced by a list of hashes that grows linearly with the data size, which remains problematic.

To overcome this linear cost, Merkle introduced a method that is able to hash large pieces of data by chunking the data and computing a tree of hashes, where leaves are hashes of data chunks and nodes are hashes of the concatenations of their children's

values (Merkle, 1987). The single root hash is sufficient to identify all the data, and the validity of a single block can be checked by walking through the path to the corresponding leaf, thus verifying only a logarithmic number of hashes provided that the tree is well balanced. This verification can also occur without having access to the whole data.

The original Merkle trees are binary trees, however the core principle of Merkle trees can be generalized and used in other contexts. Most notably, projects such as ZFS (Y. Zhang et al., 2010) and IPFS (Benet, 2014) make use of recursive hashing in the fashion of Merkle trees in order to guarantee authenticity and integrity of whole file systems. Such recursive hashing of data structures also allows for data de-duplication (Y. Zhang et al., 2010; Benet, 2014), in which case the Merkle tree becomes a DAG where nodes may have several parents.

In distributed systems, Merkle trees have shown to be particularly useful: given a deterministic way to encode data sets in Merkle trees, two nodes can determine whether they have the same version of the data simply by exchanging and comparing their root hashes. If their versions differ, they are able to identify which branches of the tree contain changes, as branches that represent the same contents have the same hash on both nodes, therefore allowing them to exchange only the differing parts of the data set.

Merkle trees come in different shapes. The standard binary Merkle trees can be used to identify and exchange numbered sequences of data blocks, but they are not able to detect differences efficiently on arbitrary key spaces when the sets of keys present at two nodes might not be the same. Database systems such as Dynamo (DeCandia et al., 2007) use Merkle trees to exchange and repair states that consist of arbitrary key-value mappings, however to ensure that a tree is well balanced they must destroy key ordering and project the keys into a uniform distribution by hashing them, which makes this method sub-optimal in scenarios where sequences of close-by keys are updated. A Merkle trees construction that preserves key order while staying balanced on arbitrary key ranges was introduced by Tamassia and Triandopoulos (2007), however this method is not deterministic: a same dataset may have several Merkle tree representations, which makes it unfit for anti-entropy. A first deterministic balanced Merkle tree construction that preserves key order on arbitrary key ranges was introduced using binary treaps (Crosby and Wallach, 2009). The construction we propose achieves the same properties with wider tree nodes, thus making for shallower trees and reducing the number of round-trips required for remote difference computation. This also reduces the number of hashes that need to be computed and stored compared to binary Merkle trees.

4.1.3 B-trees

The Merkle search tree construction we propose uses a structure similar to B-trees (Bayer and McCreight, 1970), albeit with some key differences. B-trees are indexing data structures used in relational database management systems (RDBMS). They implement shallow search trees by having nodes with large out-degrees, allowing faster traversal than standard binary search trees. Nodes typically have out-degrees such that the machine representation of a tree node is the size of a memory page. In a B-tree, a node is composed of a set of values that define how the space of keys is split, and of a set of pointers to the sub-trees containing the values for each interval in the split. The Merkle search trees we introduce also contain intermediate nodes whose values serve to split the key space for their children. Their nodes, however, do not have a constant size and their size is bounded only probabilistically.

4.2 The Merkle Search Tree Data Structure

To implement an efficient reconciliation procedure for CRDTs implementing large maps and sets, we propose a tree construction, which we call a *Merkle search tree* (abbreviated MST), that is to our knowledge the first to combine the following features:

— a given set of items has a unique deterministic representation as a tree;
— key order is preserved;
— trees are always balanced (probabilistically);
— nodes can be wide in order to reduce tree depth, with the average size of nodes being controllable by an algorithm parameter (B).

The combination of these properties makes remote tree comparison for anti-entropy extremely efficient. They also make Merkle search trees efficient for storing database indexes and accessing ordered data in blockchain systems.

4.2.1 MST construction

Merkle search trees implement an ordered set with elements taken in a totally ordered space $\mathbb{K}$. In order to implement maps, elements can be accompanied by a tag from a set $\mathbb{V}$ (the *values*, in which case $\mathbb{K}$ becomes the set of *keys*). $\mathbb{V}$ contains a default element $\perp$ ('bottom'), used to indicate that a key is absent from the map f from $\mathbb{K}$ to $\mathbb{V}$.

The MST construction is based on a hash function over arbitrary byte strings that is

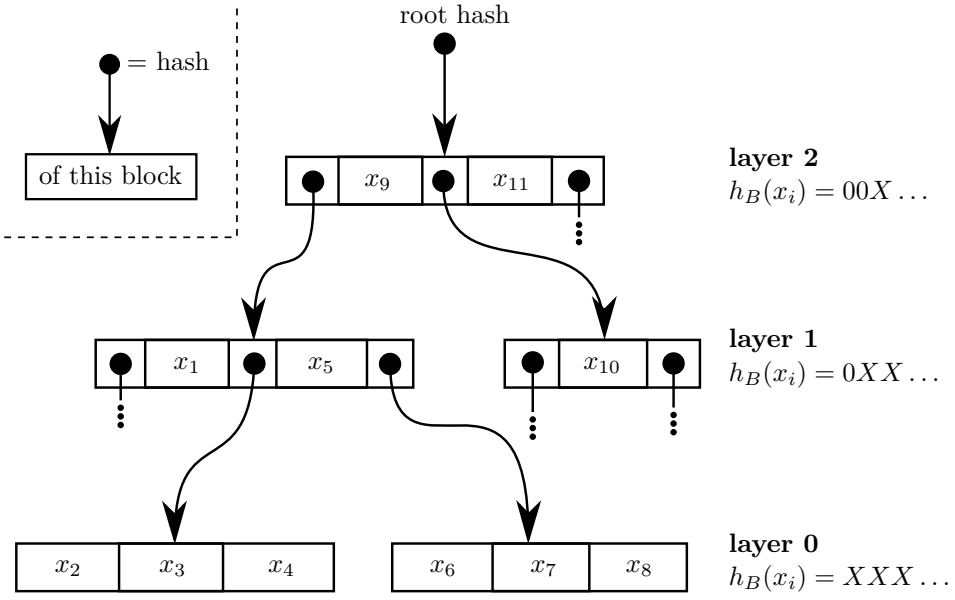


Figure 4.1 – Structure of a Merkle search tree

assumed to be collision-resistant: we assume the probability of finding two strings that have the same hash to be negligible. This is a common assumption which is essential to the Merkle tree construction. We also assume that the hash function projects uniformly to values of a certain interval whose size is a power of B. This additional property is required in our construction as we also use the hash function for the secondary purpose of providing a deterministic source of randomness over the space $\mathbb{K}$. Both of these properties are achieved in practice by modern hash functions such as SHA-256.

An MST is a search tree, similar to a B-trees in the sense that internal tree nodes contain several values that define a partition of $\mathbb{K}$ in which the children values are separated. The tree is divided in layers which are numbered starting at layer 0 which corresponds to the layer of leaf nodes (Figure 4.1). The tree nodes in a layer l are blocks of consecutive items whose boundaries corresponds to items of layers $l' > l$. For instance, in Figure 4.1, the first nodes shown for layer 0 stored the values $\{x_2, x_3, x_4\}$ which are contained between the boundaries x_1 and x_5, with x_1 and x_5 stored in the layer 1. Similarly to a previous construction (Crosby and Wallach, 2009), deterministic randomness obtained by hashing the values is used to determine tree shape. Values stored in the MST are assigned a layer by computing their hash and writing that value in base B, which we will refer to as $h_B(x)$. The layer $l(x)$ to which an item x is assigned is the layer whose number is the length of

the longest prefix of $h_B(x)$ constituted of only zeros.

4.2.2 MST unicity and operations

The construction we have just described is deterministic: it produces a single possible representation as an MST for a given set of items, an essential property to implement a reconciliation procedure. The implementation of standard operations follows from this definition.

When using MSTs as maps, we can efficiently implement read operations $\texttt{get}(f, k)$ for single items and $\texttt{getrange}(f, [k_0, k_1))$ for ranges of consecutive items in a tree f, as well as write operations $\texttt{put}(f, k, v)$ and $\texttt{delete}(f, k)$. In the case where a put operation inserts an item x at a non-leaf layer $l > 0$, some of the tree nodes at lower levels $l' < l$ might need to be split in two at boundary x. Similarly when values are deleted at non-leaf layers, some nodes at lower layers might need to be merged. At most one split or merge can happen at each layer $l' < l$ for a given put or delete operation, therefore the complexity of these operations is proportional to the depth of the tree.

As a consequence of structural unicity, inserting the same elements in different orders will always converge to the same representation, and the Merkle hash of the root will be the same. Structural unicity is also valid at intermediary nodes: two intermediary nodes have the same Merkle hash if and only if the sub-trees starting at these nodes contain exactly the same items. Therefore when comparing two trees in order to find differences, we can skip whole sub-trees when they are found to have the same Merkle hash as we know that there are no differences between them.

4.2.3 Structural properties: balance and depth

Since we use a hashing function that projects uniformly to strings of constant length that write out numbers in base B, the probability of being in layer l for an item is $p_l = \left(\frac{1}{B}\right)^l \frac{B-1}{B}$. It is easy to see that the average number of values at layer l is B times the average number of values at layer $l + 1$. Since values of layer l are split at boundaries that are all values at layers $l' > l$, nodes of a layer l have on average $B - 1$ values stored and, at non-leaf layers, B children. The probability of a node having many more values and children than these average values decreases exponentially, therefore we can bound the node size with arbitrarily high probability.

The depth of the tree is the maximum level that an item may be assigned to. The

probability of having an item at layer l is bounded by np_l (where n is the number of elements in the tree), which decreases exponentially with l. Therefore we can bound the tree depth as $log_B n$ plus a constant c with a probability that decreases exponentially in c.

Since the tree depth is about $log_B n$ and nodes have a constant number of items with high probability, we conclude that the tree has a well balanced structure and therefore reads, puts and deletes can be implemented in $O(log_B n)$ complexity.

4.2.4 Efficiency for dataset comparison

The comparison of two Merkle search trees stored at two different nodes is efficient: if we are comparing f which is on node p_1 with g which is on node p_2, then nodes p_1 and p_2 need exchange only $O(d \log_B n)$ messages, with $d = |\{x | f(x) \neq g(x)\}|$, and $n = \max(|f|, |g|)$. Supposing elements of $\mathbb{K}$ and $\mathbb{V}$ are of constant size, exchanged message are of average size proportional to B.

The average out-degree of inner tree nodes can be controlled by changing the value of B. Using larger values of B makes for nodes that contain more data, thus requiring more bandwidth expenditure when exchanged between peers, but makes for a shallower tree and therefore requires less network round trips to obtain the full set of nodes leading to a leaf. In our experiments, we fix $B = 16$.

4.2.5 Merkle search trees as CRDTs

If $\mathbb{V}$ is a CRDT with a merge operation $\sqcup_{\mathbb{V}}$ such that $\forall x, \bot \sqcup_{\mathbb{V}} x = x \sqcup_{\mathbb{V}} \bot = x$, then Merkle search trees implement a CRDT on $\mathbb{K} \to \mathbb{V}$ defined by the point-to-point application of $\sqcup_{\mathbb{V}}$: $(f \sqcup g)(x) = f(x) \sqcup_{\mathbb{V}} g(x)$.

The sequence of states a CRDT takes is supposed to be monotonic with respect to $\sqcup$, as explained in Section 4.1.1: a transition from a state x to a state x' must be of the form $x' = x \sqcup o$. In the case of Merkle search trees, this means restricting the set of allowed operations to those that lead to monotonic sequences for each individual key. Therefore operations `put` and `delete` must not be used directly, but rather updates must be of the form $\texttt{update}(f, k, v) = \texttt{put}(f, k, \texttt{get}(f, k) \sqcup v)$.

By selecting $\mathbb{V}$ appropriately, we can obtain various CRDTs. For instance if $\mathbb{V} = \{\bot, \top\}$ is a boolean indicating if an item is present or not, we obtain a grow-only set defined on $\mathbb{K}$. If $\mathbb{V}$ is a last-writer-wins register with a version number then we get a key-value store

Algorithm 4.1: Basic state-based CRDT anti-entropy protocol

```
1  initialization
2  |  state ← ⊥
3  |  fanout ← algorithm parameter
4  function do(op)
5  |  state' ← state ⊔ op
6  |  if state' ≠ state then
7  |  |  Gossip state' to fanout random peers
8  |  |  state ← state'
9  on receive Gossip new_state
10 |  state' ← state ⊔ new_state
11 |  if state' ≠ state then
12 |  |  Gossip state' to fanout random peers
13 |  |  state ← state'
```

with last-writer-wins reconciliation. Any existing CRDT type can be used as the value type $\mathbb{V}$, yielding a map CRDT construction with efficient detection of differing items.

4.3 CRDTs in Large-scale Open Networks

The Merkle search trees we have presented can be used to implement a particularly efficient pair-wise distributed reconciliation procedure for set or maps CRDTs. This procedure allows two nodes to converge to a unique state on the basis of a voluntary exchange that does not require any node to have prior knowledge of the other node's state. We combine this procedure with a simple gossip-based algorithm to allow for efficient dissemination of updates in open networks with high churn rates.

4.3.1 Gossip-based reconciliation for state-based CRDTs

Our approach builds on a simple distributed state-update method for state-based CRDTs shown in Algorithm 4.1 in its basic form, without the use of MSTs. Algorithm 4.1 adopts a reactive gossip strategy: when a node does an operation that results in a state transition from x_0 to x_1, where $x_1 = x_0 \sqcup o$, it gossips x_1 to some random peers. When a node receives an incoming state x_1 when it was in state x'_0, it does the transition to $x'_0 \sqcup x_1 = x'_1$ and in the case where $x'_1 \neq x'_0$ it in turn gossips x'_1 to some random peers. The selection of random peers assumes a peer sampling service (Jelasity et al., 2007) that

is able to give the identity of some nodes of the network taken at random.

Algorithm 4.1 implements reactive push-only exchanges. Other methods with pull or push/pull strategies are also possible, and can also be extended with a periodic component to realize a traditional anti-entropy protocol.

4.3.2 Using Merkle search trees to implement large CRDTs

When using Algorithm 4.1 to implement large map-like CRDT types such as sets or key-value stores, a naive implementation that exchanges whole states rapidly becomes intractable. In such cases, MSTs can be used to reduce dramatically network bandwidth usage at the cost of several round-trips between nodes. Algorithm 4.2 shows how this can easily be achieved in the background, without blocking local operations: the current state is not changed until all the remote information necessary for the merge is obtained (line 12), and get or update operations can continue to be executed on this state. In this algorithm, a set of remote MSTs that we wish to merge is kept in a buffer (lines 4, 14, 18) and all missing blocks are requested to the remote peer (line 13). Once all the blocks are locally available, the merge operation is done without network communication (line 16) and the local MST is updated.

4.3.3 Causal consistency

Algorithms 4.1 and 4.2 implement causal consistency in the following sense: if a node does an operation that results in a transition from x to $x' = x \sqcup o$ after having read from the state x, then the causal past of the put operation can be described as being included in x, and every other node that observes the put will observe its causal past x (since x' contains x), or a successor of x according to $\sqcup$. This algorithm and resulting property is applicable to all CRDTs that are defined by a merge $\sqcup$ and Merkle search trees extend this naturally to maps of CRDTs. This is done without any conditions on the network topology as is required by causal broadcast (Friedman and Manor, 2004; Nédelec et al., 2018a): it results from the fact of doing a full state merge at every gossip event.

If causal consistency is not required and eventual consistency is sufficient, Algorithm 4.2 can be optimized by gossiping individual single operations and applying them as soon as they are received, while executing periodic merges of the full data structure in parallel. In that case, periodic merges acts as a termination mechanism for ensuring eventual consistency. The use of explicit merges allows us to reduce the gossip fanout or

time-to-live for single operations in order to save bandwidth, to values that would result in e.g. only 99% network coverage, and count on the full merge operation as an anti-entropy procedure for the small proportion of nodes that will miss a gossip event.

4.3.4 A note on crashes

Algorithm 4.2 uses a limit on the number of simultaneous merge operations that can be happening in order to preserve bandwidth (line 8). Therefore it may get stuck if it is waiting for a reply from a node that has crashed. Removing the limit on simultaneous merges is not desirable in practice as it might spread bandwidth usage too thinly across many concurrent pairwise merges and lead to high tail latencies. Therefore a fault detector or an approximate fault detector must be employed to cancel merges when a node has crashed. Canceling a merge when a node has not crashed but is just slow is not a problem as no safety property is violated, therefore a timeout on merge operations is a viable solution. However in order for termination (eventual consistency) to be achieved, nodes must be able to complete at least some merge operations.

4.3.5 Extension to distributed MSTs

Merkle search trees are a persistent data structure in the sense of functional programming. This means that when doing a $\texttt{put}(k, v)$, the obtained MST needs not replace the previous MST but both can remain available simultaneously at a low storage cost by sharing memory for common blocks. Hashes in this case play a role similar to that of traditional pointers, with the key differences that hash values can now be used for automatic de-duplication of subtrees that contain the same values, and remain meaningful outside of the nodes on which they have been computed. Although in this chapter we use MSTs as a local data-structure that is stored on a single node, the above features could be exploited to implement *Distributed MSTs*.

Storage location A distributed MST (or D-MST) is an MST where the data structure is split over many nodes in a network, for instance using a DHT. This is easy to do with any Merkle DAG data structure, as introduced by IPFS (Benet, 2014), and has already been done for binary search trees (Tamassia and Triandopoulos, 2007). In our case, tree nodes correspond to blocks that are stored independently in the DHT and identified by their hash. If no caching is used, a D-MST requires $O(\log_B n \times \log N)$ network round trips

Algorithm 4.2: State-based CRDT anti-entropy with Merkle search trees

```
1  initialization
2  |  fanout ← algorithm parameter
3  |  max_merges ← algorithm parameter
4  |  state ← ⊥; merges ← ∅
5  function do(op)
6  |  handle_update(op)
7  on receive Gossip root_hash from peer
8  |  if root_hash ≠ h(state) ∧ |merges|≤ max_merges then
9  |  |  handle_merge(root_hash, peer)
10 function handle_merge(root_hash, peer)
11 |  Q ← missing_blocks(root_hash, local_store)
12 |  if Q ≠ ∅ then
13 |  |  Request Q to peer
14 |  |  merges ← merges ∪ {(root_hash, peer)}
15 |  else
16 |  |  update ← load(root_hash, local_store)
17 |  |  handle_update(update)
18 |  |  merges ← merges\{(root_hash, peer)}
19 on receive Request Q from peer
20 |  R ← {block ∈ local_store|h(block) ∈ Q}
21 |  Reply R to peer
22 on receive Reply R from peer
23 |  if ∃(root_hash, p) ∈ merges|p = peer then
24 |  |  save(R, local_store)
25 |  |  handle_merge(root_hash, peer)
26 function missing_blocks(root_hash, local_store)
27 |  (omitted for brevity; returns the hashes of the blocks of the tree identified by root_hash that
   |  are missing from local_store)
28 function handle_update(update)
29 |  state' ← state ⊔ update
30 |  if state' ≠ state then
31 |  |  Gossip h(state') to fanout random peers
32 |  |  state ← state'
```

for all operations, where N is the number of nodes in the network. Caching can reduce this to $O(\log_B n)$ for get operations in favorable conditions (no churn) (Tamassia and Triandopoulos, 2007).

Garbage collection Garbage collection in the D-MST scenario is harder than in the local MST case. An option is to expire all stored blocks after a certain time, and to have a distributed keep-alive mechanism that traverses the data structure resetting the expiry counter. Details of such a system are outside the scope of this thesis.

4.4 Application: a Causally-consistent Distributed Event Store

We illustrate the benefits of CRDTs built with Algorithm 4.2 on the example of a *Causally-consistent Distributed Event Store*, a data-structure that encapsulates the notion of causal broadcast. Causal broadcast is a fundamental primitive of distributed algorithms that is usually directly implemented using *send/receive* network primitives and vector clocks (Linde et al., 2016) which are the main alternative to our method and which we show do not scale efficiently. For completeness, let us note that alternative approaches to Causal Broadcast exist for static overlays (Friedman and Manor, 2004), which have been recently extended to a dynamic setting (Nédelec et al., 2018a), but the practicality of this latest development has not yet been demonstrated.

Causal Broadcast is often one of the building blocks of operation-based CRDTs, such as δ-CRDTs (Almeida et al., 2014). In this section we reverse this usual perspective: instead of deploying a message-passing algorithm to ensure the delivery properties of causal broadcast, to then implement a distributed object, we provide causal broadcast by building a storage of all the events produced by the system, which we call an *event store*. We leverage Merkle search trees to provide an efficient way to propagate missing updates in an ad-hoc manner between two peers that have no a-priori knowledge of each other's states. A direct application of such a system would allow for example to build a peer-to-peer chat room or a log system but it may also be used as a primitive for more sophisticated algorithms.

In the following we study the theoretical complexity of such an approach, and compare it to *Scuttlebutt*, an anti-entropy algorithm introduced in Renesse et al. (2008) which uses vector clocks. We confirm this analysis experimentally in the next section. Both methods

employ a push-pull gossip strategy to ensure update propagation on a large network. Our theoretical analysis is synthesized in Table 4.2.

n : number of past events
d : number of new events in anti-entropy round
p : number of nodes, past and present
N: number of nodes currently connected
λ : network latency

Table 4.1 – Notations used in our analysis of Merkle search trees

Anti-entropy Algorithm	**Dissemination Time**	**Traffic per anti-entropy round**
Scuttlebutt	$2\lambda \log N$	$O(p + d)$
Merkle search trees (ours)	$2\lambda \log N \log_B n$	$O(d \log_B n)$

Table 4.2 – Theoretical comparison of Merkle search trees and Scuttlebutt anti-entropy

4.4.1 Scuttlebutt anti-entropy

Algorithm An event in the event store is identified by pair constituted of a producer node identifier and a sequence number generated locally by the producer. Scuttlebutt uses a reactive pull-push design: new events propagate as nodes exchange vectors of the last sequence number they have for each producer node. This exchange allows nodes to compute an exact set of missing events to send to the other node.

Diffusion time An anti-entropy round is completed in one round-trip-time. The gossip protocol requires on average $\log N$ steps to reach the whole network, where N is the number of nodes currently present in the network, therefore the total diffusion time is $2\lambda \log N$ with 2λ the round-trip-time.

Bandwidth use The bandwidth use for the first step is $O(p)$, where p is the number of producer nodes for which we have to send a sequence number. p is therefore equal to the count of all the nodes that have ever participated in the network, because we do not assume we have a way of knowing which nodes are still present or not. For the second

step, only the d missing events need to be sent so the bandwidth used is $O(d)$. Therefore the total bandwidth use is $O(p + d)$.

4.4.2 Merkle search tree anti-entropy

Algorithm Merkle search trees are used as a simple grow-only set, with $\mathbb{V} = \{\bot, \top\}$ a boolean that is set to $\top$ for present items, and $\mathbb{K}$ is the space of events, ordered by their timestamp of production (either a logical clock or an approximation of real time). As a consequence, the events produced the most recently will be located at one end of the tree, increasing access locality. The gossip algorithm is the one described in Algorithm 4.2.

Diffusion time An anti-entropy round is completed in one round-trip-time if no changes are present. Otherwise, for each new item the algorithm may need to visit a leaf of the tree. The tree is of depth $\log_B n$ (plus a small constant) with high probability, where n is the number of events stored in the tree, and the algorithm requires one round-trip for each tree level, therefore an anti-entropy round is completed in $\log_B n$ round trip time. The gossip protocol requires on average $\log N$ steps to reach the whole network, therefore the total diffusion time is $2\lambda \log N \log_B n$.

Bandwidth use The total number of Merkle tree nodes that are requested by the first node and sent back by the second node is bounded by $d \log_B n$, therefore the total bandwidth use of one anti-entropy round is $O(d \log_B n)$.

4.5 Experimental Evaluation

In this section, we describe our experiment in simulating a large distributed event store using Merkle search trees and compare it to other existing approaches. Our simulator, available online [2], uses an actor-based design and consists in about 2000 lines of Elixir code including all Merkle search tree algorithms.

4.5.1 Methodology

We simulate a network of 1000 nodes with synchronous communication rounds and no crashes. Events are generated with increasing timestamps on nodes chosen at random

2. `https://gitlab.inria.fr/aauvolat/mst_exp/`

in the network. We consider two scenarios: in the first scenario, events are generated with an average rate of 0.1 events per round in the whole network, while they are generated at an average rate of 1 event per round in the second.

We compare our algorithm (Section 4.4.2) with two competitors: the Scuttlebutt reconciliation algorithm described in Section 4.4.1, and a second construction based on Merkle trees, a Merkle-encoded prefix tree on the hashes of the items (called *Merkle prefix tree*, or MPT for short).

The MPT competitor finds missing elements in the same manner as with Merkle search trees, and uses the same gossip-based algorithm. An MPT is built by hashing the key values and then building a prefix tree, thus the main difference with Merkle search trees is that this construction does not preserve key ordering. However this construction does build a tree that is on average well balanced, since the probability of hash prefix collisions decrease exponentially with the prefix length.

The experiments are run several times with different algorithm parameters, as shown in Table 4.3.

Our aim is to show that when the modified values have similar keys, the ordering property of Merkle search trees improves the performance of the anti-entropy algorithm because fewer intermediate tree nodes will need to be transmitted: in the case of the distributed event store, the new events are those with the highest timestamps and thus form a compact subtree at the right of the tree, whereas if a prefix tree on the hashes is used, the events are randomly disseminated and the paths to the newly added leaves share only few intermediate nodes.

4.5.2 A metric for event dissemination

In order to evaluate the inconsistencies in message spreading in the network, we define a metric based on the definition of binary entropy: at a certain simulation time step, the entropy of the network is defined as the sum for each message of the binary entropy of the fraction of nodes that have received the message. More formally, if M is the set of messages and p_m the proportion of nodes in the network to have received a message $m \in M$, the entropy measure we use is defined by:

$$\begin{aligned} H &= \textstyle\sum_{m \in M} H_{\mathrm{b}}(p_m) \\ &= \textstyle\sum_{m \in M} -p_m \log_2 p_m - (1 - p_m) \log_2 (1 - p_m) \end{aligned} \tag{4.1}$$

Common Parameters	
Number of nodes	1000
Events generated per round[a]	0.1 (Table 4.4, Figures 4.2, 4.4)
	1 (Table 4.5, Figure 4.3)
Scuttlebutt Anti-Entropy	
Fanout	$1^{\dagger}$, 2^{*}, 4
Gossip interval	1^{*}, 2, $4^{\dagger}$, 8 rounds
MST and MPT Anti-Entropy	
Fanout	6
Max simultaneous merges	1, 2, 3, $4^{*\dagger}$, 6, 8

[a]in the whole network

Table 4.3 – Simulation configuration

Method	Bandwidth use[a]	Entropy[b]	99% delivery delay
Scuttlebutt	1.3 MB	1.61	64 rounds
MPT	0.51 MB	1.44	56 rounds
MST (ours)	**0.44 MB**	**1.06**	**44 rounds**
Gain vs. SB	-66%	-34%	-31%
Gain vs. MPT	-13%	-26%	-21%

[a]per round, on average
[b]at each round, on average

Table 4.4 – Simulation results on scenario with light load, for representative configurations of the simulated algorithms (indicated by * in Table 4.3). Lower entropy values indicate more uniform diffusion of events and therefore better consistency. Lower bandwidth usage is better. Trade-off for different configurations is presented in Figure 4.2.

With this measure, lower entropy values indicate more uniform diffusion of events and therefore better consistency of the overall network state.

4.5.3 Results

Entropy-bandwidth trade-off The first scenario, with low rate of events, is the most favorable scenario for the Merkle search tree approach. Figure 4.2 shows the curve obtained by plotting the bandwidth usage and entropy for different parameters of the algorithms (see Table 4.3) on this scenario. Numerical results corresponding to the most successful configuration of each method are reported in Table 4.4. The lowest bandwidth usage is achieved by the two Merkle tree reconciliation methods, both of which are much more efficient than Scuttlebutt reconciliation without providing worse entropy measures or sac-

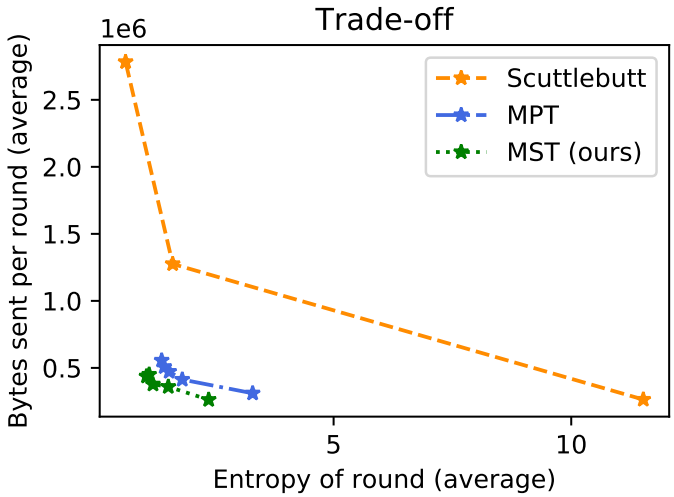


Figure 4.2 – Consistency vs bandwidth usage compromise. x-axis: average entropy measure of the diffusion of events in the network (lower is better). y-axis: bandwidth usage (lower is better). MST: Merkle search tree. MPT: Merkle prefix trees on hashes of items. SB: Scuttlebutt anti-entropy.

rificing message delivery delays. Figure 4.4 plots the bandwidth usage over time of the different methods. Merkle search trees also show a slight advantage compared to Merkle prefix trees on all evaluated metrics.

The second scenario, with ten times higher event rate, is more ambivalent: both approaches perform equally well with 1000 nodes (Figure 4.3a). We increase the number of nodes to 2000 and show that the Merkle search tree approach scales better than the Scuttlebutt approach (Figure 4.3b and Table 4.5). This scenario also clearly demonstrates the advantage of Merkle search trees over the Merkle prefix tree order: the latter show worst result than both methods for 1000 nodes, and the experiment on 2000 nodes did not terminate for Merkle prefix trees due to an explosion of the number of messages in the network, thus we were not able to complete the simulation.

Message delivery delay We measure the delivery delay in rounds of events in the network. We study the 99th percentile worst case scenario. We show that in the light load case, Merkle search trees provide an advantage over Scuttlebutt (Table 4.4), and in the higher load scenario our method provides an acceptable degradation of about 50% (Table 4.5).

1000 nodes

Method	Bandwidth use	Entropy	99% delivery delay
Scuttlebutt	**2.1 MB**	**15.4**	**50 rounds**
MPT	3.9 MB	26.9	100 rounds
MST (ours)	2.2 MB	17.5	74 rounds

2000 nodes

Scuttlebutt	7.6 MB	**14.9**	**54 rounds**
MPT	-[a]	-	-
MST (ours)	**4.2 MB**	21.0	88 rounds

[a]Experiment did not terminate

Table 4.5 – Simulation results on scenario with heavy load with 1000 and 2000 nodes, for representative configurations of the simulated algorithms (indicated by † in Table 4.3). Trade-off for different configurations is presented in Figure 4.3.

When are we better? We plot in Figure 4.5 the theoretical boundary that separates conditions for which Merkle search trees are better suited than the Scuttlebutt method. This boundary was used by writing an equivalence at the boundary based on the theoretical results of Table 4.2 and replacing the number of differences d in an anti-entropy round by the rate of events in the network r:

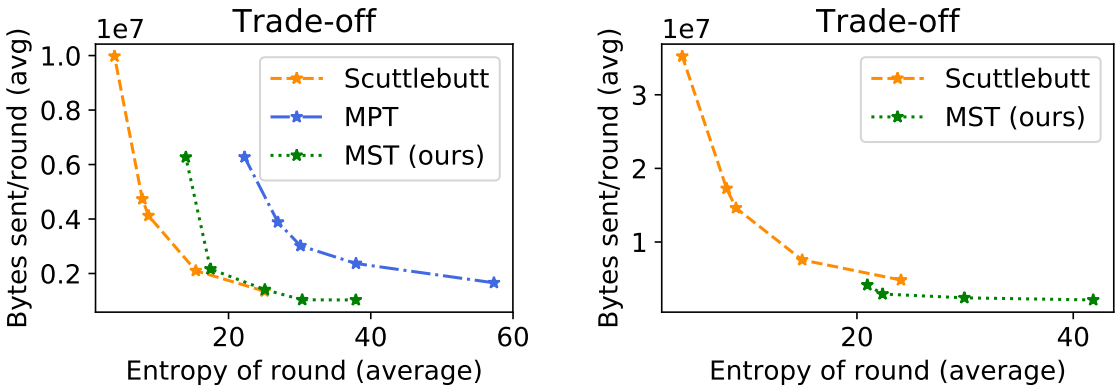
$$\alpha p + \beta r = r \log_B n$$

where p is the number of processes in the simulation. We consider $\log_B n$ to be a constant (equal to $\log_B n_{\max}$ where $n_{\max}$ is the total number of events produced in our simulation). After reordering terms, we obtain an affine boundary:

$$r = ap + b$$

which we calibrate using two simulations where both methods seemed to perform as well, one of which is shown in Figure 4.3(a) with $p = 1000, r = 1$ and the other is not shown and yields $p = 500, r = .33$. The plot of Figure 4.5 also shows the experimental configurations for which we ran a complete set of simulations, and the zones in which each method have an advantage are clearly identified.

Overall evaluation As expected, Merkle search trees are efficient when the rate of events is moderate to low: in this situation, the overhead of the Scuttlebutt method



(a) 1000 nodes

(b) 2000 nodes

Figure 4.3 – Results with a more intense workload. Merkle search trees are still competitive and allow for a high-entropy low-bandwidth configuration. The limits of Merkle prefix trees on hashes of items become apparent in this configuration. By comparing a 1000 node scenario with a 2000 node scenario we also show that Merkle search trees scale better than Scuttlebutt with the number of nodes.

becomes prohibitive.

In the light load scenario, Merkle search trees outperform all the other approaches in the three studied metrics: entropy, bandwidth usage and 99th percentile delivery delay. In the higher load scenario, Merkle search trees are not able to provide as low entropy as the Scuttlebutt protocol but they are able to extend the trade-off started with the different Scuttlebutt configurations in the direction of lower bandwidth usage and higher entropy. Merkle search trees scale better with the number of nodes present in the network, as the communication costs only depends of the rate of messages and not of the number of participating nodes.

We also show that Merkle search trees clearly outperform standard prefix trees in all possible scenarios and on both entropy and bandwidth usage, confirming our prediction that randomized prefix trees are not optimal for this application.

While we have only studied configuration where no nodes join or leave, we conjecture that the overhead of the Scuttlebutt method becomes even worse in open networks as the vector clocks will accumulate values for inactive nodes, leading to useless metadata being exchanged at every anti-entropy round.

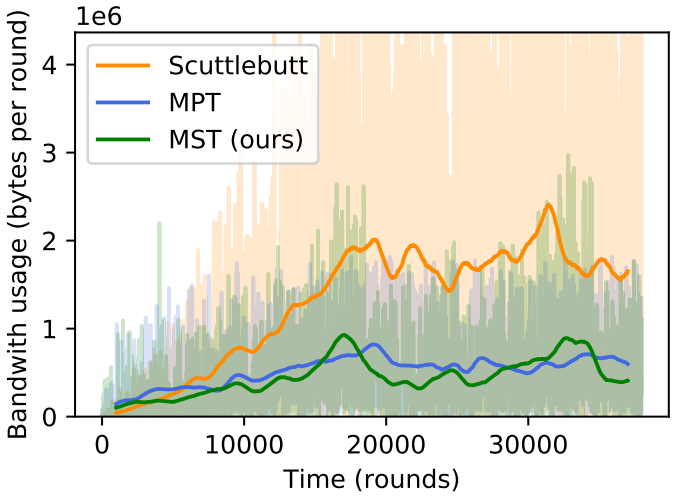


Figure 4.4 – Bandwidth usage over time for experiments shown in Table 4.4.

4.6 Conclusion and Outlook

In this chapter, we have introduced a new data structure based on Merkle trees that allows for efficient remote comparison of sets of values. We have shown that this data structure can be used to implement the CRDT merge operator between two full states in an efficient manner even for very large states. This allows to replicate CRDTs in open networks without requiring primitives such as causal broadcast which are a necessity for common delta-based approaches, giving our approach a unique edge in the world of open networks with churn. We have shown in a simulation that Merkle search trees are more efficient than vector clocks for detecting changes without any a-priori knowledge in particular in situations of low update rates in very large networks.

Our experimental evaluation of Merkle search tree was restricted to a very narrow problem, building an event store, which is equivalent to a grow-only set, one of the simplest known CRDTs. We have also focused on a very simple gossip-based algorithm to drive the distributed reconciliation process. Many other applications and more complex algorithms can be envisioned. This section discusses some of these perspectives.

4.6.1 Adaptive algorithms

In our experimental evaluation, we studied two different methods: Scuttlebutt anti-entropy which is based on vector clocks and Merkle search trees. Our experimental evalua-

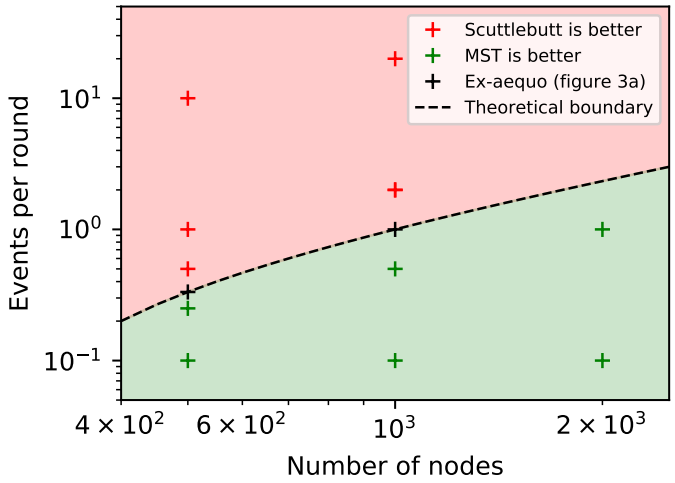


Figure 4.5 – Compromise between number of nodes and event rate in the network. Our method is better in regards to bandwidth consumption in the green area, Scuttlebutt anti-entropy is better in the red area. Crosses correspond to experiments we ran. In red, Scuttlebutt showed a clear advantage. In green, Merkle search trees showed a clear advantage.

tion has shown that Merkle search trees clearly outperform Scuttlebutt when networks are large and events are produced below a threshold frequency (Figure 4.5). One could build an adaptive algorithm that automatically selects the best anti-entropy method depending on observed system metrics. Such an algorithm could dynamically switch algorithms when the workload conditions vary in order to obtain the best performance at a lowest cost in bandwidth usage.

4.6.2 Merkle DAGs as persistent data structures

Many data structure can be encoded in a Merkle tree or Merkle DAG (Benet, 2014), simply by changing standard pointers into the hashes of the blocks they reference. This enables manipulation of the whole data structure by manipulating the root hash in the manner of persistent data structures in functional programming languages (Driscoll et al., 1986). Contrarily to standard pointers that are local to a machine, Merkle hashes allow us to reference data that is present on another node, making it practical to build very large distributed data structures. CRDTs are a specific case where the data structure is deterministic and operation order-agnostic, which allows for efficient implementations of comparison and reconciliation.

Seeing root Merkle hashes as easily swappable pointers to persistent data structures leads the way to a functional programming approach to distributed programming and opens up we believe interesting avenues to reason about and implement system-wide consistency. This implies using the Merkle data structure as the storage itself, and not only as a way to achieve reconciliation or reparation after node failure.

4.6.3 Other applications of Merkle search trees

As a primitive for optimizing other CRDTs Tree-shaped CRDTs have already been proposed (Shapiro, 2011) and are used in sequence CRDTs such as Treedoc (Preguica et al., 2009) and LSEQ (Nédelec et al., 2013). These approach could benefit from a construction similar to the Merkle search tree to avoid balancing issues, which were already identified as an important limitation (Shapiro, 2011). Previous solutions suggested to synchronize the whole network so that the trees are re-balanced simultaneously at all nodes. Using a construction that is always balanced by default removes this requirement, which is impractical in large or open networks.

Composed with other CRDTs By using various CRDTs as the value type $\mathbb{V}$, the Merkle search tree data structure can be used to create many composed CRDT types. An ordered key-value store could be naturally implemented by using last-writer-wins registers or multi-value registers as the value type $\mathbb{V}$. For CRDT types that need to track causality, such as sets that support deletion or multi-value maps, this simple construction would imply storing the information required for causality tracking alongside each item, thus generating potentially large quantities of redundant data. This issue could be addressed by storing the causality metadata separately.

Transactions, in a sense similar but weaker to snapshot isolation, could be implemented easily in such a system, with no need for distributed locking as in other approaches (Schütt et al., 2008): the first guarantee of snapshot isolation, that the reads of a transaction are all done on a consistent snapshot of the database, can be obtained simply by not integrating updates from other peers during the transaction. Once a transaction has completed we propagate the set of updates produced and also integrate updates that have been produced on other nodes, all using the CRDT reconciliation rule to resolve conflicts. The use of a Merkle data structure allows such snapshots to be kept easily even when the data is distributed over many nodes, as all data is content-addressed and consists of immutable blocks identified by their hashes. In this model, write operations consist only of reading

previously existing blocks and creating new blocks that do not impact the presence and usability of other blocks currently referenced by other nodes.

In the context of Blockchains As shown in Figure 1.1, a typical blockchain system will combine two data management tasks in a single system: first, it will collect primary sources in the form of *operations* or *transactions*, which are stored directly in each block. Second, it will compute at each block the state of the *global database*, encoding it in a Merkle tree, whose root hash is embedded in the block. In a cryptocurrency or smart contract platform, the global database is a prefix tree that maps addresses (which are cryptographic public keys) to the state of the associated account or smart contract. In this setting, the space of addresses has no useful structure: addresses appear to be random, and the only query one would realistically need to do is looking up or modifying the state associated with a single particular address. Encoding the dataset in a prefix tree is therefore adapted, and will automatically lead to a balanced data structure as addresses are dispersed randomly.

However in many cases, one would wants to store in the global database a set of items with some kind of structure between them, such as a set of events ordered by their timestamp, so that a consumer can easily find all of the most recent messages or all of the messages in a given time range. Further, one wants to encode these items in a balanced data structure so that access time to all items is logarithmic in the tree size. These properties are easily provided by database indexes in classical SQL databases, however encoding such an index in a Merkle data structure fit to be stored on a blockchain is not a trivial task. One can encode a B-tree in a Merkle data structure in order to do so, but the B-tree construction is not deterministic, meaning that if parallelism is introduced, nodes that compute the tree by taking items in different order will not converge to the same result. In other words, there may be several Merkle trees, represented by different root hashes, that contain exactly the same data items. Merkle search trees are a new data structure that solve this issue, as they are able to store items preserving their order, similarly to a B-tree, while being a deterministic construction such that a certain set of data items has a single unique representation as a Merkle search tree.

Chapter 5

Basalt

As we explained in our introduction, there are some kinds of decentralized applications that cannot be implemented using only weak coordination methods such as causal broadcast. Access control mechanisms for instance are equivalent to consensus in some cases. In the context of cryptocurrencies, smart contracts also require total order to be implemented, as they are designed specifically to allow users to implement arbitrary state machines in a sequentially consistent fashion. Weak broadcast abstractions such as causally ordered broadcast can still be used to build certain kinds of system: in particular in Chapter 3 we saw how it can be employed to build a simple money transfer object, albeit subject to some limitations. We also saw in our state-of-the-art review that systems such as ZeroNet, SSB or Matrix are able to build interesting communication platforms using only weak coordination protocols. However, building fully-fledged applications in a trustless setting using such rudimentary primitives reports the complexity of the problem to the application layer, as illustrated by the complexity of the Matrix protocol and in particular of its state-resolution algorithm (Jacob et al., 2020; Jacob et al., 2021). Instead, a software engineering-oriented perspective indicates that we should rather try to build powerful abstractions as efficiently as possible, to allow for a variety of general applications to be built easily, abstracting away the intrinsic complexity of the trustless open network setting.

In this chapter and the next, we therefore return to the question of how to build totally ordered broadcast, or equivalently BFT consensus, in large-scale trustless open networks and in the presence of churn. We have identified that epidemic algorithms are a particularly promising area of research in this domain, as they have extremely good scalability properties and are intrinsically insensitive to churn.

Epidemic methods however critically depend on a secure random peer sampling service: a service that provides a stream of random network nodes where no attacking entity can become over-represented. Unfortunately, classical RPS algorithms (Jelasity et al., 2007; Nédelec et al., 2018b; Voulgaris et al., 2005) are not resilient to malicious behavior:

Byzantine nodes can easily disrupt their execution by flooding honest nodes with Byzantine identifiers. Left unchecked, this strategy has the potential to isolate honest nodes in a so-called *Eclipse attack* (Heilman et al., 2015; Singh et al., 2006), or to partition the system. Moreover, a scheme where peers are sampled with uniform probability is vulnerable to so-called *Sybil attacks* (Douceur, 2002) where a malicious entity creates arbitrarily many network node identifiers that it controls, thus gaining unlimited influence on the network.

Current deployments of epidemic BFT algorithms, such as the AVA cryptocurrency platform [1], rely on a Proof-of-Stake mechanism (Gilad et al., 2017) to ensure that nodes are sampled in a secure way, i.e. that the cost for an attacker of biasing samples in their favor is very high. However, Proof-of-Stake has several known limitations (F. Zhang et al., 2017). In essence, Proof-of-Stake consists in building an abstraction of a closed (permissioned) system, where system membership can however evolve dynamically according to the various parties' economic investments (in the form of token staking). We argue that such an abstraction is too restrictive and in fact not required. Particularly in the case of epidemic BFT algorithms, we show that the required Byzantine-tolerant random peer sampling service can be implemented directly in a much more open fashion, without resorting to Proof-of-Stake to ensure security.

In this chapter, we revisit the problem of secure peer sampling in large-scale P2P systems, and propose Basalt, a novel Byzantine-tolerant random peer sampling algorithm. Basalt exhibits close to optimal Byzantine fault tolerance, thus significantly improving on the state-of-the-art (Bortnikov et al., 2009; Jesi et al., 2010). Basalt is designed to operate in Internet-scale permissionless systems while resisting to Eclipse and Sybil attacks. At the core of Basalt lies what we have termed a *stubborn chaotic search*, a greedy epidemic procedure (Voulgaris and Steen, 2013) towards random nodes that are implicitly defined in a way that makes it extremely hard for malicious nodes to manipulate the decisions of correct ones. This procedure is parametrized by a target distribution on the IP addresses of nodes, providing defense against Sybil attacks by malicious entities that control nodes in a subset of the IP address space.

We comprehensively analyze Basalt under a theoretical model based on the *power* f of the attack, which captures the (ideal) probability of sampling malicious nodes as defined by the target distribution. We show that Basalt provides samples in which the proportion of malicious nodes is very close to f, its theoretical optimum, and that

1. `https://www.avalabs.org/`

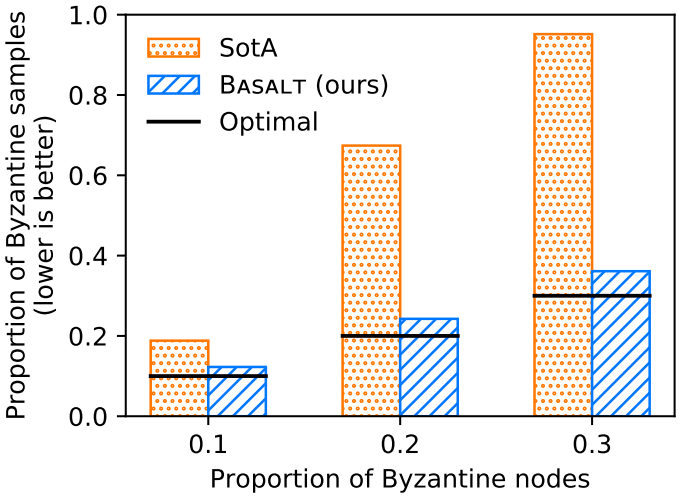


Figure 5.1 – Basalt produces samples with a proportion of malicious nodes that is close to the theoretical optimal (horizontal line) even for large fractions of Byzantine nodes, making it far more resilient that the best approach so far (simulation with $N = 10000$ nodes).

f is acceptably small in several real-world scenarios including institutional attacks and botnet attacks. We complement our theoretical model with Monte Carlo simulations that confirm our analysis. Finally, we demonstrate the feasibility and concrete benefits of our technique by deploying Basalt within a live cryptocurrency network using a prototype implementation of Basalt for AvalancheGo [2], the reference engine powering the AVA cryptocurrency network. Our experiments on the AVA network confirm that the samples obtained using Basalt are equitably distributed amongst nodes, allowing for a system which is both open and such that no single entity can gain excessive power. Our prototype is publicly available, fully functional, and compatible with the existing AVA network without requiring any protocol changes. [3]

2. `https://github.com/ava-labs/avalanchego`

3. The work presented in this chapter is currently under submission, and is already available on arXiv: Alex Auvolat, Yérom-David Bromberg, Davide Frey, and François Taïani (2021a), « Basalt: A Rock-Solid Foundation for Epidemic Consensus Algorithms in Very Large, Very Open Networks », arXiv: `2102.04063`

5.1 Problem Statement

A random peer sampling (RPS) service can be defined as a service that produces a continuous stream $(p_i)_{i\geq 0}$ of random nodes selected in the network. A secure random peer sampling service is faced with the double task of *(i)* ensuring that the network stays well connected by providing a good diversity of peers in the stream $(p_i)_{i\geq 0}$, while *(ii)* limiting as much as possible the appearance of malicious nodes.

5.1.1 System model

We assume a very large system composed of nodes that can either be honest (a.k.a. correct) or malicious (a.k.a. Byzantine). Byzantine nodes may deviate arbitrarily from the prescribed protocol in order to manipulate the decisions taken by correct nodes, for instance to isolate correct nodes or to increase malicious nodes' representation in the peer sampler's output. We write Q the number of correct nodes in the system.

We consider a communication network where any node can send a message to any other node, and assume that more than a fixed fraction of the messages sent to a node by other non-malicious nodes arrive within a certain delay. Byzantine nodes may collude (share information, coordinate their behaviors), and may send arbitrary messages to an arbitrarily large number of correct nodes per time unit. They cannot however block completely the communication between two correct nodes, or read the local memory of correct nodes.

Nodes are granted each a unique identifier, which we assume to be their IP address. We will use the same notation to refer to a node and to its identifier. We assume that Byzantine nodes may not spoof the IP addresses of other nodes, which can be ensured using a handshaking mechanism (Ehrenkranz and Li, 2009). [4]

We do not consider network-level attacks such as BGP hijacks in our attack model, assuming that ISPs have correctly set up their infrastructure in order to avoid propagating

4. Any modern communication protocol that includes a security layer such as TLS or Secure Scuttlebutt's Secret Handshake protocol (`https://secret-handshake.club/`) implements such a handshaking mechanism. Modern protocols routinely integrate such security primitives, and it is natural to assume that Basalt will be implemented over such a protocol. In terms of costs, this handshake protocol only needs to be done when one needs to communicate with the targeted node, i.e. it only adds a marginal cost to an already existing communication cost. Further, we know of no attack that can falsify a communication that is cryptographically secured in this way, other than network attacks through BGP which we discuss in Section 5.7.2. Even assuming a BGP attack, such an attack can only be achieved if the attacker has a certificate with the correct host name (in the case of TLS) or knows the private key of the intended node (in the case of SSB's Secret Handshake).

invalid BGP announces and thus preventing such BGP hijacks. This may not be perfectly the case in practice: we discuss in Section 5.7.2 the impact that such attacks could have on real-world Basalt deployments.

5.1.2 Sybil attacks

Random peer sampling is often considered under the assumption of a *closed*, or *permissioned* system (e.g. Jelasity et al. (2007) and Bortnikov et al. (2009)), where the whole set of nodes is known and the proportion of malicious nodes is equal to (or bounded by) a small fixed fraction f. In such a situation, a perfect random peer sampler could be defined as one that samples all nodes uniformly, thus returning a fraction f of malicious nodes in the samples it produces.

This assumption is however not adapted to an open network such as the public Internet, which is more akin to a *permissionless* (open) systems. In such a setting, an attacker may control nodes with many times more IP addresses than there are correct nodes, which may then be used to perform a Sybil attack, leading to an increased influence of the attacker in the peer sampler's output. In particular, an RPS that samples peers uniformly based on their IP addresses is especially vulnerable to such Sybil attacks.

To counter Sybil attacks on the Internet, we consider properties on the IP address distribution of nodes controlled by the attacker. The attacks that we consider fall on a spectrum whose two extremes are defined by the following:

1. **Perfect attacks**, where the attacker controls nodes whose IP addresses have the same variety as those of correct nodes, in which case no biasing method in the sampling algorithm will be effective to limit the attacker's influence; and
2. **Single prefix attacks**, where the attacker is only able to control nodes whose IP addresses are in a single IP address prefix, in which case biasing for variety of IP prefixes will severely hamper the power of the attacker.

All practical attacks fall somewhere between these two extremes, and we call them **prefix-limited attacks**: they correspond to scenarios where the attacker controls machines whose IP addresses span a variety of IP prefixes, but limited to fewer prefixes than those in which correct nodes exist. This property allows us to implement efficient defenses by biasing our sample selection to limit the influence of any given entity (Section 5.2.3). The definition of perfect attacks is however important as it will allow us to temporarily ignore the features of the IP address space in order to simplify our analysis.

The following two scenarios, taken from the classification of Heilman et al. (2015), correspond to the definition of prefix-limited attacks:

— **Institutional attacks**, launched by an institution or an organization that owns large IP address blocks; and

— **Botnet attacks**, where many infected machines are controlled by an attacker.

These two attacks differ in the fact that in an institutional attack, the attacker may control extremely large numbers of IP addresses located in a limited number of continuous address blocks, whereas in a botnet attack the attacker may control a smaller number of addresses in a larger variety of IP address blocks. Note that in practice, while botnet attacks are closer to perfect attacks, they are in fact prefix-limited. Indeed, the identifiers controlled by a botnet might be biased towards certain blocks (e.g. in the case of botnets built by targeting certain organization, or specific vulnerabilities) that differ from those of honest nodes. Section 5.4 is devoted to quantifying precisely the resilience of Basalt to these two practical attacks.

5.2 The Basalt Algorithm

Basalt leverages three main components. First it employs a novel sampling approach, termed *stubborn chaotic search*, that exploits random ranking functions to define a dynamic target random graph (i.e. a set of v target neighbors for each node) that cannot be controlled by Byzantine nodes. Second, it adopts a *hit-counter* mechanism that favors the exploration of new peers even in the presence of Byzantine nodes that flood the network with their identifiers. Finally, it incorporates hierarchical ranking functions that ensure that nodes sample their peers from a variety of address prefixes. The first two mechanisms ensure that the number of Byzantine nodes in a node's view cannot be increased arbitrarily by attackers. This offers protection from general Byzantine behaviors in all attack scenarios. The third mechanism ensures that nodes sample their peers from a variety of address prefixes, thus improving the resilience of Basalt specifically in the context of prefix-limited Sybil attacks.

Table 5.1 shows an overview of the parameters of our algorithm and of its environment, while Algorithm 5.1 shows its pseudocode. For the sake of clarity, in the following, we use the generic term *node* to refer to protocol participants, but we use the term *peer* to refer to a node's neighbor or potential neighbor.

Environment parameters		
N	Number/equivalent number of nodes	1000, 10 000
f	Fraction/equivalent fraction of malicious nodes	10%, 30%
Q	Number of correct nodes	$= (1-f)N$
F	Attack force (described in Sec. 5.3.2)	≥ 0
I	Bootstrap set size	$\sim v$
f_0	Fraction of malicious nodes in bootstrap set	50%
Algorithm parameters		
v	View size	50 to 200
τ	Exchange interval	1 time unit
ρ	Sampling rate (peers per time unit)	~ 1
k	Replacement count	up to $v/2$
Theoretical model variables		
t	Time	
$c(t)$	Number of correct node identifiers seen	$0 \leq c(t) \leq Q$
$b(t)$	(Equivalent) number of malicious node identifiers seen	$0 \leq b(t) \leq fN$
$B(t)$	Probability of sampling a Byzantine node	$= \frac{b(t)}{b(t)+c(t)}$

Table 5.1 – Parameters of the Basalt algorithm and of its environment

5.2.1 Stubborn chaotic search

Basalt nodes implicitly identify a dynamic target random graph by defining target neighbors using a set of random ranking functions. Then, each node greedily attempts to converge towards this implicit definition by repeatedly exchanging neighbor lists with other peers, discovering at each step peers that better match its ranking functions. In the following, we first detail the use of ranking functions to identify target neighbors. We then discuss how nodes update these ranking functions to make the random graph dynamic.

Identifying neighbors through ranking functions Each node maintains a view, view$[\cdot]$, composed of v *slots*. For each slot, $i \in \{1, \ldots, v\}$, it chooses a random seed, noted seed$[i]$ (line 5 of Algorithm 5.1, and Figure 5.2) that defines a corresponding random *ranking* function, $\text{rank}_{\text{seed}[i]}(\cdot)$. We then define a node's i-th out-neighbor in the target graph as the (correct or malicious) node p that minimizes $\text{rank}_{\text{seed}[i]}(p)$. The function $\text{rank}_{\text{seed}[i]}(\cdot)$ can be selected to implement specific sampling distributions. For instance, using a simple hash function $\text{rank}_{\text{seed}[i]}(p) = h(\langle \text{seed}[i], p \rangle)$ (where angle brackets represent a tuple) leads to a uniform sampling function, since each peer identifier has the same probability of producing the lowest rank. In Section 5.2.3, we present how a hierarchical

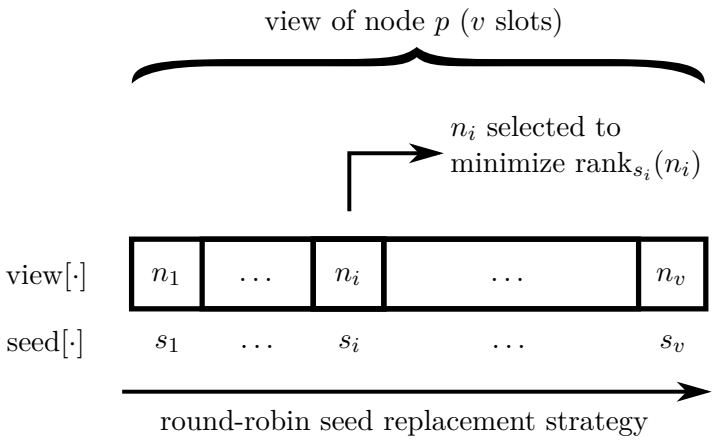


Figure 5.2 – The neighbor selection mechanism of Basalt

ranking function allows Basalt to withstand prefix-limited Sybil attacks. For simplicity, we use the shortcut of saying that a peer p better matches seed[i] than a peer p' if $\text{rank}_{\text{seed}[i]}(p) < \text{rank}_{\text{seed}[i]}(p')$.

When selecting seed[i], a node cannot know the corresponding target identifier. Rather, it stores in view[i] the identifier that has so far produced the smallest value of $\text{rank}_{\text{seed}[i]}(\text{view}[i])$ amongst those seen since selecting seed[i]. At startup, each node selects the best matching peers, view[i], from a set of bootstrap peers (line 6). [5] Nodes then periodically exchange the current contents of their views at lines 7-9 in order to discover new peers that can serve as better matches for the slots in their views. Specifically, every τ time units (exchange interval), each correct node selects a random peer from its view and sends it a *pull* request (line 8) to which the recipient, if correct, replies by sending the contents of its current view (line 11). Then, the node selects another peer from its view and sends it a *push* message containing its current view (line 9). When it receives the reply to the pull request, the node greedily updates any slot view[i] that can be brought closer to its corresponding seed, seed[i], using one of the received identifiers (lines 24-25). The peer to which a push message was sent does the same on its side.

Making the graph dynamic To generate a dynamic random graph and enable nodes to continuously generate fresh samples from the network, nodes regularly reset some of their seeds to new random values. This periodic operation, at lines 14-19, first provides the application with k peer identifiers representing a random sample of the network (line 17), and then resets the k corresponding slots by selecting new seeds seed[r] (line 18). These

5. We discuss the influence of the composition of this bootstrap set in Section 5.3.2.

k slots are selected in a round-robin fashion every k/ρ time units. This yields ρ random samples per time unit on average as indicated in Table 5.1. It then sets the corresponding entries, view[r], to the identifiers from the current view that best match the new seeds (line 19). When the algorithm returns view[r] as a sample to the application, view[r] effectively results from a random selection amongst all the peer identifiers received since the last reset of seed[r]. A node has no way of knowing if it has found the peer p that best matches seed[r] globally (i.e. its target neighbor in the random graph), but selecting which seeds to reset in a round-robin fashion and by sampling view[r] just before resetting seed[r], the algorithm ensures that a maximum number of identifiers have been seen for each seed when returning the corresponding sample. This optimizes the randomness of the sample for a given budget of peer exchanges and view size. [6]

Parameter ρ controls the number of random samples per time unit, and so the number of slots whose seeds are refreshed at each time unit. With a view size of v, this means that each slot is refreshed on average every v/ρ time units. The value of v/ρ must therefore be large enough with respect to the exchange interval, τ, so that the view slots have a chance of converging before being refreshed. Parameter k controls, instead, the number of slots that are reset at the same time. A large value of k causes the algorithm to explore many slots in parallel, thereby obtaining more diverse samples that help the k slots converge faster together. A small value of k (e.g. $k = 1$), instead, causes the exploration to occur with most slots in a quasi-converged state. This increases the probability of contacting peers that have already been contacted recently, thereby leading to slower convergence for the k unconverged slots. Our experiments by simulation confirm that BASALT better resists the presence of Byzantine peers using a batch sampling-and-replacement strategy where k can be as high as $v/2$ (in which case k/ρ, like v/ρ, must also be at least several exchange intervals, τ), rather than replacing seeds one by one (i.e. setting $k = 1$).

6. Other than using more memory, increasing the view size has a non-negligible networking cost as one would typically keep an open TCP connection ready for each peer of the current view. Alternatively, the samples' randomness could be increased by keeping a log of recently closed connections and re-injecting these peer identifiers when selecting new seeds.

Algorithm 5.1: The Basalt algorithm

algorithm parameters
  see Table 5.1

initialization
  for $i \in 1, \ldots, v$ **do**
    seed[i] $\leftarrow$rand_seed(); view[i] $\leftarrow \perp$; hits[i] $\leftarrow 0$
  $r \leftarrow 1$; updateSample(*bootstrap_peers*)

every τ time units
  $p \leftarrow$selectPeer(); **Send** $\langle$Pull$\rangle$ **to** p
  $q \leftarrow$selectPeer(); **Send** $\langle$Push, view[$\cdot$]$\rangle$ **to** q

on receive $\langle$Pull$\rangle$ **from** p
  Send $\langle$Push, view[$\cdot$]$\rangle$ **to** p
on receive $\langle$Push, $[p_1, \ldots, p_v]\rangle$ **from** p
  updateSample($[p_1, \ldots, p_v, p]$)

every k/ρ time units
  for $i = 1, \ldots, k$ **do**
    $r \leftarrow (r \bmod v) + 1$
    Sample view[r]
    seed[r] $\leftarrow$rand_seed()
  updateSample(view[$\cdot$])

function updateSample($[p_1, \ldots, p_m]$)
  for $i \in 1, \ldots, v$, $p \in [p_1, \ldots, p_m]$ **do**
    if $p =$ view[i] **then**
      hits[i] $\leftarrow$ hits[i] $+ 1$
    else if view[i] $= \perp$ **or** $\text{rank}_{\text{seed}[i]}(p) < \text{rank}_{\text{seed}[i]}(\text{view}[i])$ **then**
      view[i] $\leftarrow p$; hits[i] $\leftarrow 1$
function selectPeer()
  $i \in \text{argmin}_{j=1}^{v}(\text{hits}[j])$
  hits[i] $\leftarrow$ hits[i] $+ 1$
  return view[i]

5.2.2 Hit counter hardening mechanism

The graph-generation mechanism described above prevents Byzantine peers from influencing the target graph, and thus the views of correct nodes once the network has converged. However, depending on the speeds at which correct nodes discover other correct or Byzantine nodes, their intermediate views may suffer from a bias in favor of Byzantine nodes. If this happens, the algorithm will tend to select malicious nodes to push to and pull from (lines 8 and 9), further slowing down convergence.

BASALT mitigates this issue by introducing a hit-counter mechanism that effectively makes the protocol harder to attack. Each node maintains a *hit counter* variable, hits[i], for each slot $i \in \{1, \ldots, v\}$ in its view. A node sets hits[i] to 1 when initializing the slot, as well as whenever updating view[i] with a peer that better matches the corresponding seed (line 25). Every time a node receives another peer's view that also contains peer view[i], it increases hits[i] by one (line 23).

When deciding which neighbors to contact, a node always selects one of the peers with the lowest value of hits[i] (line 27). Finally, the node increases the hit counter of the selected peer by 1 to make it less likely to be selected the next time (line 28).

This mechanism has no impact on honest nodes as they should each appear as often in expectation. However, it creates a trade-off for (possibly colluding) malicious nodes that try to be over-represented, as nodes attempting to appear more often will automatically be contacted less. We further discuss this aspect and the possibility of attacks on the hit counter mechanism in Section 5.3.3.

5.2.3 Hierarchical ranking

Central to Algorithm 5.1, the function $\text{rank}_{\text{seed}[i]}(\cdot)$ induces a specific sampling distribution of node identifiers. For instance, using simply a hashing function for $\text{rank}_{\text{seed}[i]}(\cdot)$ yields uniform node sampling. Unfortunately, uniform sampling makes it relatively easy for attackers to gain an overwhelming influence in the system with a Sybil attack, by controlling a large number of IP addresses, such as in the two scenarios of Section 5.1.2 (institutional attacks and botnet attacks). For instance, as we will see in Section 5.4, controlling one of the largest ISPs would grant an attacker about 10^8 IPv4 addresses, a number large enough to thwart most existing decentralized BFT systems. However, as observed in Section 5.1.2, such attacks tend to be prefix-limited, i.e. heavily concentrated in a limited number of address ranges ($\sim$5700 address blocks in the above example). The

key idea for countering them therefore consists in not sampling peers uniformly, but using ranking functions that induce some diversity in the sampled node identifiers, and thus reduce the probability of sampling Sybil peers.

Ranking functions and target distributions Formally, let S be a uniform random variable on 256-bit integers, which corresponds to the sampling of a seed. Let X be the random variable corresponding to the *best matching peer sample* for S, defined as:

$$X = \text{argmin}_{p \in \mathcal{N}} \text{rank}_S(p) \tag{5.1}$$

where $\mathcal{N}$ denotes the set of all network nodes. Depending on the definition of $\text{rank}_S(p)$, X can implement a specific probability distribution on network nodes. This allows us to define the attacker's *power*, f, as the probability of X being a malicious node for a specific definition of the ranking functions $\text{rank}_S(\cdot)$.

If $\text{rank}_S(\cdot)$ is a simple hashing function, $\text{rank}_S(p) = h(\langle S, p \rangle)$, nodes are selected uniformly, and f corresponds to the true fraction of malicious nodes in the system. In the more general case, this probability is no longer equal to this true fraction, however as we will discuss in Section 5.4.1, Basalt behaves similarly as if the actual fraction of malicious nodes was equal to f and the ranking function was uniform: we thus call f the *equivalent fraction* of malicious nodes.

Selecting the ranking function In order to counter prefix-limited Sybil attacks, we need to select a ranking function that minimizes f by giving malicious nodes a low probability of being selected as best-matching peers (i.e. chosen by the distribution X). Basalt adopts a ranking function that spreads sampled peers amongst different subnets by exploiting the structure of IP addresses. IP addresses can indeed usually be decomposed in two parts, a prefix, that designates the *subnet* to which the address belongs (linked to a given Internet service provider), and a local part that identifies a node within that subnet.

A first grouped ranking function To illustrate this intuition, suppose that $G(p)$ corresponds to the prefix of a given length of the IP address, or a country code determined from the address. The following ranking function (based on a lexicographical ordering on values) can be used to sample uniformly amongst the different values of property $G(p)$, and then uniformly amongst all the peers that have the selected value of $G(p)$:

$$\text{rank}_S(p) = \langle\, h(\langle S, G(p) \rangle)\,,\; h(\langle S, p \rangle)\,\rangle$$

Using such a ranking function makes an attack against BASALT harder. In order to gain a power of f in the network, a malicious entity would need to control a large number of nodes at least in a fraction f of all the values of $G(p)$ where network nodes exist. For instance, consider an attacker that owns a full IP address block. Uniform node sampling gives the attacker a power of $f = \frac{q}{N}$, where q is the size of the IP block and N the total number of nodes in the network. In the group-based sampling model, since an address block is usually associated with a single group (a single country, a single IP address prefix), the attacker only has a power of $f = \frac{1}{|G|}\frac{q}{g}$, where $|G|$ is the number of different groups, and g is the number of nodes present in the group that contains attacker's address block. This attacking power is trivially bounded by $\frac{1}{|G|}$, and the only way to increase it consists in taking control of many IP addresses in other groups, making such an attack much more costly.

Basalt's hierarchical ranking function In BASALT, we take the grouping approach described above one step further, taking into account the fact that CIDR blocks on the Internet correspond to IP prefixes of various lengths. We adopt a hierarchical ranking function that descends the address hierarchy by sampling uniformly at levels /8, then /16, then /24, and then finally at the level of individual addresses, defined as follows:

$$\text{rank}_S(p) = \left\langle h(\langle S, p_0^8\rangle), h(\langle S, p_0^{16}\rangle), h(\langle S, p_0^{24}\rangle), h(\langle S, p\rangle)\right\rangle \tag{5.2}$$

where p_0^i corresponds to the prefix constituted of the i most significant bits of p's IP address. The efficiency of this ranking function in countering practical Sybil attacks (which are prefix-limited) is demonstrated numerically in Section 5.4.

5.3 Theoretical Analysis: Perfect Attacks

We now use a theoretical continuous model to estimate the value of $B(t)$, the probability at a time t that a given slot of a correct process contains a Byzantine peer identifier, as a function of f, the attacker's power. In this section, we first focus on perfect attacks in order to study the dynamic properties of BASALT. In the next section, we will use real-world data and an adaptation of our continuous model applied to IP prefixes to demonstrate the resilience of BASALT to practical Sybil attacks.

5.3.1 Parameters, notations and assumptions

Scenario parameters and node distribution

In order to derive a continuous model of Basalt's behavior, we consider a theoretical perfect attack in which Byzantine identifiers follow the same distribution as those of honest nodes. In this case, the attacker's power f that we introduced in Section 5.2.3 is simply equal to the fraction of Byzantine nodes in the network, and is independent of Basalt's hierarchical ranking function. To analyze this attack, we note N the total number of network nodes (i.e. the network size). The product fN denotes the number of Byzantine nodes, and $Q = (1 - f)N$ denotes the number of correct nodes.

Notations

The probability $B(t)$ of selecting a Byzantine node in a given slot of a node p at time t depends on two sets of identifiers: the set of correct identifiers seen at a time t by p on this slot since the last reset, which we note $\mathcal{C}(t)$, and the set of Byzantine identifiers seen by p over the same period, which we note $\mathcal{B}(t)$. Since the distributions of correct and Byzantine peer identifiers are indistinguishable, peers in either set have on average the same influence on the algorithm. We therefore focus only on the sizes $c(t)$ and $b(t)$ of these two sets: $c(t)$ is the size of $\mathcal{C}(t)$, and $b(t)$ is the size of $\mathcal{B}(t)$.

Deriving an expression of $B(t)$

The probability $B(t)$ depends on the distribution of correct and Byzantine identifiers across the three levels of blocks used in Equation 5.2.

We fix one node p selected randomly amongst $\mathcal{C}(t) \cup \mathcal{B}(t)$, and write $\mathsf{selected}(p)$ the event that p is selected by the ranking function $\text{rank}_S(\cdot)$:

$$\mathsf{selected}(p) \equiv \Big(p = \text{argmin}_{q \in \mathcal{C} \cup \mathcal{B}} \text{rank}_S(q)\Big). \tag{5.3}$$

With this notation we have $B(t) = \Pr\left(\, p \in \mathcal{B} \,|\, \mathsf{selected}(p) \,\right)$.

In our model, a perfect attack corresponds to the (ideal) case in which Byzantine and honest nodes follow the same distribution across IP blocks. As a result, they are indistinguishable from the point of view of $\text{rank}_S(\cdot)$, which means here that the events $p \in \mathcal{C}(t)$ and $\mathsf{selected}(p)$ are independent. This independence implies that

$$
\begin{aligned}
B(t) =& \Pr\left(p \in \mathcal{B}(t) | \mathsf{selected}(p)\right) \\
=& \frac{\Pr\left(p \in \mathcal{B}(t) \wedge \mathsf{selected}(p)\right)}{\Pr\left(\mathsf{selected}(p)\right)} \\
=& \frac{\Pr\left(p \in \mathcal{B}(t)\right) \times \Pr\left(\mathsf{selected}(p)\right)}{\Pr\left(\mathsf{selected}(p)\right)} \\
=& \Pr\left(p \in \mathcal{B}(t)\right) \\
=& \frac{|\mathcal{B}(t)|}{|\mathcal{C}(t)|+|\mathcal{B}(t)|} \\
=& \frac{b(t)}{c(t)+b(t)}.
\end{aligned} \tag{5.4}
$$

Assumptions

One key observation is that, for a fixed $c(t)$, $B(t)$ increases as p hears of new Byzantine identifiers and $b(t)$ grows, i.e. $c(t) = c(t') \wedge b(t) \leq b(t') \implies B(t) \leq B(t')$. As a consequence, for a given $c(t)$, $B(t)$ is maximum when the node p has learned all Byzantine identifiers circulating in the system.

In the following analysis, we therefore assume a worst-case scenario in which correct nodes have been flooded with all existing Byzantine identifiers (we discuss the actual implementation of this worst-case scenario in Section 5.3.2). For perfect attacks, we have assumed that correct and Byzantine identifiers follow the same distribution, implying that

$$
B(t) = \frac{b_{\max}}{b_{\max} + c(t)}, \tag{5.5}
$$

where $b_{\max}$ is the total number of Byzantine identifiers, i.e. $b_{\max} = fN$. The probability of selecting a Byzantine node, $B(t)$, becomes therefore driven by $c(t)$.

For simplicity, we study a version of BASALT without the hit counter-based hardening mechanism, and later discuss its impact in Section 5.3.3. Algorithm 5.2 shows the pseudocode corresponding to the hit counter-less version being analyzed. To approximate the system's behavior, we will reason using the mean values of $c(t)$ over all nodes and slots, and assume that the values of individual nodes tend to concentrate around their means in practice with high probability, as is usually the case in such stochastic systems.

Algorithm 5.2: Simplification of Algorithm 5.1 for the theoretical analysis of Section 5.3.2

```
1 function updateSample([p_1, ..., p_v])
2 |   for i ∈ 1, ..., v do
3 |   |   for p ∈ [p_1, ..., p_v] do
4 |   |   |   if view[i] = ⊥ or h(⟨seed[i], p⟩) < h(⟨seed[i], view[i]⟩) then view[i] ← p
5 function selectPeer()
6 |   i ←rand(1, ..., v); return view[i]
```

5.3.2 Analysis of the core mechanism

We first discuss in more detail the worst-case attack on the Basalt algorithm. We then study the risk of a node becoming isolated under this attack model (i.e. of an Eclipse attack succeeding), before moving on to studying the convergence properties of Basalt assuming no node is ever isolated.

Identifying the worst-case attack

To identify the worst-case attack, we observe that attackers cannot influence the choices correct nodes make (at line 6 of Algorithm 5.2, and at lines 16-18 of Algorithm 5.1); thus they can only manipulate the peer-sampling process by increasing their representation in the views of correct nodes, i.e. the value of $B(t)$. The fact that $B(t)$ grows with the number of Byzantine identifiers the node is aware of, $b(t)$, suggests that the worst-case scenario arises when Byzantine nodes flood the network with their identifiers in order to increase $b(t)$ as much as possible. We model this attack scenario as follows:

— A malicious node that receives a pull request returns a view composed of v nodes selected uniformly at random amongst the malicious nodes.

— Regularly, a malicious node sends a push request to randomly selected correct peers, containing similarly a view of v uniformly random malicious peers.

We define the *force* of the attack, F (distinct from the attacker's power, f), as the ratio between the number of push requests sent by a Byzantine node and number of push requests sent by a correct node in a given time interval. For example, if a Byzantine node sends push requests at the same rate as correct nodes, a force of F corresponds to sending push requests to F distinct correct nodes rather than to only one. Alternatively, the force of the attack can also model a situation in which Byzantine nodes send requests more often, or where the network loses more messages from correct nodes than from Byzantine ones.

The worst case corresponds to an arbitrarily large value of F, arising when correct nodes receive all the identifiers of Byzantine peers in any arbitrarily small (but non-empty) time interval. This means that apart from the initial state, $b(t)$ is constant equal to $b_{\max} = fN$, which gives Equation 5.5 for $B(t)$. The analysis that follows shows that even in this case, BASALT causes $B(t)$ to converge to a value that is only slightly larger than the attacker's power, f. The experimental results of Section 5.5 analyze instead the actual performance with finite values of F.

Bounding the probability of isolation

We start by showing that nodes have a low probability of being isolated. Isolation can happen in two ways: either when a node joins the network for the first time, or when it evicts all correct peers from its view and replaces them with Byzantine peers.

Isolated joining node In the first case, the unfortunate joining node receives all of the identifiers of Byzantine nodes as soon as it joins. At time ϵ after joining we have $b(\epsilon)$ becomes maximal equal to $b_{\max}$ and $c(\epsilon) = (1 - f_0)I$, where f_0 is the fraction of Byzantine nodes in the bootstrap sample and I is the size of the bootstrap sample. Since we defined $B(t)$ as the probability of a given slot in the view being occupied a Byzantine peer, we can write the probability that a node has only Byzantine neighbors as $B(t)^v$.

$$B(t)^v = \left(\frac{b_{\max}}{b_{\max} + c}\right)^v = \left(\frac{1}{1 + (1 - f_0)\frac{I}{fN}}\right)^v \tag{5.6}$$

We can reduce this probability exponentially by increasing v, by increasing I or by assuming a lower f_0. For instance, supposing $f_0 = 50\%$ of malicious nodes in our bootstrap peer list, by taking a view size of $v = 200$ and a bootstrap peer list size 25% of the number of malicious nodes in the network ($I = \frac{1}{4}fN$), this probability becomes smaller than 10^{-10}. Supposing for instance a network of size $N = 10000$ with a fraction $f = 0.1$ of Byzantine nodes, this only requires a bootstrap set of size $I = 250$ nodes, of which only 125 are required to be correct.

Convergence to isolated state The second way for a node to become isolated results from resetting the seeds for the slots that still contain correct peers to new seeds that select Byzantine nodes. When such a reset occurs, the probability that all of the non-reset

slots are already owned by Byzantine peers is equal to $B(t)^{v-k} = \left(\frac{b_{\max}}{b_{\max}+c(t)}\right)^{v-k}$. When the number of correct nodes seen locally, $c(t)$, is large enough, this probability is negligible.

Let us now study the value of $c(t)$ at the time of a reset, depending on the value of $c(t)$ at the time of the previous reset. For this analysis, we look at a single node of the network and make the hypothesis that other network nodes are well-converged. As we discuss further in this section and in Section 5.5, this implies that the fraction of Byzantine nodes in their views approaches f with appropriate algorithm parameters. We write c_0 the value of $c(t)$ at the previous reset. The expected number of correct peer identifiers received during the period between the two resets is lower bounded by $\frac{k}{\rho}\frac{v}{\tau}\frac{c_0}{fN+c_0}(1-f)$. We write Δc the corresponding increase in $c(t)$, i.e. the number of *distinct* correct peer identifiers received during this time period.

Based on the result from the coupon collector's problem, the expected number of uniformly distributed (non-distinct) correct peer identifiers that must be received in order to learn Δc new *distinct* correct peer identifiers amongst Q, when c_0 are already known, is:

$$\frac{Q}{Q-c_0} + \frac{Q}{Q-c_0-1} + \cdots + \frac{Q}{Q-c_0-\Delta c+1} \tag{5.7}$$

The number of uniformly distributed peer identifiers received between the two resets is at least the following expression:

$$\frac{k}{\rho}\frac{v}{\tau}\frac{c_0}{fN+c_0}(1-f) \tag{5.8}$$

where $\frac{k}{\rho}$ is the duration of the considered time slice, v is the number of peer identifiers exchanged at each exchange step, τ is the time between two exchange steps, $\frac{c_0}{fN+c_0}$ is the probability that the exchange was conducted with a correct peer, and $(1-f)$ is the probability that each of the peers of the returned view is correct.

We bound the value of (5.7) as follows:

$$(5.7) \leq \Delta c \frac{Q}{Q-c_0-\Delta c} \tag{5.9}$$

Moreover, we have (5.7) $\geq$ (5.8). Thus:

$$\Delta c \frac{Q}{Q-c_0-\Delta c} \geq \frac{k}{\rho}\frac{v}{\tau}\frac{c_0}{fN+c_0}(1-f)$$

thus

$$\Delta c Q \tau \rho (fN+c_0) \geq kvc_0(Q-c_0-\Delta c)(1-f)$$

thus

$$\Delta c \geq \frac{kvc_0(1-f)(Q-c_0)}{Q\tau\rho(fN+c_0)+kvc_0(1-f)}. \tag{5.10}$$

Suppose for instance a network of $N = 10000$ nodes with a proportion $f = 0.1$ of malicious nodes, with algorithm parameters $v = 100$ and $k = 50$. In this system, taking $\tau = 1$ and $\rho = 1$, and supposing that the node we are considering has just joined the network and knows only of $c_0 = f_0\frac{1}{4}fN = 125$ correct node identifiers, we obtain that $\Delta c \geq 467$, i.e. $c(t)$ at the next reset is expected to be at least 592. $B(t)^{v-k}$ is smaller than 10^{-10} as soon as the number $c(t)$ of correct node identifiers seen is more than 585. In other words, the probability that the node becomes isolated at the next reset is negligible. This guarantee can be made even stronger by increasing the view size v. Moreover, if the node is already in a better-converged state with a relatively large c_0, the probability of becoming isolated during a reset becomes even smaller.

Non-isolated execution

Now that we have shown that the probability of a node becoming isolated can be made arbitrarily low, we make the following assumption in the rest of the analysis.

Assumption 1 *No node is isolated, and all nodes have at least some correct neighbors. In particular, $B(t)^v$ is negligible at all times t.*

Deriving a continuous model We introduce the notation $C(t) = 1 - B(t)$ which corresponds to the probability of a given slot of a given node to contain the identifier of a correct peer.

When Assumption 1 holds, and in the worst-case scenario discussed above (Byzantine nodes have propagated all their identities to all correct nodes) $b(t)$ is constant equal to $b_{\max} = fN$, which allows us to write the evolution of $c(t)$ over time as a differential equation, as the sum of contributions resulting from the various parts of the system:

— *Pull exchange*: every τ rounds, a node pulls from one peer in its view, which replies by sending v node identifiers. With probability $B(t)$, the node contacts a Byzantine peer. In this case, it receives only Byzantine peer identifiers that it is already aware of (by the worst-case assumption $b(t) = b_{\max} = fN$). With probability $C(t)$, the node contacts instead a correct peer. In this case, each returned identifier will itself be correct with probability $C(t)$, and if correct, it will have a probability $\frac{c(t)}{(1-f)N}$ of

being already known ($(1-f)N$ being the total number of correct nodes). Thus we can express the variation of $c(t)$ over time as a result of a pull operation as follows:

$$\frac{dc}{dt} = \frac{C(t)^2 v}{\tau}\left(1 - \frac{c(t)}{(1-f)N}\right).$$

— *Push exchange*: every τ rounds, a node pushes to a random node in its view. This push has a $C(t)$ probability of being sent to a correct node. In this case we can apply the same reasoning as above and derive the same contribution to $\frac{dc}{dt}$.

— *Sampling and view renewal*: every ρ rounds, a node resets one of its v slots and forgets the identifiers collected for this slot. Let us write $c(t)$ as $c(t) = \frac{1}{v}\sum_{i=1}^{v} c_i(t)$, where $c_i(t)$ is the number of correct nodes taken into account by slot i. Then a single c_i is set to zero every ρ rounds. On average, this yields the following contribution to $\frac{dc}{dt}$:

$$\frac{dc}{dt} = -\rho\frac{c(t)}{v}.$$

By summing all three above contributions, we obtain our final differential equation:

$$\frac{dc}{dt} = \frac{2C(t)^2 v}{\tau}\left(1 - \frac{c(t)}{(1-f)N}\right) - \rho\frac{c(t)}{v}. \tag{5.11}$$

Solving the continuous model We now solve Equation 5.11 under Assumption 1 and show that the network converges to a state where the proportion B of Byzantine peers in nodes' views is small even for arbitrarily large values of the attack force, F. To this end, we can express $\frac{dB}{dt}$ as $\frac{dB}{dt} = -\frac{b_{\max}}{(b_{\max}+c)^2}\frac{dc}{dt}$ and then substitute $\frac{dc}{dt}$ in Equation 5.11. We start by taking the derivative of (5.5), and using the definition of $b_{\max} = fN$, yielding

$$\frac{dB}{dt} = -\frac{b_{\max}}{(b_{\max}+c)^2}\frac{dc}{dt} \tag{5.12}$$

$$= -\frac{fN}{(fN+c)^2}\frac{dc}{dt}, \tag{5.13}$$

where we use c as short-hand for $c(t)$ for conciseness.

From (5.5) and $b_{\max} = fN$, we also have

$$fN + c = b_{\max} + c = \frac{b_{\max}}{B} = \frac{fN}{B}, \tag{5.14}$$

where B is used as short-hand for $B(t)$. Substituting (5.14) in (5.13) gives

$$\frac{dB}{dt} = -\frac{B^2}{fN}\frac{dc}{dt}. \tag{5.15}$$

We now want to express $\frac{dc}{dt}$ as a function of B, n, f, τ, ρ and v. Using C as short-hand for $C(t)$, we start from Equation 5.11, repeated bellow,

$$\frac{dc}{dt} = \frac{2C^2 v}{\tau}\left(1 - \frac{c}{(1-f)N}\right) - \rho\frac{c}{v}, \tag{5.11}$$

in which we substitute $C = 1 - B$, and

$$c = \frac{fN}{B} - fN = \frac{fN(1-B)}{B}, \tag{5.16}$$

which we obtain from (5.14). These two substitutions yield

$$\frac{dc}{dt} = \frac{2(1-B)^2 v}{\tau}\left(1 - \frac{fN(1-B)}{Bn(1-f)}\right) - \rho\frac{fN(1-B)}{Bv} \tag{5.17}$$

$$= \frac{2(1-B)^2 v}{\tau}\left(\frac{B-f}{B(1-f)}\right) - \rho\frac{fN(1-B)}{Bv} \tag{5.18}$$

$$= \frac{fN(1-B)}{B}\left(\frac{2v(1-B)(B-f)}{\tau f(1-f)N} - \frac{\rho}{v}\right). \tag{5.19}$$

Substituting (5.19) in (5.15) results in the following:

$$\frac{dB}{dt} = B(1-B)\left(\frac{\rho}{v} - \frac{2v(1-B)(B-f)}{\tau f(1-f)N}\right) \tag{5.20}$$

Constant regime of Equation 5.20 To study the constant regime of this system, we write $\frac{dB}{dt} = 0$ and exclude the solutions $B = 0$, which is not compatible with $b_{\max} = fN$, and $B = 1$, which corresponds to the case where Byzantine nodes take over the whole network. We also simplify by setting $\tau = 1$ as its role is symmetrical with that of ρ. We obtain after a few steps:

$$(1-B)(B-f) = \frac{\rho f(1-f)N}{2v^2}. \tag{5.21}$$

The equation exhibits two roots $B_1 < B_2$.

$$B_{1,2} = \frac{1}{2}\left(1 + f \mp \sqrt{(1-f)^2 - 2\frac{\rho f(1-f)N}{v^2}}\right) \tag{5.22}$$

When the quantity on the right-hand side of Equation 5.21 approaches zero, B_1 approaches f from above, while B_2 approaches 1 from below. Since $\frac{dB}{dt} > 0$ for $B < B_1$ and $B > B_2$, while $\frac{dB}{dt} < 0$ for $B_1 < B < B_2$, B_1 corresponds to a stable equilibrium, while B_2 corresponds to an unstable one. We therefore focus our analysis on B_1.

With respect to B_1, the right-hand side of Equation 5.21 represents the difference between the proportion of malicious peers in nodes' views, $B(t)$, and their overall proportion in the network, f. Ideally, we want to keep this quantity as small as possible, making B only slightly larger than f.

To this end, we observe that the term $\frac{\rho f(1-f)N}{2v^2}$ shrinks proportionally to the square of the view size, v^2. Thus, choosing a large enough view size allows the network to converge to a globally well mixed state where Byzantine nodes control only slightly more peers in the view than their overall proportion in the network. Moreover, in order to obtain the same stable state value of B, for fixed values of f and N, the view size v should grow proportionally to the square root of the sampling rate $\sqrt{\rho}$, while, for fixed values of f and ρ, it needs to increase proportionally to $\sqrt{N}$.

5.3.3 Analysis of the hardening mechanism

BASALT's hit counter-based hardening mechanism allows nodes to detect which peers have appeared more often in incoming messages, and prioritize other peers for network exploration. In the case of a simple attack where malicious peers flood their own identifiers, the hit counter favors the choice of correct peers over malicious ones.

However, the fact that we analyzed a simplified version of BASALT without the hardening mechanism raises the legitimate question of whether the hit counter may degrade the security of the approach by enabling some other attack. To answer this question, let us consider a malicious node or a coalition of malicious nodes that want to influence the sampling operations performed by a correct node.

We start by observing that malicious nodes can neither write nor read the local memories of correct nodes. So they cannot influence sampling operations directly: their only strategy consists in increasing the hit counters of correct peers in order to make them less likely to be contacted for peer exchanges, thus reducing the growth rate of $c(t)$. To this end, malicious nodes can repeatedly advertise the identifiers of correct peers. But again, they cannot guess which correct peers are in the target correct node's view slots. So their only option consists in advertising a possibly large random set of correct peers in the hope that some of them will be in the correct node's view.

But even this turns out to be counterproductive. If malicious nodes advertise a large number of correct identifiers, it is indeed possible that they may increase the hit counter of some entries in the target's view. But they do so at the cost of increasing their target's value of $c(t)$, i.e. the number of correct peers known to it. Since we already assumed the worst-case scenario of $b_{\max} = fN$, the increase in $c(t)$ can only decrease $B(t)$ thereby making the attack counterproductive.

5.4 Real-world Scenarios: Numerical Analysis of the Hierarchical Ranking Function

Until now we focused on analyzing BASALT under the assumption of a perfect attack, i.e. an attack where the distribution of Byzantine nodes is the same as the distribution of correct nodes. This means that BASALT would sample all nodes uniformly and thus be vulnerable to Sybil attacks. In practice this will not be the case, as an attacker will likely have access to IP addresses in a limited subset of all available internet prefixes, which we have called a prefix-limited Sybil attack. The hierarchical ranking function we have designed (Section 5.2.3) specifically helps BASALT resist such attacks, including in cases where the number of malicious nodes is order of magnitudes larger than that of correct nodes, as long as these addresses are in a small subset of the IP prefixes in which correct nodes exist.

5.4.1 Linking f to the equilibrium value of $B(t)$

With the hierarchical ranking function, the power f of the attacker corresponds to the probability of a given slot of a correct node to contain a malicious peer identifier supposing that it knows of all of the peer identifiers (correct and Byzantine) circulating in the system. f thus gives a lower bound on the expected number of Byzantine samples returned by BASALT in this scenario, and estimating its practical value is thus crucial to determining BASALT's usefulness. Note that in practice, this perfect knowledge state is never reached as nodes reset their seeds regularly in order to provide fresh samples to the application. The dynamics of BASALT in this situation can be extrapolated from the analysis above, by applying it not to individual IP addresses but to IP prefixes: thanks to these dynamic properties on IP prefixes, BASALT converges rapidly to a proportion of Byzantine peers in correct nodes' views close to f.

Indeed, the continuous analysis of Section 5.3.2 assumes a perfect attack, in which the attacker controls nodes whose IP addresses have the same variety as those of correct nodes. It no longer holds when this is not the case, but its general principle can nevertheless be recycled to analyze prefix-limited scenarios. For illustration, consider a prefix-limited attack in which an attacker controls all the addresses of $m_B^{\max}$ top-level /8 blocks, and no address outside of these $m_B^{\max}$ blocks (so a total of $m_B^{\max} \times 2^{24}$ addresses). Further assume, as in Section 5.3.2, a worst-case scenario, in which correct nodes have been flooded with all existing Byzantine identifiers, and in particular have observed Byzantine nodes of all $m_B^{\max}$ /8 blocks.

By assumption, an /8 block can only fall in three categories: either *(i)* it does not contain any identifiers of nodes involved in the system (it is empty), or *(ii)* it only contains Byzantine identifiers (is fully compromised), or *(iii)* it only contains correct identifiers (it is "correct"). Because of this unambiguous partition, we can introduce $m_C(t)$, the number of "correct" /8 blocks identifiers seen at time t by a node p, which is the /8 counterpart of the quantity $c(t)$ used in Section 5.3.2.

By construction, the behavior of Basalt's hierarchical ranking function, $\text{rank}_S(p')$ (Eq. 5.2), will in this case be fully determined by its first component, $h(\langle S, p'^8_0 \rangle)$. If among the $m_B^{\max} + m_C(t)$ blocks known by p, $h(\langle S, p'^8_0 \rangle)$ is minimal for a Byzantine block, then a Byzantine node will be selected, otherwise a correct node will. More formally, the probability $B(t)$ of selecting a Byzantine node in a given slot of p is

$$B(t) = \frac{m_B^{\max}}{m_B^{\max} + m_C(t)}, \tag{5.23}$$

where $m_B^{\max}$ plays the role of $b_{\max}$ in Eq. 5.5, and $m_C(t)$ that of $c(t)$.

When Algorithm 5.1 executes, we can focus on the /8 prefixes of the identifiers exchanged and manipulated by the participating nodes, and thus perform for $m_C(t)$ an analysis similar to that performed for $c(t)$ in Section 5.3.2, yielding the following differential equation

$$\frac{dm_C}{dt} = \frac{2C(t)^2 v}{\tau}\left(1 - \frac{m_C(t)}{(1-f)N_0^8}\right) - \rho\frac{m_C(t)}{v}, \tag{5.24}$$

where $C(t) = 1 - B(t)$ is the probability of selecting a correct node in a view slot, N_0^8 is the total number of /8 blocks in which system identifiers appear (rather than the total number of identifiers N used in Eq. 5.5), and $f = \frac{m_B^{\max}}{n_0^8}$ is the power of the attack (the equivalent fraction of malicious nodes), as defined in Section 5.2.3.

Performing similar substitutions to those of Section 5.2.3, leads to a stable equilibrium for this prefix-limited attack that has a similar form to that of Equation 5.22, except that it is defined in terms of the total number of /8 blocks N_0^8 in which system identifiers are present, and the equivalent fraction of malicious nodes f:

$$B_1 = \frac{1}{2}\left(1 + f - \sqrt{(1-f)^2 - 2\frac{\rho f(1-f)N_0^8}{v^2}}\right). \tag{5.25}$$

5.4.2 Estimating f in practical scenarios

To illustrate the robustness of Basalt's hierarchical ranking function against a prefix-limited Sybil attack, we calculate the power f (here an equivalent fraction of malicious nodes) of a real-world attacker using data corresponding to two such possible attackers.

Our first dataset is the GeoLite2 Block/ASN dataset (MaxMind, 2020): we assume that the attacker is an internet autonomous system (AS), that exploits all the IP addresses it owns to attack Basalt, and that a certain number of honest nodes (100, 1000, 10 000) are uniformly spread amongst the remaining currently active IP addresses. Table 5.2a shows the power f of such an attacker, supposing that the attacker is the Internet AS with the largest number of currently active addresses (106 million in the dataset we used, spread over 5739 blocks). This calculation shows that the hierarchical sampling method reduces the power of the attacker down to 21% when only 1000 honest nodes run Basalt, where it would have been above 99.99% (i.e. full control on the network) using uniform sampling. Figure 5.3a similarly shows the equivalent fraction f of malicious nodes using different sampling methods for the 100 biggest Internet ASes, assuming 1000 uniformly spread honest nodes: hierarchical sampling limits the power of the adversary to around 20% even for the biggest possible attackers.

Our second dataset, provided by an industrial partner, is a record of 833 554 IP addresses which have been observed to have malicious behaviour such as trying to break into SSH servers due to being infected with malicious software. We assume a hypothetical attacker that would consist in a botnet that controls the entirety of these 833 554 addresses, which is extremely unlikely. Analysis of the data shows that even such a large-scale adversary is prefix-limited, and Basalt's hierarchical sampling method can be effectively employed to reduce the attacker's power to about 42% with 10 000 honest nodes, and 10% with 100 000 honest nodes (see Table 5.2b).

One could argue that it will be hard to bootstrap a Basalt network containing enough nodes to effectively counter such large-scale botnet attacks. We note that this problem is

exactly the same as in PoW-based cryptocurrencies, as an attacker that gains $> 50\%$ of the network's hashing power can overturn the network in their favor (which is easy to do for smaller cryptocurrencies that don't have a lot of hashing power allocated to them). A PoW-based cryptocurrency network is secured by members investing in providing lots of hashing power, as is the case e.g. for Bitcoin, in order to make a $> 50\%$ attack so costly that it is impossible in practice (or simply not worth it compared to the value of the cryptocurrency that could be stolen). A Basalt-based cryptocurrency is similarly secured by participants investing in running as many nodes as possible from many different IP prefixes, which they have an incentive to do in order to keep the system safe. Moreover, Basalt has the advantage that this investment does not require the waste of tremendous quantities of energy.

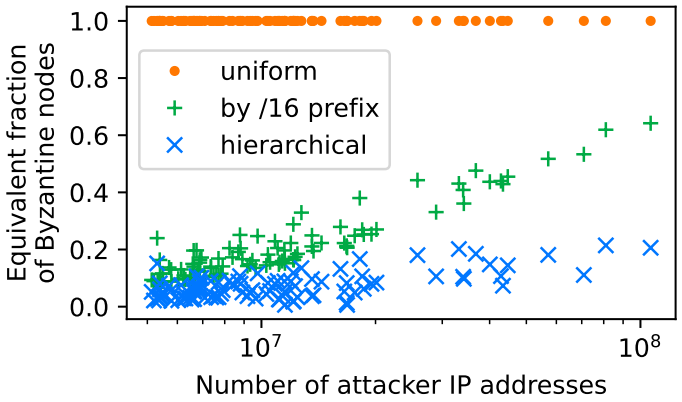
# of honest IPs (Q)	**100**	**1000**	**10 000**	**100 000**
Uniform	99.9999%	99.999%	99.99%	99.9%
By /8 prefix	49%	28%	27%	25%
By /16 prefix	95%	64%	17%	4.4%
By /24 prefix	99.98%	99.8%	98%	81%
Hierarchical	**47%**	**21%**	**10%**	**3.9%**

(a) Institutional attack: we suppose our adversary is the biggest Internet service provider ($\sim 10^8$ IP addresses, distributed over 5739 blocks), on the IPv4 network. Data sourced from the GeoLite2 Block/ASN dataset

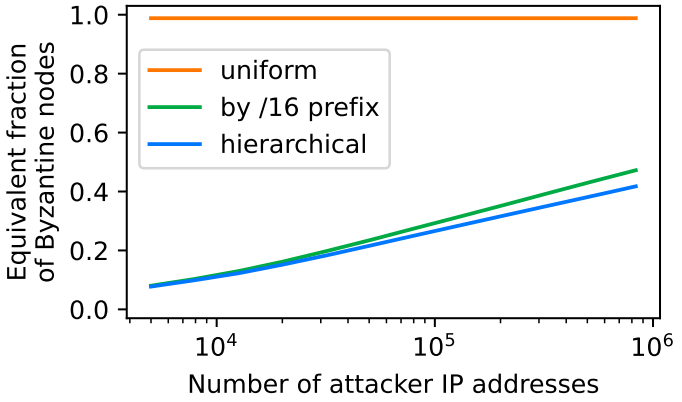
# of honest IPs (Q)	**100**	**1000**	**10 000**	**100 000**
Uniform	99.99%	99.9%	98.8%	89%
By /8 prefix	97.6%	89%	60%	19%
By /16 prefix	98.9%	90%	47%	**9.97%**
By /24 prefix	99.7%	96.6%	74%	22%
Hierarchical	**96.7%**	**82%**	**42%**	10.03%

(b) Botnet attack: we suppose our adversary is a botnet controlling the entirety of the 833 554 IP addresses in our dataset (see Figure 5.3b)

Table 5.2 – Power f of the adversary, defined as the equivalent fraction of malicious nodes, for different sampling methods and two possible large-scale adversaries



(a) Institutional attack: equivalent fraction f calculated for the 100 biggest Internet ASes supposing that each of them is the attacker, and that 1000 honest nodes are uniformly spread amongst remaining IP address space. Data sourced from the GeoLite2 Block/ASN dataset. Horizontal axis: number of IP addresses controlled by the AS.



(b) Botnet attack: equivalent fraction f calculated assuming a certain number of machines have been compromised by a botnet, and that 10 000 honest nodes are uniformly spread in IP address space. IPs of botnet nodes are randomly sampled from a dataset of 833 554 observations of malicious behavior on the Internet (such as trying to break into SSH servers) provided by an industrial partner. Horizontal axis: number of IP addresses controlled by the botnet.

Figure 5.3 – Equivalent fraction f of Byzantine nodes with different ranking functions, for two different attack scenarios

5.4.3 Remarks on the hierarchical ranking function

Attacks by focusing on less-populated prefixes A possible topic for concern when analyzing Basalt's hierarchical ranking function is the possibility that an attacker could gain important power in the network by focusing on under-represented IP address prefixes. This situation can only occur if the addresses of honest nodes are limited to a few prefixes For instance, if the honest nodes running Basalt happen to be all in the datacenters of a few large cloud providers, their IP addresses will necessarily concentrate in a few prefixes. As an other example, suppose the case from Table 5.2a where the actors are swapped: the honest nodes are now the 10^8 IP addresses in 5739 blocks owned by a single organization, and the attacker controls 1000 well-distributed IPs in the Internet. The attacker would then be able to control almost 80% of the samples.

Having more dispersed honest IPs is a strong defense against these attacks. In practice, a concentration of honest IPs in a small subset of IP space does not correspond to the distribution we can expect in a true deployment. A variety of individuals from different horizons (different countries, organizations, etc. – individuals of which a majority are assumed to be honest) will necessarily have access to more IP networks at a lower cost (thus more prefixes more easily obtained) than a single attacker, meaning that a secure Basalt network can be built. The simple necessity of securing the network could be a sufficient incentive to do so. In the context of cryptocurrencies, a crypto-economic incentive that encourages users to place nodes in varied prefixes could be built, by finding means to link the probability of obtaining a reward to the probability of being sampled by Basalt (we discuss this possibility in our conclusion in Section 7.1). Note however that, while we expect that the obtained distribution of IP addresses will be dispersed enough to secure the network, we have no way of knowing what the distribution of honest IPs would be in such a deployment before seeing Basalt widely deployed in practice, as having an incentive for the variety of IP prefixes means that the distribution will be very different from the distribution of IPs in e.g. Bitcoin, AVA or other existing open networks.

Consequences of a non-uniform sampling distribution Nodes in less populated prefixes being selected more often than others is not a bug but a feature of our method. In particular, it does not impact the properties of epidemic algorithms built upon Basalt. In our real-world evaluation (Section 5.6), it does not lead to strong imbalance, in the sense that no single node is selected a majority of the time, or even more than 2% of the time (reading Figure 5.8). Similarly, the fact that nodes in more densely populated

prefixes are selected less often (as in a Sybil attack, even if it is not one) is also not an issue: these nodes simply have less value in securing the network, which can be ratified through crypto-economic incentives (see above). The only property which is necessary to guarantee is that the random graph provided by BASALT has good connectivity properties, such as a low mean path length (see Table 5.3).

Sampling method	**MPL**[1]	**CC**[2]	**In-degree d9 - d1**[3]
Uniform sampling	2.18	0.0328	26
/16 prefix sampling	2.34	0.0409	181
BASALT hierarchical sampling	2.46	0.0968	203

[1]: mean path length, [2]: clustering coefficient, [3]: in-degree difference between 9th and 1st decile

Table 5.3 – Graph quality metrics according to the sampling method, using a list of 6150 IPv4 Bitcoin nodes as a dataset. Calculated on average of 10 random graphs, nodes have a view size of $v = 100$.

5.5 Experimental Evaluation in Simulation

We complement our theoretical analysis with Monte Carlo simulations that illustrate BASALT's dynamic behaviour. In this section, we focus on simulating a closed (permissioned) system with a known fraction of malicious nodes. We do not simulate the IP address distribution and use the uniform ranking function. As explained above, our observations can be transposed to a permissionless setting, where the attacker's power defined by the hierarchical ranking function plays the role of an equivalent fraction of malicious nodes. We show that BASALT consistently produces samples with fewer malicious peers than the state-of-the-art algorithms Brahms (Bortnikov et al., 2009) and SPS (Jesi et al., 2010) over a wide range of scenarios. We also show that BASALT converges faster on metrics quantifying the random connectivity of the graph generated by the algorithm, such as the clustering coefficient and mean path length. These metrics are relevant for information dissemination and may thus have an influence on the convergence time of epidemic agreement algorithms.

5.5.1 Experimental setting

We evaluate the tested algorithms by simulating a system with N nodes, of which a fraction f implement the malicious behaviour described in Section 5.3.2. We do not

simulate message loss or variable link latencies, as our model parameter F (the attack force) already integrates the possibility of message loss (see Section 5.3.2), and variable link latencies can also be modeled as losing messages that arrive after a certain delay. We do not simulate node churn, but consider instead an extreme scenario in which all nodes have just joined the system—this can be seen as an ultimate churn event, in which all nodes are replaced. We vary the two parameters v, the view size, and ρ, the sampling rate, of the algorithm, as well as the force of the attack, F. We fix the exchange interval to $\tau = 1$ (1 simulation time step). Unless stated otherwise, we use $F = 10$ and $\rho = 1$. All algorithms were implemented in a same simulation framework written in Rust, totaling about 2500 lines of code [7].

We compare Basalt (Algorithm 5.1) and its variant without the hardening mechanism (Algorithm 5.2, Basalt-simple) to two state-of-the-art competitors: Brahms (Bortnikov et al., 2009) and SPS (Jesi et al., 2010).

SPS was unable to function at all in the tested scenarios: for instance for $N = 1000$, $f = 30\%$, and even with an attack force F of 0, 90% of correct nodes become isolated in the network rapidly using SPS and remain so during the whole simulation. In contrast, both Basalt and Brahms were able to prevent all correct nodes from becoming isolated. We have thus decided to exclude SPS from our comparison charts, and concentrate on the comparison of Basalt against Brahms.

To compare Brahms and Basalt on similar grounds, we add to the Brahms algorithm a mechanism that resets some of the hash functions regularly, using the same round-robin strategy as Basalt. Without such a mechanism Brahms would always return the same fixed set of samples, limiting its usability as a random peer sampling algorithm. As we will show just below, adding a reset rate to Brahms makes it less resilient to malicious nodes.

In terms of communication overhead, Brahms and Basalt have the same cost. Indeed, both algorithms send a set of peer identifiers of size v when replying to a pull request. For push requests, Basalt uses larger messages since Brahms does not send the view with a push message, only the sending node's identifier, whereas we send the whole view of size v. However, supposing $v = 200$ (the maximum in our experiment) and node identifiers of size 4 bytes (such as IPv4 addresses), the size of the communicated information is smaller than one MTU (maximum transmission unit, i.e. maximum size of a single packet, which is about 1500 bytes on the Internet), thus the same number of Internet packets need to

7. Available publicly at `https://github.com/basalt-rps/basalt-sim`.

be sent by both algorithms.

5.5.2 Proportion of Byzantine samples

In our first experiment, we measure the proportion of Byzantine nodes present in correct nodes' samples on average after 200 simulation time steps. For this experiment, we simulate a network of $N = 10000$ nodes. We fix base parameter values of $f = 10\%$ of malicious nodes, a sampling rate of $\rho = 1$, a view size of $v = 160$ and an attack force of $F = 10$. We then vary the parameters f, ρ, v and F individually. Figure 5.4 shows how this proportion evolves for the three algorithms evaluated, as one of the parameters f, ρ, v and F varies.

Figure 5.4a shows how the algorithms behave when the proportion of Byzantine nodes in the system varies. BASALT provides close to optimal proportions of Byzantine samples even with many Byzantine nodes, whereas Brahms fails to contain the attack in this domain.

Figure 5.4b shows the sensitivity of the algorithms to the force of the attack F. These plots show that BASALT is almost insensitive to F, whereas Brahms shows an increasing proportion of Byzantine samples when F increases.

Figure 5.4c shows how the algorithms behave for various values of the sampling rate ρ. For low values of ρ, both Brahms and BASALT are able to converge to high quality samples, however such a setting does not provide much utility as the algorithm is unable to frequently return new samples to the application. Increasing the sampling rate ρ results, however, in more disruption of the views, where view slots have a higher risk of being reset before they converge to their target peer. This disruption causes Brahms to collapse for higher values of ρ: the network becomes fully disconnected, and the views of correct nodes end up completely polluted by malicious peers. This plot also shows how the hit-counter variant helps BASALT attain better states when ρ is high.

Figure 5.4d shows how the algorithms behave for various view sizes. For small view sizes, all algorithms are unable to keep the network in a connected state and correct nodes all end up isolated. The plots show that BASALT can keep the network connected using smaller views than Brahms.

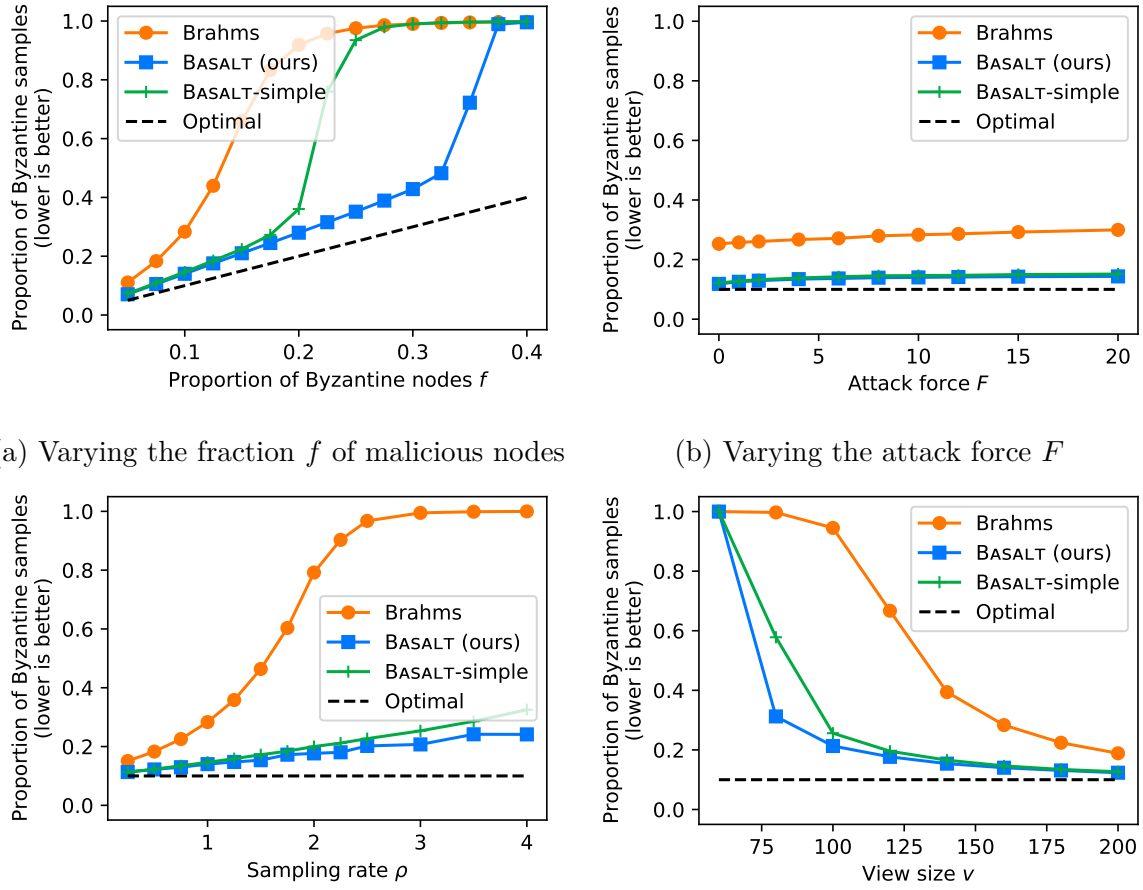
The results of Figure 5.1 were obtained using the same methodology for $N = 10000$, f varying from 10% to 30%, $\rho = 1$, $v = 200$ and $F = 10$.

5.5.3 Evaluating convergence speed

In this second experiment, we study the speed at which the algorithms converge to good network states, where they provide samples with low proportions of malicious nodes. Figure 5.5 shows the time that Brahms and Basalt take to converge to proportions of Byzantine samples that are within 25% of the optimal proportion, for $N = 1000, v = 100, F = 10$ and $\rho = 1$ and for varying proportions of Byzantine nodes in the network. We show that the convergence time of Basalt remains low for up to 30% of Byzantine nodes, whereas Brahms takes much longer to converge (starting at 20% of Byzantine nodes, it did not converge within the experiment's time).

Figure 5.6 shows the evolution of several metrics through time, starting with the number of Byzantine nodes in the view, in our experiment for $N = 10000$, in a favorable situation with $f = 10\%$, $\rho = 0.5$ and $F = 1$. These plots show that Basalt converges much faster than Brahms to a good network state (Brahms does not converge according to the previous criterion within the time of the experiment). The other plots show metrics for graph quality, where the algorithms exhibit a similar convergence behaviour: clustering coefficient, mean path length and the concentration of in-degrees measured by the difference between the last and the first decile. The clustering coefficient is computed by averaging the local clustering coefficient of correct nodes in a graph where malicious nodes are assumed to be all connected to one another.

The mean path length is measured in a graph where malicious nodes are assumed to have no connection in either direction, which models the situation where they do not cooperate in transmitting information between correct nodes.



(a) Varying the fraction f of malicious nodes

(b) Varying the attack force F

(c) Varying the sampling rate ρ

(d) Varying the view size v

Figure 5.4 – Our algorithm (boxes, blue) consistently provides samples that contain fewer Byzantine nodes than our competitor, Brahms, in a variety of situations. A proportion of 1 of Byzantine samples, as exhibited in Fig 5.4a for the highest values of f, corresponds to a situation where malicious nodes are able to cause a network partition. Results shown for a network size of 10 000 nodes, with a base proportion $f = 0.1$ of malicious nodes. Base values for other parameters are $v = 160$, $\rho = 1$, $F = 10$. BASALT corresponds to the complete version of our algorithm, whereas BASALT-simple corresponds to Algorithm 5.1 without the hardening mechanism (modifications of Algorithm 5.2).

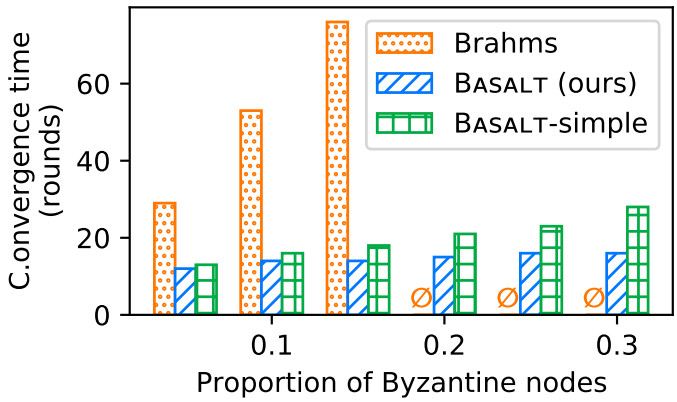


Figure 5.5 – Time to convergence within 25% of optimal proportion of Byzantine samples, for $N = 1000$, $v = 100$ (on the right part, Brahms does not converge within experiment time)

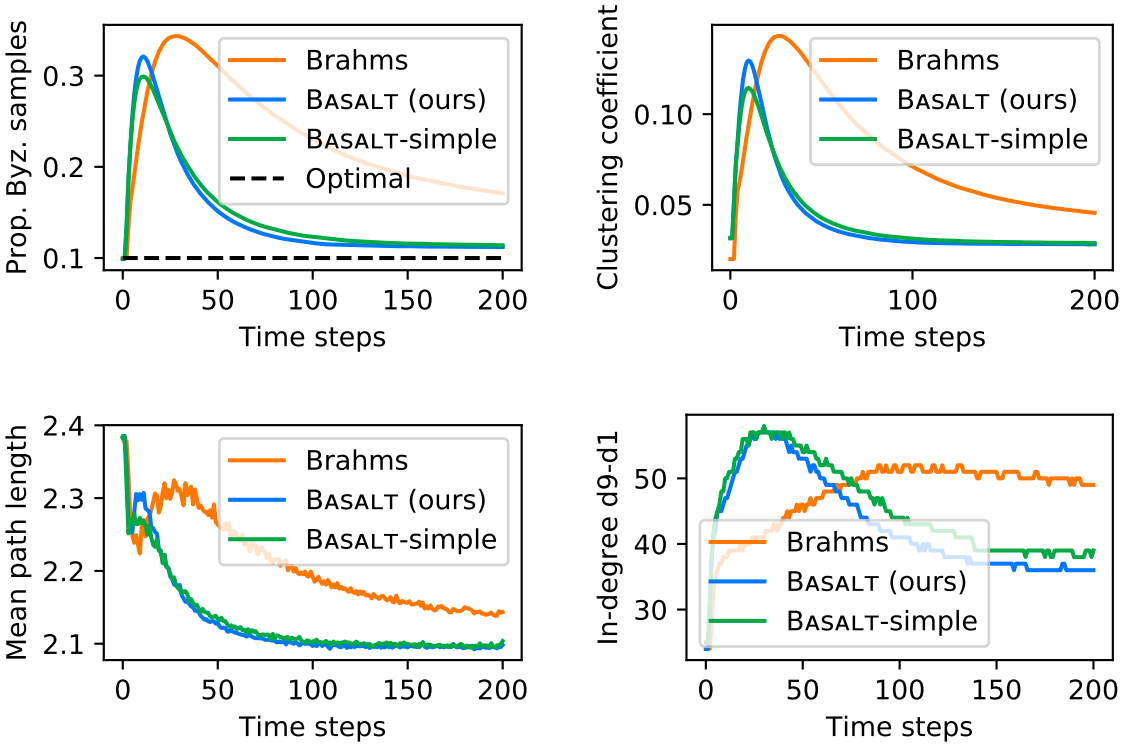


Figure 5.6 – Algorithm convergence on several graph quality metrics, for $N = 10000$, $f = 10\%$, $F = 1$, $\rho = 0.5$, $v = 160$. On all metrics, lower is better: we see that Basalt converges much more rapidly than Brahms.

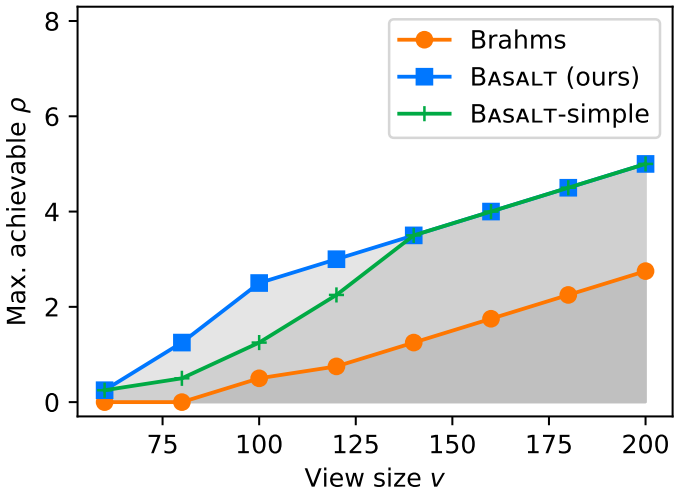


Figure 5.7 – Maximum achievable sampling rate ρ (see Section 5.5.4) for 10000 nodes, $f = 10\%$

5.5.4 Node isolation vs. sampling rate

We have seen earlier (Figure 5.4c) that both Brahms and BASALT are sensitive to increased sampling rates, and return more malicious samples when the sampling rate, ρ, is high, with Brahms failing completely for too large values of ρ.

To investigate this effect further, we run both algorithms for various values of v and ρ, and plot the maximum value of ρ that can be used for a given v without causing a network partition. More precisely, a run for a given set of parameters v, ρ is successful if starting from half of the allocated simulation time, no correct node is ever isolated by the malicious peers. Otherwise it is failed. We plot the successful runs with highest values of ρ for a given v. The results of this experiment are shown in Figure 5.7 for $N = 10000, f = 10\%$ and $F = 10$. The areas delineated in Figure 5.7 correspond to the parameter sets that give successful runs. Our results show that for similar view sizes, BASALT achieves much higher sampling rate than Brahms, thus providing more utility to the application.

5.6 Live Deployment

We implemented Basalt in the AvalancheGo engine [8], the main implementation of the AVA network which uses the Avalanche consensus algorithm [9]. We picked AVA, as it is the main cryptocurrency network that uses an epidemic, sampling-based consensus, which is the target use case of Basalt. Our implementation, a 500-lines patch to the Go source code of AvalancheGo, replaces peer sampling based on stake in a proof-of-stake system by peer sampling based on Basalt, including the hierarchical ranking function described in Section 5.2.3.

Our implementation integrates seamlessly with the AVA protocol and is fully compatible with the existing network. Our implementation supports managing current outgoing connections according to the Basalt algorithm, instead of keeping connections open to all reachable network nodes as done by the original AvalancheGo implementation [10].

To show that Basalt can be applied as a sampling method that reduces the risk of an institutional attack, we ran a 10-hour experiment where we launched 100 "adversarial" Avalanche nodes on the public AVA network (corresponding to about 20% of total active nodes) in an attempt to bias sampling in their favor using a Sybil attack against one of our nodes. The nodes we launched all had IP addresses located in the same /24 prefix, owned by our research institution. Note that our nodes were not acting maliciously and did not attempt to take over the network. In fact, our nodes were not reachable from the outside: they simply contacted existing nodes to retrieve the current state of the AVA network, and then consisted in a "local-only" extension of the network, which we observed from therein. Samples were measured at witness nodes running the Basalt sampling algorithm, as well as the non-hierarchical variant of Basalt and a sampling algorithm based on full network knowledge. Results shown in Table 5.4 show that using Basalt, the probability of sampling one of our adversarial nodes is brought to about 1%, meaning that the influence of our nodes in the network is extremely limited.

To show the wider benefit of Basalt, we plot in Figure 5.8 the number of times the

8. `https://github.com/ava-labs/avalanchego`

9. Our code is publicly available at `https://github.com/basalt-rps/avalanchego-basalt`. Our implementation is forked from the official AvalancheGo repository. Our changes are identified by "Basalt RPS Authors".

10. Unfortunately, we had to disable this behaviour as it led to too many connection attempts and some nodes appeared to have banned our IP addresses as a consequence. A simple modification allows our code to never close connections intentionally: the view maintained by Basalt is only used to sample peers for the Avalanche consensus algorithm, and connections are kept in the background to nodes that have been removed from the view.

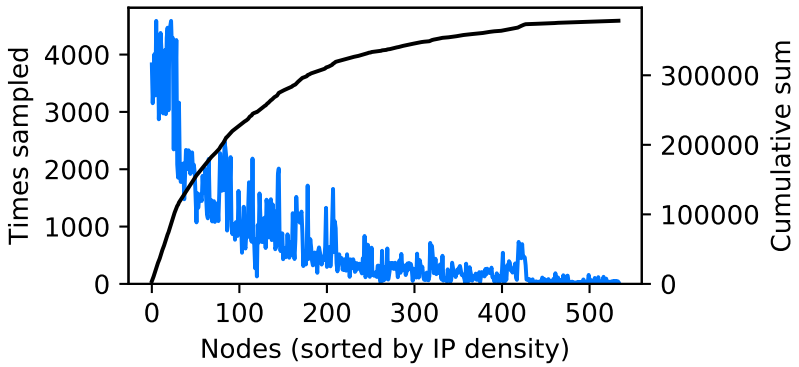


Figure 5.8 – Behaviour of AvalancheGo modified with Basalt running on the public AVA network, 5-hour experiment. On the left, nodes that are alone in their IP prefix are sampled the most frequently. On the right, nodes that belong to IP prefixes where other network nodes are present are sampled less often.

various nodes of the AVA network were sampled in the experiment. Sorting nodes by a density metric which counts the number of other nodes in the same /8, /16 and /24 prefix reveals that nodes which are isolated in their prefix (on the left of the graph) are sampled more often than nodes which share their IP prefixes with other nodes (on the right). However all network nodes have a chance of being sampled, and no single node is sampled exceedingly more often than others. This stands in contrast with Proof-of-Stake-based sampling, where sampling frequency is proportional to the stake invested, a mechanism that gives disproportionate power to rich nodes and totally excludes nodes that are not able to invest any stake in the network.

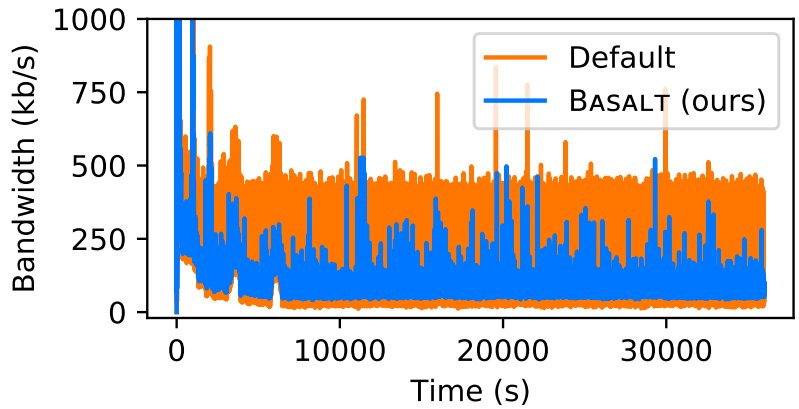


Figure 5.9 – Bandwidth usage of AvalancheGo with and without Basalt (up + down)

Algorithm	**Adversary samples**
Full knowledge uniform sampling	18.4%
Basalt-uniform	17.5%
Basalt (hierarchical)	**1.13%**
True proportion of Byzantine nodes	18.8%

Table 5.4 – Observed proportion of samples that are nodes controlled by the adversary in our live experiment (see Section 5.6)

5.7 Discussion

5.7.1 Comparison with previous works

Surprisingly, very few works have sought to develop Byzantine-tolerant RPS protocols. State-of-the art methods such as Brahms (Bortnikov et al., 2009) and Secure Peer Sampling (Jesi et al., 2010) (SPS) are based on a classical RPS algorithm, to which is adjoined a mechanism that tries to correct for the over-representation of malicious nodes. As we have shown, these methods are less efficient than Basalt. This is due to the fact that they need to acquire sufficient statistics on input data before they can correct biases by filtering the stream.

Our protocol, on the other hand, circumvents these limitations by eschewing filtering altogether, in which a first set of imperfect neighbors are pruned to produce robust samples. Instead, Basalt directly exploits a minimization of hash functions, which makes it capable to effectively handle the above Byzantine flooding attacks. Moreover, the majority of these systems do not address risks that exist in real-world networks, such as Sybil attacks. To our knowledge, the only exception is HAPS (Bouchra Pilet et al., 2020), which is designed specifically to handle Sybil attacks. HAPS, however, only addresses Sybil attacks in which attackers are concentrated in a few IP blocks ("institutional attacks"), by using random walks on a carefully crafted probabilistic tree. Due to its design, it is not immediately clear how HAPS could be extended to counter attackers that are spread out, which Basalt does thanks to its stubborn chaotic search.

5.7.2 On network attacks

Recent works on blockchains have also brought to lighten the risk of attacks at a more fundamental level than those described in Section 5.1.2. Network adversaries are malicious entities that gain control of part of the routing infrastructure (internet autonomous systems, or ASes), in which case they can intercept and modify all the traffic that they are routing, or attack the routing algorithm itself by advertising Internet prefixes that they do not own, thus attracting traffic that should have gone through another path, a so-called *BGP hijack* (Maria et al., 2017).

Note that most ISPs implement BGP filtering and many organizations closely monitor BGP announces for changes and potential hijacks in order to counter them as soon as possible. As a result, BGP hijacking attacks are necessarily limited to one or a few IP prefixes, and even then they cannot extend for long periods of time. By spreading connections over a variety of IP prefixes through its rank function, Basalt builds intrinsic resilience to these attacks as at most only a small fraction of nodes' neighbors will be located in hijacked prefixes. In this way, the global Basalt network is not at risk of being taken down or manipulated by a malicious entity.

However, network attacks might also be used to target specific nodes, to remove them from the global network and make them believe false information about the network's state (an Eclipse attack). Defenses have been proposed against Eclipse attacks at the network level: for instance, the SABRE network (Apostolaki et al., 2019) proposes to use additional communication channels, in the form of a network of specialized nodes that are all connected to one another using dedicated links, and that are located close to end-users so that they can provide a safe service directly to them even in the case of a hijack.

In the case of a blockchain, where the most crucial property to guarantee safety is that all nodes are made aware of new blocks rapidly, the SABRE method is able to help by providing reliable block delivery. For sampling-based methods that use Basalt, SABRE could provide a security mechanism at the application layer to enable detection of network attacks and stop all activity in case they happen, for instance by detecting a discrepancy between a node's local state and the state of SABRE nodes. This mechanism however cannot be used to allow eclipsed nodes to make progress in such a situation, as it does not provide the secure random peer sampling service itself. Finding mechanisms to allow nodes that are eclipsed by a network attack to continue functioning normally when running a sampling-based algorithm is, to the best of our knowledge, still an open problem.

5.8 Conclusion

We have presented a new algorithm for Byzantine-tolerant random peer sampling on the Internet that uses biased sampling to prevent Sybil attacks. Compared with the approaches described in Chapter 3 and 4, this new algorithm helps us tackle simultaneously the challenge of Byzantine fault tolerance on the one side, and of very large open network subject to churn on the other side. Its applications are mainly for the implementation of sampling-based consensus algorithms, a first example of which being Avalanche. Contrary to sampling algorithms based on Proof-of-Stake, such as those currently in use on the AVA network, Basalt allows the network to be truly open by allowing any Internet user to join the consensus without having to own any cryptocurrency tokens. In the next chapter, we will exploit Basalt with a new consensus algorithm, SBO, in order to implement a fully-fledged total order epidemic broadcast primitive in open, trustless networks.

Chapter 6

Simultaneous Block Ordering

As discussed in previous chapters, many kinds of decentralized application benefit from having access to a totally ordered broadcast primitive, in order to allow for the implementation of arbitrary replicated state machines. However currently known algorithms for totally ordered broadcast in the setting of open, trustless networks suffer from several well-known limitations that restrict their usefulness. In particular, algorithms based on Proof-of-Work are extremely power-hungry and provide only low transaction throughputs. Algorithms based on Proof-of-Stake alleviate these issues, but at the cost of limiting the openness of the system, as only members with significant stake can have an active role in the consensus algorithm. Proof-of-Stake also requires the application platform to be intrinsically linked with a cryptocurrency, an unwanted dependency in many cases.

We investigate instead epidemic algorithms as an alternative to PoW- and PoS-based blockchains. To build such algorithms, we have to consider two aspects: Chapter 5 considered the problem of building a Byzantine-tolerant, Sybil resilient random peer sampling algorithm as a secure foundation for epidemic algorithms in the setting of trustless open networks, and proposed the Basalt algorithm to solve this issue. In this chapter, we build upon Basalt and provide a new epidemic algorithm for totally ordered delivery of messages in this context.

Totally ordering messages with epidemic algorithms is not a new problem, but previous approaches fall short of the requirements of modern blockchain systems. EpTO (Matos et al., 2015) for instance is able to provide high-throughput totally ordered epidemic message delivery, but is not tolerant to the presence of Byzantine nodes. Doerr et al. (2011) propose a primitive for a single instance of Byzantine-tolerant epidemic consensus, using an aggregation rule based on the median, but they do not propose an extension to several consecutive broadcasts with low latency and high throughput. On the side of cryptocurrencies, Avalanche (Team Rocket, 2018) proposed an epidemic algorithm for money transfer that is very efficient and Byzantine-tolerant, but this algorithm does not provide total order (which is not required to build the simpler money transfer object) and

is therefore not generalizable.

In this chapter, we propose a new probabilistic Byzantine-resilient consensus algorithm, which we call Simultaneous Block Ordering (SBO), that provides total order on messages and achieves near-optimal throughput for a given bandwidth budget, without depending on a Proof-of-Stake mechanism. SBO is based on epidemic dissemination and a novel ordering technique based on median computations, that allows many messages to be processed simultaneously by the network. SBO is the first epidemic total order consensus algorithm to achieve both Byzantine fault tolerance and high message throughput. Its epidemic nature makes it suitable to open, large-scale networks, while its parallel ordering strategy allows it to deliver a high message throughput in spite of Byzantine attacks.

SBO's core principles are inspired by the method of Doerr et al. (2011), but it is significantly more complex in order to achieve high throughput. Unfortunately, this makes SBO hard to analyze from a mathematical point of view, and we are therefore not able to give a full model with a proof of its Byzantine resilience. Instead, we focus on evaluating SBO from both simulations and a real-world implementation. The contribution of this chapter should therefore be considered more as a preliminary exploration of a new domain of algorithmics, rather than a fully-fledged solution ready for use.

On one hand, we run various simulations of SBO showing that SBO scales perfectly to huge networks (up to 10 000 active consensus nodes in our simulation, but with no true limit currently identified), thanks to its epidemic nature. In addition, simulations enable us to evaluate SBO under a variety of possible attacks and further show that none of them has significant impact on SBO when 10% or less of sampled peers are nodes controlled by the malicious actor [1]. With tolerance to only 10% of malicious nodes, the Byzantine resilience of SBO seems to be significantly weaker than classical BFT algorithms, that often handle up to 1/3 of Byzantine nodes in a closed system. This however does not mean that SBO cannot be used for large-scale collaboration systems: in the absence of monetary value (the case of cryptocurrencies), there is less incentive for a malicious party to try and attack the system, meaning that weaker Byzantine resilience is not necessarily a big issue. More importantly, we see SBO as a preliminary first step in this new domain of epidemic BFT algorithms, and we hope that it will lead to new contributions on this problem.

On the other hand, we build a real-world implementation of SBO where messages

1. Importantly, this does not mean that such an attack is impossible, but it seems to suggest that SBO might have reasonably good theoretical properties.

consist in blocks of transactions, and we evaluate it in an (emulated) geo-distributed deployment with 1000 nodes in 20 cities around the world. We show the following properties:

1. Assuming standard 100 Mbps connectivity for all nodes, SBO is able to process up to 5000 transactions per second, corresponding to several MB/s of effective applicative throughput.
2. The average latency of messages delivered by SBO remains under 30 seconds for all cities, independently of their position on the globe, aligning SBO with the temporal requirements of a typical planetary payment system.

Finally, by allowing nodes to securely bootstrap from any position in the blockchain, SBO removes the requirement for any node to store the whole history of transaction, thus making blockchains much more space-efficient for high-throughput workloads. Indeed, by trusting the majority of nodes to have calculated the correct computation of the blockchain state, we avoid the need to download everything from the beginning in order to verify the whole past of the blockchain. This is possible thanks to an adaptation of the sampling-based mechanism that can be used to reliably obtain the consensus state at any arbitrary blockchain height.

6.1 Problem Statement and Assumptions

Our goal is to implement total order broadcast, i.e. a broadcast primitive where all correct nodes receive messages in the same order, in a very large system whose complete participant list is unknown and subject to constant change due to churn, and in the presence of malicious participants.

In our context, total order broadcast (or atomic broadcast), is defined as follows:

Definition 6.1 *A totally ordered broadcast primitive consists in a broadcast operation where the following properties are satisfied with probability* $1 - \exp(-\lambda)$*, for a tunable security parameter* λ*:*

1. ***Correctness:*** *if at a certain time a correct node has delivered the sequence of messages* $m_1, \ldots, m_k$*, and another correct node has delivered the sequence of messages* $m'_1, \ldots, m'_{k'}$*, then* $\forall i \leq \min(k, k'), m_i = m'_i$*.*
2. ***Termination:*** *if a correct node broadcasts a message* m*, all correct nodes eventually deliver* m*.*

Note that this definition does not require that the delivered messages be unique: in practice, we do not prevent nodes from broadcasting the same message several times. In such a case, correct nodes may deliver the message one or many times, depending on the state of the algorithm. It is thus up to the application to ensure that message processing is idempotent by implementing a way to ignore duplicates, e.g. using signatures and sequence numbers.

In the context of decentralized applications in trustless open networks, a message m may consist in either a single operation (also called a *transaction* in the context of cryptocurrencies [2]), or a block composed of many operations. As we will see later, SBO has a non-negligible overhead for the delivery of a single message, but this overhead does not depend on message size and can thus be compensated by batching operations in blocks as large as possible. From this section until Section 6.3, we focus purely on the message delivery properties of SBO, leaving discussion of batching methods to Section 6.4.

As other epidemic protocols (Guerraoui et al., 2019a; Team Rocket, 2018), SBO relies on a peer sampling services to obtain a unbiased continuous stream of random peers from the network. In the context of this thesis, we assume that this service is provided by Basalt.

SBO follows a query/response exchange pattern (i.e. a pull protocol). Periodically, queries are sent to k random nodes obtained from the RPS service, where k is a parameter of the algorithm (see Table 6.1 for a list of common parameters we use). Responses are then processed together, once enough have been received.

More specifically, our system model makes the following assumptions:

1. The network is composed of an unknown, potentially very large, number of nodes. Nodes may join and leave the system at any time.
2. A random peer sampling (RPS) service is available, that is able to sample at any time a set of k random peers in the network.
3. The probability of each of the k sampled nodes of being Byzantine is no more than a fixed probability f. This probability also counts nodes that misbehave in one way or another, without malicious intention. Note that many kinds of misbehaviour, such as bad network connectivity, can be transient, and misbehaving nodes at one time

2. We will use the terms *operation* and *transaction* interchangeably. In particular, we will use the term of *transactions per second* (tps) to designate a measure of the consensus throughput of an algorithm, staying in line with the vocabulary used in the literature on consensus algorithms in open trustless networks, which often considers exclusively the case of cryptocurrencies.

can be correct at another time. We suppose that at all times, the probability of a sampled node to be malicious or misbehaving during the associated query/response cycle is less than f.

4. There is a timeout Δ such that when a query is sent to a correct nodes, the response will be received before Δ time has passed (Δ is an upper bound on the round trip time). Responses that are lost or delayed for more than Δ count in the probability f of sampling Byzantine nodes.
5. Nodes joining the network are assumed to have already successfully bootstrapped the consensus algorithm.

Algorithm parameters		
k	Query count	20
w	Maximum ordering window	1000
r	Validation threshold (security parameter)	20
b	Safety margin	0.2
Environment parameters		
N	Number of nodes (known only in simulation)	
f	Fraction of malicious nodes in samples	$< b$
Δ	Maximum message delay for correct nodes	
Other variables		
ρ	Throughput (messages per second)	

Table 6.1 – Parameters of the SBO algorithm and the environment

6.2 The SBO Algorithm

6.2.1 Algorithm overview

The SBO algorithm progressively orders incoming messages into a totally ordered log (which we call a *history*) by working on a *window* of pending messages that are currently being ordered before delivery. This ordering window is updated iteratively using periodic epidemic exchanges: in each iteration, SBO queries k random nodes (obtained using the RPS service) for their current ordering window. Once enough replies have been received, the responses are used to re-order the messages in the current node's window. A message

stays in the window until its position stabilizes, at which point it moves to the node's history.

The reordering step exploits median computations, shown on a toy example in Figure 6.1: for each message m (a, b, c, and d in the figure) in the received windows, SBO computes the median of the positions of this message in the incoming windows (2 for a, 3 for b, etc.), and reorders messages by the obtained medians (argsort operation in the figure, with ties decided deterministically, for instance using message hashes).

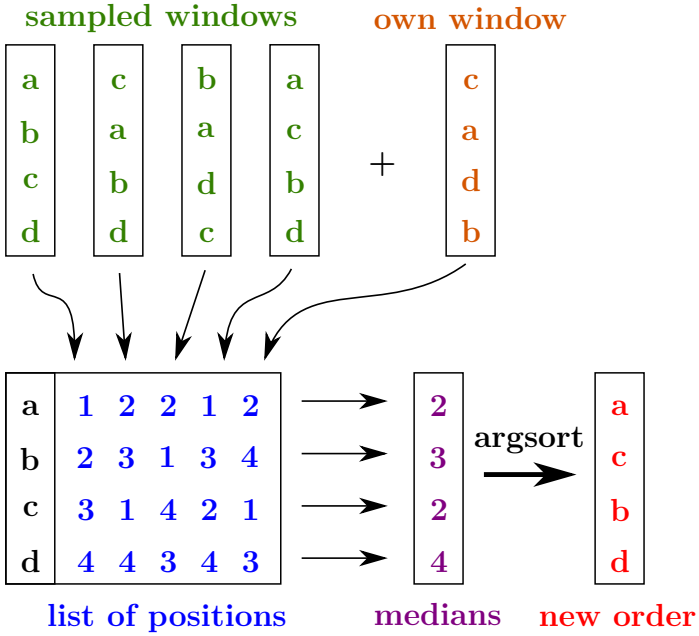


Figure 6.1 – The simultaneous median-based reordering mechanism

Once a message has stayed long enough in the same position in the ordering window, it moves to the node's local history. To this end, SBO keeps track of how many consecutive times a message has stayed in the same position. Once this count reaches a security parameter r, the message is considered stable and is thus removed from the ordering window and appended to the node's history.

The ordering window thus serves as the tip of the sequence of all messages to be ordered: messages contained in the node's *history* are considered validated by the network, i.e. their position is considered stable and definitive, while messages in the ordering window are in the process of being ordered.

In practice, different processes may not have the same start position for their window during a query/response cycle, i.e. they may not have validated the exact same number

of messages in their history. To handle this situation, a node specifies the offset of the beginning of its current window when requesting windows from other peers. A queried peers replies by sending a window of up to w messages starting at the requested offset, rather than its own window.

To further boost safety, nodes also exchange hashes of their history as part the query/response cycle, in order to detect when the consensus algorithm has failed, in which case they can signal failure to the user or automatically take corrective action such as backtracking or bootstrapping from scratch. These mechanisms are omitted in the pseudo-code of Algorithm 6.1 for simplicity.

6.2.2 Algorithm description

The pseudo code of the SBO protocol is shown in Algorithm 6.1. The SBO algorithm works with consecutive rounds of queries sent to random peers. A first round of queries is initiated as soon as the algorithm starts, on line 8. On line 14, SBO samples k random peers using the RPS protocol and sends them a query message containing the offset of the ordering window, i.e. the length of the history. SBO stores these peers in the variable Queried, so that when responses are received it is able to know that the message is a legitimate response to a query and not a spurious message from a Byzantine peer. On line 20, correct peers that receive a query with a certain offset reply with the list of at most w messages in their current ordering starting from the specified offset, taking from the history or the current ordering window as necessary. On line 22, when SBO receives a reply, it verifies its origin and saves it in the Incoming buffer. The peer that sent the reply is then removed from the Queried set (line 24), so that a possible second reply from the same peer is not accepted. Once enough replies are received, i.e. at least $(1 - b)k$, SBO processes them (line 26, described in next paragraph), and starts a new round of queries (line 27). If not enough replies are received within a timeout interval Δ (which is initiated in SendQuery at line 17), SBO knows by Assumption 4 than more than a proportion b (with $b > f$) of the sampled nodes are Byzantine. It thus scraps all currently buffered responses and starts a new query round from scratch (line 29).

The ReorderAndValidate procedure is invoked each time enough replies are received to a round of queries. This procedure consists in producing an order of known pending messages based on the replies received at line 21, and adapting the local ordering window to become closer to that target. On line 36, SBO adds its own window to the set of received replies so that it is counted as a supplementary input to the reordering algorithm. Each of

these inputs consists in a different ordered list of messages. On lines 56-62, SBO calculates the set E of all messages that appear in at least one of the input lists and that is not already in the process' local history, and for each $m \in E$ it stores in a list $P[m]$ the indices (the positions) at which m appear in the input lists. On lines 63-65, SBO completes the lists $P[m]$ for all messages m so that they have the same size: the number of input lists. $P[m]$ is completed by making the assumption that if a message m was not present in one of the input list (which is of size at most w for correct processes), it might have been present at position $w + 1$, therefore SBO completes the lists $P[m]$ with enough times the value $w + 1$ to reach the targeted size. SBO then reorders all of the messages m of E, sorting them by the median value of $P[m]$ (line 66), which is stored in a temporary buffer L.

This new order of messages is not directly accepted as the new algorithm's window. SBO begins by identifying the length c of the longest common prefix between the current window and the temporary buffer (line 38). SBO then adapts the stability counters associated with each item of the current window: messages within the common prefix, i.e. whose order stays the same in the new order, have their counters incremented (lines 39-40). On the contrary, messages after this prefix have their counters decremented (lines 41-42). Once the new counters have been calculated, SBO identifies the position i_0 of the first message in the ordering window that has a zero counter (line 43). Messages at positions 1 to $i_0 - 1$ stay in their previous positions, while all messages starting from message i_0 are removed (line 44) and will be replaced by messages from the temporary buffer in the order in which they appear in it. Before calculating this new message order, SBO validates the messages that appear at the beginning of the window and whose counters have reached the security threshold r (line 46). These items are moved out from the window and into the history, and are simultaneously delivered to the application (lines 70-72). Once these messages have been validated, SBO fills the window with messages from the temporary buffer L in the order in which they appear in L, until it reaches its maximum size w (line 51). The stability counters of these messages is set to zero. If messages remain in L after the window has reached its maximum size w, these messages are inserted in the queue to be processed later. On the contrary, if messages were previously in the queue and space is now available in the window due to messages being validated and freeing up slots, they are moved from the queue to the window to be processed by the network (lines 54, 31-34).

Algorithm 6.1: Simultaneous Block Ordering

```
1  algorithm parameters
2      w, k, r, b ← parameters
3  initialization
4      history ← []
5      window ← []; window_cnt ← []
6      queue ← ∅
7      queried ← ∅; incoming ← []
8      sendQuery()
9  function to_broadcast(m)
10     queue ← queue ∪ {m}
11     dequeue()
12 function sendQuery()
13     incoming ← []
14     queried ← k random peers sampled in the
         network
15     for p ∈ queried do
16         Send ⟨QUERY, |history|+1⟩ to p
17     reset timeout
18 on receive ⟨QUERY, t_0⟩ from p'
19     to_send ← (history +
         window)[t_0, ..., t_0 + w − 1]
20     Send ⟨REPLY,to_send⟩ to p'
21 on receive ⟨REPLY, x⟩ from p
22     if p ∈ queried then
23         incoming.append(x)
24         queried ← queried\{p}
25         if |incoming| ≥ (1 − b)k then
26             reorderAndValidate()
27             sendQuery()
28 after timeout Δ
29     sendQuery()
30 function dequeue()
31     while |window| < w ∧ queue ≠ ∅ do
32         e ← min(queue); queue ← queue\{e}
33         window.append(e)
34         window_cnt.append(0)
35 function reorderAndValidate()
36     incoming.append(window)
37     L ← reorder(incoming)
38     c ← |max_common_prefix(window, L)|
39     for i = 1, ..., c do
40         window_cnt[i] ← window_cnt[i] + 1
41     for i = c + 1, ..., |window| do
42         window_cnt[i] ←
             max(0, window_cnt[i] − 1)
43     i_0 ← position of first 0 in window_cnt, or
         |window_cnt|+1
44     window ← window[1, ..., i_0 − 1]
45     window_cnt ← window_cnt[1, ..., i_0 − 1]
46     validate()
47     for e ∈ L[c, ...] do
48         if e ∉ history ∪ window then
49             if |window| < w then
50                 window.append(e)
51                 window_cnt.append(0)
52             else
53                 queue ← queue ∪ {e}
54     dequeue()
55 function reorder(X)
56     E ← ∅; P ←empty map
57     for x ∈ X do
58         for i ∈ 1, ..., |x| do
59             if x_i ∉ history ∧ x_i ∉ [x_1, ..., x_{i−1}]
                 then
60                 if x_i ∉ E then
61                     E ← E ∪ {x_i}; P[x_i] ← []
62                 P[x_i].append(i)
63     for e ∈ E do
64         while |P[e]| < |incoming| do
65             P[e].append(w + 1)
66     return E sorted by λe ∈ E.median(P[e])
67 function validate()
68     while |window| > 0 ∧ window_cnt[1] ≥ r do
69         to_deliver (window[1])
70         history.append(window[1])
71         window ← window[2, ...]
72         window_cnt ← window_cnt[2, ...]
```

6.2.3 Link with previous approaches and arguments for correctness

SBO builds upon previous works on Byzantine-tolerant epidemic consensus algorithms, namely Doerr et al. (2011) and Avalanche (Team Rocket, 2018).

The algorithm provided by Doerr et al. focuses on achieving consensus on a single value, given any number of inputs. It works in a similar fashion as SBO with synchronous rounds where peers are sampled and their values are used to calculate a new state. However the state maintained by the algorithm consists only of a single value, and only two peers are sampled in each round. The algorithm of Doerr et al. assumes that there is a total order on the set of all possible values (e.g. ordering them by the hashes of their serialized representations), and the aggregation rule consists in selecting as a new state the value which is the median of the input set (the set of values consisting in the processes' previous local state and the values received at this round from two other processes). Doerr et al. show that their method converges in polynomial time assuming no more than $\sqrt{N}$ nodes exhibit Byzantine behavior.

When limited to the case of two input values (say 0 and 1), Doerr et al. and SBO behave in extremely similar fashions: the median value selected by the algorithm of Doerr et al. is the value which has the majority of votes in the input set. Instead of selecting either 0 or 1, SBO will produce one of two possible orderings: 0 then 1, or 1 then 0. When focusing on the first value of the produced order, SBO also selects the value which has the majority of first positions in the input set. In other words, selecting a value in the first position in SBO is equivalent to selecting the single value that makes up the state in the algorithm of Doerr et al. This means that the theoretical analysis by Doerr et al. can be transposed to SBO to show that, in this simple case with two messages to order, SBO is correct and terminates in polynomial time with high probability when $\sqrt{N}$ of the nodes at most are Byzantine. It remains to be shown that this proof extends to the general SBO case where the messages to be ordered come in a continuous stream.

SBO also takes strong inspiration in the intermediary algorithms Snowflake and Snowball presented as precursors to the full Avalanche algorithm (Team Rocket, 2018). In particular the stability counters in SBO function almost in the same way as the counters in Snowball: these counters are increased when an option has a majority vote in the input set, and decreased in the case when another option has the majority vote. SBO also borrows the mechanism where a selected value cannot change until the stability counter is

decreased to zero, in order to tolerate cases when Byzantine nodes are a majority in a particular input set due to the randomness of the sampling. Again when focusing only on the selection of the first message in the queue in SBO, the mechanism is close enough to Snowball that the same kind of analysis should apply. Completing such a theoretical analysis and extending it to the general SBO algorithm is left to future works.

Compared to these previous approaches, SBO innovates by letting nodes build a total order on a large set of messages simultaneously, thus producing consensus much faster than by selecting messages one by one.

6.3 Evaluation in Simulation

We evaluate the performance and robustness of SBO using two experimental set-ups: with simulations (this section), and using an actual prototype implementation executed on an institutional grid platform emulating a geo-distributed planetary deployment (Section 6.4).

6.3.1 Model, metrics, and attacks

For our simulations, we make the following simplifying assumptions:

— *Synchronous rounds:* we assume all messages take the same time to arrive.

— *Full network knowledge:* we assume N nodes of which fN behave maliciously, and that correct nodes sample other peers uniformly among the N network nodes. This is equivalent to assuming a perfect RPS service in a system without churn.

We are interested in evaluating the following values:

— *Message throughput:* the number of messages delivered per synchronous round. The equivalent number of transactions depending on block size is studied later (in Section 6.4), based on real-world networking constraints.

— *Latency:* the average number of rounds between a message being broadcast and it being delivered.

We are also interested in evaluating the resistance of SBO to Byzantine behaviors. We do not know exactly what would be a worst-case Byzantine behavior to implement in our simulator, therefore we focus on the following five possible scenarios of malicious nodes trying to disrupt the network, that cover a diverse spectrum of possible attacks:

— (S1) *Reverse and truncate responses:* Byzantine nodes run the same algorithm as

correct nodes in which they try to evaluate the current state of the network. When they answer queries about their current state, they attempt to disrupt the stable order that correct nodes are trying to build by reversing their current window and truncating the last half of it before sending it as a response.

— (S2) *Shuffle response:* Byzantine nodes shuffle their ordering window before sending it.

— (S3) *Divide:* Always answer with the same two Byzantine-generated messages m, m', putting message m before m' when answering to one half of the correct nodes, and putting m' before m when answering to the other half. This attack attempts to destabilize the network by making different nodes validate different message orderings, in which case the algorithm will detect an inconsistency and stall.

— (S4) *Garbage:* Replace the hash of the validated history by a random value, thus announcing an inconsistent state that is ignored by the recipients.

— (S5) *Reply-self:* Reply to correct nodes with their own state from the last step (this attack assumes that Byzantine nodes know the state of correct nodes slightly in advance).

Note that at the exception of (S5), all these attacks are easy to implement as they do not require the malicious nodes to have any knowledge of the network state that they could not easily obtain when behaving as correct nodes.

We set SBO's parameters to their default value, indicated in Table 6.1. In particular, the window size w is fixed at 1000, which is a good compromise in practical scenarios as we will see when varying its value in Section 6.4.

6.3.2 Key findings

— SBO provides an application message throughput of up to 17 messages per synchronous round with a delivery latency of under 30 synchronous rounds, in a 1000-node simulation (Figure 6.2).

— This result is valid with up to 10% of Byzantine nodes implementing (S3). Other Byzantine attacks strategies have less impact on SBO: the algorithm is able to tolerate up to 30% of Byzantine nodes implementing (S2) (Figure 6.3).

— Delivery latency increases indefinitely when the system is under higher load and SBO is no longer able to deliver messages as fast as they are sent (Figure 6.2).

— The behavior of SBO stays similar when the network size increases up to 10 000 nodes. In particular, SBO tolerates 10% of Byzantine nodes (a linear fraction)

independently of network size (Figure 6.4). This is surprising given the results of Doerr et al. (2011) that predicted tolerance to only $O(\sqrt{N})$ Byzantine nodes.

6.3.3 Detailed experimental results

Figure 6.2 shows how SBO behaves in simulation when the load increases, i.e. when more messages are sent by network nodes, in a simulated network of 1000 nodes and with a maximum window size $w = 1000$. Figure 6.2 is plotted with different fractions f of Byzantine nodes (from 0 to 10%) with the (S3) Byzantine behavior (which tries to divide nodes in two groups). Other parameters are set to their default value of Table 6.1. SBO has a saturation point that is defined by the size w of the window: since each message has to stay at least r consecutive rounds in the window before being validated, the maximum throughput per step is w/r messages. However this best case can happen only if all messages are initially in the correct positions at all nodes. Otherwise there is an intermediate period (visualized in the next section) where messages get reordered, adding to the r rounds. Figure 6.2 shows that in the case of a window size of $w = 1000$ the maximum achievable throughput is around 17 messages per round, and that this saturation point (and the corresponding algorithm's latency) is independent of the fraction of Byzantine nodes within the considered range $[0, 0.1]$. In the non-saturated regime, SBO is able to maintain an almost optimal latency (close to the security parameter $r = 20$). This behavior is independent of the proportion of Byzantine nodes $f \in \{0, 0.05, 0.1\}$.

Due to the nature of the simulation, which consists in consecutive synchronous rounds where the amount of data that can be transmitted between nodes at each round is unbounded, increasing the window size w would allow SBO to reach arbitrarily high throughputs in terms of messages delivered per round. However in a real-world setting, as we will see in the next section, network congestion becomes an issue when w is too large, in which case the throughput can drop sharply. It is therefore important to select w small enough as to avoid reaching a state of network congestion.

Figure 6.3 shows the impact of the five different malicious attacks we implemented, in a regime where SBO is not saturated (10 messages per round), in a simulated network of 1000 nodes with a maximum window size of $w = 1000$ and 10 messages generated per simulated round in the whole network (other parameters set as in Table 6.1). For small fractions of Byzantine nodes, SBO is able to deliver the messages at the rate at which they are sent (Figure 6.3a). The proportion f of nodes required to stop the algorithm depends on the implemented attack strategy: in the worst case (S3), 15% of Byzantine

nodes are enough to stop SBO almost entirely. SBO tolerates up to 30% of malicious nodes with the weaker malicious behavior (S2). Figure 6.3b similarly shows that the protocol's correctness is maintained for Byzantine fractions up to 10%.

Figure 6.4 shows the behavior of SBO as the network size scales from 1000 to 10000 nodes, with a constant message sending rate (10 messages sent in the whole network at every round). Note that communication cost per node per round stays almost constant as network size scales. f represents the fraction of malicious nodes, which implement (S3) as in Figure 6.2. In line with classical results on epidemic algorithms, SBO scales perfectly with network size, delivering messages at the rate at which they are produced in all cases. Latency only slightly increase with network size, while network usage per node per round is a constant.

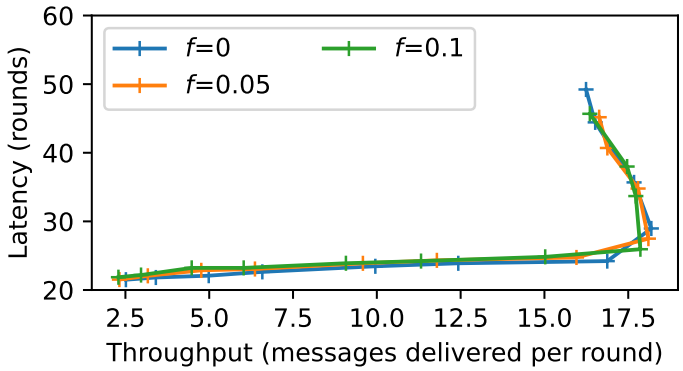
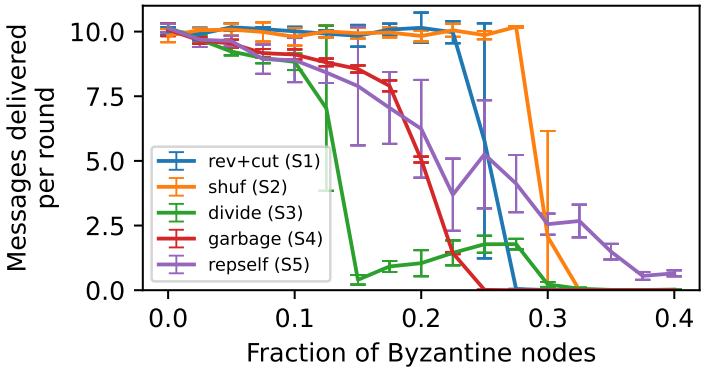
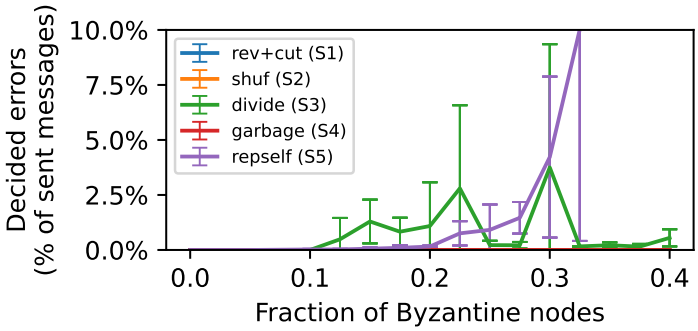


Figure 6.2 – Latency/throughput plot of SBO in simulation as load increases, for different proportions f of Byzantine nodes that implement (S3)

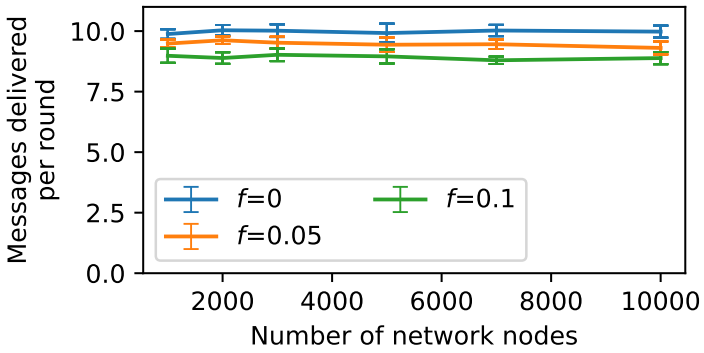


(a) Throughput

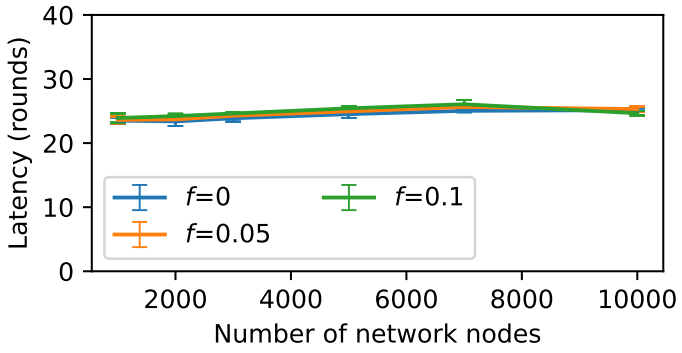


(b) Inconsistent histories (consensus failures)

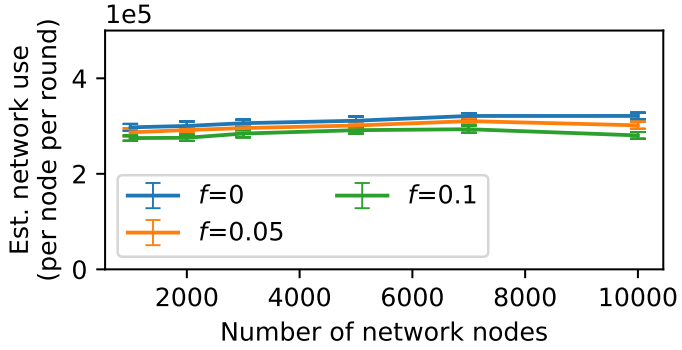
Figure 6.3 – Behaviour of SBO in simulation when the fraction f of Byzantine nodes increases, depending on the Byzantine node's attack strategy



(a) Throughput



(b) Latency



(c) Network usage

Figure 6.4 – Behaviour of SBO in simulation as network size scales

6.4 Real World Experiment

To evaluate the real-world performances of SBO, we implement a fully functional prototype and deploy it on an institutional cluster with artificial latencies that emulate the distribution of 20 large cities around the world. We use SBO in a batching mode, in which each message carries a payload that aggregates several application-level operations (or transactions), and we experiment with payload of different sizes. We assume that there are no Byzantine nodes in the system for this experiment.

Figure 6.5 – Dataset of 20 cities for emulating latencies

6.4.1 Set-up and research questions

Our experimental setup consists in the following components:

— 20 physical machines, each running 50 instances of SBO in Docker containers.

— Emulated pairwise latencies based on data from WonderNetwork [3], where each of the 20 physical machines corresponds to one of 20 large cities around the world (see Fig 6.5).

— Emulated bandwidth limitation to 100 Mbps (symmetrical) for each Docker container.

In this section, we focus on investigating the following research questions:

— *RQ1:* Is SBO able to run with latencies typical for Internet communications between cities around the world, i.e. of the order of hundreds of milliseconds?

— *RQ2:* What is the maximum throughput SBO is able to achieve in such conditions?

3. `https://wondernetwork.com/pings`

— *RQ3:* What is the network usage of SBO in kilobytes per second in this setting?
— *RQ4:* Does SBO provide a uniform performance independent of the (emulated) geographical location?
— *RQ5:* Can SBO reach performances close to the optimal defined by the link speeds?

6.4.2 Theoretical maximal throughput

Our environment consists in nodes with a symmetrical bandwidth of 100 Mbps (∼10 MB/s). This means that no consensus algorithm can achieve a throughput of more than 10 MB/s of application data. In practice, the consensus protocol has an overhead meaning that the effective throughput will be lower. This section discusses the theoretical maximal throughput given the overheads of the SBO protocol.

The SBO protocol exchanges messages identified by their 32-byte SHA-256 hash. With the default values $r = 20$ and $k = 20$, Fig 6.2 shows that in a nominal unsaturated regime messages take at most 30 algorithm rounds to be delivered, even in the presence of (limited) Byzantine behavior from certain nodes. The cost of the consensus algorithm itself for a single message therefore is: $32 \times 20 \times 30$ bytes = 19.2 KB per message.

Assuming that the payload contained in a message is negligible, which would be the case for messages that contain a single transaction which is a few hundred bytes, our 10 MB/s network connection should allow us to validate 500 messages per second. For 400-byte transactions (Bitcoin's average transaction size), this means that the consensus throughput of SBO could theoretically reach up to 200 KB/s in this setting (equivalent to 500 tps, transactions per second).

Suppose now that we group transactions in blocks, and assume for instance that the average block size is 100 KB. The cost of the consensus algorithm is still only of 20KB per validated block, which is an overhead of only 20%. The network should now be able to validate no more than 83 blocks per second (10 MB / 120 KB), however this now corresponds to a consensus throughput of 8.3 MB/s, which is close to the maximum allowed by the network connection. Again assuming Bitcoin's average transaction size of 400 bytes, this means slightly more than 20,000 transactions could theoretically be validated per second with SBO, assuming that it is able to fully utilize the available bandwidth.

6.4.3 Key findings

— SBO is able to provide a stable application throughput of 2 MB/s (24% of theoretical maximum) with delivery latency below 30 seconds in all cities (Figure 6.8).
— Network usage varies two-fold between different geographical locations (Figure 6.8).
— Aggressively setting algorithm parameters allows SBO to reach an application throughput of over 3 MB/s at peak usage (40% of theoretical maximum), however this scenario is very unstable as latency rapidly increases under saturation (Figures 6.6 and 6.7), and the algorithm throughput may even drop to zero with too large window sizes (Figure 6.7).
— Optimal payload size is around 100 KB per message (i.e. blocks of size 100 KB), with larger sizes providing only marginal throughput improvement and potentially increasing delivery latencies much faster (Figure 6.6).
— Optimal window size w is between 1000 and 5000 messages, with larger sizes providing no marginal throughput improvement and risking congestion that completely kills the algorithm's performance (Figure 6.7).

6.4.4 Detailed experimental results

While theoretically a 10 MB/s bandwidth should allow SBO to validate around 8 MB/s of data utilizing block sizes of 100 KB, this analysis does not take in account the effects of congestion and latency. Our experiments show that in fact SBO stalls at lower consensus rates. However the loss is not orders of magnitude: we were able to achieve consensus throughputs around 40% of the theoretical maximum.

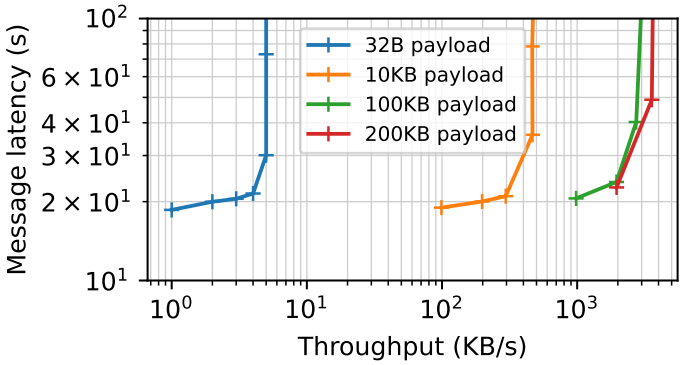
Figure 6.6 shows the behavior of SBO, with a constant window size of $w = 1000$, for various payload sizes between 32 bytes and 200 KB. Bigger payloads enable to reach higher consensus throughputs at lower messages rates. However transmitting the payload has an impact on overall network latencies as the transmission of payload messages competes for bandwidth with the consensus messages of SBO, introducing delays in the consensus. The maximum achievable consensus throughput was slightly above 3 MB/s (∼7500 tps) for a payload of 200 KB. However the latency increases sharply when the network becomes saturated: to keep the lowest possible block latency of around 20 to 30 seconds (depending on node location, see Figure 6.8b), the network must not be processing more than about 2 MB/s (∼5000 tps).

Figure 6.7 show the evolution of SBO's throughput when varying the window size w

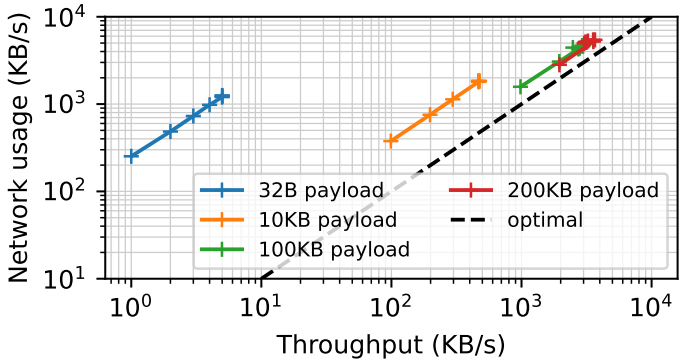
and increasing the load in terms of messages to be transmitted, for a payload of 100 KB per message. Larger window sizes allow for more messages to be validated per second, here reaching almost 4 MB/s of application data with a window size of $w = 5000$. However there is limited benefit to increasing the window size when messages are larger, as shown also in results from Figure 6.6: when messages are too large they compete for bandwidth with the consensus algorithm, preventing it from reaching maximum throughput. For smaller window sizes, the limit is reached when the window is always full, in which case SBO's throughput stagnates at its maximum value. In these particular experimental setting, the optimal compromise to achieve maximum consensus throughput with blocks of average size 100 KB is using a window size between $w = 1000$ and $w = 2000$, as it allows to reach almost optimal throughput without degrading under heavy load. This limits the network throughput to slightly less than 40 messages per second (4 MB/s of application data, ∼10 000 tps). Note that in the saturated regime, latency is virtually infinite as the message queue can never be fully processed and keeps growing.

Figures 6.8 and 6.10 show the results for a single run of the experiment in the optimal conditions identified above: with $w = 1000$ and 20 messages sent per second, and with a payload of 100 KB per message, SBO is far from being saturated and is able to produce 2 MB/s of consensus at low latencies (20 to 30 seconds). We observe that some cities that have better global connectivity, such as London or Toronto, have a more active role in the network: the network bandwidth used by these nodes is larger, and they are able to run more steps of the algorithm per second on average. This translates into higher bandwidth usage for these nodes (Figure 6.8c), and lower message delivery latency (Figure 6.8b). However all cities are able to deliver all the messages at the rate at which they are sent (Figure 6.8a).

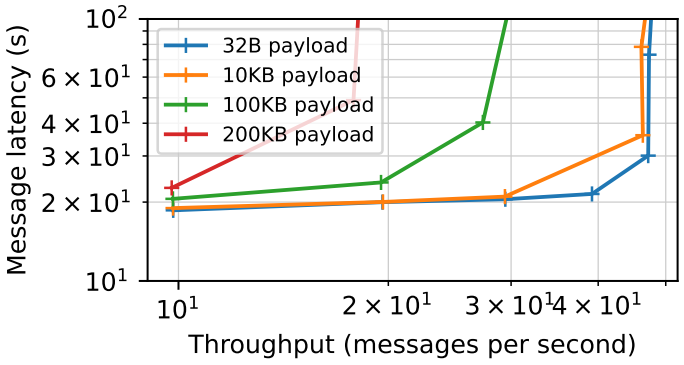
Figure 6.9 shows the positions of a single message in the queues of all of the nodes in the network and its evolution over time, in the same experiment as Figures 6.8 and 6.10. Each dot corresponds to an observation of the position of the message in a particular node's state buffer and at a particular point in time, with on the x axis the time since first observation of message (in seconds), and on the y axis the position (or height) of the message in the blockchain produced by the system. Points with the same color correspond to nodes in the same city. In about 6 seconds all nodes agree on a single position that will be the definitive position of the message once it is validated. However to ensure that this position is indeed definitive, the algorithm waits 20 or 30 seconds more (depending on the node's location, see Figure 6.8b) before validating the message.



(a) Latency-throughput plot for varying payload sizes

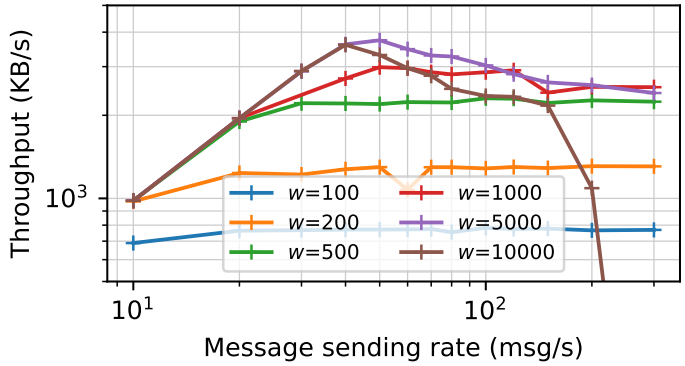


(b) Associated network usage

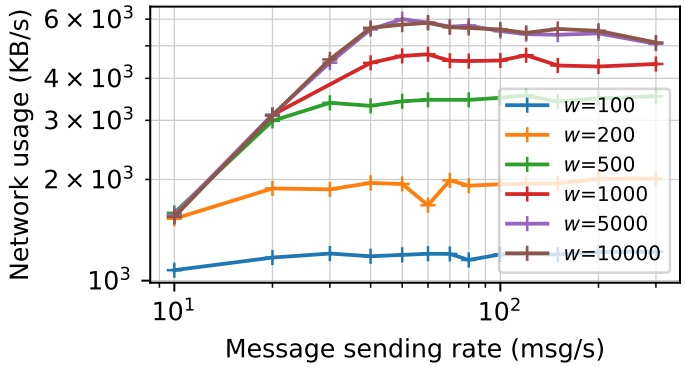


(c) Throughput/latency by message number

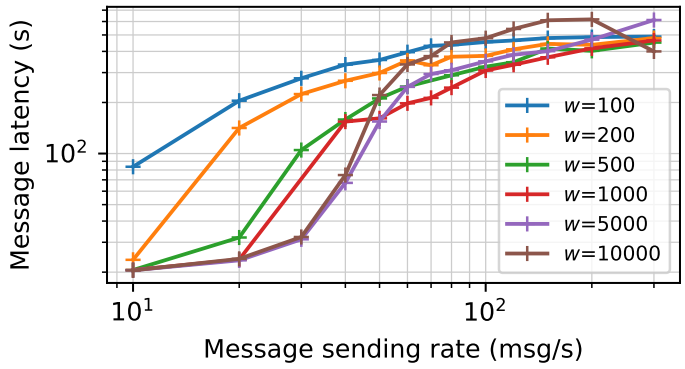
Figure 6.6 – Evolution of SBO in a real-world setting, for different payload sizes, as the production rates of messages increases in the network. Window size is fixed at $w = 1000$.



(a) Throughput as load increases for various window sizes

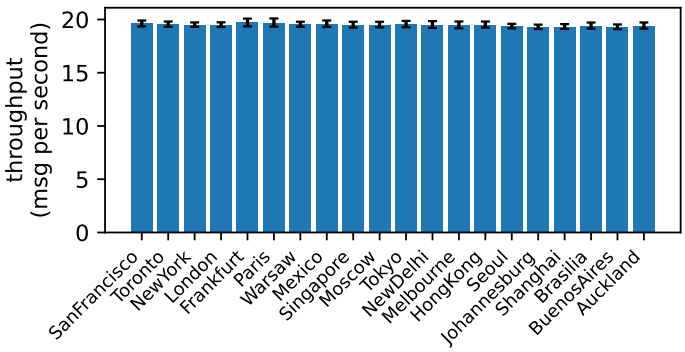


(b) Associated network usage

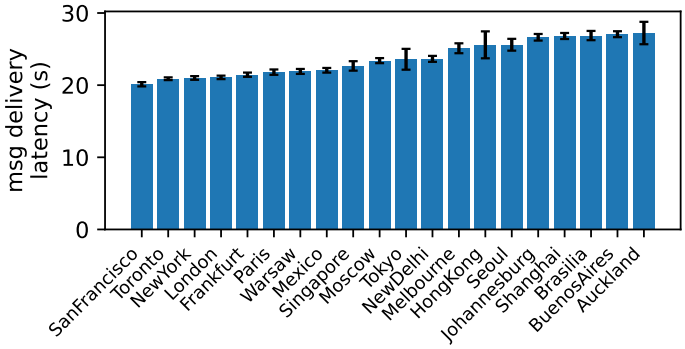


(c) Associated message latency

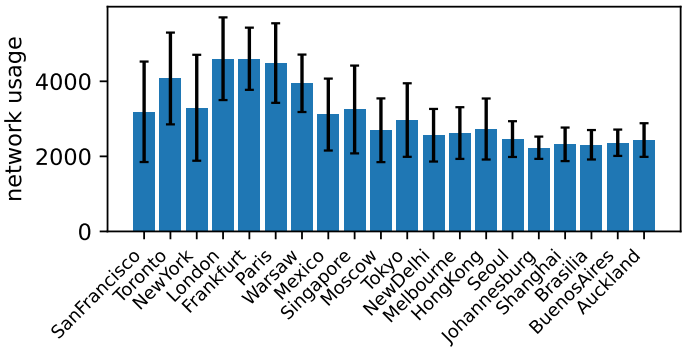
Figure 6.7 – Impact of varying the window size w in a real-world setting. Payload size is fixed at 100 KB per message.



(a) Throughput



(b) Latency



(c) Network usage

Figure 6.8 – Detail between emulated cities, window size $w = 1000$, 20 messages emitted per second, payload size 100 KB per message.

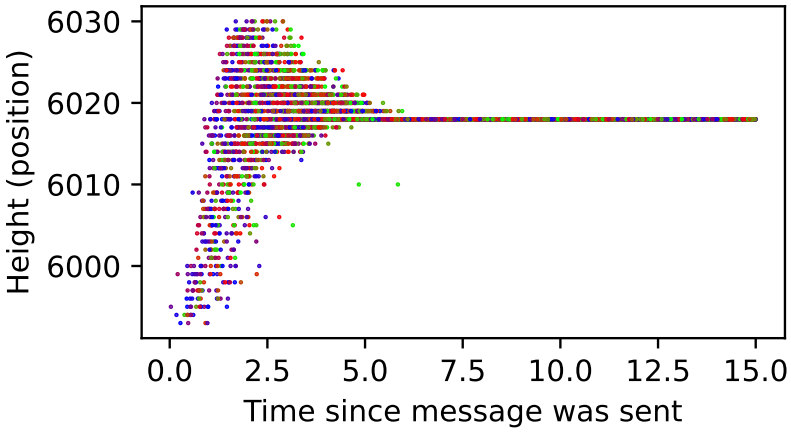


Figure 6.9 – Observation of the positions of a single message over time in our real-world experiment

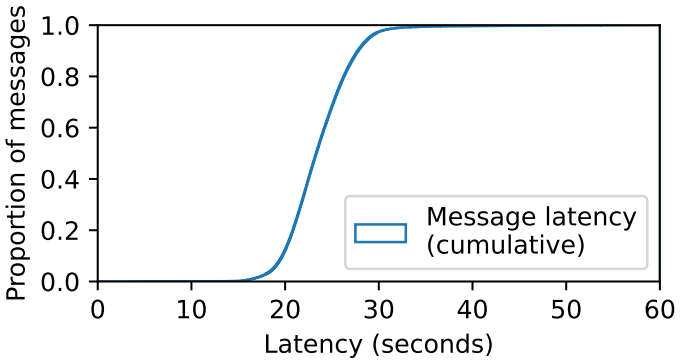


Figure 6.10 – CDF of message delivery latencies (over all simulated cities), for the same setting as Figure 6.8

6.5 Discussion

6.5.1 Alternative Deployment Scenarios

Although we argue for the deployment of SBO in very large networks with open participation, and without access control mechanisms such as Proof-of-Stake, this is not the only context in which SBO can be employed. SBO is versatile and can be adapted to many other deployment scenarios, where the random peer sampling is provided using methods appropriate to the context. For instance, in a consortium network, i.e. a closed system, nodes could have a full list of the network participants, which would enable them to sample uniformly. The security properties of SBO would then be defined in relation to the proportion of such nodes that are malicious.

Implementations in networks that implement a Proof-of-X scheme (Proof-of-Stake, Proof-of-Work, Proof-of-Elapsed-Time, Proof-of-Space-Time, Proof-of-Burn, ...) are also an option: these schemes allow the network to affect a *power* to all participants based on a given resource they own or control. This power can be directly used to build a Sybil-resistant random peer sampling service, by sampling peers with probability proportional to their power in the network, thus allowing for SBO to be deployed.

In both of these cases, SBO has advantages over existing options in the fact that it has $O(n \log n)$ message complexity thanks to the properties of epidemic dissemination, compared to $O(n^2)$ for most PoS algorithms (Gilad et al., 2017; Miller et al., 2016). This mean that in theory SBO could scale effortlessly to tens of thousands of active consensus nodes. SBO is also a conceptually simple algorithm, compared to classical BFT algorithms that have been adapted to PoX schemes, as it does not require precise network membership knowledge and supports node churn transparently. SBO's implementation is thus much more straightforward and less subject to mistakes caused by complexity.

6.5.2 Secure bootstrapping without history

A common problem in blockchains today is the need for all consensus nodes to store the whole history starting from the genesis block, for validation purposes. Indeed, a new node that arrives in the system needs to check that this whole history is valid before accepting it as the current state of the blockchain.

Building upon the principles of SBO, a protocol could be built where joining nodes do not need to retrieve all this information to validate the network state. Indeed, supposing

that the hash-linked chain produced when SBO validates messages also contains the state that results from the application of all transactions contained in the validated blocks, a bootstrapping procedure based on random sampling similar to SBO can be used to assess with high probability the current consensus state of the blockchain starting at any arbitrary height. Once this consensus state has been retrieved and accepted by the joining node, it can start participating normally in the consensus algorithm, with its window position set to the start height it selected for bootstrapping. This means that no network node needs to store the whole history of the blockchain. In fact, drawing comparison from similar analysis on other blockchains (Kim et al., 2021), a SBO-based blockchain platform could likely function perfectly while only storing a few days of transaction history.

6.6 Conclusion

We have presented a new total ordering algorithm, SBO, that works in very large very open networks and in presence of malicious actors. SBO is based on a random peer sampler, which is expected to return a small fraction of malicious nodes. This means that the peer sampler must implement resistance to a variety of attacks including Sybil attacks, which is the case with the Basalt algorithm we presented in Chapter 5.

We have shown in simulation that SBO continues to function when up to 10% of nodes returned by the peer sampler are malicious, for a variety of malicious behaviors. While this fraction is lower than in classical BFT algorithms, which often tolerate up to 1/3 of Byzantine nodes, SBO remains of interest as it is, to our knowledge, the first Byzantine-fault-tolerant total ordering algorithm that uses epidemic dissemination, allowing it to scale to arbitrarily large networks where all nodes have an active role, while providing high throughputs (close to the maximum of the available bandwidth) in the case of few attackers. Remark also that in the case of collaboration systems that are not linked to a cryptocurrency, the value in attacking a system is much lower, and therefore an attacker is less likely to expend excessive resources in performing an attack. This means that even 10% Byzantine tolerance can be sufficient for many applications in this domain.

We have also comprehensively tested SBO's performance on a prototype deployment emulating planetary network delays. Our experimental results demonstrate that SBO can serve as the foundation to implement geodistributed ledger platforms delivering up to 5000 tps, and a block latency below 30 seconds. These results are obtained with consumer-grade connectivity (100 Mbps upload and download speed for all nodes) and could be vastly

improved by using higher-end gigabit network connections.

In this study we were able to provide only limited mathematical proof for SBO's correctness, therefore we will focus in future works on proving SBO's correct behavior in a set of precisely defined conditions, and/or showing SBO's limitations by exhibiting new attack scenarios which have worse impact on the algorithm than the ones demonstrated in this chapter.

We believe that with sufficient care, SBO or similar algorithms can thus pave the way to next-generation blockchain platforms, in combination with Basalt, in particular in the case of collaboration systems that are not linked to cryptocurrencies, alleviating much of the limitations of current blockchain consensus protocols in terms of energy consumption, throughput limitations and limited public participation.

Chapter 7

Conclusion

The problem of building decentralized applications in the context of an open trustless network is complex due to questions related to the scale of the network: how to disseminate information with minimal communication costs, and in particular without requiring all-to-all communication, and how to ensure that information is propagated rapidly in the system, as well as questions of security: how to ensure malicious actors cannot interfere with the correct functioning of the system, and how to stave off tentatives of taking over the system with Sybil attacks. Moreover, particular care is needed to ensure that users keep a consistent view of the system.

Coordination-free protocols have proved interesting due to the performance benefits they offer: unrelated pieces of information can be processed simultaneously and independantly, possibly by different parts of the network. In this thesis, we proposed two contributions on this kind of systems: a formalization of Byzantine causal broadcast, albeit in closed systems, and an algorithm for epidemic causal broadcast in large-scale open networks but without Byzantine tolerance, using a new data structure which we called a Merkle search tree.

These coordination-free protocols are however insufficient for certain kinds of applications, such as group-based access control or smart contracts which are computationally equivalent to the problem of Byzantine consensus. To solve Byzantine consensus in the open trustless setting, current technologies employ blockchain-based consensus algorithms such as Proof-of-Work or Proof-of-Stake, which both have significant limitations in terms of energy consumption, throughput, and openness. With the two main contributions of this thesis, Basalt and SBO, we believe we have contributed significantly in alleviating these limitations by providing a solid foundation for applications based on totally ordered message delivery in very large, very open networks. Basalt and SBO work together and form a stand-alone method which allows these properties to be obtained without building a cryptocurrency network and without making use of cryptoeconomic incentives.

7.1 Future Work

Epidemic Byzantine-tolerant causal broadcast in open networks In Chapters 3 and 4, we studied causal broadcast first in presence of Byzantine nodes but in closed systems, and then in open systems but in the absence of Byzantine nodes. It would be interesting to combine these two contributions to obtain a Byzantine-tolerant causal broadcast primitive that works in open systems and in presence of churn. This leads to the following research questions: how to define the required properties of a Byzantine-tolerant causal broadcast primitive in an open system, and in particular in presence of churn? Can the definitions of BCO-broadcast proposed in Chapter 3 be adapted to this context? On the side of algorithms, could an epidemic algorithm based on anti-entropy such as the one proposed in Chapter 4 be used in presence of Byzantine nodes, and does it respect an interesting definition of BCO-broadcast? Or should one rather use algorithms based on event DAGs such as the one proposed by Sanjuan et al. (2020) or the one used in the Matrix protocol?

Combining weakly consistent and strongly consistent methods We explained in our introduction that certain kinds of applications intrinsically require a strong consistency model such as sequencial consistency, which is provided by total order broadcast. This is the case for instance of access control mechanisms. However the costs of total order broadcast are significant as it intriniscally requires all nodes in a system to handle all messages to be ordered. We however believe that in many applications, only a subset of the messages really need to be totally ordered (those that cover crucial control information such as membership changes), whereas others can simply be distributed by causal broadcast without impeding the application's functionality. Defining a framework in which decentralized applications could make use of both primitives according to their needs, allowing them to properly integrate together data coming from both sources in order to preserve some global form of consistency between them, is a future line of research we think would be interesting to explore.

Theoretical analysis of SBO The SBO algorithm that we presented in Chapter 6 is promising as it is one of the first epidemic algorithms that provide high-throughput totally ordered delivery (a previous such algorithm being EpTO (Matos et al., 2015), which does not focus on handling Byzantine faults). A core property of SBO that makes it useable in the trustless setting is its ability to support a small number of Byzantine (malicious)

actors. We were able to show in our experiments that SBO was able to tolerate 10% of malicious nodes that implemented a specific kind of attack, however this leaves open many questions to be answered about SBO, in particular regarding the formal analysis and proof of the algorithm. We discussed how SBO works on similar principles as a method that has a mathematical proof (Doerr et al., 2011), however we did not make explicit the link between the two, formally proving that the results of Doerr et al. carry over to SBO. Further, the proofs presented by Doerr et al. demonstrated that their algorithm was tolerant to $\sqrt{N}$ Byzantine nodes only, whereas our experiments seem to suggest that SBO tolerates a linear fraction of Byzantine nodes, at least on the range of system sizes for which we have gathered experimental data (Figure 6.3 shows that the behavior of SBO does not change for constant proportions of Byzantine nodes when network size scales). Finding the root cause for this discrepancy, and overall better analyzing the mathematical properties of SBO, should be the object of a future study.

SBO broadcast guilds The SBO algorithm as we presented it in Chapter 6 requires each node to act independently, executing the consensus algorithm at its own pace. This has the consequence that nodes must each reach a consensus state independently, which is not necessarily optimal in terms of convergence time. In particular when using SBO in combination with Basalt, participants will need to make use of as many IP addresses in different prefixes as possible in order to better defend against Sybil attacks, as we saw that Basalt is more and more secure as more honest nodes join the network (this has the consequence of reducing the probability of sampling an attacker's nodes, especially in the case where the IP addresses of honest nodes are well dispersed in IP address space). With the current SBO algorithm, to exploit all of the IP addresses they have at disposition a participant must run a full SBO node for each of these IP addresses, where each of these nodes must do the full query/response process at each step, which is costly in bandwidth. However these nodes all being controlled by the same participant offers the possibility for optimizing SBO by introducing stronger coordination between these nodes.

We will call a *guild* of nodes a subset of nodes that trust one another in a global trustless network. Typical examples of guilds include the set of nodes controlled by a single actor, or the nodes controlled by actors that trust one another. In the case of a guild of nodes, SBO's performance could be vastly improved by having only a single full node per guild (the guild's leader) which is responsible of doing SBO's query/response at each step. All other nodes are simple follower nodes that receive the leader's state each

time it is updated. Follower nodes participate in the peer sampling, and answer SBO queries as would full nodes, simply replying with their leader's state. In this way, guild members can exploit the full diversity of IP addresses at their disposition without having to bear the cost of the full SBO query/response algorithm at each node. This naturally makes the network more scalable as the ratio of nodes able to answer queries by number of active consensus nodes increases. This also has the advantage that decisions taken by the leader are immediately adopted by its followers, thus better propagating their opinion (on the ordering of future messages) in the network, meaning that the overall time to consensus might be decreased. Note that having a single leader in a guild means that the leader becomes a single point of failure. This can easily be avoided by having secondary fallback leaders, or by letting nodes run the entire SBO algorithm autonomously whenever the leader is unavailable.

Split blockchain: decoupling broadcast from computation As shown in a recent work on the Ethereum network (Saraph and Herlihy, 2019), the current scalability bottleneck in blockchain technology might be not so much in the broadcast layer than in the computation layer. Indeed, by increasing block size and block frequency in PoW blockchains, more transactions can be successfully delivered in the network, while retaining their property of totally ordered delivery. However, the Ethereum community has been reluctant to increase the block size too much, as another barrier is then hit by miner nodes: the time to compute the effects of transactions. Indeed, computing the effect of transactions (i.e. running the Ethereum state machine) is a costly computational process due to the involvement of many cryptographic primitives such as hash functions and digital signatures. Most Ethereum transactions take between 10 and 100 milliseconds to be calculated on modern CPUs, with many taking much more time. In addition, Saraph et al. show that there is not much room for concurrent execution of Ethereum transactions, as many have data dependency between them and must thus be resolved sequentially. The computation model of Ethereum also implies that transactions might have effects to read or write at arbitrary places in the Ethereum state, which cannot be known before executing the transaction. Overall, this makes parallelization very hard, thus intrinsically limiting the throughput of the Ethereum blockchain to a few dozen transactions per second at most.

A first step in solving this problem would consist in separating the broadcast aspect and the computation aspect in two different hash-linked chains. A first blockchain, called

the "broadcast chain", would contain only the totally ordered transactions, but not the current state of the network that results from computing the transitions corresponding to these transactions. A second chain would contain the state calculated at each block of the broadcast chain. This second chain could be built asynchronously, thus potentially alleviating the issues of contention by reporting the computations in high-contention periods to later periods of lower contention. This also means that the computation model can better be studied independently of the broadcast layer, in order to find models with more opportunities for parallelism. In the case of a guild of nodes running SBO and Basalt, such an architecture would also have the effect that the guild could have only a single powerful node to do the computation, while all nodes still participate independently as full broadcast nodes by running the SBO algorithm.

Sharding and validation guilds Existing blockchain platforms process transactions one after the other, and as we saw, there is in practice little opportunity for parallelization (Saraph and Herlihy, 2019). This limitation is exacerbated by single smart contracts concentrating a large proportion of the transaction in a block: this means that these transactions necessarily have data dependencies on one another. On the Ethereum blockchain, the Cryptokitties smart contract has had exactly this effect during the recent years, with a peak in 2017-2018 that caused significant congestion on the Ethereum network.

To alleviate this issues, the Ethereum blockchain is set on transitioning to a sharded model, where the blockchain is divided in 64 sub-blockchains, called "shards", for which consensus is obtained independently and whose computations are done on separate nodes. Each shard can therefore handle the same transaction volume as the entire Ethereum blockchain today. A special mechanism is defined to allow smart contracts in different shards to communicate in order to exchange assets. This model has the inconvenient that the shards must be coordinated a-posteriori in a trustless fashion. What we suggest instead is an architecture that exploits a separation between broadcast and computation, with computation guilds to compute the results of transactions in all shards. A single blockchain could be used to broadcast transactions for all shards, ensuring a consistent set of inputs to the computation nodes. Once these inputs are determined, the inputs are split by shard and each shard is set to a different node in the guild. Once all of the guild nodes have reported their computation result, the resulting global blockchain is deduced and stored on the computation chain. Such an architecture is only possible thanks to the trust relationships within a guild. Note that in such an architecture, guilds still do

not need to have trust relationships between them. The architectures of guild in which local trust relationships exist is already in place with mining pools on the Bitcoin and Ethereum networks for instance, where miners put their computing power in common and trust a central party to fairly redistribute the rewards to all pool participants. In our guild validation architecture, in the case where guild nodes are not fully trusted, the computation of each shard could be done by multiple nodes, so that a node giving an incorrect result could be detected. Since blockchain computations are ran on Merkle data structure, it is possible to provide a proof that a computation was ran correctly when that is the case: a computation on a blockchain is deterministic given its inputs (a set of key/value pairs it reads in the previous state), and produces a deterministic output (a set of key/value pairs it writes in the next state), therefore the Merkle inclusion proofs for the key/value pairs the computation reads from and writes to is effectively a proof of the correct execution of a transaction. It is thus easy for guild members to confirm that a node has been misbehaving, and to exclude that node from the guild when such misbehavior is detected.

Including Basalt-verified properties in the SBO blockchain It might be of interest for blockchain application developers to be able to rely on the sampling properties of Basalt and the identity verification mechanism it provides to implement identity-based properties in smart contracts. This requires a mechanism in which the identity of participants can be securely ratified in the blockchain. Suppose for instance that we are interested in ratifying observations of the following kind: "at time t, the node that runs on the IP address xx.xx.xx.xx has the public key k". This property can be verified by each node individually by contacting said node on its IP address, however this does not make it a consensus in the network. To build such a consensus, and ratify this observation in the blockchain, the following mechanism could be used: once a node has observed this property, it could emit an operation whose role is to register that observation in the blockchain. Other nodes running SBO would identify that operation as an unconfirmed information that they must verify by themselves. They will thus try to contact the target node by themselves to verify its public key indeed corresponds to the one announced. Only once they have been able to confirm it, they will accept to take the ratify operation in their ordering window for consensus. If they are not able to verify it, they will reject that operation and it will not be confirmed by the network.

7.2 Outlook

Replacing centralized systems such as those of Google, Facebook, Microsoft, etc. by decentralized alternatives, that are respectful of users and concerned with the public good instead of their own self-interest, is a crucial step in making the Internet a more free and democratic space. The potential of the Internet to change society for the better is immense, as a tool for communication and collaboration, helping citizens to become better educated and obtain valuable information necessary to the democratic decision process on questions of public interest. These benefits have been appropriated, privatized and monopolized by the GAFAM in order to replace them with commercial advertising as a profit-generating tool in their own interest. We have argued in this thesis that this monopoly must end, and we are taking steps to end it.

We hope that our work will have impact on the future developments in the domain of decentralized applications. Our work, however, only touches limited aspects of this emerging domain. New technologies (protocols, frameworks, etc) for decentralized applications are being created every day to try to solve the challenges posed by the necessity of making decentralized applications work, in particular to tackle the challenging constraints of trustlessness. These include blockchain-based technologies, as well as a variety of technologies that do not rely on blockchains and try to find alternative ways of providing similar or better levels of functionality to the end user with lighter constraints on network nodes (by allowing nodes to only handle a small portion of the overall network's content, a property not easily achieved with blockchain technologies). In this domain, we are particularly interested in following the advances on the Matrix and Secure Scuttlebutt protocols and their descendants.

Appendix A

Two Signature-free Multi-shot BR-broadcast Algorithms

To complete the presentation of Byzantine causal broadcast (Chapter 3), this section presents multi-shot extensions of the one-shot signature-free BR-broadcast algorithms introduced by Bracha (1987) and Imbs and Raynal (2016). The presentation is inspired by pages 64-71 of Raynal (2018), where the reader can also find proofs of the original single-shot algorithms. In the text of these two extensions, sn denotes the sequence number of the corresponding BR-broadcast instance, hence a process invokes br_broadcast(sn, m).

A.1 Underlying basic communication system

Both the algorithms described below, the processes communicate by exchanging messages through an asynchronous reliable point-to-point network. "Asynchronous" means that a message that has been sent is eventually received by its destination process, i.e., there is no bound on message transfer delays. "Reliable" means that the network does not loose, duplicate, modify, or create messages. "Point-to-point" means that there is a bi-directional communication channel between each pair of processes.

A process p_i sends a message to a process p_j by invoking the primitive "send TAG(m) to p_j", where TAG is the type of the message and m its content. To simplify the presentation, it is assumed that a process can send messages to itself. A process receives a message by executing the primitive "receive()". The macro-operation "broadcast TAG(m)" is a shortcut for "**for** $j \in \{1, \cdots, N\}$ **do** send TAG(m) to p_j **end for**".

A.2 Multi-shot version of Bracha's BR-broadcast algorithm

This algorithm assumes $t < N/3$. When, on its client side, a process p_i invokes br_broadcast(sn, m), it invoke the macro-operation broadcast() with the protocol message INIT(sn, m) (line 1).

Algorithm A.1: Multi-shot version of Bracha's BR-broadcast algorithm ($t < N/3$, code for p_i)

operation br_broadcast(sn, m) **is**
(1) broadcast INIT(sn, m).

when a message INIT(sn, m) **is received from** p_j **do**
(2) discard the message if it is not the first message INIT$(sn, -)$ from p_j;
(3) broadcast ECHO$(\langle j, sn\rangle, m)$.

when a message ECHO$(\langle j, sn\rangle, m)$ **is received from any process do**
(4) **if** (ECHO$(\langle j, sn\rangle, m)$ received from strictly more than $\frac{N+t}{2}$ different processes)
$\wedge$ (READY$(\langle j, sn\rangle, m)$ not yet broadcast)
(5) **then** broadcast READY$(\langle j, sn\rangle, m)$
(6) **end if**.

when a message READY$(\langle j, sn\rangle, m)$ **is received from any process do**
(7) **if** (READY$(\langle j, sn\rangle, m)$ received from at least $(t + 1)$ different processes)
$\wedge$ (READY$(\langle j, sn\rangle, m)$ not yet broadcast)
(8) **then** broadcast READY$(\langle j, sn\rangle, m)$
(9) **end if**;
(10) **if** (READY$(\langle j, sn\rangle, m)$ received from at least $(2t + 1)$ different processes)
$\wedge$ $(\langle j, sn\rangle, m)$ not yet br_delivered from p_j)
(11) **then** br_delivery of (sn, m) **from** p_j
(12) **end if**.

On it server side a process p_i may receive three different types of protocol messages: INIT(), ECHO(), and READY(). A message INIT carries an application message, while the messages ECHO() and READY() carry a process identity and an application message [1].

1. The fact that the ECHO() and READY() messages carry a process identity makes redundant the use

— When p_i receives INIT(sn, m) for the first time from a process p_j (line 2), it broadcasts the protocol message ECHO$(\langle j, sn\rangle, m)$ (line 3). If this message is not the first message INIT$(sn, -)$ from p_j, p_i discards it (in this case, p_j is Byzantine).

— When p_i receives the protocol ECHO$(\langle j, sn\rangle, m)$ for any process, it broadcasts the protocol message READY$(\langle j, sn\rangle, m)$ (line 5) if it received ECHO$(\langle j, sn\rangle, m)$ from enough different processes (where "enough" means here more than $\frac{N+t}{2}$), and READY$(\langle j, sn\rangle, m)$ has not yet been broadcast (line 4). This message exchange ensures that no two correct processes will br-deliver different message from p_j with the sequence number sn, but it is still possible that a correct process br-delivers m from p_j while another correct process does not br-deliver a message from p_j. The role of the message READY$(\langle j, sn\rangle, m)$ is to prevent a correct process from blocking on the br-delivery of m.

— When p_i receives READY$(\langle j, sn\rangle, m)$ for any process, it does the following.

  — Process p_i first broadcasts READY$(\langle j, sn\rangle, m)$ (line 8) if (i) not already done and (ii) it received READY$(\langle j, sn\rangle, m)$ from "enough" processes (where "enough" means here $(t + 1)$ processes, which means from at least on correct process, line 7). As previously indicated, this allows other correct processes not to deadlock.

  — Then, if p_i received READY$(\langle j, sn\rangle, m)$ from "enough" processes (where "enough" means here $((2t + 1)$, which means from at least $(t + 1)$ correct processes), it locally br-delivers the pair (sn, m) (from p_j), if not yet already done (lines 10-11).

This algorithm is optimal with respect to t-resilience (namely $t < N/3$). It requires three consecutive communication steps, and $(N-1)+2N(N-1) = 2N^2-N-1$ protocol messages. The proof of this algorithm relies on the following properties, which assume $n > 3t$ (see Raynal (2018) for their proofs):

— $N - t > \frac{N+t}{2}$.

— Any set containing more than $\frac{N+t}{2}$ different processes, contains at least $(t + 1)$ correct processes.

— Any two sets of processes Q_1 and Q_2 of size at least $\lfloor \frac{N+t}{2} \rfloor + 1$ have at least one correct process in their intersection.

of an identity in the pair $\langle -, sn\rangle$ that appear in the messages that are br-broadcast. (Hence Algorithm 3.1 can be modified accordingly.)

A.3 Multi-shot version of Imbs-Raynal's BR-broadcast algorithms

This algorithm assumes $t < N/5$. The code of br_broadcast(sn, m) is the same as in the previous algorithm.

Algorithm A.2: Multi-shot version of Imbs-Raynal's BR-broadcast algorithm

($t < N/5$, code for p_i)

```
operation br_broadcast(sn, m) is
(1)  broadcast INIT(sn, m).

when INIT(sn, m) is received from p_j do
(2)  discard the message if it is not the first message INIT(sn, −) from p_j;
(3)  broadcast WITNESS(⟨i, sn⟩, m).

when WITNESS(⟨j, sn⟩, m) is received from any process do
(4)  if (WITNESS(⟨j, sn⟩, m) received from (N − 2t) different processes
       ∧ WITNESS(⟨i, sn⟩, m) not yet broadcast)
(5)    then broadcast WITNESS(⟨j, sn⟩, m)
(6)  end if;
(7)  if (WITNESS(⟨j, sn⟩, m) received from (N − t) different processes
       ∧ (sn, m) not yet br_delivered from p_j)
(8)    then br_delivery of (sn, j) from p_j
(9)  end if.
```

On its server side a process p_i may receive two different types of protocol messages: INIT() and WITNESS(). The processing of INIT(sn, m) is similar to the one of Algorithm A.1. Process p_i simply broadcasts the message WITNESS$(\langle j, sn\rangle, m)$ if it is the first time it received from p_j a message INIT$(sn, -)$ (line 3). Then, when it receives a message WITNESS$(\langle j, sn\rangle, m)$ p_i does the following.

- If it received the same message WITNESS$(\langle j, sn\rangle, m)$ from "enough" processes (where "enough" means here $N - 2t$), and it has not yet broadcast this message (line 4), it does it (line 5).
- If it received WITNESS$(\langle j, sn\rangle, m)$ from "more" processes (where "more" means here $N - t$), and it has not yet br-delivered the pair (sn, m) from p_j (line 7), it br-delivers it (line 8).

Let us notice that, as $t < N/5$, we have $N - 2t > 3t$, which means that, in this case, (WITNESS(j, m) was broadcast by at least $N - 3t \geq 2t + 1$ correct processes. Then, if it received WITNESS(j, m) from more different processes, where "more" means" $(N - t)$, p_i locally br-delivers m from p_j.

As we can easily see, this algorithm requires two communication steps and $N^2 - 1$ protocol messages. This better efficiency with respect to Bracha's algorithm is obtained at the price of a weaker t-resilience, namely $t < N/5$.

Appendix B

Résumé en français

Internet est un outil formidable pour l'éducation, la communication et la collaboration, dont les usages ont été révolutionnés dans les dernières décennies. Nous pouvons désormais communiquer instantanément d'un bout à l'autre du globe, et ce à tout moment et où qu'on soit grâce aux smartphones, ce qui nous a permis de développer de nouvelles façons de nous éduquer, de collaborer et de socialiser.

Malheureusement, les usages majoritaires d'Internet passent aujourd'hui tous par des plateformes centralisées immenses, contrôlées par une poignée de multinationales telles que Google, Facebook, Amazon, Apple ou encore Microsoft (nous appelons ces plateformes les « GAFAM »). Cette monopolisation par des plateformes privées a de nombreuses conséquences néfastes pour les utilisateurs, dues au fait que ces plateformes sont pensées et développées avant tout dans leur intérêt propre et dans celui de leurs actionnaires, et non dans celui des utilisateurs. De manière non exhaustive, nous pouvons citer :

— la structuration des contenus dans un but commercial, avec l'introduction de publicités dès que possible et la mise en valeur de contenus ayant une marque vendable, ce qui n'est pas adapté à un contexte de crise écologique notoirement amplifiée par la consommation de masse et la surconsommation ;
— les questions de vie privée liée à l'utilisation des données et traces laissées par les utilisateurs sur les plateformes, pouvant être utilisées notamment à des fins de manipulation commerciale ou politique, mais également pour la surveillance d'État ;
— l'enfermement des utilisateurs dans un « jardin clos » (« walled garden » en anglais), rendant impossible la communication avec des utilisateurs en dehors de ces plateformes et rendant difficile aux utilisateurs la portabilité de leurs données d'une plateforme à une autre ;
— l'application de critères de modération et d'exclusion opaques sur les contenus et les utilisateurs, anéantissant la possibilité d'une gestion démocratique et collaborative par les utilisateurs des types de contenus qu'il souhaitent promouvoir ou au

contraire rejeter ;

— l'application de critères de modération automatisés, dans une perspective de réduction des coûts qui conduit à la suppression d'un possible « service après-vente » qui mettrait des interlocuteurs humains à disposition des utilisateurs, faisant que les producteurs de contenu n'ont parfois aucun recours pour contester une décision d'exclusion d'une de ces plateformes, ce qui peut avoir des conséquences graves par exemple lorsqu'ils en sont dépendants pour les revenus qu'ils en tirent.

Étant donné l'impact massif de ces nouvelles technologies sur nos vies modernes, il nous semble dangereux de laisser autant de pouvoir à ces compagnies multinationales. Au contraire, Internet étant un lieu d'échange public où chacun doit pouvoir trouver sa place afin d'en tirer un bénéfice personnel et collectif, nous défendons l'idée que les plateformes numériques doivent être dirigées de façon démocratique, avec des règles de gestion inspirées par celles proposées par Elinor Ostrom pour la gouvernance des communs (Ostrom, 1990). Afin de chercher des solutions à ces problématiques, nous nous orientons vers la conception d'architectures techniques alternatives ayant pour propriété fondamentale la *décentralisation*. Celle-ci assure que le contrôle du système n'est jamais aux mains d'un seul acteur en position dominante, mais d'une multiplicité d'acteurs pouvant agir en leur intérêt propre et en celui de leurs utilisateurs directs, et que la collaboration entre ces acteurs est possible dans un cadre technique unifié et avec une possibilité de discussion et de prise de décision démocratique.

B.1 Présentation des architectures décentralisées

B.1.1 Les réseaux fédérés et leurs limites

Une première méthode pour la conception de réseaux décentralisés consiste en la création de plateformes *fédérées*. Dans une *fédération*, plusieurs serveurs individuels sont configurés afin de fournir un service de communication à un ensemble d'utilisateur locaux à chaque serveur. Les serveurs disposent ensuite d'un protocole permettant l'échange d'informations entre eux, afin de rendre visible les utilisateurs d'un serveur A et les contenus qu'ils produisent sur un serveur B. Un serveur dans le cadre d'un réseau fédéré est généralement appelé une *instance*. Dans ce système, chaque instance retient son autonomie propre, et peut choisir par exemple la manière dont les contenus sont présentés et mis en page, ou peut choisir d'appliquer des critères de promotion et de modération du contenu

spécifiquement appropriés aux besoins de la communauté d'utilisateurs représentée par cette instance.

Malheureusement, les réseaux fédérés soufrent de quelques problèmes qui rendent leur usage parfois peu pratique et/ou risqué dans certaines situation :

— **Une tendance à la centralisation sur de très grosses instances.** En effet, la plupart des utilisateurs souhaitant rejoindre le réseaux ne connaissent pas une instance particulière adaptée à leurs besoin, voire même ne comprennent pas le concept de la fédération de serveurs. Cela fait qu'ils auront tendance à s'inscrire en grand nombre sur une instance majoritaire ayant une grande visibilité, et les autres instances resteront nécessairement limitées à un ensemble plus restreint d'utilisateurs enthousiastes et engagés. Les risques de la centralisation sont donc également présents sur ces plateformes.

— **Une vulnérabilité accrue aux pannes du réseau.** En effet, les petites instances sont souvent gérés par une seule personne ayant un temps et des ressources limités pour l'administrer correctement. Cela signifie que ces petits serveurs peuvent à tout moment tomber en panne ou disparaître totalement du réseau. En d'autres termes, la fiabilité du réseau dépend de la disponibilité à tout instant de tous les administrateurs de serveurs, sans quoi une partie des utilisateurs est toujours à risque d'être déconnectée.

B.1.2 Les principes de la décentralisation *trustless*

Afin de palier à ces problèmes avec les réseaux fédérés, il convient d'abord d'observer que ceux-ci reproduisent en fait la même structure *technique* que les réseaux centralisés, avec pour simple ajout l'existence d'un protocole de communication pour le partage des données entre les serveurs. Cette structure technique se définit par le fait qu'un serveur reste une entité *centralisée*, fortement identifiable (par son nom de domaine), et qui a le contrôle absolu sur les données des utilisateurs qui en dépendent. Cela signifie que pour interagir avec le réseau, les utilisateurs n'ont d'autre choix que de contacter leur serveur désigné et de lui faire confiance sur tous les aspects : authentification des utilisateurs, stockage des contenus, implémentation de règles d'accès, etc.

Pour sortir de cette conception centralisée, nous proposons à la place de se baser sur des principes dits *« sans confiance »* ou *« trustless »*. Dans un système sans confiance, le contrôle sur l'authentification et la transmission des informations dans le réseau n'est pas effectué par des serveurs agissant en leur nom propre, mais par des mécanismes de

vérification distribués ayant pour base les principes de la cryptographie asymétrique. Par exemple, identifier un contenu comme ayant été produit par un certain utilisateur ne se fait plus en interrogeant le serveur de cet utilisateur qui héberge le contenu, mais en vérifiant une signature cryptographique qui accompagne le contenu en question et qui est produite directement par l'utilisateur lui-même. Cela a pour conséquence notamment que n'importe quelle machine est en mesure de stocker et de relayer ce contenu pour les clients : en effet, ceux-ci ne sont plus dépendant du fait de contacter un serveur précis pour vérifier l'authenticité du contenu, ils peuvent à la place simplement vérifier la signature cryptographique peu importe d'où l'information a été reçue.

L'utilisation de tels principes permet d'implémenter facilement les fonctionnalités suivantes :

— **Redondance et tolérance aux pannes :** puisque n'importe quelle machine est en mesure de fournir n'importe quel contenu aux utilisateurs, un contenu peut être trivialement répliqué sur de nouvelles machines sans autorisation préalable. Lorsqu'une redondance de ce type est correctement implémentée, les serveurs ne sont plus individuellement indispensables : un serveur en panne ne pose pas de problème puisque les contenus qu'il héberge peuvent également être récupérés depuis d'autres serveurs.
— **Scalabilité horizontale des lectures :** de même, si un contenu fait un *buzz* et est très demandé à un instant t, celui-ci peut être répliqué de manière transparente et automatique sur un grand nombre de machines, afin de le mettre plus rapidement à disposition de tout le monde. C'est par exemple comme cela que fonctionne le protocole BitTorrent, où après avoir téléchargé un fichier, toute machine devient automatiquement un *« seeder »* depuis laquelle des nouveaux utilisateurs peuvent télécharger le fichier à leur tour.
— **Migration facile des profils utilisateurs :** les données produites par un certain utilisateur ne sont plus liées à un serveur en particulier, puisque leur validité est maintenant attestée par des signatures cryptographiques pouvant être relayées par n'importe quelle machine. Cela signifie donc que les profils utilisateurs ne sont pas liés à un seul serveur : ils sont au contraire intrinsèquement *nomades*, et peuvent passer facilement d'un serveur à un autre selon les besoins.

B.1.3 Les architectures pair-à-pair trustless

L'implémentation d'architectures trustless pose néanmoins un certain nombre de problèmes, notamment en termes de coordination entre les différents serveurs. En effet, dans le cas d'objets collaboratifs comme un salon de discussion ou un document partagé, plusieurs utilisateurs seront amenés à faire des apports individuels, qu'ils signeront avec leur clef cryptographique personnelle. Néanmoins pour avoir une vision de l'objet final, il convient de combiner ces apports individuels selon des règles bien définies, afin que tous les utilisateurs aient une vision cohérente. Ces règles de combinaison doivent également permettre l'implémentation de mécanismes de contrôle d'accès, afin de limiter l'écriture aux utilisateurs autorisés, et de permettre l'exclusion d'utilisateurs du système lorsque cela s'avère nécessaire.

Dans les cas les plus favorables, l'union des contributions des utilisateurs individuels peut être récupérée par chaque serveur et traitée localement, en s'appuyant uniquement sur la signature présente dans chaque contribution pour vérifier son origine. C'est par exemple le cas des réseaux ZeroNet et Scuttlebutt, qui fonctionnent selon ce mécanisme. Le réseau Matrix implémente un mécanisme similaire mais plus évolué : en s'appuyant sur des relations de causalité entre les contributions des utilisateurs, un algorithme de résolution de conflits peut être appliqué par exemple pour déterminer si un message envoyé par un utilisateur ayant été banni du système doit être ignoré. Dans tous les cas, ces systèmes ont la propriété cruciale de fonctionner *sans coordination* : cela signifie que les nœuds (les machines des utilisateurs ou les serveurs) peuvent produire des messages correspondant aux contributions des utilisateurs, et les traiter immédiatement en local. Par la suite et de manière asynchrone, ces contributions sont diffusées aux autres nœuds du système. Peu importe l'ordre dans lesquelles ces contributions sont propagées, les autres nœuds sont capable d'appliquer les mêmes règles de combinaison afin d'obtenir un résultat final identique. On parle alors de *cohérence à terme* : après propagation de tous les messages à tous les nœuds, et indépendamment de l'ordre dans lequel ceux-ci sont reçus, les nœuds peuvent effectuer un calcul déterministe qui donne le même résultat pour l'état de l'objet partagé.

Néanmoins, certains types d'objets partagés ne peuvent pas être implémentés d'une manière aussi simple. Nous prenons l'exemple (voir texte en anglais) d'un groupe d'utilisateurs avec contrôle d'accès : le fait que deux utilisateurs puissent tous deux demander l'exclusion l'un de l'autre fait qu'en cas de conflit, le réseau n'est pas capable de choisir de manière fiable l'un ou l'autre à garder ou à exclure. À ce titre, nous montrons que cet objet

« contrôle d'accès » a en fait un *numéro de consensus* infini, c'est-à-dire que son implémentation nécessite l'utilisation d'une primitive plus puissante en termes computationnels que la simple diffusion asynchrone de messages entre des nœuds. Cette primitive, nommée *consensus*, permet de décider d'un ordre total entre les messages, partagé d'un commun accord par tous les nœuds. L'utilisation de cet ordre total permet de résoudre de façon déterministe les situations de conflit. De manière générale, la présence d'un ordre total permet de réduire l'implémentation d'un objet distribué à la confection d'une machine à état qui le représente par des transitions séquentielles déterministes, dont les entrées sont les messages totalement ordonnés, et dont le calcul de l'état de façon déterministe par tous les nœuds leur permet de conserver une vision cohérente de l'objet. On parle alors de *cohérence forte*, ou plus précisément de *cohérence séquentielle*.

L'implémentation de cette primitive de *consensus* a fait l'objet de nombreux travaux dans des réseaux fermés, composés de N nœuds bien identifiés. On sait notamment implémenter le consensus dans un tel réseau avec passage de message asynchrone en présence de jusqu'à $t < N/3$ nœuds malicieux (dits *Byzantins*), qui ne respectent pas le protocole et tentent de le faire échouer. Ces protocoles ne sont néanmoins pas adaptés au contexte des applications décentralisées sur Internet, où il est attendu que n'importe quel machine puisse rejoindre le réseau et participer à tout moment, selon les besoins. Par ailleurs les applications décentralisées présentent un problème de *passage à l'échelle* : on peut s'attendre à ce que des milliers de nœuds participent à ces réseaux s'ils deviennent populaire, ce qui poserait un problème pour les algorithmes de consensus susmentionnés, qui ont un coût de communication quadratique en le nombre de nœuds.

Dans le contexte du réseau public Internet, et plus particulièrement pour la création de *cryptomonnaies*, de nouveaux protocoles de consensus ont émergé, basés sur la technologie dite de la *blockchain* (chaîne de blocs en Français). Ces protocoles utilisent notamment des mécanismes de consensus distribué où une « puissance de vote » est attribuée à tous les nœuds du réseau. Afin d'éviter la monopolisation par une entité, la puissance de vote doit être difficile à obtenir. Dans le premier réseau de ce type, Bitcoin, la puissance de vote est définie en rapport avec la puissance de calcul dont chacun dispose : par le calcul répété d'une opération de hachage, les nœuds sont en mesure de prouver qu'ils ont dépensé un certain temps de calcul sur la blockchain, ce qui permet de leur octroyer une puissance de vote en proportion (mécanisme dit *Proof-of-Work*). Le mécanisme Proof-of-Work est néanmoins très coûteux en énergie, d'autant plus que Bitcoin attribue des récompenses aux utilisateurs qui dépensent de la puissance de calcul (les mineurs), ce qui a pour

conséquence perverse de les encourager à consommer le plus d'électricité possible pour obtenir ces récompenses. Dans un contexte de crise écologique où l'énergie est la cause principale de nos émissions de CO_2, cela ne nous semble pas tolérable. Afin de pallier à cet échec critique de la technologie Proof-of-Work, d'autres solutions ont été proposées, dont notamment *Proof-of-Stake*, qui promet d'attribuer une puissance de vote à chacun en fonction de l'argent qu'il possède et qu'il investit à titre d'*enjeu* (*stake*) dans le réseau. Néanmoins, on voit bien qu'un tel système ne permet pas une participation démocratique, ni même ouverte à tous, puisque seuls les utilisateurs disposant d'une masse monétaire suffisante peuvent jouer un rôle actif dans le consensus. Par ailleurs, nous soutenons qu'il n'est pas souhaitable d'adosser un algorithme de consensus à un système monétaire, non seulement pour des raisons de complexité d'implémentation, mais également car cela limite la possibilité d'utiliser cet algorithme dans des situations de collaboration non monétisée, ou pour créer des univers monétaires alternatifs.

B.2 Contributions de cette thèse

Dans cette thèse, nous proposons de nouvelles contributions pour faciliter le développement et l'analyse théorique des applications décentralisées dans le contexte des réseaux publics large échelle et sans confiance.

B.2.1 Diffusion causale tolérante aux nœuds Byzantins

Bien que cette thèse se concentre principalement sur la construction de la primitive de diffusion totalement ordonnée (le consensus), l'étude des propriétés de primitives plus faibles en présence de nœuds Byzantins reste très importante. En effet, un certain nombre d'approches dont le réseau Matrix utilisent une simple diffusion causale (encodée dans un graphe acyclique des événements), ce qui leur permet malgré tout d'implémenter des applications décentralisées complexes (en l'occurrence, un système de messagerie instantanée). À ce titre, elles représentent un espoir important pour le développement d'applications décentralisées sans avoir à subir le coût associé à la construction d'un ordre total.

L'abstraction de diffusion causale s'assure que deux messages qui sont liés par la relation de causalité de Lamport sont reçus par tous les nœuds dans un ordre compatible avec cette causalité. Par exemple dans le cas d'un réseau social, cela permet de s'assurer qu'un utilisateur verra le message d'Alice avant de voir la réponse que Bob y apporte.

Plusieurs algorithmes de diffusion causale existent dans les systèmes distribués sans fautes ou ne subissant que des pannes et non des comportements malicieux.

Notre Chapitre 3 est dédié à l'étude de la diffusion causale en présence de nœuds Byzantins dans des systèmes fermés de n nœuds dont t Byzantins. Bien que ce contexte soit très restrictif en comparaison avec celui de l'Internet public dans lequel fonctionnent les technologies de blockchain, cette étude n'avait à notre connaissance pas encore était mené, et constitue une première étape d'analyse qui pourra servir à une meilleure compréhension de protocoles similaires dans le contexte Internet (comme par exemple Matrix). Nous commençons par donner une définition formelle de la diffusion causale en présence de nœuds Byzantins, que nous appelons BCO-broadcast. Cette définition est donnée sous la forme de deux caractérisations équivalentes. Nous présentons ensuite un algorithme qui l'implémente dans un système à passage de messages asynchrones, algorithme dont nous prouvons qu'il est correct.

B.2.2 Arbres de recherche de Merkle

En continuation de notre étude de la diffusion causale, notre Chapitre 4 étudie l'implémentation de cette primitive sous la forme d'un « event store » causalement cohérent, que l'on peut facilement construire dans un réseau ouvert en présence de « churn » (l'existence de nœuds qui quittent ou rejoignent le réseau à tout moment) et en l'absence de nœuds Byzantins. Pour ce faire, nous proposons une nouvelle structure de données pour l'anti-entropie que nous appelons un arbre de recherche de Merkle. Nous montrons que cette structure de donnée peut être utilisée pour améliorer les performances de l'échange de données en anti-entropie dans un protocole épidémique de dissémination de rumeur. Nous comparons cette approche avec une autre approche basée sur les vecteurs d'horloge (Renesse et al., 2008), ainsi qu'avec une approche basée sur un arbre de Merkle qui ne préserve pas l'ordre des éléments insérés. Nous montrons que dans des grands réseaux et en présence d'un faible taux de messages envoyés par unité de temps, la méthode à base d'arbre de recherches de Merkle donne le meilleur compromis : une réduction de 66% de la bande passante utilisée, ainsi qu'une amélioration de 34% sur notre mesure de l'état de cohérence du réseau, et une amélioration de 32% sur le 99e percentile de délai de livraison des messages. En comparaison avec les arbres de Merkle sans ordre, les arbres de recherche de Merkle sont également meilleurs sur les trois métriques. Notons qu'en plus d'être utiles pour la diffusion causale, les arbres de recherche de Merkle peuvent également être utilisés dans le contexte des blockchains dans des réseaux large échelle sans confiance

et en présence de nœuds malicieux.

B.2.3 Basalt : échantillonnage de pairs résistant aux attaques Sybil

Nous revenons ensuite sur l'idée de construire une abstraction de diffusion totalement ordonnée (équivalente au consensus) à l'aide de blockchains. Nous nous posons notamment la question de comment construire un algorithme de consensus efficace qui n'ait pas les défauts de Proof-of-Work et Proof-of-Stake. Nous envisageons les algorithmes épidémiques comme un paradigme alternatif qui promet de résoudre ces problèmes.

Les algorithmes épidémiques dépendent néanmoins d'un échantillonneur de pairs sécurisé, qui fournit des échantillons de bonne qualité, dans le sens suivant : la proportion de nœuds Byzantins dans les échantillons doit être la plus faible possible, et la diversité des pairs échantillonnés doit simultanément être maximisée pour optimiser la connectivité du réseau et donc la vitesse de diffusion des informations.

Dans le Chapitre 5, nous proposons un nouvel échantillonneur de pairs avec un design de sécurité radicalement différent, qui utilise des propriétés de la distribution des adresses IP des nœuds pour résister aux attaques Sybil, et non le Proof-of-Stake comme le fait par exemple le réseau AVA. Nous proposons un nouvel algorithme appelé Basalt, qui implémente ce nouveau design avec une méthode de *recherche chaotique tenace*, afin d'empêcher les tentatives des nœuds Byzantins de devenir sur-représentés. Nous montrons à l'aide d'une étude théorique et de simulations de Monte Carlo que Basalt donne des échantillons qui sont très proches de la distribution optimale que l'on pourrait en espérer, y compris dans des scénarios adverses tels qu'une tentative d'attaque Eclipse. Une expérience sur une plateforme réelle confirme que Basalt produit des échantillons distribués équitablement entre les nœuds, ce qui donne un système réellement ouvert à la participation publique et dans lequel aucune entité unique ne peut gagner une puissance excessive.

B.2.4 SBO : consensus Byzantin épidémique à haut débit

Les algorithmes de consensus épidémiques restent un domaine de recherche émergent, avec seulement quelques méthodes connues (Doerr et al., 2011 ; Matos et al., 2015 ; Team Rocket, 2018) et, à notre connaissance, aucune qui ne permette une livraison à haut débit des messages en présence de nœuds Byzantins.

Dans le Chapitre 6, nous proposons le premier algorithme de consensus tolérant aux nœuds Byzantins à haut débit, que nous appelons SBO. SBO est basé sur la diffusion épidémique et sur une nouvelle technique d'ordonnancement basée sur un calcul de médianes qui permet à un grand nombre de messages d'être traités simultanément dans le réseau. Les nœuds du réseau interrogent répétitivement leurs pairs pour connaître l'ordre qu'ils prédisent pour les messages à délivrer, et adaptent à leur tour leur propre estimation de cet ordre pour se rapprocher des informations obtenues. Les nœuds gardent une trace du niveau d'accord globalement observé dans le réseau sur cet ordre, et une fois qu'un niveau d'accord fort est observé, les nœuds valident les messages qui ont une position stable.

Étant donné la complexité de l'algorithme SBO, il est difficile de l'évaluer entièrement à l'aide d'un modèle mathématique pour en prouver qu'il est correct. Nous nous concentrons donc plutôt sur une évaluation à base de simulations et d'un déploiement en conditions réelles. Notre évaluation montre que SBO peut passer à l'échelle en supportant jusqu'à 10 000 nœuds actifs en simulation, et qu'en conditions réelles il permet d'atteindre un débit de 5000 transactions par seconde et une latence de moins de 30 secondes en utilisant simplement une connexion Internet 100 Mbps facilement accessible au grand public, et ce dans un scénario de déploiement planétaire avec 1000 nœuds dispersés dans 20 grandes villes autours du globe.

Bibliography

Afek, Yehuda, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit (1993), « Atomic snapshots of shared memory », *Journal of the ACM (JACM)* 40.*4*, pp. 873–890, DOI: 10.1145/153724.153741.

Akram, Rida, Natasha, Shah Fahad, Muhammad Zaffar Hashmi, Abdul Wahid, Muhammad Adnan, Muhammad Mubeen, Naeem Khan, Muhammad Ishaq Asif Rehmani, Muhammadd Awais, Mazhar Abbas, Khurram Shahzad, Shakeel Ahmad, Hafiz Mohkum Hammad, and Wajid Nasim (2019), « Trends of electronic waste pollution and its impact on the global environment and ecosystem », *Environmental Science and Pollution Research* 26.*17*, pp. 16923–16938, ISSN: 1614-7499, DOI: 10.1007/s11356-019-04998-2.

Alizart, Mark (2019), « 9. L'appropriation des moyens de production monétaire », vol. Cryptocommunisme, Presses Universitaires de France, pp. 85–95, ISBN: 9782130814597.

Almeida, Paulo Sérgio, Ali Shoker, and Carlos Baquero (2014), « Efficient State-based CRDTs by Delta-Mutation », DOI: 10.1007/978-3-319-26850-7_5, arXiv: 1410.2803.

Anceaume, Emmanuelle, Yann Busnel, and Sébastien Gambs (2013a), « On the Power of the Adversary to Solve the Node Sampling Problem », *Trans. Large Scale Data Knowl. Centered Syst.* 11, pp. 102–126, DOI: 10.1007/978-3-642-45269-7_5.

Anceaume, Emmanuelle, Yann Busnel, and Bruno Sericola (2013b), « Uniform node sampling service robust against collusions of malicious nodes », *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE Computer Society, pp. 1–12, DOI: 10.1109/DSN.2013.6575363.

Andrychowicz, Marcin and Stefan Dziembowski (2015), « PoW-based Distributed Cryptography with No Trusted Setup », *Annual Cryptology Conference*, Springer, pp. 379–399.

Apostolaki, Maria., Marti Gian, Müller Jan, and Vanbever Laurent (2019), « SABRE: Protecting Bitcoin against Routing Attacks. », *NDSS*, pp. 1–15, DOI: 10.14722/ndss.2019.23252.

Ariès, Paul (2005), *Décroissance ou barbarie*, Golias Lyon.

Attiya, Hagit, Faith Ellen, and Adam Morrison (2016), « Limitations of highly-available eventually-consistent data stores », *IEEE transactions on parallel and distributed systems* 28.*1*, pp. 141–155, DOI: 10.1145/2767386.2767419.

Auvolat, Alex (2019), « Making Federated Networks More Distributed », *38th Symposium on Reliable Distributed Systems (SRDS)*, IEEE, pp. 383–384, DOI: 10.1109/SRDS47363.2019.00058.

Auvolat, Alex, Yérom-David Bromberg, Davide Frey, and François Taïani (2021a), « BASALT: A Rock-Solid Foundation for Epidemic Consensus Algorithms in Very Large, Very Open Networks », arXiv: 2102.04063.

Auvolat, Alex, Yérom-David Bromberg, and François Taïani (n.d.), « SBO: Fast Epidemic Byzantine-Tolerant Consensus in Truly Open Networks ».

Auvolat, Alex, Davide Frey, Michel Raynal, and François Taïani (2020), « Money Transfer Made Simple: a Specification, a Generic Algorithm, and its Proof », *Bull. EATCS* 132.

— (2021b), « Byzantine-tolerant Causal Broadcast », *Theoretical Computer Science*, pp. 55–68, DOI: 10.1016/j.tcs.2021.06.021.

Auvolat, Alex, Michel Raynal, and François Taïani (2019), « Byzantine-tolerant Set-constrained Delivery Broadcast », *International Conference on Principles of Distributed Systems (OPODIS)*, ACM, 6:1–6:23, DOI: 10.4230/LIPIcs.OPODIS.2019.6.

Auvolat, Alex and François Taïani (2019), « Merkle Search Trees: Efficient State-based CRDTs in Open Networks », *38th Symposium on Reliable Distributed Systems (SRDS)*, IEEE, pp. 221–230, DOI: 10.1109/SRDS47363.2019.00032.

Azariadis, Costas (1981), « Self-fulfilling prophecies », *Journal of Economic Theory* 25.*3*, pp. 380–396, ISSN: 0022-0531, DOI: 10.1016/0022-0531(81)90038-7.

Baillargeon, Normand (2001), *L'ordre moins le pouvoir, histoire et actualité de l'anarchisme*, Agone, ISBN: 2-910846-29-6.

Barnosky, Anthony D., Paul R. Ehrlich, Elizabeth A. Hadly, and Anne R. Kapuscinski (2016), « Avoiding collapse: Grand challenges for science and society to solve by 2050Avoiding collapse », *Elementa: Science of the Anthropocene* 4.

Baudet, Mathieu, George Danezis, and Alberto Sonnino (2020), « FastPay: High-Performance Byzantine Fault Tolerant Settlement », *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pp. 163–177, DOI: 10.1145/3419614.3423249.

Bayer, Rudolf and Edward M. McCreight (1970), « Organization and Maintenance of Large Ordered Indexes », ACM, DOI: 10.1007/BF00288683.

Beau, Denis (2020), *Crypto-actifs, stablecoins et banques centrales : risques, enjeux et perspectives*, URL: https://www.banque-france.fr/intervention/crypto-actifs-stablecoins-et-banques-centrales-risques-enjeux-et-perspectives-0 (visited on 06/14/2021).

Bégaudeau, François (2021), *Peut-on s'émanciper de ses déterminismes ?*, URL: https://www.youtube.com/watch?v=owPFdSEdSKA (visited on 03/16/2021).

Benet, Juan (2014), « IPFS-content addressed, versioned, P2P file system », arXiv: 1407.3561.

Bessani, Alysson, Joao Sousa, and Eduardo Alchieri (2014), « State machine replication for the masses with BFT-SMART », *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, IEEE, pp. 355–362, DOI: 10.1109/DSN.2014.43.

Birman, Kenneth P. (1994), « A response to Cheriton and Skeen's criticism of causal and totally ordered communication », *ACM SIGOPS Operating Systems Review* 28.*1*, pp. 11–21, DOI: 10.1145/164853.164858.

Birman, Kenneth P. and Robert Cooper (1991), « The ISIS project: Real experience with a fault tolerant programming system », *ACM SIGOPS Operating Systems Review* 25.*2*, pp. 103–107, DOI: 10.1145/122120.122133.

Birman, Kenneth P. and Thomas A Joseph (1987), « Reliable communication in the presence of failures », *ACM Transactions on Computer Systems (TOCS)* 5.*1*, pp. 47–76, DOI: 10.1145/7351.7478.

Birman, Kenneth P., André Schiper, and Pat Stephenson (1991), « Lightweight causal and atomic group multicast », *ACM Transactions on Computer Systems (TOCS)* 9.*3*, pp. 272–314, DOI: 10.1145/128738.128742.

Bloom, Burton H. (1970), « Space/Time Trade-offs in Hash Coding with Allowable Errors », *Commun. ACM* 13.*7*, pp. 422–426, DOI: 10.1145/362686.362692.

Borg, Anita, Jim Baumbach, and Sam Glazer (1983), « A message system supporting fault tolerance », *ACM SIGOPS Operating Systems Review* 17.*5*, pp. 90–99, DOI: 10.1145/773379.806617.

Bortnikov, Edward, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer (2009), « Brahms: Byzantine resilient random membership sampling », *Computer Networks* 53.*13*, pp. 2340–2359, DOI: 10.1145/1400751.1400772.

Bouchra Pilet, Amaury, Davide Frey, and Francois Taiani (2020), « Foiling Sybils with HAPS in Permissionless Systems: An Address-based Peer Sampling Service », *IEEE Symposium on Computers and Communications*, IEEE.

Bouget, Simon, Yérom-David Bromberg, Adrien Luxey, and François Taiani (2018), « Pleiades: Distributed structural invariants at scale », *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE, pp. 542–553, DOI: 10.1109/DSN.2018.00062.

Bounie, David (2020), « Cryptomonnaies : « Il existe une course à la création de monnaies digitales publiques » », *La Croix*, ISSN: 0242-6056, URL: https://www.la-croix.com/Economie/Cryptomonnaies-Il-existe-course-creation-monnaies-digitales-publiques-2020-09-13-1201113666 (visited on 06/14/2021).

Bracha, Gabriel (1987), « Asynchronous Byzantine agreement protocols », *Information and Computation* 75.*2*, pp. 130–143, DOI: 10.1016/0890-5401(87)90054-X.

Bracha, Gabriel and Sam Toueg (1985), « Asynchronous consensus and broadcast protocols », *Journal of the ACM (JACM)* 32.*4*, pp. 824–840, DOI: 10.1145/4221.214134.

Browne, Ryan (2021), « The global chip shortage is starting to hit the smartphone industry », en, *CNBC*, Section: Technology, URL: https://www.cnbc.com/2021/07/29/the-global-chip-shortage-is-starting-to-hit-the-smartphone-industry.html (visited on 09/16/2021).

Buchman, Ethan, Jae Kwon, and Zarko Milosevic (2018), « The latest gossip on BFT consensus », arXiv: 1807.04938.

Byers, John, Jeffrey Considine, and Michael Mitzenmacher (2002), « Fast Approximate Reconciliation of Set Differences », p. 17.

Cachin, Christian, Rachid Guerraoui, and Luís Rodrigues (2011), *Introduction to reliable and secure distributed programming*, Springer Science & Business Media, DOI: 10.1007/978-3-642-15260-3.

Cachin, Christian, Klaus Kursawe, Frank Petzold, and Victor Shoup (2001), « Secure and efficient asynchronous broadcast protocols », *Annual International Cryptology Conference*, Springer, pp. 524–541, DOI: 10.1007/3-540-44647-8_31.

Cachin, Christian and Olga Ohrimenko (2018), « Verifying the consistency of remote untrusted services with conflict-free operations », *Information and Computation* 260, pp. 72–88, DOI: 10.1016/j.ic.2018.03.004.

Cass, David and Karl Shell (1983), « Do Sunspots Matter? », *Journal of Political Economy* 91.*2*, pp. 193–227, ISSN: 0022-3808, DOI: 10.1086/261139.

Castro, Miguel, Barbara Liskov, et al. (1999), « Practical Byzantine fault tolerance », *OSDI*, vol. 99, 1999, pp. 173–186.

Chen, Adrian (2013), « Much Ado About Bitcoin », *The New York Times*, URL: https://web.archive.org/web/20131211065800/http://www.nytimes.com/2013/11/27/opinion/much-ado-about-bitcoin.html (visited on 08/18/2021).

Chevalier, Martin and Benjamin Vignolles (2014), « Le bitcoin : défi à la souveraineté monétaire des États et ressource pour le blanchiment d'argent », *Regards croises sur l'economie* n° 14.*1*, pp. 122–125, ISSN: 1956-7413.

Clark, David D. (1985), « The structuring of systems using upcalls », *Proceedings of the tenth ACM symposium on Operating systems principles*, pp. 171–180, DOI: 10.1145/323627.323645.

Cohen, Benjamin J. (1998), *The Geography of Money*, Cornell University Press, ISBN: 978-0-8014-3513-3.

Colin, Gérard (2018), « 10. Les données numériques au cœur de nouveaux conflits géopolitiques », *Regards croises sur l'economie* n° 23.*2*, pp. 138–143, ISSN: 1956-7413.

Collins, Daniel, 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 », *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE, pp. 26–38, DOI: 10.1109/DSN48063.2020.00023.

Cormode, Graham and Shan Muthukrishnan (2005), « An improved data stream summary: the count-min sketch and its applications », *Journal of Algorithms* 55.*1*, pp. 58–75, DOI: 10.1007/978-3-540-24698-5_7.

Coulon, Fabien, Alex Auvolat, Benoit Combemale, Yérom-David Bromberg, François Taïani, Olivier Barais, and Noël Plouzeau (2020), « Modular and Distributed IDE », *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*, pp. 270–282, DOI: 10.1145/3426425.3426947.

Coulouris, George F., Jean Dollimore, and Tim Kindberg (2005), *Distributed systems: concepts and design*, Pearson Education.

Coulouris, George F., John M. Evans, and R. W. Mitchell (1972), « Towards content-addressing in data bases », *Comput. J.* 15.*2*, pp. 95–98, DOI: 10.1093/comjnl/15.2.95.

Crain, Tyler, Vincent Gramoli, Mikel Larrea, and Michel Raynal (2018), « Dbft: Efficient leaderless byzantine consensus and its application to blockchains », *2018 IEEE*

17th International Symposium on Network Computing and Applications (NCA), IEEE, pp. 1–8, DOI: 10.1109/NCA.2018.8548057.

Crosby, Scott A. and Dan S. Wallach (2009), « Super-Efficient Aggregating History-Independent Persistent Authenticated Dictionaries », *ESORICS*, ISBN: 978-3-642-04444-1, DOI: 10.1007/978-3-642-04444-1_41.

Davis, Ann E (2017), *Money as a Social Institution: The Institutional Development of Capitalism*, Taylor & Francis.

De Filippi, Primavera (2018), « Chapitre IV. Implications politiques et sociales », *Blockchain et Cryptomonaies*, Presses Universitaires de France, pp. 104–120, ISBN: 9782130811459.

DeCandia, Giuseppe, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels (2007), « Dynamo: Amazon's highly available key-value store », *ACM SIGOPS operating systems review* 41.*6*, pp. 205–220.

Defontaine, Galaad (2020), « La sobriété : indispensable à une transition réussie ? », *Regards croises sur l'economie* n° 26.*1*, pp. 153–160, ISSN: 1956-7413.

Demers, Alan, Dan Greene, Carl Houser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry (1987), *Epidemic algorithms for replicated database maintenance*, DOI: 10.1145/43921.43922.

DeVault, Drew (2021), *Cryptocurrency is a disaster*, URL: https://drewdevault.com/2021/04/26/Cryptocurrency-is-a-disaster.html (visited on 08/26/2021).

Digiconomist (2020), *Bitcoin Energy Consumption Index - Digiconomist*, URL: https://digiconomist.net/bitcoin-energy-consumption (visited on 03/05/2020).

Doctorow, Cory (2021), *Tech Monopolies and the Insufficient Necessity of Interoperability*, URL: https://locusmag.com/2021/07/cory-doctorow-tech-monopolies-and-the-insufficient-necessity-of-interoperability/ (visited on 07/16/2021).

Doerr, Benjamin, Leslie Ann Goldberg, Lorenz Minder, Thomas Sauerwald, and Christian Scheideler (2011), « Stabilizing consensus with the power of two choices », *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, pp. 149–158, DOI: 10.1145/1989493.1989516.

Douceur, John R. (2002), « The sybil attack », *International workshop on peer-to-peer systems*, Springer, pp. 251–260, DOI: 10.1007/3-540-45748-8_24.

Driscoll, James R., Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan (1986), « Making Data Structures Persistent », STOC '86, New York, NY, USA: ACM, ISBN: 978-0-89791-193-1, DOI: 10.1145/12130.12142.

Duan, Sisi, Michael K. Reiter, and Haibin Zhang (2017), « Secure causal atomic broadcast, revisited », *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE, pp. 61–72, DOI: 10.1109/DSN.2017.64.

Dupré, Denis (2020), *La Finance: pyromane des effondrements?*, URL: https://files.inria.fr/steep/comprendreagir/effondrements-et-finance/ (visited on 08/18/2021).

Dupré, Denis, Jean-François Ponsot, and Jean-Michel Servet (2015), « Le bitcoin contre la révolution des communs », *5ème congrès de l'Association Française d'Economie Politique (AFEP) " L'économie politique de l'entreprise : nouveaux enjeux, nouvelles perspectives "*, Lyon, France.

Eckel, Mike, Izida Chania, and Anaid Gogoryan (2020), « Bitcoin Blackouts: Russian Cryptocurrency 'Miners' Minting Millions While Sucking Abkhazia's Electricity Grid Dry », *RadioFreeEurope/RadioLiberty*, URL: https://www.rferl.org/a/bitcoin-blackouts-russian-cryptocurrency-miners-minting-millions-sucking-abkhazia-electricity-grid-dry/30968307.html (visited on 08/26/2021).

Ehrenkranz, Toby and Jun Li (2009), « On the state of IP spoofing defense », *ACM Transactions on Internet Technology (TOIT)* 9.*2*, pp. 1–29, DOI: 10.1145/1516539.1516541.

Enes, Vitor, Paulo Sérgio Almeida, Carlos Baquero, and João Leitão (2019), « Efficient synchronization of state-based CRDTs », *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, IEEE, pp. 148–159, DOI: 10.1109/ICDE.2019.00022.

Erdős, Paul and Alfréd Rényi (1959), « On random graphs I. », *Publicationes Mathematicae*.

Eugster, Patrick T., Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Massoulié (2004a), « Epidemic information dissemination in distributed systems », *Computer* 37.*5*, pp. 60–67, DOI: 10.1109/MC.2004.1297243.

— (2004b), « From epidemics to distributed computing », *IEEE Computer*.

Falque-Pierrotin, Isabelle (2018), « 12. L'économie de la donnée peut-elle se développer contre ses utilisateurs ? », *Regards croises sur l'economie* n° 23.*2*, pp. 161–169, ISSN: 1956-7413.

Feuer, Alan (2013), « The Bitcoin Ideology », *The New York Times*, URL: https://web.archive.org/web/20180701222302/https://www.nytimes.com/2013/12/15/sunday-review/the-bitcoin-ideology.html (visited on 08/17/2021).

Fidge, Colin J. (1987), « Timestamps in message-passing systems that preserve the partial ordering ».

Fourcade, Marion (2018), « 8. La mouche et le traqueur : alignement et désaxement dans le capitalisme au XXIe siècle », *Regards croises sur l'economie* n° 23.*2*, pp. 114–125, ISSN: 1956-7413.

Fowler, Martin (2004), « Module assembly [programming] », *IEEE Software* 21.*2*, pp. 65–67.

Fox, Armando and Eric A. Brewer (1999), « Harvest, yield, and scalable tolerant systems », *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*, IEEE, pp. 174–178, DOI: 10.1109/HOTOS.1999.798396.

Friedman, Roy and Shiri Manor (2004), *Causal Ordering in Deterministic Overlay Networks*, tech. rep. CS Technion report CS-2004-04.

Garay, Juan, Aggelos Kiayias, and Nikos Leonardos (2015), « The Bitcoin Backbone Protocol: Analysis and Applications », *Advances in Cryptology - EUROCRYPT 2015*, Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 281–310, ISBN: 978-3-662-46803-6, DOI: 10.1007/978-3-662-46803-6_10.

Gilad, Yossi, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich (2017), « Algorand: Scaling byzantine agreements for cryptocurrencies », *Proceedings of the 26th Symposium on Operating Systems Principles*, ACM, pp. 51–68, DOI: 10.1145/3132747.3132757.

Gilbert, Seth and Nancy A. Lynch (2002), « Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services », *Acm Sigact News* 33.*2*, pp. 51–59, DOI: 10.1145/564585.564601.

— (2012), « Perspectives on the CAP Theorem », *Computer* 45.*2*, pp. 30–36, DOI: 10.1109/MC.2011.389.

Giraud, Gaël (2013), *Illusion financière*, Editions de l'Atelier.

Goncalves, Ricardo Jorge Tome, Paulo Sergio Almeida, Carlos Baquero, and Vitor Fonte (2017), « DottedDB: Anti-Entropy without Merkle Trees, Deletes without Tombstones », *SRDS '17*, ISBN: 978-1-5386-1679-6, DOI: 10.1109/SRDS.2017.28.

Gonzalez, Oscar (2021), « Cryptocurrency pump-and-dump schemes: Everything you should know about these scams », en, *CNET*, URL: https://www.cnet.com/personal-finance/investing/cryptocurrency-pump-and-dump-schemes-everything-you-should-know-about-these-scams/ (visited on 09/16/2021).

Guerraoui, Rachid, Nikola Knežević, Vivien Quéma, and Marko Vukolić (2010), « The Next 700 BFT Protocols », EuroSys '10, New York, NY, USA: ACM, pp. 363–376, ISBN: 978-1-60558-577-2, DOI: 10.1145/1755913.1755950.

Guerraoui, Rachid, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi (2019a), « Scalable Byzantine reliable broadcast », *33rd International Symposium on Distributed Computing (DISC 2019)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Guerraoui, Rachid, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi (2019b), « The consensus number of a cryptocurrency », *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pp. 307–316, DOI: 10.1145/3293611.3331589.

— (2019c), « The consensus number of a cryptocurrency », *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pp. 307–316.

Guo, Bingyong, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang (2020), « Dumbo: Faster asynchronous bft protocols », *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 803–818.

Guo, Eileen (2021), *How YouTube's rules are used to silence human rights activists*, URL: https://www.technologyreview.com/2021/06/24/1027048/youtube-xinjiang-censorship-human-rights-atajurt/ (visited on 06/24/2021).

Habib, Komal, Lorie Hamelin, and Henrik Wenzel (2016), « A dynamic perspective of the geopolitical supply risk of metals », *Journal of Cleaner Production* 133, pp. 850–858, ISSN: 0959-6526, DOI: 10.1016/j.jclepro.2016.05.118.

Hadzilacos, Vassos and Sam Toueg (1994), *A modular approach to fault-tolerant broadcasts and related problems*, tech. rep., Cornell University.

Hansen, James, Makiko Sato, Pushker Kharecha, David Beerling, Robert Berner, Valerie Masson-Delmotte, Mark Pagani, Maureen Raymo, Dana L Royer, and James C Zachos (2008), « Target atmospheric CO2: Where should humanity aim? », arXiv: 0804.1126.

Hayek, Friedrich A. von (1990), *Denationalisation of money: the argument refined ; an analysis of the theory and practice of concurrent currencies*, 3. ed, Hobart paper 70, London, ISBN: 978-0-255-36239-9.

He, Dong (2018), « La politique monétaire à l'ère du numérique », *Finances & Développement L'argent, Autrement*, p. 4.

Heilman, Ethan, Alison Kendler, Aviv Zohar, and Sharon Goldberg (2015), « Eclipse attacks on bitcoin's peer-to-peer network », *24th USENIX Security Symposium (USENIX Security 15)*, pp. 129–144.

Heinberg, Richard (2010), *Peak everything: waking up to the century of declines*, New Society Publishers.

Heinberg, Richard (2011), *The end of growth: Adapting to our new economic reality*, New Society Publishers.

Herlihy, Maurice (1991), « Wait-free synchronization », *ACM Transactions on Programming Languages and Systems (TOPLAS)* 13.*1*, pp. 124–149, DOI: 10.1145/114005.102808.

Hopkins, Rob and Peter Lipman (2009), « Who we are and what we do », *Transition Network, Totnes*, URL: https://www.transitionnetwork.org/sites/www.transitionnetwork.org/files/WhoWeAreAndWhatWeDo-lowres.pdf (visited on 09/07/2021).

Illich, Ivan (1973), *Tools for conviviality*, Harper & Row New York.

Imbs, Damien and Michel Raynal (2016), « Trading off t-resilience for efficiency in asynchronous byzantine reliable broadcast », *Parallel Processing Letters* 26.*04*, p. 1650017, DOI: 10.1142/S0129626416500171.

Inequality.org (2021), *Global Inequality*, URL: https://inequality.org/facts/global-inequality/ (visited on 08/23/2021).

IPCC (2014), *AR5 Synthesis Report: Climate Change 2014*, tech. rep., URL: https://archive.ipcc.ch/report/ar5/syr/ (visited on 06/23/2021).

— (2018), *Special Report: Global Warming of 1.5 ºC*, tech. rep., URL: https://www.ipcc.ch/sr15/ (visited on 06/23/2021).

— (2021), *AR6 Synthesis Report: Climate Change 2022*, tech. rep., not yet published, URL: https://www.ipcc.ch/report/sixth-assessment-report-cycle/ (visited on 06/23/2021).

Jacob, Florian, Luca Becker, Jan Grashöfer, and Hannes Hartenstein (2020), « Matrix Decomposition: Analysis of an Access Control Approach on Transaction-based DAGs without Finality », *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*, pp. 81–92.

Jacob, Florian, Carolin Beer, Norbert Henze, and Hannes Hartenstein (2021), « Analysis of the Matrix Event Graph Replicated Data Type », *IEEE access* 9, pp. 28317–28333.

Jelasity, Márk, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen (2007), « Gossip-based Peer Sampling », *ACM Trans. Comput. Syst.*, ISSN: 0734-2071, DOI: 10.1145/1275517.1275520.

Jesi, Gian Paolo, Alberto Montresor, and Maarten van Steen (2010), « Secure peer sampling », *Computer Networks* 54.*12*, pp. 2086–2098, DOI: 10.1016/j.comnet.2010.03.020.

Jungnickel, Tim, Lennart Oldenburg, and Matthias Loibl (2017), « Designing a Planetary-Scale IMAP Service with Conflict-free Replicated Data Types », *OPODIS 2017*, DOI: 10.4230/lipics.opodis.2017.23.

Kermarrec, Anne-Marie, Laurent Massoulié, and Ayalvadi J. Ganesh (2003), « Probabilistic reliable dissemination in large-scale systems », *IEEE Transactions on Parallel and Distributed Systems* 14.*3*, pp. 248–258, ISSN: 1045-9219, DOI: 10.1109/TPDS.2003.1189583.

Kermarrec, Anne-Marie and Maarten Van Steen (2007), « Gossiping in distributed systems », *ACM SIGOPS operating systems review* 41.*5*, pp. 2–7, DOI: 10.1145/1317379.1317381.

Kim, Jae-Yun, Junmo Lee, Yeonjae Koo, Sanghyeon Park, and Soo-Mook Moon (2021), « Ethanos: efficient bootstrapping for full nodes on account-based blockchain », *Proceedings of the Sixteenth European Conference on Computer Systems*, pp. 99–113.

Kleppmann, Martin and Alastair R. Beresford (2017), « A Conflict-Free Replicated JSON Datatype », *IEEE Transactions on Parallel and Distributed Systems* 28.*10*, pp. 2733–2746, ISSN: 1045-9219, DOI: 10.1109/TPDS.2017.2697382.

Kshemkalyani, Ajay D. and Mukesh Singhal (1998), « Necessary and sufficient conditions on information for causal message ordering and their optimal implementation », *Distributed Computing* 11.*2*, pp. 91–111, DOI: 10.1007/s004460050044.

Kwon, Jae (2014), « Tendermint: Consensus without mining », *Draft v. 0.6, fall* 1.*11*, URL: https://tendermint.com/static/docs/tendermint.pdf (visited on 09/07/2021).

Laborde, Stéphane (2012), *Relative Theory of Money*, URL: http://en.trm.creationmonetaire.info/ (visited on 03/17/2021).

Lakomski-Laguerre, Odile and Ludovic Desmedt (2015), « L'alternative monétaire Bitcoin : une perspective institutionnaliste », *Revue de la régulation. Capitalisme, institutions, pouvoirs 18*, ISSN: 1957-7796, DOI: 10.4000/regulation.11489.

Lamport, Leslie (1978), « Time, Clocks, and the Ordering of Events in a Distributed System », vol. 21, 7, pp. 558–565, DOI: 10.1145/359545.359563.

— (1984), « Using time instead of timeout for fault-tolerant distributed systems. », *ACM Transactions on Programming Languages and Systems (TOPLAS)* 6.*2*, pp. 254–280, DOI: 10.1145/2993.2994.

Lamport, Leslie, Robert E. Shostak, and Marshall C. Pease (1982), « The Byzantine Generals Problem », vol. 4, 3, pp. 382–401, DOI: 10.1145/357172.357176.

Latouche, Serge (2006), *Le pari de la décroissance*, Fayard.

Lesani, Mohsen, Christian J. Bell, and Adam Chlipala (2016), « Chapar: certified causally consistent distributed key-value stores », *ACM SIGPLAN Notices* 51.*1*, pp. 357–370, DOI: 10.1145/2914770.2837622.

Linde, Albert van der, João Leitão, and Nuno Preguiça (2016), « Δ-CRDTs: Making Δ-CRDTs Delta-based », PaPoC '16, ACM, 12:1–12:4, ISBN: 978-1-4503-4296-4, DOI: 10.1145/2911151.2911163.

Lloyd, Wyatt, Michael J. Freedman, Michael Kaminsky, and David G. Andersen (2011), « Don't settle for eventual: Scalable causal consistency for wide-area storage with COPS », *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 401–416, DOI: 10.1145/2043556.2043593.

Malkhi, Dahlia and Michael K. Reiter (1997), « A high-throughput secure reliable multicast protocol », *Journal of Computer Security* 5.*2*, pp. 113–127, DOI: 10.3233/JCS-1997-5203.

Marandi, Parisa Jalili, Marco Primi, and Fernando Pedone (2011), « High performance state-machine replication », *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, IEEE, pp. 454–465.

Maria, Apostolaki, Zohar Aviv, and Vanbever Laurent (2017), « Hijacking Bitcoin: Routing Attacks on Cryptocurrencies », *Security and Privacy (SP), 2017 IEEE Symposium on*, IEEE.

Matos, Miguel, Hugues Mercier, Pascal Felber, Rui Oliveira, and José Pereira (2015), « EpTO: An Epidemic Total Order Algorithm for Large-Scale Distributed Systems », Middleware '15, New York, NY, USA: ACM, pp. 100–111, ISBN: 978-1-4503-3618-5, DOI: 10.1145/2814576.2814804.

Matrix Foundation (2021), *Room Version 2*, URL: https://spec.matrix.org/unstable/rooms/v2/ (visited on 08/12/2021).

Mattern, Friedemann (1989), « Virtual time and global states of distributed systems », *Parallel and Distributed Algorithms*, pp. 215–226.

MaxMind (2020), *GeoLite2 ASN CSV Database*, URL: https://dev.maxmind.com/geoip/geoip2/geolite2-asn-csv-database/ (visited on 10/12/2020).

May, Timothy (1988), *The Crypto Anarchist Manifesto*, URL: https://groups.csail.mit.edu/mac/classes/6.805/articles/crypto/cypherpunks/may-crypto-manifesto.html (visited on 08/18/2021).

McLachlan, Jon, Andrew Tran, Nicholas Hopper, and Yongdae Kim (2009), « Scalable onion routing with torsk », *Proceedings of the 16th ACM conference on Computer and communications security (CCS)*, pp. 590–599, DOI: 10.1145/1653662.1653733.

Meissner, Falk and Thomas Kirschstein (2021), *Semiconductor crisis in the automotive industry*, en, URL: https://www.rolandberger.com/en/Insights/Publications/Semiconductor-crisis-in-the-automotive-industry.html (visited on 09/16/2021).

Merkle, Ralph C. (1987), « A digital signature based on a conventional encryption function », *Conference on the theory and application of cryptographic techniques*, Springer, pp. 369–378, DOI: 10.1007/3-540-48184-2_32.

Messéant, Élodie (Nov. 18, 2020), « Les cryptomonnaies: l'argent des criminels? », *La Tribune 7034*, p. 50.

Micali, Silvio, Michael Rabin, and Salil Vadhan (1999), « Verifiable random functions », *40th Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE, pp. 120–130, DOI: 10.1109/SFFCS.1999.814584.

Mien, Edouard (2020), « Y-a-t-il des limites à la croissance ? Le « Rapport Meadows » et ses prolongements actuels », *Regards croises sur l'economie* n° 26.*1*, ISSN: 1956-7413.

Miller, Andrew, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song (2016), « The honey badger of BFT protocols », *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 31–42, DOI: 10.1145/2976749.2978399.

Minsky, Yaron, Ari Trachtenberg, and Richard Zippel (2003), « Set reconciliation with nearly optimal communication complexity », *IEEE Transactions on Information Theory* 49.*9*, pp. 2213–2218, ISSN: 0018-9448, DOI: 10.1109/TIT.2003.815784.

Mostefaoui, Achour, Matthieu Perrin, Michel Raynal, and Jiannong Cao (2019), « Crash-tolerant causal broadcast in O (n) messages », *Information Processing Letters* 151, p. 105837, DOI: 10.1016/j.ipl.2019.105837.

Murty, Venkatesh V. and Vijay K. Garg (1997), « Characterization of message ordering specifications and protocols », *Proceedings of 17th International Conference on Distributed Computing Systems*, IEEE, pp. 492–499, DOI: 10.1109/ICDCS.1997.603392.

Nakamoto, Satoshi (2009a), *Bitcoin open source implementation of P2P currency*, URL: https://satoshi.nakamotoinstitute.org/posts/p2pfoundation/1/ (visited on 08/17/2021).

— (2009b), *Bitcoin: A peer-to-peer electronic cash system*, URL: https://bitcoin.org/bitcoin.pdf (visited on 09/07/2021).

Nédelec, Brice, Pascal Molli, and Achour Mostefaoui (2016), « CRATE: Writing Stories Together with our Browsers », *Proceedings of the 25th International Conference Companion on World Wide Web - WWW '16 Companion*, Montréal, Québec, Canada: ACM Press, pp. 231–234, ISBN: 978-1-4503-4144-8, DOI: 10.1145/2872518.2890539.

Nédelec, Brice, Pascal Molli, Achour Mostefaoui, and Emmanuel Desmontils (2013), « LSEQ: An Adaptive Structure for Sequences in Distributed Collaborative Editing », DocEng '13, ACM, ISBN: 978-1-4503-1789-4, DOI: 10.1145/2494266.2494278.

Nédelec, Brice, Pascal Molli, and Achour Mostéfaoui (2018a), « Breaking the Scalability Barrier of Causal Broadcast for Large and Dynamic Systems », DOI: 10.1109/SRDS.2018.00016, arXiv: 1805.05201.

Nédelec, Brice, Julian Tanke, Davide Frey, Pascal Molli, and Achour Mostéfaoui (2018b), « An adaptive peer-sampling protocol for building networks of browsers », *World Wide Web* 21.*3*, pp. 629–661, ISSN: 1386-145X, 1573-1413, DOI: 10.1007/s11280-017-0478-5.

Needleman, Sarah E. (2018), « The Computer Part People Are Hoarding: 'I Felt Like I Was Buying Drugs' », *Wall Street Journal*, ISSN: 0099-9660, URL: https://www.wsj.com/articles/the-computer-part-people-are-hoarding-i-felt-like-i-was-buying-drugs-1518195876 (visited on 08/26/2021).

OECD (2015), *In It Together: Why Less Inequality Benefits All*, p. 336, URL: https://www.oecd-ilibrary.org/content/publication/9789264235120-en.

Oster, Gérald, Pascal Urso, Pascal Molli, and Abdessamad Imine (2006), « Data consistency for P2P collaborative editing », *Proceedings of the 2006 ACM Conference on Computer Supported Cooperative Work, CSCW 2006, Banff, Alberta, Canada*, ACM, pp. 259–268, DOI: 10.1145/1180875.1180916.

Ostrom, Elinor (1990), *Governing the commons: The evolution of institutions for collective action*, Cambridge university press.

Palo Alto Networks (2013), *Application Usage & Threat Report*, URL: https://web.archive.org/web/20131031153132if_/http://researchcenter.paloaltonetworks.com/app-usage-risk-report-visualization/#.

Papp, Alizé (2018), « L'infobésité, une épidémie à l'âge des nouvelles technologies de l'information et de la communication ? », *Regards croises sur l'economie* n° 23.*2*, pp. 105–113, ISSN: 1956-7413.

Parrique, Timothée (2019), « The political economy of degrowth », PhD thesis, Université Clermont Auvergne.

Pass, Rafael and Elaine Shi (2017a), « Fruitchains: A fair blockchain », *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pp. 315–324.

— (2017b), « Hybrid consensus: Efficient consensus in the permissionless model », *31st International Symposium on Distributed Computing (DISC 2017)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Porcherot, Raphael (2019), *Les cryptomonnaies sociales, ou la convergence des contestations monétaires*, URL: http://theconversation.com/les-cryptomonnaies-sociales-ou-la-convergence-des-contestations-monetaires-109278 (visited on 06/14/2021).

Preguica, Nuno, Joan Manuel Marques, Marc Shapiro, and Mihai Letia (2009), « A Commutative Replicated Data Type for Cooperative Editing », *29th IEEE International Conference on Distributed Computing Systems*, DOI: 10.1109/ICDCS.2009.20.

Premalatha, M., Tabassum-Abbasi, Tasneem Abbasi, and S. A. Abbasi (2014), « The Generation, Impact, and Management of E-Waste: State of the Art », *Critical Reviews in Environmental Science and Technology* 44.*14*, pp. 1577–1678, ISSN: 1064-3389, DOI: 10.1080/10643389.2013.782171.

Raynal, Michel (2018), *Fault-Tolerant Message-Passing Distributed Systems - An Algorithmic Approach*, Springer, ISBN: 978-3-319-94140-0, DOI: 10.1007/978-3-319-94141-7.

— (2019), « An Informal Visit to the Wonderful Land of Consensus Numbers and Beyond », *Bull. EATCS* 129.

Raynal, Michel, André Schiper, and Sam Toueg (1991), « The Causal Ordering Abstraction and a Simple Way to Implement it », *Inf. Process. Lett.* 39.*6*, pp. 343–350, DOI: 10.1016/0020-0190(91)90008-6.

Reiter, Michael K. and Kenneth P. Birman (1994), « How to Securely Replicate Services », *ACM Trans. Program. Lang. Syst.* 16.*3*, pp. 986–1009, DOI: 10.1145/177492.177745.

Renesse, Robbert van, Dan Dumitriu, Valient Gough, and Chris Thomas (2008), « Efficient Reconciliation and Flow Control for Anti-entropy Protocols », LADIS '08, ACM, ISBN: 978-1-60558-296-2, DOI: 10.1145/1529974.1529983.

Robinson, Brett H. (2009), « E-waste: An assessment of global production and environmental impacts », *Science of The Total Environment* 408.*2*, pp. 183–191, ISSN: 0048-9697, DOI: 10.1016/j.scitotenv.2009.09.044.

Rudgard, Olivia (2018), « The tech moguls who invented social media have banned their children from it », *Independent.ie*, URL: https://www.independent.ie/life/

family/parenting/the-tech-moguls-who-invented-social-media-have-banned-their-children-from-it-37494367.html (visited on 07/19/2021).

Sanjuan, Hector, Samuli Poyhtari, Pedro Teixeira, and Ioannis Psaras (2020), « Merkle-CRDTs: Merkle-DAGs meet CRDTs », arXiv: 2004.00107.

Sapirshtein, Ayelet, Yonatan Sompolinsky, and Aviv Zohar (2016), « Optimal selfish mining strategies in bitcoin », *International Conference on Financial Cryptography and Data Security*, Springer, pp. 515–532.

Saraph, Vikram and Maurice Herlihy (2019), « An empirical study of speculative concurrency in ethereum smart contracts », arXiv: 1901.01376.

Schiesser, Tim (2021), « GPU Availability and Pricing Update: August 2021 », *TechSpot*, URL: https://www.techspot.com/article/2311-gpu-pricing-2021-update/ (visited on 08/26/2021).

Schmuck, Frank B. (1988), *The use of efficient broadcast protocols in asynchronous distributed systems*, tech. rep.

Schneider, Fred B. (1990), « Implementing fault-tolerant services using the state machine approach: A tutorial », *ACM Computing Surveys (CSUR)* 22.*4*, pp. 299–319, DOI: 10.1145/98163.98167.

Schütt, Thorsten, Florian Schintke, and Alexander Reinefeld (2008), « Scalaris: reliable transactional p2p key/value store », *ERLANG '08*, ACM Press, p. 41, ISBN: 978-1-60558-065-4, DOI: 10.1145/1411273.1411280.

Schwarz, Reinhard and Friedemann Mattern (1994), « Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail », *Distributed Comput.* 7.*3*, pp. 149–174, DOI: 10.1007/BF02277859.

Servigne, Pablo and Raphaël Stevens (2015), *Comment tout peut s' effondrer. Petit manuel de collapsologie à l'usage des générations présentes: Petit manuel de collapsologie à l'usage des générations présentes*, Média Diffusion.

Shapiro, Marc (2011), « A comprehensive study of Convergent and Commutative Replicated Data Types », *Encyclopedia of Database Systems*, Springer New York, pp. 1–5, ISBN: 978-1-4899-7993-3, DOI: 10.1007/978-1-4899-7993-3_80813-1.

Shapiro, Marc, Nuno Preguiça, Carlos Baquero, and Marek Zawirski (2011), « Conflict-Free Replicated Data Types », *Stabilization, Safety, and Security of Distributed Systems*, Springer Berlin Heidelberg, DOI: 10.1007/978-3-642-24550-3_29.

Singh, Atul, Tsuen-Wan Ngan, Peter Druschel, and Dan S. Wallach (2006), « Eclipse Attacks on Overlay Networks: Threats and Defenses », *Proceedings IEEE INFOCOM*

2006. 25TH IEEE International Conference on Computer Communications, pp. 1–12, DOI: 10.1109/INFOCOM.2006.231.

Stephant, Aurore (2021), *Under the hood of Sustainable IT - Promesses de dématérialisation et matérialité minérale*, URL: https://www.youtube.com/watch?v=QW9udH0vwlE (visited on 09/16/2021).

Stiglitz, Joseph (2010), « US Does Not Have Capitalism Now », *CNBC*, URL: https://www.cnbc.com/id/34921639 (visited on 08/18/2021).

Tamassia, Roberto and Nikos Triandopoulos (2007), « Efficient Content Authentication in Peer-to-Peer Networks », *Applied Cryptography and Network Security*, Springer Berlin Heidelberg, ISBN: 978-3-540-72738-5, DOI: 10.1007/978-3-540-72738-5_23.

Team Rocket (2018), *Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies*, URL: https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNP (visited on 09/07/2021).

Tolia, Niraj, Michael Kozuch, Mahadev Satyanarayanan, Brad Karp, Thomas C. Bressoud, and Adrian Perrig (2003), « Opportunistic Use of Content Addressable Storage for Distributed File Systems », *Proceedings of the General Track: 2003 USENIX Annual Technical Conference, June 9-14, 2003, San Antonio, Texas, USA*, USENIX, pp. 127–140.

Tu, Wenling and Yujung Lee (2009), « Ineffective environmental laws in regulating electronic manufacturing pollution: examining water pollution disputes in Taiwan », *2009 IEEE International Symposium on Sustainable Systems and Technology*, pp. 1–6, DOI: 10.1109/ISSST.2009.5156734.

Turak, Natasha (2021), « Iran bans bitcoin mining as its cities suffer blackouts and power shortages », *CNBC*, URL: https://www.cnbc.com/2021/05/26/iran-bans-bitcoin-mining-as-its-cities-suffer-blackouts.html (visited on 08/26/2021).

Urdaneta, Guido, Guillaume Pierre, and Maarten van Steen (2011), « A Survey of DHT Security Techniques », *ACM Comput. Surv.* 43.*2*, DOI: 10.1145/1883612.1883615.

Vasselin, Françoise (2020), « Bitcoin et souveraineté monétaire », *La Vie des idées*, URL: https://laviedesidees.fr/Bitcoin-et-souverainete-monetaire.html (visited on 06/14/2021).

Vogels, Werner (2008), « Eventually Consistent », *Queue* 6.*6*, ISSN: 1542-7730, DOI: 10.1145/1466443.1466448.

Voulgaris, Spyros, Daniela Gavidia, and Maarten Van Steen (2005), « Cyclon: Inexpensive membership management for unstructured p2p overlays », *Journal of Network and systems Management* 13.*2*, pp. 197–217, DOI: 10.1007/s10922-005-4441-x.

Voulgaris, Spyros and Maarten van Steen (2013), « VICINITY: A Pinch of Randomness Brings out the Structure », *Middleware 2013*, Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 21–40, ISBN: 978-3-642-45065-5, DOI: 10.1007/978-3-642-45065-5_2.

Weyl, Glen (2018), « 4. Vers la dignité numérique », *Regards croises sur l'economie* n° 23.*2*, pp. 56–63, ISSN: 1956-7413.

Zamani, Mahdi, Mahnush Movahedi, and Mariana Raykova (2018), « Rapidchain: Scaling blockchain via full sharding », *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 931–948.

Zhang, Fan, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert Van Renesse (2017), « REM: Resource-efficient mining for blockchains », *26th USENIX Security Symposium (USENIX Security 17)*, pp. 1427–1444.

Zhang, Yupu, Abhishek Rajimwale, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau (2010), « End-to-end Data Integrity for File Systems: A ZFS Case Study. », *FAST*, pp. 29–42.

Zimmermann, Claus D. (2013), *A Contemporary Concept of Monetary Sovereignty*, Oxford Monographs in International Law, Oxford, New York: Oxford University Press, ISBN: 978-0-19-968074-0.

Zimmermann, Till and Stefan Gößling-Reisemann (2013), « Critical materials and dissipative losses: A screening study », *Science of The Total Environment* 461-462, pp. 774–780, ISSN: 0048-9697, DOI: 10.1016/j.scitotenv.2013.05.040.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Méthodes Probabilistes pour les Systèmes Collaboratifs dans les Réseaux Large-échelle Sans Confiance

Mot clés : Systèmes distribués, consensus distribué, consensus Byzantine, primitives de diffusion, cohérence faible, échantillonnage de pairs, algorithmes épidémiques, blockchain

Résumé : Internet est un outil formidable pour l'éducation, la communication et la collaboration, mais ses usages majoritaires sont actuellement sous monopole de grandes multinationales (GAFAM), ce qui a des conséquences sur le respect des droits humains et des libertés individuelles. Cette thèse propose des outils pour le développement d'applications décentralisées : des applications sur Internet qui fournissent des fonctionnalités similaires aux plateformes des GAFAM, mais de manière décentralisée, afin de rendre le pouvoir aux utilisateur pour décider démocratiquement de leur fonctionnement et de leurs usages. Nous nous concentrons sur les algorithmes épidémiques qui sont particulièrement adaptés dans le cadre de réseaux ouverts à large échelle. Nous proposons des contributions sur la diffusion causale de messages tolérante aux nœuds Byzantins, la diffusion causale épidémique à l'aide d'un stockage d'événements synchronisé par anti-entropie, l'échantillonnage aléatoire de pairs résistants aux attaques Byzantines et aux attaques Sybil, ainsi qu'un nouvel algorithme épidémique de diffusion totalement ordonnée qui tolère les nœuds malicieux et fournit un débit de messages élevé.

Title: Probabilistic Methods for Collaboration Systems in Large-scale Trustless Networks

Keywords: Distributed systems, distributed consensus, Byzantine consensus, broadcast primitives, eventual consistency, random peer sampling, blockchain

Abstract: The Internet is a formidable tool for education, communication and collaboration, however it is currently being monopolized by large corporations (GAFAM), which has consequences for many social issues such as respect of human rights and individual freedoms. This thesis focuses on ways to build decentralized applications: Internet applications that provide levels of functionality similar to those provided by the GAFAM, but that function in a decentralized manner, empowering the users to democratically decide of their functioning and their uses. We focus on epidemic algorithms, which are particularly suited to the context of very large open networks. We make contributions on causal broadcast in presence of Byzantine nodes, epidemic causal broadcast using an event store synchronized with an anti-entropy algorithm, random peer sampling in presence of Byzantine nodes and Sybil attacks, as well as a new epidemic total order broadcast which is tolerant to malicious nodes and provides high throughput message delivery.