Exploring heterogeneity in loosely consistent decentralized data replication

Pierre-Louis Roman

UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1
COMUE UNIVERSITÉ BRETAGNE LOIRE

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

Pierre-Louis ROMAN

Exploring heterogeneity in loosely consistent decentralized data replication

Thèse présentée et soutenue à Rennes, le 2018-12-18
Unité de recherche : IRISA (UMR 6074)

Rapporteurs avant soutenance :

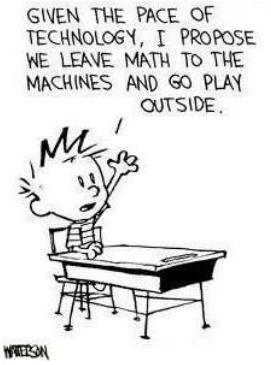
Dick EPEMA	Full professor	Delft University of Technology
Pascal FELBER	Professeur	Université de Neuchâtel

Composition du jury :

Président :	Gaël THOMAS	Professeur	Telecom SudParis
Rapporteurs :	Dick EPEMA	Full professor	Delft University of Technology
	Pascal FELBER	Professeur	Université de Neuchâtel
Examinatrice :	Emmanuelle ANCEAUME	Chargée de recherche	CNRS IRISA
Dir. de thèse :	François TAÏANI	Professeur	Université de Rennes 1
Co-encadrant :	Davide FREY	Chargé de recherche	Inria Rennes Bretagne-Atlantique

Invités :

Pascal MOLLI	Professeur	Université de Nantes
Spyros VOULGARIS	Assistant professor	Athens University of Economics and Business



ACKNOWLEDGMENT

I would first and foremost like to thank both Dick Epema and Pascal Felber for accepting the time-consuming task of reviewing this thesis, I am truly grateful for your time. I extend my thanks to the examiners Emmanuelle Anceaume, Pascal Molli, Gaël Thomas and Spyros Voulgaris for accepting to be part of my jury.

This thesis would not have been possible without my supervisors François Taïani and Davide Frey whom I thank for their extensive support throughout these ~~three~~ four (!) years. François should be acclaimed for his unrivaled patience and broad knowledge, and Davide for his acute perspicacity and high availability.

Apart from my supervisors, I would like to acknowledge other researchers I have learned from and have had the pleasure to discuss extensively with in these past few years: Marc X. Makkes, Achour Mostéfaoui, Matthieu Perrin, Michel Raynal, Spyros Voulgaris and numerous members of the ANR SocioPlug project.

My special thanks go to Cécile Bouton and Virginie Desroches for helping me survive the administrative madhouse that PhD students all experience.

I have enjoyed the time I spent at the ASAP/WIDE research group, as well as at the VU during my short stay, and I must thank its members for that. I consider most of you as friends by now.

I feel obliged to praise a couple more key people who had a great impact on my life: Mr Pouant, my biology teacher in high school, and Olivier Marin, my distributed systems teacher during my Licence and Master. They have both helped me when I needed it the most and I clearly would not be here if it were not for them.

Last and most importantly, un grand merci à ma famille.

Grazie Gracias धन्यवाद ευχαριστώ Dankjewel Mulțumesc Obrigado Tack
धन्यवाद Շնորհակալություն Trugarez Thank you Merci Gratias

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The printing press has enabled the people to read. Internet will enable him to write.

Benjamin Bayart

If you have ever lost your personal data due to a machine failure, you know how important backups are. For service suppliers, backups are vital for the robustness of their services, as losing their user's data is not an option. Data is therefore usually replicated in several geo-distributed regions to avoid even the worst failures. However, in this situation, each update must also be replicated by every instance to maintain consistent replicas, which rapidly gets more difficult as the number of users, and hence updates, increases. The spectacular growth of the number of people connected to the Internet and the increasing time they spend on it clearly lay a challenge to data replication. This challenge and its ramifications lay at the heart of this thesis.

1.1 Democratization of Internet access

According to the *International Telecommunication Union* (ITU), the worldwide adoption rate of the Internet increased from 22 % in 2007 to 48 % in 2017 [1], [2] resulting in 3.6 billion connected people. The combined drop in price of broadband connection—particularly in least developed countries where it decreased by a 2.5 factor between 2013 and 2016 [2]—and the drop in price of entry-level Internet-ready devices such as smartphones have made accessing the Internet much more affordable and strongly increased its usage.

Alongside the increase in adoption rate, the engagement rate has also been rising. According to Flurry Analytics [3], US citizens went from spending 2h40 per day on their smartphone in 2013 to an impressive 5h per day by 2016, a third of which is being spent on online social networks.

The rebound effect on the adoption rate can be seen through the growth of the userbase of popular social networks and applications such as Facebook or Pokémon Go. It took just a little over 4 years for Facebook, the largest online social network, to get from 1 billion active users in October 2012 to 2 billion active users in January 2017 [4], in the process enabling hundreds of

thousands of content updates and uploads per minute [5] and dozens of millions of messages per minute via their messaging platform WhatsApp [6]. In a different genre, Pokémon Go broke all adoption rate records [7]: with more than 100 million game downloads in their release month and an additional 400 million [8] in the following month, Pokémon Go became the biggest mobile game launch in history.

Since Internet is advocated as both a mean of fundamental freedom of expression [9] and a mean of societal and economic development, it is more than likely that an ever growing fraction of the worldwide population will be provided with access to the Internet. Highly populated regions such as Sub-Saharan Africa, South Asia and Southeast Asia are yet to have a high rate of Internet adoption. However, according to the ITU, the Least Developed Countries that are mostly present in these regions have the highest growth in fixed and mobile broadband subscriptions [2] and will hopefully catch up with the adoption rates of developed countries in the coming decade.

Today's personal devices support a wide range of applications and abilities, and as they have embraced different forms, they can be found almost everywhere in our lives. From owning a single desktop or laptop machine in the 1980s-90s, people started also owning smartphones and tablets in the 2000s. These pocket devices can be quickly accessed anywhere and at any time, thus simplifying Internet access. On a related note, the emergence of the Internet of Things (IoT) is another reason to expect a growth of the number of Internet-ready devices per capita. Devices in the IoT approach are not general purpose anymore but rather tailor-fitted to their tasks. These devices can be designed for entertainment purposes, such as televisions and consoles, enhancing the comfort of homes, such as CCTVs and various sensors managements, and can even be wearable, such as watches and fitness trackers. There exists other types of devices in IoT, alongside personal devices, which are for instance fit for urban management (Smart city), or enterprise-related tasks (Industry 4.0).

The stunning growth of the number of devices connected to the Internet combined with the increase of their usages pose a foreseeable strain to the availability of the accessed services and must be handled by the service providers. In the case of Pokémon Go, the launch of the awaited game was riddled with server outages [10]–[12] preventing a smooth adoption for its users and subsequently attracting bad press for the app. Even the Google cloud engine that supports the app had problems scaling due to that unique flash growth [13]. As users avoid unreliable services, scaling correctly is imperative not only for the growth of a service, but also for its success and survival.

1.2 Architectures for scalable systems

Providers have to multiply the number of servers they deploy to improve the performance and reliability of their services and to deliver highly scalable services able to sustain a growing userbase [14]. Adding servers increases the resilience to hardware failures and prevents resource from depleting as the userbase grows. Servers are typically distributed over different geographical regions [15], [16]; doing so prevents catastrophic failures happening in a region from affecting the entire system, and reduces the average latency for geo-distributed clients by bringing the servers closer to them. However, deploying a large-scale geo-distributed system forces the use of wide area networks that are inherently less reliable and exhibit higher latencies than local networks in datacenters. Thus, data replication within a geo-distributed system has to cope with the increased probability of communication deterioration on top of higher delays. This problems is further compounded by the fact that these architectures are frequently tightly coupled, which helps coordination between servers but hinders the overall scalability of systems.

Another approach to scaling systems is to use loosely coupled architectures that are scalable by design. Decentralized systems are an instance of such systems [17]; they offer strong robustness to failures and network degradation as well as efficient load distribution [18]. As nodes in these systems are less dependent on each other, (i) their users benefit from a stronger control over the resources they choose to contribute (realizing in effect a decentralization of control) and (ii) honest nodes are less impacted by malicious behaviors from other nodes. Coordination in decentralized systems is however more difficult to achieve than in tightly-coupled systems. Typical coordination protocols such as Paxos [19] are very costly in the number of messages they use and generally rely on a leader to reduce this cost. Because a leader tends to tighten the coupling of a network, recent leaderless coordination protocols offer a noteworthy prospect for architectures that aims at maximizing decentralization. Blockchains are an instance of leaderless coordination systems, which we return to later on in this introduction due to their importance for the work we present.

Tightly coupled systems are simpler and more intuitive to manage than loosely coupled systems and are thus more prevalent in production. However, as the load of a service increases, its developers must often trade the simplicity of centralization and tight coupling for the efficiency of geo-distribution and loose coupling. In the context of data replication, tight coupling offers data consistency properties which are easy for developers to work with, but this simplicity is paid for by performances that are lower than those of highly available geo-replicated systems.

1.3 Data replication approaches

More precisely, traditional data replication algorithms provide *strong consistency* by following the "Safety first" approach. These algorithms focus on maintaining exact replicas at all time at each node which is often performed with the help of a single, albeit rotating, coordinator to ensure consensus. Typically, one node acts as a leader [19], [20] among the servers to dictate the order of events to the other machines to ensure a total order and exact replicas (e.g., Apache ZooKeeper [21] and Apache Kafka [22]).

Albeit simple and intuitive, this method does not scale well when the load increases rapidly, due to the performance bottlenecks created by the presence of a single coordinator. Even worse, quorum-based consensus may need higher resource consumption in case of a failing or missing leader. Additionally, the latency due to the geo-distribution drastically impairs the communication performances required for these algorithms and the need to rely on quorum mechanisms in case of failures prevents strong consistent systems from providing high availability on wide area networks subject to delays and transient partitions [23], [24].

Taking into consideration the changes in system scales, the opposite approach of *eventual consistency* favors availability over safety when replicating data. Some major middleware systems (e.g., Apache Cassandra [25] and Riak [26]) provide eventual consistency by construction. As eventual consistency requires less coordination between the nodes, latency and faults have less impact on the operation of the system. Eventual consistency unfortunately suffers from its own disadvantages. Its most important weakness is the likeliness for users to experience transient inconsistencies or observe stale state, which goes against a fluid and seemingly natural user experience. Strong consistency nowadays is not a de facto standard when designing a system but rather a costly property that should be achieved only when necessary.

The recent emergence of large-scale fault-tolerant distributed ledgers has however challenged our understanding of the tension between scalability and consistency: systems such as Bitcoin [27] offer solutions that are strongly consistent with high probability, while scaling in principle to an arbitrary and dynamic number of participants.

1.4 Permanent data with distributed fault-tolerant ledgers

Distributed ledgers appeared 10 years ago with Bitcoin [27], and have since made their mark in the media and in institutions [28]. Behind the hype fueled by financial speculation, Bitcoin and its underlying technology, the blockchain, enable radically new services to exist. The best known examples of these services are decentralized platforms for transfers of value, also known as cryptocurrencies (e.g., Dogecoin [29], Monero [30]), and decentralized computing platforms (e.g., Ethereum [31]) that allow decentralized applications to execute, providing services such

as automated contracts and decentralized autonomous organizations.

From a system point of view, blockchains are a new family of decentralized systems which operate a robust agreed-upon ledger with read and write accesses that may be public. A key property of blockchains is their *verifiability*: any joining member can ensure for herself the correctness of the ledger by downloading and verifying all of its history. The core novelty behind Bitcoin resides in its open-membership consensus, eponymously referred as the Nakamoto consensus, which ensures the consistency of the ledger. Unlike classic Byzantine fault-tolerant consensus [32], the Nakamoto consensus does not need to know how many participants are in the system, making it the first open-membership consensus protocol. The Nakamoto consensus offers new perspectives to decentralized systems by making accessible to the public the most powerful abstraction from distributed computing that is consensus.

However, the Nakamoto consensus is not a perfect solution either. Its main drawback relevant to this thesis is the lack of determinism in this consensus. The Nakamoto consensus only provides probabilistic consistency guarantees to the ledger it protects, in the sense that recent consensus decisions have a non-zero probability of being reverted. Another highly debated issue revolves around the core mechanism used by Bitcoin-inspired blockchains, which requires enormous computing resources, and lends more decision power to participants with more computing power. This mechanism encourages participants to compete in a race where the highest computational power, hence highest energy consumer, wins.

Thanks to their decentralized consensus and the fact that anyone can verify their correctness, blockchains are described as trustless systems. No particular member needs to be trusted for the system to remain correct, thus following the "*Don't trust, verify.*" motto.

We chose in this thesis to establish a clear distinction between decentralized and centralized ledgers—in the sense of centrality of control—and to denote blockchains only as decentralized ledgers. Centralized ledgers, also called permissioned blockchains (e.g., HyperLedger Fabric [33], R3 Corda [34]), operate with a known number of participants and can therefore benefit from the decades of both theoretical and practical work invested in consensus among known participants. Both decentralized and centralized ledgers do provide a similar service, but whereas decentralized ledgers needed a new type of consensus to exist, centralized ledgers could have emerged regardless of that novelty. From a distributed systems' standpoint, decentralized ledgers present a breakthrough in technology while centralized ledgers are the outcome of successive optimizations.

1.5 Research challenges and thesis contributions

The general challenge we explore in this thesis is the need to strengthen coordination in a decentralized system without delegating decisions to a central coordinator that could limit the

scalability of the system. Interpreted in the context of data replication, this challenge is reduced to achieving the best possible consistency guarantees while remaining in a weak consistency model that is imposed by the lack of centrality.

Additionally, we defend the claim that the increase in device heterogeneity and system size calls for systems that offer heterogeneous levels of quality of service. Our vision is that we should be able to build decentralized systems with guarantees that are personalized for each device.

To fulfill this vision, we harness the tension between data consistency and availability in data replication for personalization. We must also consider that, in the era of ubiquitous computing, devices have bounded capacity and would benefit from personalization that restrain resource usage such as computation and bandwidth.

The overall approach of our work is to develop novel mechanisms providing personalizable trade-offs to well-known algorithms and systems. To that end we propose two contributions. The first one focuses on personalizing consistency and latency in epidemic broadcast protocols and the second one focuses on personalizing security and resource consumption requirements in blockchains. We detail each contribution below.

1.5.1 Gossip Primary-Secondary

Our first contribution consists of a combination of two novel protocols, dubbed *Gossip Primary-Secondary* (GPS) and *Update consistency Primary-Secondary* (UPS), which rely on epidemic dissemination to offer differentiated guarantees of data consistency and update delivery latency to a system's participants.

To enable personalization, the network is divided in two distinct subsets with opposing performances: a small subset of primary nodes (e.g., 1 %) experiences lower message latency, while the larger subset of secondary nodes experiences better message ordering leading to stronger data consistency. Explained shortly, the GPS protocol ensures that primary nodes receive messages first then direct the dissemination towards secondary nodes. The better message ordering is achieved by having the set of primary nodes acts as a soft message orderer for secondary nodes: by receiving messages first and caching them shortly before forwarding them, primary nodes effectively remove some of the randomness generated by the chaotic nature of gossip protocols. Instead of using a central coordinator, we use a whole subset of nodes as a decentralized soft coordinator and obtain clear consistency gains for secondary nodes.

1.5.2 Dietcoin

Our second contribution also considers a problem arising from heterogeneous capacity in data replication systems, but in the context of Bitcoin. The Bitcoin network is composed of two

types of nodes: full nodes which check the correctness of the entire blockchain during bootstrap, and light nodes (also called SPV nodes in Bitcoin) which only check the difficulty of each block in the chain during bootstrap and assume that a difficult enough block is a correct block. This assumption is motivated by the prohibitive cost of a full node bootstrap on low resource devices. This cost is represented by the download and verification of far more than 150 GiB of data (as of mid 2018).

To address the inherent limitation of existing light node verification, we propose *Dietcoin* and its *diet nodes* which expend the capabilities of Bitcoin light nodes by enabling them to verify the correctness of a block or a subchain of blocks. A diet node can securely query parts of the UTXO set (i.e., the aggregated state of the chain) to perform said verification while keeping low bandwidth requirements. For instance, a diet node willing to increase its trust in the latest block of the chain can choose to verify the latest ℓ blocks to ensure that the block of its interest is built upon a correct subchain and is thus more likely to be part of the correct chain of Bitcoin. The mechanisms used in Dietcoin also enable fast full node bootstrap and efficient double-spending checks.

1.6 Thesis outline

Before we present in details the contributions brought in this thesis, we first cover in the next chapter some background knowledge this thesis builds upon. We then present our first contribution *GPS* and *UPS* and evaluate it in Chapter 3. Our second contribution *Dietcoin* is presented in Chapter 4. Finally, we conclude and propose some perspectives in Chapter 5.

CHAPTER 2

BACKGROUND

This chapter serves as a foundation for the remaining of this thesis. We overview the fundamental background work at the base of this thesis as well as more recent and relevant work. We first survey the concepts and definitions related to consistency in Section 2.1 and the consensus abstraction in Section 2.2. We present in Section 2.3 the most recent usages of consensus in distributed ledgers, and we continue by describing Bitcoin in depth in Section 2.4. Lastly, we detail gossip protocols in Section 2.5. We finally summarize this chapter and anchor our contributions within the existing background work in Section 2.6.

2.1 Data consistency

As stated in the Introduction, consistency is a key aspect of data replication. Consistency ensures that all replicas of an object hold the same value; it abstracts the replication into one unique virtual entity: the replicated object. To keep all the replicas consistent with one another, operations on the object must be applied in the same order on every replica. We talk about a consistent *history of operations* if all the nodes of a system experience events in the same order. Inconsistencies therefore happen when a node experiences events in a different order than expected.

More formally, each data type (e.g., an integer, a set, a queue) has its own *sequential specification*. The sequential specification of a data type T is the set of histories of operations that are valid when executed in a sequential manner on an object of type T. For instance, the history $< \mathrm{push}(1), \mathrm{push}(2), \mathrm{pop}() \rightarrow 1, \mathrm{pop}() \rightarrow 2 >$ is a valid sequential history of a queue and therefore respects its sequential specification. The same history however doesn't respect the sequential specification of a stack.

The sequential specification of a data type is however insufficient to define its behavior in a distributed setting. In a distributed setting, operations must be projected into a single history to determine whether they respect the sequential specification of the data type. The set of valid projections are defined by the used *consistency criterion*. Consistency criteria are typically sorted into strong consistency criteria or weak consistency criteria.

2.1.1 Strong consistency

Strong consistency groups all consistency criteria that enable replicas to hold identical values of an object, which notably include sequential consistency and linearizability.

Sequential consistency [35] is a criterion in which every operation on the object are seen in the same order by all the nodes. The local order of operations is respected but the order of concurrent operations may be rearranged to fit the sequential specification.

Linearizability [36] enriches sequential consistency by adding a real time property to the operations of a replicated object. Each operation appears as if instantaneous in their linearization point that is located between the start and end of the operation. The order of operations seen by each node is fixed and is determined by the time of each operation linearization point.

2.1.2 Weak consistency

On the other end of the consistency spectrum lie weak consistency criteria. These criteria do not provide the same safety guarantees on replicated objects that strong consistency enable. But by relaxing the consistency guarantees, weak consistency criteria offer better availability guarantees. This direct trade-off is formulated in the pivotal CAP theorem [23], [24] which demonstrates the innate balance between Consistency, Availability (i.e., performances) and Partition tolerance in data replication protocols. Pushing the lever in one direction to strengthen the guarantees on one of these properties results in weakening the guarantees on (at least one of) the other properties.

Causal consistency [37] is enabled by having a causal order on partial histories of operations. This criterion is characterized by the happens-before relation between operations as defined by Leslie Lamport [38] and is therefore limited to partial histories. Causal consistency only provides order on partial histories without specifying any order for concurrent operations. Causal consistency provides some level of guarantees of local consistency but since no total order of operations is obtainable in the presence of concurrent operations, this criterion does not provide global consistency guarantees. Since only local orders are guaranteed, causal consistency is more affordable than strong consistency criteria and is hence an interesting choice for large-scale systems [39]–[42].

Eventual consistency [43], [44] is the weakest consistency criteria. It only ensures that convergence is eventually attained once updates on an object cease for long enough. The timing implications of eventual consistency are further discussed in Chapter 3.1. A formalization of eventual consistency for any data type has been realized in the form of Update-Query Consistency [45] that we develop in Section 2.1.3 due to its importance for Chapter 3. As eventual consistency is the criterion that makes the weakest assumptions, it is the easiest to guarantee in large-scale systems.

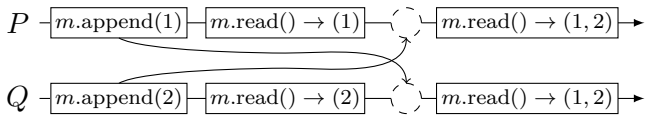


Figure 2.1 – An eventually consistent append-only queue.

Order-free consistency

Eventual consistency can also be achieving by removing the existence of conflict-inducing concurrency in the data type specification. Conflict-free Replicated Data Types (CRDTs) [46]–[49] have emerged as such objects; provided that all the replica have the same initial state and set of operations of an object, they then own a consistent replica of the object regardless of the order of operations. There is however no generic CRDT that functions for any data type; developers must therefore produce additional effort as they are limited by the available data types.

2.1.3 Update consistency

We focus in Chapter 3 on update consistency [45] applied on a specific data type. Let us consider the append-only queue object m of Figure 2.1. Its sequential specification consists of two operations:

— $\mathrm{append}(x)$, appends the value x at the end of the queue;

— $\mathrm{read}()$, returns the sequence of all the elements ever appended, in their append order.

When several node update the queue, update consistency requires the final convergence state to be the result of a total ordering of all the append operations which respects the program order. For example, the scenario in Figure 2.1 satisfies update consistency because the final convergence state results from the ordering $< m.\mathrm{append}(1), m.\mathrm{append}(2) >$, which itself respects the program order. An equivalent definition [45] states that an execution history respects update consistency if it contains an infinite number of updates or if it is possible to remove a finite number of reads from it, so that the resulting pruned history is sequentially consistent. In Figure 2.1, we achieve this by removing the read operation that returns 2.

Unlike CRDTs that rely on conflict-free operations, the generic update consistent data type [45] exploits a broadcast operation together with Lamport clocks, a form of logical timestamps that makes it possible to reconstruct a total order of operations after the fact. Relying on this after-the-fact total order allows update consistency to support conflicting operations, like append-only queue in this case.

We show in Algorithm 1 of Chapter 3 an application of the generic update consistent data type to the append-only queue of Figure 2.1.

2.1.4 Hybrid consistency

A large number of works have looked at hybrid consistency conditions, originally for distributed shared memory [50]–[52], and more recently in the context of geo-distributed systems [53]–[57].

Fisheye consistency [53] provides a generic approach in which network topology affects the choice of consistency criterion. Nodes that are topologically close satisfy a strong consistency criterion, such as sequential consistency, while remote nodes satisfy a weaker one, such as causal consistency.

Salt [54] enables the coexistence of both strongly consistent transactions and eventually consistent transactions within the same database. Salt refines the isolation level of transactions to permit concurrency between targeted eventually consistent transactions while ensuring strong isolation for the others.

Terry et al. [55] describe the session guarantees provided by the Bayou system. As they are accessing the same database, applications observe a consistent history while they each may have different session guarantees.

RedBlue consistency [56] offers a trade-off in consistency and speed of operations. Blue operations are fast and eventually consistent while red operations are slow and strongly consistent.

Incremental consistency [57] helps developers with a new abstraction that provides several levels of consistencies for all operations. Each operation returns several results over time, starting from fast results with weak consistency guarantees to slow results with stronger guarantees.

Our first contribution presented in Chapter 3 proposes differentiated trade-offs in consistency and latency to different nodes within the same network.

2.2 Consensus

Consensus is a universal object in distributed computing since all the other synchronization objects can be expressed with the consensus object [58]. Consensus enables a set of processes—nodes can be abstracted as processes—to agree on a value. Consensus protocols enable strong consistency in the context of data replication; they notably play an important role in distributed ledgers which we explore in Section 2.3.

More formally, a consensus is started on a value v when a process invokes $\mathrm{propose}(v)$. Upon completion of the consensus, all correct processes may decide on v. The consensus abstraction is defined by the following four properties (taken from Chapter 14 of Michel Raynal's 2013 book [58]):

— Validity: a decided value is a proposed value;
— Integrity: a process decides at most once;
— Agreement: no two processes decide different values;
— Termination: an invocation of $\mathrm{propose}()$ by a correct process terminates.

There are various assumptions that can be made on the settings in which consensus operates, assumptions that change the performance and above properties of the consensus. The consensus abstraction depends on the assumptions made on processes and on the communication channels. In message-passing consensus, we denote the following possible assumptions: failure model, synchrony model and the use of signatures.

The **failure model** describes the types of incorrect behaviors that can be assumed from processes. Process may for instance gracefully stop, suddenly stop (crash) or even behave arbitrarily (Byzantine [32]).

The **synchrony model** describes the timing properties of the communication channels between processes. While synchronous communication channels have a known fixed latency upper bound, asynchronous channels have no latency upper bound at all. Several other synchrony models lies in between synchronous and asynchronous systems. Partial synchrony [59] for instance uses an existing yet unknown upper bound on latency, while eventual synchrony [60] provides an upper bound for messages if they are sent after a certain time threshold.

Using signatures enable the sources of relayed messages to be identified which can reduce the overall number of messages needed for consensus in the presence of Byzantine processes. Signatures incur a computation overhead for processes but may reduce the overall complexity in number of messages as well as help counter Byzantine processes.

Failure bounds

In a system of n processes that is at least eventually synchronous and where f processes crash, the upper bound on the number of tolerated faults is $n = 2f + 1$. On the other hand, assuming that processes can be Byzantine, the threshold of tolerated failures lowers at $n = 3f + 1$ [32].

2.2.1 Consensus implementations

Paxos [19] relies on quorums of nodes to provide consensus within a crash-tolerant system. Since nodes are assumed honest, a single node can oversee and facilitate the consensus. After receiving a request from a client, a proposer initiates a two-phase commit with a quorum of acceptors. In the first phase, the proposer sends the request to the quorum and aggregates the quorum answers. In the second phase, the proposer sends the result of the decision to the quorum that acknowledges the changes in its reply to the proposer, which ends the protocol.

Since the original Paxos publication, plenty of variations have been proposed to make the protocol faster, cheaper, and even Byzantine-resilient. In the same vein as Paxos lies Raft [61], a protocol with equivalent performance to Paxos but which advocates simplicity of design for easier implementation and analysis.

Practical Byzantine-Fault Tolerance [20] (PBFT) does not assume honest participants. PBFT depends on a primary node acting as leader to order the requests it receives from clients. Once a received request is ordered, the primary initiates a three-phase commit on this request with a quorum of backup nodes. The added phase compared to Paxos ensures the safety of the protocol in case of primary failure. Contrary to Paxos however, no node oversees the consensus and all messages are broadcast within the quorum to let every member in the quorum decide locally.

2.3 The anatomy of distributed ledgers

Distributed (digital) ledgers are distributed systems that securely store records of data though time, preventing the records from being tampered by anyone. Like their physical counterparts, distributed ledgers strengthen the trust relations between their users by making them accountable for the records they author. Users can consult the history of records and build irrefutable proofs of wrongdoing by other users, thus inciting correct behavior from everyone to avoid repercussions. In addition to a posteriori control, records are automatically verified upon insertion into the ledger to guarantee their conformity to a set of predefined rules.

Because they involve multiple nodes, the content of distributed ledgers must be agreed-upon by every participating node and must cope with malicious behaviors. Distributed ledgers can therefore be abstracted as byzantine-fault tolerant replicated state machines [38] with additional validity constraints [62] on the state they hold. The ledger abstraction only offers two operations to preserve its state: reading a record from the ledger, and appending a record to the ledger. We note that ledgers have a similar sequential specification to the append-only queue described in Section 2.1.3. We refer our readers to Chapter 16.7 of Michel Raynal's 2018 book [63] for an in-depth look at ledgers from a distributed computing perspective.

Distributed ledgers are formed by a combination of the following components:

1. The **membership protocol** dictates who can commit records to the ledger, which can range from a centrally managed authority to an open-membership protocol;
2. The **consensus protocol** enables the nodes handling the ledger to agree on a common version of its content, it can provide deterministic or probabilistic guarantees;
3. The **record ruleset** defines rules that records must follow to be appended to the ledger; records may only hold plain data, but may also contain executable scripts written in languages with varying expressive power, the extreme being that of a Turing-complete language.

In the following we first cover each of these components in turn (Sections 2.3.1-2.3.3), before discussing blockchains (Section 2.3.4).

2.3.1 Sybil-proof membership protocols

The membership protocol specifies which nodes are able to append the ledger with new committed records; the membership protocol therefore defines whom controls the ledger. Membership protocols must be resilient to Sybil adversaries to prevent takeovers of the ledger by malicious nodes. There currently exists several such protocols, each enabling its own distribution of control on ledgers. Additionally, membership protocols can be combined to provide a

hybrid distribution of control. We note that the main difference between permissioned (private) and permissionless (public) ledgers resides in the membership protocol they use.

In the rest of this section we discuss three common approaches to realize Sybil-proof membership protocols: relying on a trusted authority, based on Proof-of-Stake, or based on Proof-of-Work.

Trusted authority

The classic approach to manage membership consists in a sole entity that takes the role of authority on the network. Since the authority single-handedly decides the membership of the distributed ledger, it retains a tight control over it. In particular, the authority must prevent Sybil attacks and must cope with the presence of malicious nodes in the network. Thanks to its tight control over the membership, the authority can offer an equal participation weight to each node in the voting process.

However, the strong coupling in this membership protocol poses a threat to the well-being of the entire system in case of dishonest behavior from the authority. Decentralized membership protocols offer robust alternatives by removing the need for trust in a potentially flawed authority.

Proof-of-Stake membership

Proof-of-Stake membership prevents the existence of Sybil adversaries by using coins that are impossible to copy instead of using duplicable identities subject to counterfeit. A node in Proof-of-Stake membership is therefore represented by its stake (i.e., the amount of coins it owns), and its participation power weighted by the size of its stake. The coins are either created at the bootstrap of the ledger, or over time with an agreed-upon rule such that all nodes agree on the total amount of coins in the membership at all time.

Any member owning enough coins can share a fraction of its coins with an outsider node to let them join the membership. This ability that any member has to invite other nodes in the membership drastically differs from a trusted authority-based membership in which only the authority has inviting capabilities. Proof-of-Stake is therefore a closed-membership protocol managed in a decentralized fashion by all its participants.

Proof-of-Work membership

Proof-of-Work prevents Sybil attacks by relying on the computational resources of nodes. Nodes must prove their authenticity by solving a computational puzzle [64], [65] that is probabilistically lengthy and costly to solve, but fast to verify. Proof-of-Work therefore anchors the membership protocol into the physical world by enforcing the consumption of energy to participate in the protocol.

Alike Proof-of-Stake, each node's participation is weighted by its computational power. Contrary to Proof-of-Stake however, the ability to join the membership depends on requirements external to the ledger. The only needed requirement to join the membership is the ability of a node to perform computation. Because Proof-of-Work membership does not need any setup, it is open to everyone.

2.3.2 Consensus protocols

The second component of distributed ledgers, consensus protocols, have already been discussed in Section 2.2 of this chapter. In particular we saw that the most famous deterministic consensus protocols use quorums to avoid concurrent rounds.

Alternatively to deterministic consensus protocols, several cryptocurrencies (e.g., Bitcoin, Algorand) use probabilistic consensus for the creation of blocks. Each new block is created by a random node in the membership with a weighted probability proportional to its stake/work. Consensus is formed on the new block if no concurrent block is created, which happens with high probability. This block creation process resembles a random leader election with a new election for every block.

Different combinations of Sybil-proof membership protocols and consensus protocols are possible depending on which properties one wishes to achieve. For instance, the following combinations have been proposed in the literature:

— Proof-of-Work + random leader election: Bitcoin [27], Ethereum [31], Bitcoin-NG [66]
— Proof-of-Work + quorum: PeerCensus [67], ByzCoin [68], OmniLedger [69], RapidChain [70]
— Proof-of-Stake + random leader election: Algorand [1] [71], Ouroboros [72], [73]
— Proof-of-Stake + quorum: Tendermint [74]
— Single authority + quorum: HyperLedger Fabric [33], R3 Corda [34]

In particular, the Nakamoto consensus, eponymously dubbed after the author of Bitcoin, is a singular combination of Proof-of-Work membership and random leader election. A main property of the Nakamoto consensus is its probabilistic behavior regarding recent committed records that can still be revoked with low probability. While it may appear as a purely negative property, it also enables resilience against temporary faults and it provides eventual convergence regardless of the power of malicious nodes as long as the attacks remain transient.

Consensus protocols used by distributed ledgers in production and in particular in cryptocurrencies have been extensively studied by the distributed computing community [75]–[77].

1. Algorand relies on random committees—not on one single leader—selected via cryptographic sortition.

2.3.3 Record rulesets

Records must fit agreed-upon rules before being committed in the distributed ledger. Since a ledger is used to improve the trust between its users, the more verifiable a record, the less the need to trust the record and its author. The record ruleset of Bitcoin for example ensures that coins cannot be spent multiple times and that coins can only be spent by their owners.

Records can range from plain data to computable scripts. For instance, financial transactions in Bitcoin are written in a limited scripting language with access to a local stack that is used to sign coins and verify the integrity of signatures. Ethereum on the other hand enables so-called *smart contracts*, which are written in more expressive language, and are thus more powerful than Bitcoin's transactions from a computability perspective. Enabling the storage of Turing complete records effectively transforms the distributed ledger into a virtual machine executed by every participating node.

Records may contain sensitive information and storing these records in a distributed ledger exposes its author to the rest of the system. As a countermeasure, records can hide sensitive content with zero-knowledge proofs (e.g., zk-SNARKs [78], [79], zk-STARKs [80]), as in zkLedger [81], to increase the privacy of the users linked to the committed records.

2.3.4 Blockchains

The differences between the definitions of blockchains and distributed ledgers are not yet clear within the community. Whereas we denote in this thesis distributed ledgers as the overall abstraction, blockchains are instances of such distributed ledgers in decentralized environments.

The term blockchain comes from its data structure. Its chaining is designed to be verifiable, alongside the records it stores, when used in pair with the Nakamoto consensus. Blocks contain records that are secured from tampering once the block is appended to the blockchain. Appending a block to the chain is equivalent to committing the records it contains in the ledger.

Because Bitcoin plays a central role in the second contribution of this thesis (Chapter 4), we discuss it and its blockchain in detail in the next section of this chapter (Section 2.4).

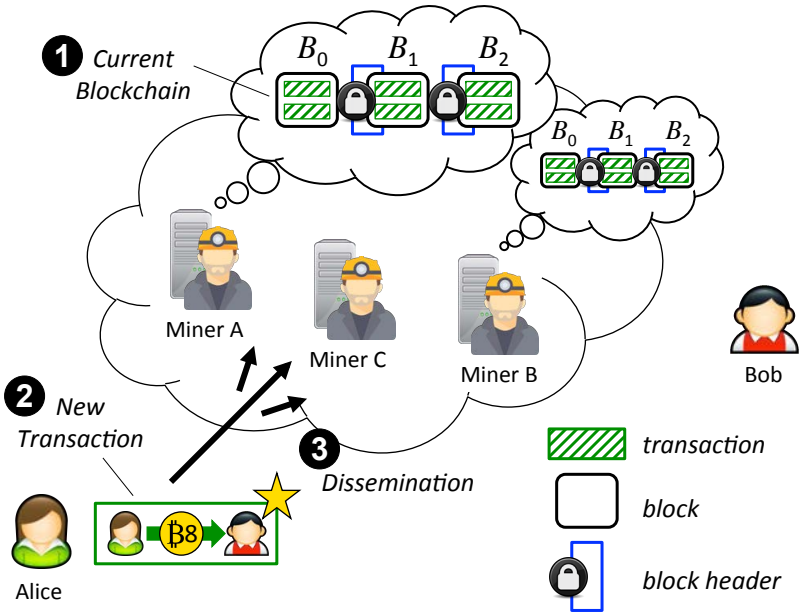


Figure 2.2 – The Bitcoin blockchain is formed of a sequence of blocks containing transactions. The current state of the chain (here (B_0, B_1, B_2)) is stored by each individual miner.

2.4 Bitcoin

After presenting the abstraction of distributed ledgers in Section 2.3, we expose in this section the detailed inner workings of Bitcoin as well as some of its limitations. Our second contribution Dietcoin, depicted in Chapter 4, builds upon Bitcoin.

2.4.1 Overview

Bitcoin uses a blockchain combined with Nakamoto consensus to secure the transactions it stores. The transactions, the blocks, and the resulting chain obey a few core rules that ensure the system remains tamper-proof.

The blockchain proper ($(B_i)_{i\in\mathbb{N}}$, label ❶ in Figure 2.2) is maintained by a peer-to-peer network of miners. Each block B_i links to the previous block B_{i-1} by including in its header a cryptographic hash that is (i) easy to verify, but (ii) particularly costly to create. The leftmost block B_0 is known as the *genesis block*: it is the first and oldest block in the blockchain, and is the only one with no predecessor.

Recording a new transaction

To transfer 8 bitcoins from herself to Bob, the user Alice must first create a valid transaction (label ❷) that contains information proving she actually owns the 8 bitcoins (using a crypto-

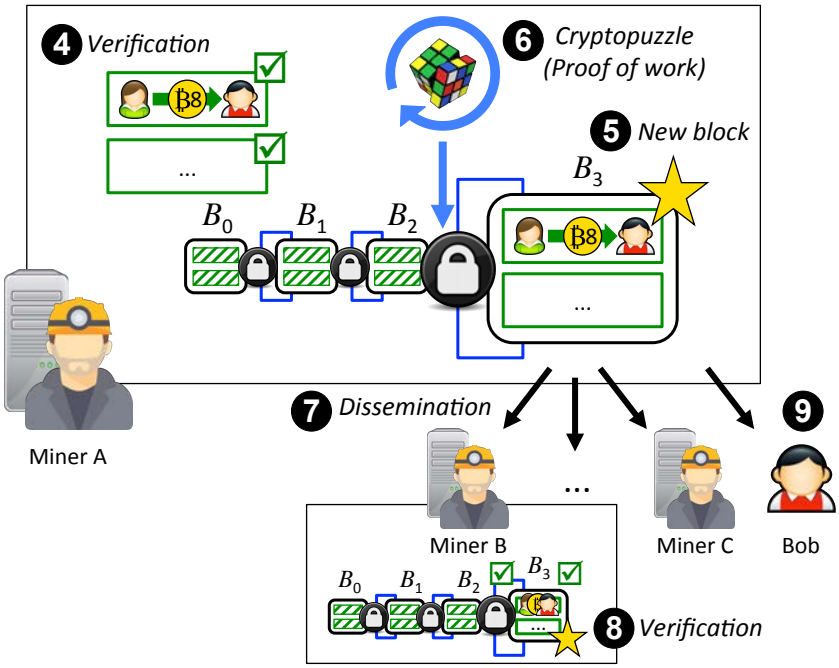


Figure 2.3 – To add a new transaction to the Bitcoin blockchain, a miner first verifies the validity of the transaction. It must then solve a costly cryptopuzzle to encapsulate this transaction in a new block (here B_3), before disseminating this block to other miners.

graphic signature), and encode the resulting transaction output with Bob's public key (such that, in turn, only Bob will be able to demonstrate ownership of the transaction's output).

Alice then broadcasts this new transaction to the network of miners ❸, in order for it to be included in the blockchain. Before adding Alice's transaction into the blockchain, Miner A first verifies that the transaction is valid (label ❹ in Figure 2.3, a point we revisit below).

Miner A then includes Alice's transaction together with other transactions received in parallel into a new block (B_3, ❺), and attempts to link it to the current tip of the blockchain. This linkage operation requires Miner A to solve a probabilistically difficult cryptopuzzle ❻ that regulates the frequency at which blocks are created (or *mined*) by the whole network. (In Bitcoin, this periodicity is set to one block every 10 minutes.) If Miner A succeeds, the new block B_3 is now linked to the existing block chain (B_0, B_1, B_2), and is disseminated to the other miners ❼. When a miner receives a new block (here Miner B), it checks that the transactions included are valid, along with the cryptopuzzle, before including the new block in its local copy of the blockchain. The new block ultimately reaches Bob ❾, who can check that the transaction has been properly recorded.

Miners are incentivized to produce blocks by receiving some new bitcoins for each new block they create (which constitutes Bitcoin's money creation mechanism), and by receiving fees from users (such as Alice here) whose transactions they have committed in the chain.

Irrevocability of deep blocks

Because blocks are produced at a very slow rate compared to network latency, all miners are extremely likely to have a consistent view of the blockchain. The views of individual miners may however diverge in problematic cases, either because of network delays and failures or malicious behavior, causing branches to appear. When a branch occurs, honest miners resolve the divergence by choosing as valid branch the one that was the most difficult to create. Blocks that are left out of the chain are said to be *orphan*.

The risk of being made orphan decreases exponentially as a block lies deeper in a chain, ensuring the *practical irrevocability* of deep blocks and the transactions they contain [82].

2.4.2 Transactions, blocks, and UTXO set

To benefit from the full security of Bitcoin, Bob should verify the validity of the new block that contains Alice's payment to him (label ❾ in Figure 2.3) in addition to verifying the validity of Alice's transaction. This is because Alice could collude with a miner (or launch herself a miner), and produce an invalid block that she would advertise to Bob. Bitcoin relies on a number of built-in validity checks on blocks, transactions, and an intermediary set known as the set of *Unspent TransaCTion Outputs* (UTXO set) to conduct this verification. However, whereas *full nodes* exploit all of these checks, *Simple Payment Verification nodes* (SPV nodes) only perform a limited verification.

Checking block validity

A block is valid if and only if it meets the following two conditions.

— **(BV1)** Its header respects the blockchain's Proof-of-Work difficulty predicate;

— **(BV2)** It only contains valid transactions.

BV1: The Proof-of-Work difficulty predicate makes it very difficult for malicious actors to alter the blockchain in an attempt to edit the ledger. It is enforced on each block header, whose simplified structure is shown in Figure 2.4. The header of each block B_k points both to the header of the previous block B_{k-1} (using a hash function, ❶), and to the transactions contained in the current block B_k ❷. To fulfill the difficulty predicate ❹, a header must contain a *nonce* ❸ such that the hash of the header is less than a *difficulty target*. The difficulty target is adjusted regularly so that a new block is created every ten minutes by the miners as a whole, regardless of the computation power of the overall network. Finding such a nonce is computationally very expensive, which prevents attackers from easily tampering with the chain.

To establish a secure and verifiable link between the header of B_k and B_k's content, the pointer to B_k's transactions ❷ consists of the root of a *Merkle tree*. A Merkle tree is a hashing

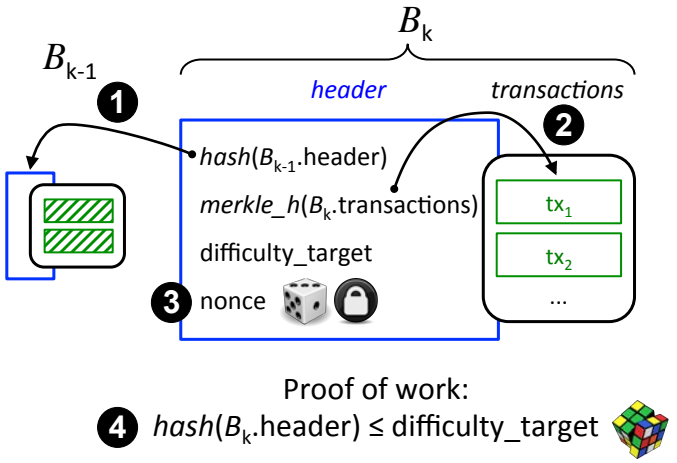


Figure 2.4 – Content of a Bitcoin block header (simplified).

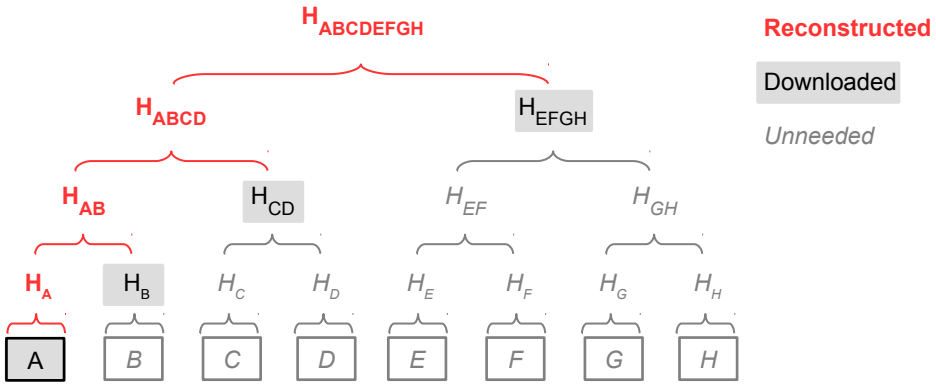


Figure 2.5 – Example of a Merkle tree root reconstruction that only needs $log(n)$ hashes.

mechanism for finite sets that allows a verifier to efficiently test whether an item (here a transaction) belongs to the initial set. The Merkle tree of a set S is constructed by first hashing each of its elements ($(H_X)_{X\in\{A,..,H\}}$ in Figure 2.5), and then recursively hashing pairs of hashes, until a final *root hash* ($H_{ABCDEFGH}$) is produced (at the top of Figure 2.5). Only this root hash is kept. To verify the presence of transaction A in S, a node then only needs to (i) download the root $H_{ABCDEFGH}$ from a secured communication channel (e.g., the blockchain), (ii) obtain the three intermediate hashes shown in red (H_B, H_{CD} and H_{EFGH}) from full nodes storing blocks, and (iii) reconstruct the root from the downloaded hashes.

BV2: In addition to the Proof-of-Work difficulty predicate (**BV1**), all the transactions included in a block must also be valid for the overall block to be valid. A typical Bitcoin transaction is made of inputs (❶ in Figure 2.6, owned by Alice) and outputs ❷ (to Bob and Tux in this example), complemented by metadata and cryptographic proofs. Only coins created in earlier transactions may be spent: each of Alice's inputs therefore points back to the output of an earlier transaction

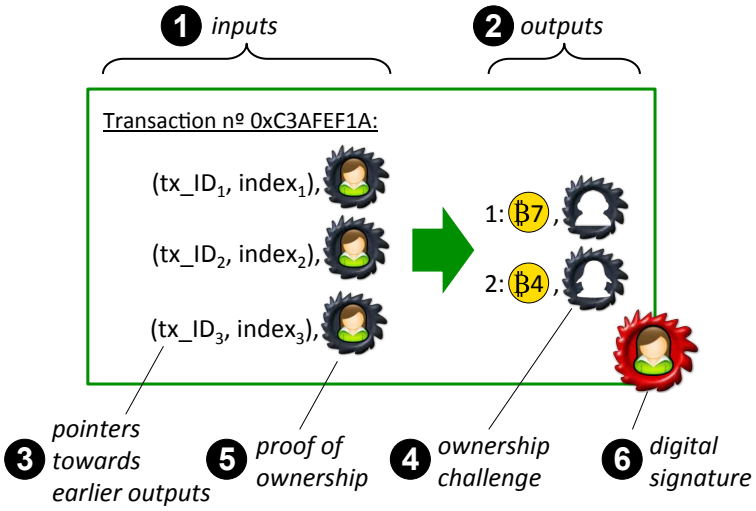


Figure 2.6 – Structure of a Bitcoin transaction.

❸. To ensure that only the recipients (Bob and Tux) are able to spend the output, each new coin contains an ownership challenge (a hashed public key), that must be solved to spend this coin ❹.

Alice's transaction is only valid if the following three conditions are met:

— **(TV1)** The inputs do exist, and Alice owns them;

— **(TV2)** No money is created in the transaction:

$$\sum_{in \in \mathsf{inputs}(t)} \mathsf{value}(in) \geq \sum_{out \in \mathsf{outputs}(t)} \mathsf{value}(out).$$

The difference $\sum \mathsf{value}(in) - \sum \mathsf{value}(out)$ is given as a fee to the miner of the block containing the transaction;

— **(TV3)** The transaction's inputs $(\mathsf{tx\_ID}_i, \mathsf{index}_i)$ have not been spent yet.

The set of Unspent TransaCTion Outputs (UTXO set)

While the validity of a block's header (**BV1**) only requires access to the current block B_k, and to the header of its predecessor B_{k-1}, verifying transactions (**BV2**) requires access to the transactions recorded in earlier blocks. Worse, verifying that inputs coins have not yet been spent (**TV3**) potentially requires parsing and verifying the entire blockchain.

To avoid performing such a costly operation for each new block, nodes that verify transactions maintain an intermediary set known as the set of *Unspent TransaCTion Outputs* (UTXO set). The UTXO set contains all spendable coins, i.e. the coins that have been created in the chain but not spent yet.

A node verifying transactions can prevent double spends (**TV3**) by simply ensuring that all the inputs of a transaction appears in its UTXO set. The UTXO set evolves as new correct blocks are added to the chain: transaction outputs are removed when they are spent, and new transaction outputs are added.

Despite its benefits, maintaining a local UTXO set is costly: in order to obtain the set, a node must download the entire chain (comprising hundreds of thousands of blocks) and validate it.

Because of this cost, Bitcoin supports several levels of verification. *Miners* and users running *full nodes* construct the UTXO set and check both block headers (**BV1**) and transactions (**TV1,2,3** and as a result **BV2**). Because they perform all the possible checks, full nodes benefit from the maximum security that the Bitcoin system has to offer.

By contrast, *Simple Payment Verification* nodes (SPV nodes) do not construct the UTXO set. Instead, they only download the chain's block headers (rather than full blocks), and verify that these headers are valid (**BV1**). With the headers only, SPV nodes are able to verify the well-formedness of the blockchain including the crucial Proof-of-Work difficulty predicate that seals the links of the chain.

2.4.3 Limitations of Bitcoin

In spite of its many innovations at the time it was first introduced, Bitcoin suffers from a number of important limitations, some of which have since then been addressed by other competing systems. We discuss the most important of these limitations below.

SPV nodes partial verification

Because they limit themselves to checking the difficulty of the blockchain, SPV nodes are unable to detect that a new block contains an invalid transaction. They must rely on full nodes to ensure the correctness of transactions. The second contribution of this thesis, Dietcoin presented in Chapter 4, proposes mechanisms to enable SPV nodes to verify the correctness of transactions.

Reward unfairness

As shown in Bitcoin-NG [66], the block reward is not fairly distributed among Bitcoin miners due to both the low rate of block creation and the size of blocks. Additionally, selfish mining strategies [83], [84] reduces fairness even more by having miners retain blocks, instead of immediately broadcasting them, to waste their competitors resources while they mine on an outdated chain tip. Selfish strategies can be enhanced by combining them with eclipse attacks [85], [86] to partition competitor miners from the rest of the network and control their environment.

Security considerations

Gervais et al. [87] have quantitatively studied selfish mining and double-spending strategies in several cryptocurrencies including Bitcoin. Karame et al. [88] showed the ease and efficiency of double-spending with fast payments (uncommitted transactions). As SPV nodes must send a bloom filter of their public keys to full nodes to register for committed transactions affecting them, they are at risk of displaying their public keys to others [89].

2.4.4 Scalability improvements

The scalability of Bitcoin in particular and distributed ledgers in general has attracted considerable attention as deployed ledgers have progressively attained their intrinsic limits. Several publications have notably surveyed the challenges in scalability of distributed ledgers [90] and of Bitcoin [91]–[93] in particular.

Among the approaches proposed to overcome these limits, Bitcoin-NG [66] decouples the leader election block from transaction blocks. By tuning the frequency of block creation and the size of blocks, Bitcoin-NG reaches near optimal number of transaction per second and greatly enhances the fairness of block reward distribution.

ByzCoin [68] merges PBFT [20] and collective signatures [94] with Bitcoin-NG to produce a strongly consistent version of Bitcoin [95]. The voting committee is formed by selecting the last x block miners ($x = \{144, 1008\}$ in the evaluation) while collective signatures are transmitted via a broadcast tree to optimize both computation and bandwidth costs.

Chainiac [96] integrates a skiplist into a blockchain to produce a skipchain, a blockchain with both backward and forward long-distance links. Chainiac relies on collective signatures to implement forward links, which are not available in Nakamoto consensus chains such as Bitcoin.

Trustchain [97] lowers the transaction validity requirements to improve the scalability of the distributed ledger. Each node has its own ledger which is regularly synchronized thanks to checkpoints with all the other per-node ledgers in the system. Contrary to most distributed ledgers, Trustchain allows invalid transactions to exist, letting the applications handle accountability in case of wrongdoing.

While payment channels [98], [99] increase the throughput of distributed ledgers by tightening the assumptions on the application, we focus here on the scalability of ledgers regardless of the upper application layer. The verification process can be parallelized between nodes of a ledger to greatly improve the throughput of the overall system.

Sharding distributed ledgers

Elastico [100], OmniLedger [69] and RapidChain [70] propose a permissionless distributed ledger using multiple classical PBFT consensus protocols each executing within a subset of nodes (i.e., *a shard*).

Elastico limits the number of shards a malicious nodes may join (under different identifies) by tying the shard of a node to the result of a Proof-of-Work puzzle.

OmniLedger [69] extends ByzCoin [68] with the ideas proposed by Elastico [100] to increase the size of the shards (and thus reduce the probability of failures), and allow for cross-shards transactions thanks to a Byzantine shard-atomic commit protocol called Atomix. OmniLedger handles up to a few thousands transactions per second in their evaluation.

RapidChain [70] goes even further than OmniLedger by performing various optimizations such as using smaller committees, pipelining consensus instances and batching cross-shard transactions. RapidChain improves upon the throughput results of OmniLedger by another few thousands transactions per second.

2.5 Gossip protocols

Gossip protocols, also known as epidemic protocols [101], [102], are probabilistic broadcasting protocols [103] that are particularly efficient [104] in large-scale systems due to the little assumptions they make on the network. Gossip protocols are robust to nodes crashing, unreliable communication channels, topology changes, and churn (i.e., nodes joining and exiting the network). Gossip protocols are often used in combination with a Random Peer Sampler [105] (e.g., Cyclon [106], Scamp [107]) which itself is a gossip protocol that provides them with a partial view of the network. As broadcast protocols, gossip protocols are originally designed to disseminate information in large-scale systems, but they can also be used to aggregate information, and to create and organize nodes in topologies (e.g., with Random Peer Sampling).

We generally classify gossip protocols as push-based, pull-based, or push-pull. In a push gossip protocol, nodes send messages without requiring an answer. In pull-only protocols, they send messages with actual content only when requested by other nodes (e.g., anti-entropy [101]). While in push-pull protocols they send messages and expect replies.

Dissemination

To start a broadcast in a push gossip protocol, a node sends a message to a random subset of *fanout* nodes; in turn, each receiving node also forwards the message to a random subset of nodes. This simple forwarding process is sufficient to disseminate the message to an overwhelming majority of nodes and even to all the nodes in some failure-free executions with a large enough fanout in the order of $\log_{\text{fanout}}(n) + c$, with n network size and c constant [104]. The forwarding can be reactive to target fast delivery speed, or proactive (cyclic) to reduce the load on the network.

In addition, protocols may employ push, pull, or push-pull strategies. Push protocols tend to be the most effective in the initial phases of gossip dissemination, while pull protocols start being effective when messages have already spread to a large set of nodes. This justifies the use of push-pull protocols or of specific combinations that use push protocols in the first rounds of dissemination and pull protocols in the later rounds [108].

Aggregation

Additionally to bare dissemination, nodes can enrich messages as they are forwarding them. For example, this makes it possible to aggregate values over the network [109] in order to compute averages or other aggregate functions in large-scale networks. For instance, gossip aggregation can be used to count the number of nodes in the network, or even to perform gradient-descent-based machine learning [110], [111].

Topology management

Lastly, gossip protocols make it possible to build overlay networks. A first example consists of the already mentioned peer-sampling protocols. Each node maintains a local *view*, a data structure that lists references to other nodes. Nodes exchange the content of their views with each other; upon receiving another node's view, a node updates its view by keeping a subset of the two. Depending on how nodes select these subsets, the process leads to more or less random topologies [105]–[107]. Moreover, this process can also define global structures starting from local knowledge held by individual nodes [112], [113]. Each node applies a distance function to each remote node in its partial view of the network, and only keeps the closest neighbors in its view. Thanks to topology management gossip protocols, applications can use algorithms with well-known results on certain topologies (e.g., LCR [114] uses a rotating sequencer on a ring for total order broadcast).

2.5.1 Ordering in gossip protocols

Having a total order in the events happening in the system is equivalent to achieving consensus in a distributed system. In particular, totally ordered events enable strong consistency in data replication protocols. EpTO [115] is a gossip-based total order broadcast with probabilistic strong ordering guarantees. ecBroadcast [116] orders events but it performs local rollbacks if events are not received in the correct order. ecBroadcast can only provide eventual consistency guarantees by ensuring convergence of state. Optimistic total order [117] assumes low channel-latency variability and adds delays to the delivery of messages depending on the observed latency with the sender of the message.

2.5.2 Biased gossip protocols

Some protocols bias their behavior to favor or to better exploit specific classes of nodes. Directional gossip [118] favors weakly connected nodes in order to improve its overall reliability. The work in [119] reduces the message complexity of a broadcast by disseminating to good nodes first using a reactive epidemic protocol, while bad nodes are reached through a slower periodic push procedure. Similarly, Gravitational gossip [120] proposes a multicast protocol with differentiated trade-offs in cost and reliability to better balance the communication workload between nodes, according to their user-defined capacities. Hierarchical gossip [121] reduces overheads induced by the physical network topology by favoring gossip targets that are close in the network hierarchy. In the context of video streaming, HEAP [122] adapts the fanout of nodes to reduce delivery latency in the presence of heterogeneous bandwidth capabilities. In addition, late messages harm the usability of the application and are thus ignored.

Our first contribution, GPS, harnesses the capabilities of biased gossip protocols to offer differentiated ordering guarantees to its members.

2.6 Summary

We have shown throughout this chapter the foundations as well as some of the most recent advances in decentralized data replication. As deterministic coordination protocols, performed for instance with consensus or total order broadcast, quickly reach their limits as systems expend, designers of large-scale distributed systems have turned towards scalable solutions that offer weaker guarantees such as eventual consistency.

In the next chapters we propose two solutions that help large-scale decentralized systems personalize their quality of service to better fit the heterogeneous needs of their users. In Chapter 3 we propose the novel hybrid consistency criterion *Update consistency Primary-Secondary* that exploits the differentiated trade-offs in message latency and message ordering provided by our novel biased gossip protocol *Gossip Primary-Secondary*. In Chapter 4 we propose to enrich Bitcoin with a new class of nodes that can adapt their safety and resource consumption trade-off. *Dietcoin* enables low-resource devices to verify the correctness of the Bitcoin blockchain which, due to its prohibitive cost, is in practice never performed by such devices.

CHAPTER 3

DIFFERENTIATING CONSISTENCY GUARANTEES IN EPIDEMIC PROTOCOLS

In the first contribution of this thesis, we tackle the problem of requirement heterogeneity in terms of consistency and latency in large-scale systems. Eventual consistency is often used because it can be realized by highly scalable protocols, but it tends to be uniformly applied to an entire system. We argue that there is a growing demand for differentiated eventual consistency requirements. We address this demand at the communication layer to ensure the genericity of our solution.

More precisely, we propose in this chapter *Update-Query consistency Primary-Secondary* (UPS), a novel consistency mechanism that offers differentiated eventual consistency and delivery speed to two classes of nodes (Primary and Secondary nodes). UPS combines the update-query consistency protocol [45] with a novel two-phase epidemic broadcast protocol *Gossip Primary-Secondary* (GPS).

After a short overview and motivation, we present GPS in Section 3.2. We then propose a closed-form analysis of our approach's delivery speed in Section 3.3. We formally define in Section 3.4 a novel and scalable consistency metric operating at runtime that enables us to quantify the amount of temporary inconsistencies of each node. We then evaluate in Section 3.5 our complete mechanism experimentally on a simulated network of one million nodes. To measure the consistency trade-off, we formally define in Section 3.4 a novel and scalable consistency metric that operates at runtime. We show in particular that the cost paid by each class of nodes is in fact very small compared to an undifferentiated system: Primary nodes experience a lower latency with levels of inconsistency that are close to those of undifferentiated nodes, while Secondary nodes observe less inconsistencies at a minimal latency cost. Finally we conclude in Section 3.7.

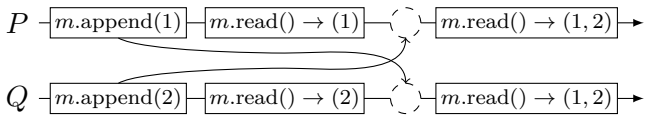


Figure 3.1 – An eventually consistent append-only queue (repeated from Chapter 2.1.3).

3.1 Motivation and problem statement

One of the key motivations of this thesis is the observation that modern large-scale distributed computer systems cannot avoid replication, and that replicated data is unfortunately difficult to keep consistent. *Strong consistency*, such as linearizability or sequential consistency, is particularly expensive to implement in large-scale systems, and cannot be simultaneously guaranteed together with availability, when using a failure-prone network [24].

3.1.1 Exploiting the compromise between speed and consistency

As discussed in the previous chapter, weak consistency conditions [39]–[41] offer a welcome trade-off between the competing needs for scalability and coordination. Among these weaker consistency conditions, *eventual consistency* [44], [46] aims to strike a balance between agreement, speed, and dynamicity within a system. Intuitively, eventual consistency allows the replicas of a distributed shared object to temporarily diverge, as long as they eventually converge back to a unique global state. An eventually consistent object must however find a compromise between speed, how fast are changes visible to other nodes, and consistency, to which extent do different nodes agree on the system's state.

We will illustrate this compromise on the example of the distributed append-only queue discussed in Chapter 2.1.3 (shown again in Figure 3.1). In this figure, the distributed append-only queue m is manipulated by two processes P and Q. m supports two operations: $\mathrm{append}(x)$, which appends an integer x to the queue, and $\mathrm{read}()$, which returns the current content of m. In Figure 3.1, both P and Q eventually converge to the same consistent global state $(1, 2)$, that includes both modifications: $m.\mathrm{append}(1)$ by P and $m.\mathrm{append}(2)$ by Q, in *this* order. However, Q experiences a temporary inconsistent state when it reads (2): this read does not take into account the append operation by P which has been ordered before $m.\mathrm{append}(2)$, and is thus inconsistent with the final state $(1, 2)$ [1]. Q could increase the odds of avoiding this particular inconsistency by delaying its first read operation, which would augment its chances of receiving information regarding P's append operation on time (dashed circle). Such delays improve consistency, but reduce the speed of change propagation across replicas, and must be chosen with care.

1. This inconsistency causes in particular Q to observe an illegal state transition from (2) to $(1, 2)$.

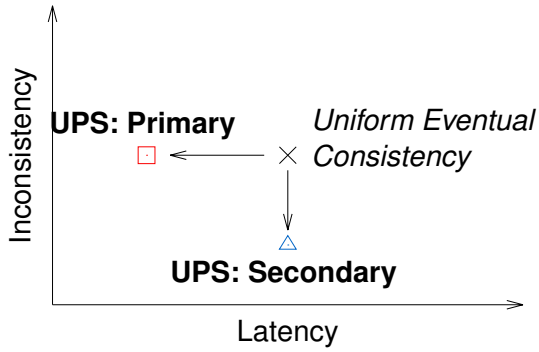


Figure 3.2 – Aimed consistency/latency trade-off in UPS.

Most existing solutions to eventual consistency resolve this tension between speed and consistency by applying one trade-off point *uniformly* to all the nodes in a system [16], [115]. However, as systems continue to grow in size and expand in geographic span, they become more diverse, and must cater for diverging requirements. In this thesis, we argue that this heterogeneity increasingly calls for differentiated consistency levels in large-scale systems. This observation has been made by other researchers, who have proposed a range of hybrid consistency conditions over the years [50], [53], [54], [56], but none of them has so far considered how eventual consistency on its own could offer differentiated levels of speed and consistency within the same system.

Designing such a protocol raises however an important methodological point: how to measure consistency. Consistency conditions are typically formally defined as predicates on execution histories; a system execution is thus either consistent, or it is not. However, practitioners using eventual consistency are often interested in the current *level* of consistency of a live system, i.e., how far the system currently is from a consistent situation. Quantitatively measuring a system's inconsistencies is unfortunately not straightforward: some practical works [16] measure the level of *agreement* between nodes, i.e., how many nodes see the same state, but this approach has little theoretical grounding and can thus lead to paradoxes. For instance, returning to Figure 3.1, if we assume a large number of nodes (e.g., $Q_1, ..., Q_n$) reading the same inconsistent state (2) as Q, the system will appear close to agreement (many nodes see the same state), although it is in fact largely inconsistent.

3.1.2 Problem statement

We focus in this chapter on *update consistency*, a particular type of eventual consistency that we have discussed in Chapter 2.1.3. The key feature of update consistency lies in the ability to define precisely the nature of the convergence state reached once all updates have

Algorithm 1 Update consistency for an append-only queue

```
1:  variables
2:      int id                                              ▷ Node identifier
3:      set <int, int, V> U ← ∅                             ▷ Set of updates to the queue
4:      int clock_id ← 0                                    ▷ Node's logical clock
5:  procedure APPEND(v)                                     ▷ Append a value v to the queue
6:      clock_id ← clock_id + 1
7:      U ← U ∪ {< clock_id, id, v >}
8:      BROADCAST (< clock_id, id, v >)
9:  upon receive (< clock_msg, id_msg, v_msg >) do
10:     clock_id ← MAX(clock_id, clock_msg)
11:     U ← U ∪ {< clock_msg, id_msg, v_msg >}
12: procedure READ( )                                       ▷ Read the current state of the queue
13:     q ← ()                                              ▷ Empty queue
14:     for all < clock, id, v > ∈ U sorted by (clock, id) do
15:         q ← q · v
16:     return q
```

been issued. However, the nature of temporary states also has an important impact in practical systems. This raises two important challenges. First, existing systems address the consistency of temporary states by implementing uniform constraints that all the nodes in a system must follow [47]. But different actors in a distributed application may have different requirements regarding the consistency of these temporary states. Second, even measuring the level of inconsistency of these states remains an open question. Existing systems-oriented metrics do not take into account the ordering of update operations (append in our case) [16], [123]–[125], while theoretical ones require global knowledge of the system [126] which makes them impractical at large scale.

In the following sections, we address both of these challenges. First we propose a novel broadcast mechanism that, together with Algorithm 1, satisfies update consistency, while supporting differentiated levels of consistency for read operations that occur before the convergence state. Specifically, we exploit the evident trade-off between speed of delivery and consistency, and we target heterogeneous populations consisting of an elite of Primary nodes that should receive fast, albeit possibly inconsistent, information, and a mass of Secondary nodes that should only receive stable consistent information, albeit more slowly as depicted in Figure 3.2. Second, we propose a novel metric to measure the level of inconsistency of an append-only queue, and use it to evaluate our protocol.

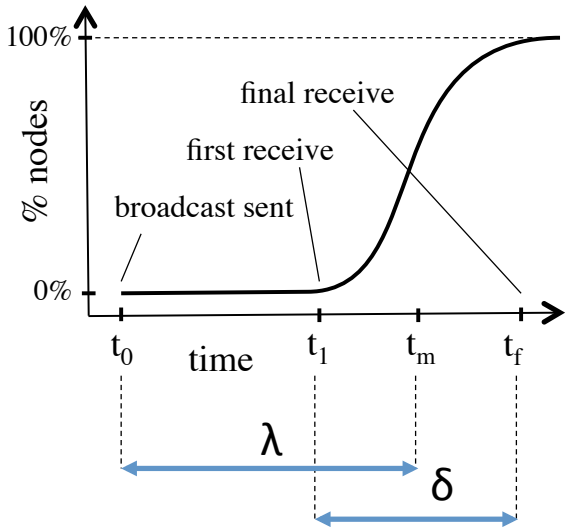


Figure 3.3 – Two sorts of speeds: latency (λ) and jitter (δ).

3.2 The GPS broadcast protocol

3.2.1 System model

We consider a large set of nodes $p_1, ..., p_N$ that communicate using point-to-point messages. Any node can communicate with any other node, given its identifier. We use probabilistic algorithms in the following that are naturally robust to crashes and message losses, but do not consider these aspects in the rest of the chapter for simplicity. Nodes are categorized in two classes: a small number of Primary nodes (the *elite*) and a large number of Secondary nodes (the *masses*). The class of a node is an application-dependent parameter that captures the node's requirements in terms of update consistency: Primary nodes should perceive object modification as fast as possible, while Secondary nodes should experience as few inconsistencies as possible.

3.2.2 Intuition and overview

We have repeatedly referred to the inherent trade-off between speed and consistency in eventually consistent systems. On deeper examination, this trade-off might appear counter-intuitive: if Primary nodes receive updates faster, why should not they also experience higher levels of consistency? This apparent paradox arises because we have so far silently confused *speed* and *latency*. The situation within a large-scale broadcast is in fact more subtle and involves two sorts of speeds (Figure 3.3): *latency* (λ, shown as an average over all nodes in the figure) is the time a message m takes to reach individual nodes, from the point in time of m's

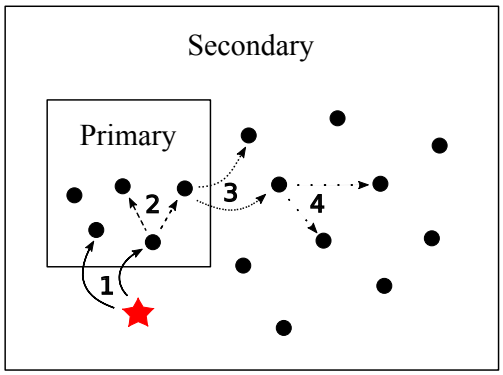


Figure 3.4 – Model of GPS and path of an update in the system.

sending (t_0). *Jitter* (δ), by contrast, is the delay between the first (t_1) and the last receipt (t_f) of a broadcast. (In most large-scale broadcast scenarios, $t_0 - t_1$ is small, and the two notions tend to overlap.) Inconsistencies typically arise in Algorithm 1 when some updates have only partially propagated within a system, and are thus predominantly governed by the jitter δ rather than the average *latency* λ.

The gossip-based broadcast protocol we propose, Gossip Primary-Secondary (GPS), exploits this distinction and proposes different δ/λ trade-offs. Primary nodes have a reduced λ (thus increasing the speed at which updates are visible) but also a slightly increased δ, while Secondary nodes have a reduced δ (thus increasing consistency by lowering the probability for a node to receive updates in the wrong order) but at the cost of a slightly higher λ.

More precisely, GPS uses the set of Primary nodes as a sort of orderer that accumulates copies of an update u before collectively forwarding it to Secondary nodes. The main phases of this sequence is shown in Figure 3.4:

1. A new update u is first sent to Primary nodes (**1**);
2. Primary nodes disseminate u among themselves (**2**);
3. Once most Primary nodes have received u, they forward it to Secondary nodes (**3**);
4. Finally, Secondary nodes disseminate u among themselves (**4**).

A key difficulty in this sequence consists in deciding when to switch from Phase **2** to **3**. A collective, coordinated transition would require some global synchronization mechanism, a costly and generally impracticable solution in a very large system. Instead, GPS relies on a less accurate but more scalable local procedure based on broadcast counts, which enables each Primary node to decide locally when to start forwarding to secondaries.

Algorithm 2 – Gossip Primary-Secondary for a node

```
 1: parameters
 2:     int fanout                                  ▷ Number of nodes to send to
 3:     int rpsViewSize                             ▷ Out-degree per class of each node
 4:     boolean isPrimary                           ▷ Class of the node
 5: initialization
 6:     set{node} Γ_P ← ∅                           ▷ Set of Primary neighbors
 7:     set{node} Γ_S ← ∅                           ▷ Set of Secondary neighbors
 8:     map{message, int} R ← ∅                     ▷ Counters of message duplicates received
 9: periodically
10:     Γ_P ← rpsViewSize nodes from Primary-RPS
11:     Γ_S ← rpsViewSize nodes from Secondary-RPS
12: procedure BROADCAST(msg)                        ▷ Called by the application
13:     R ← R ∪ {(msg, 1)}
14:     GOSSIP(msg, Γ_P)
15: procedure GOSSIP(msg, targets)
16:     for all j ∈ {fanout random nodes in targets} do
17:         SENDTONETWORK(msg, j)
18: upon receive (msg) do
19:     counter ← R[msg] + 1
20:     R ← R ∪ {(msg, counter)}
21:     if counter = 1 then DELIVER(msg)            ▷ Deliver 1st receipt
22:     if isPrimary then
23:         if counter = 1 then GOSSIP(msg, Γ_P)
24:         if counter = 2 then GOSSIP(msg, Γ_S)
25:     else
26:         if counter = 1 then GOSSIP(msg, Γ_S)
```

Algorithm 3 – Uniform Gossip for a node (baseline)

(only showing main differences to Algorithm 2)

```
 9': periodically Γ ← rpsViewSize nodes from RPS
15': procedure GOSSIP(msg)
16':     for all j ∈ {fanout random nodes in Γ} do
17':         SENDTONETWORK(msg, j)
18': upon receive (msg) do
19':     if msg ∈ R then
20':         return                                 ▷ Infect and die: ignore if msg already received
21':     R ← R ∪ {msg}
22':     DELIVER(msg)                               ▷ Deliver 1st receipt
23':     GOSSIP(msg)
```

3.2.3 The GPS algorithm

The pseudo-code of GPS is shown in Algorithm 2. GPS follows the standard models of reactive epidemic broadcast protocols [104], [127]. Each node keeps a history of the messages received so far (in the R variable, line 8), and decide whether to re-transmit a received broadcast to fanout other nodes based on its history. Contrary to a standard epidemic broadcast, however, GPS handles Primary and Secondary nodes differently.

— *First*, GPS uses two distinct *Random Peer Sampling* protocols (RPS) [105] (lines 10-11) to track the two classes of nodes. Both Primary and Secondary nodes use the RPS view of their category to re-transmit a message they receive for the first time to fanout other nodes in their own category (lines 23 and 24), thus implementing Phases **2** and **4**.

— *Second*, GPS handles retransmissions differently depending on a node's class (Primary or Secondary). Primary nodes use the inherent presence of message duplicates in gossip protocols, to decide locally when to switch from Phase **2** to **3**. More specifically, each node keeps count of the received copies of individual messages (lines 8, 13, 20). Primary nodes use this count to detect duplicates, and forward a message to fanout Secondary nodes (line 24) when a duplicate is received for the first time, thus triggering Phase **3**.

We can summarize the behavior of both classes by saying that Primary nodes *infect twice and die*, whereas Secondary nodes *infect and die*. For comparison, a standard *infect and die* gossip without classes (called Uniform Gossip in the following), is shown in Algorithm 3. We will use Uniform Gossip as our baseline for our analysis (Section 3.3) and our experimental evaluation (Section 3.5).

3.3 Analysis of GPS

In the following we compare analytically the expected performance of GPS and compare it to Uniform Gossip in terms of message complexity and latency.

Our analysis uses the following parameters:

— Network N of size: $|N| \in \mathbb{N}$;

— Fanout: $f \in \mathbb{N}$;

— Density of Primary nodes: $d \in \mathbb{R}_{[0,1]}$ (assumed $\leq 1/f$).

Uniform Gossip uses a simple *infect and die* procedure. For each unique message it receives, a node will send it to f other nodes. If we consider only one source, and assume that most nodes are reached by the message, the number of messages exchanged in the system can be estimated as

$$|msg|_{uniform} \approx f \times |N|. \tag{3.1}$$

In the rest of our analysis, we assume, following [128], that the number of rounds needed by Uniform Gossip to infect a high proportion of nodes can be approximated by the following expression when $N \to \infty$:

$$\lambda_{uniform} \approx \log_f(|N|) + C, \tag{3.2}$$

where C is a constant, independent of N.

GPS distinguishes two categories of nodes: Primary nodes and Secondary nodes, noted P and S, that partition N. The density of Primary nodes, noted d, defines the size of both subsets:

$$|P| = d \times |N|, \qquad |S| = (1-d) \times |N|.$$

Primary nodes disseminate twice, while Secondary nodes disseminate once. Expressed differently, each node disseminates once, and Primary nodes disseminate once more. Applying the same estimation as for Uniform Gossip gives us:

$$\begin{aligned} |msg|_{GPS} &\approx f \times |N| + f \times |P| \\ &\approx f \times |N| + f \times d \times |N| \\ &\approx (1+d) \times f \times |N| \\ &\approx (1+d) \times |msg|_{uniform}. \end{aligned} \tag{3.3}$$

(3.1) and (3.3) show that GPS only generates d times more messages than Uniform Gossip, with $d \in \mathbb{R}_{[0,1]}$. For instance, having 1% Primary nodes in the network means having only 1% more messages compared to Uniform Gossip.

If we now turn to the latency behavior of GPS, the latency of the Primary nodes is equivalent to that of Uniform Gossip executing on a sub-network composed only of $d \times |N|$ nodes, i.e.,

$$\lambda_P \approx \log_f(d \times |N|) + C. \tag{3.4}$$

Combining (3.2) and (3.4), we conclude that Primary nodes gain $log_f(d)$ rounds compared to nodes in Uniform Gossip:

$$\Delta\lambda_P \approx -log_f(d). \tag{3.5}$$

Considering now Secondary nodes, their latency can be estimated as a sum of three elements:

— the latency of Primary nodes;
— an extra round for Primary nodes to receive messages a second time;
— the latency of a Uniform Gossip among Secondary nodes with $d \times |N|$ nodes, corresponding to the Primary nodes, already infected;

which we approximate for $d \ll 1$ as

$$\begin{aligned}\lambda_S &\approx \log_f(d \times |N|) + 1 + \log_f(\frac{(1-d) \times |N|}{d \times |N|}) + 2C, \\ &\approx \log_f((1-d) \times |N|) + 1 + 2C. \end{aligned} \tag{3.6}$$

In summary, this analysis shows that GPS only generates a small number of additional messages, proportional to the density d of Primary nodes, and that the latency cost paid by Secondary is bounded by a constant value $(1 + C)$ that is independent of the Primary density d.

3.4 Consistency metric

As a side contribution, we propose in this chapter a novel metric that measures the consistency level of the temporary states of an update-consistent execution. As discussed in Section 3.1, existing consistency metrics fall short either because they do not capture the ordering of operations, or because they cannot be computed without global system knowledge. Our novel metric satisfies both of these requirements.

3.4.1 A general consistency metric

We start by observing that the algorithm for an update-consistent append-only queue (Algorithm 1) guarantees that all its execution histories respect update consistency. To measure the consistency level of temporary states, we therefore evaluate how the history deviates from a stronger consistency model, *sequential consistency* [35]. An execution respects sequential consistency if it is equivalent to some sequential (i.e., totally ordered) execution that contains the same operations, and respects the sequential (process) order of each node.

Since update consistency relies itself on a total order, the gist of our metric consists in counting the number of read operations that do not conform with a total order of updates that leads to the final convergence state. Given one such total order, we may transform the execution into one that conforms with it by removing some read operations. In general, a data object may reach a given final convergence state by means of different possible total orders, and for each such total order we may have different sets of read operations whose removal makes the execution sequentially consistent. We thus count the level of inconsistency by taking the minimum over these two degrees of freedom: choice of the total order, and choice of the set.

More formally, we define the *temporary inconsistencies* of an execution Ex as a *finite* set of read operations that, when removed from Ex, makes it sequentially consistent. We denote the set of all the temporary inconsistencies of execution Ex over all compatible total orders by $TI(Ex)$. We then define the *relative inconsistency* RI of an execution Ex as the minimal

number of read operations that must be removed from Ex to make it sequentially consistent.

$$RI(Ex) = \begin{cases} \min\limits_{E \in TI(Ex)} |E| & \text{if } TI(Ex) \neq \emptyset \\ +\infty & \text{otherwise} \end{cases}$$

For example, in Figure 3.1, if we consider the total order $< m.\text{append}(1)$, $m.\text{append}(2)$, $m.\text{read}(1)$, $m.\text{read}(2)$, $m.\text{read}(1,2)$, $m.\text{read}(1,2) >$ then we need to remove both $m.\text{read}(1)$ and $m.\text{read}(2)$ to make the execution sequentially consistent. Removing only $m.\text{read}(2)$, instead, suffices to make the execution sequentially consistent with respect to $< m.\text{append}(1)$, $m.\text{read}(1)$, $m.\text{append}(2)$, $m.\text{read}(2)$, $m.\text{read}(1,2)$, $m.\text{read}(1,2) >$. Therefore the level of inconsistency of the execution in Figure 3.1 is 1.

The metric RI is particularly adapted to compare the consistency level of implementations of update consistency: the lower, the more consistent. In the best case scenario where Ex is sequentially consistent, $TI(Ex)$ is a singleton containing the empty set, resulting in $RI(Ex) = 0$. In the worst case scenario where the execution never converges (i.e., some nodes indefinitely read incompatible local states), every set of reads that needs to be removed to obtain a sequentially consistent execution is infinite. Since TI only contains finite sets of reads, $TI(Ex) = \emptyset$ and $RI(Ex) = +\infty$.

3.4.2 The case of our update-consistent append-only queue

In general, $RI(Ex)$ is complex to compute: it is necessary to consider all possible total orders of events that can fit for sequential consistency and all possible finite sets of reads to check whether there are temporary inconsistencies. But in the case of an append-only queue implemented with Algorithm 1, we can easily show that there exists exactly one minimal set of temporary inconsistencies.

To understand why, we first observe that the append operation is non-commutative. This implies that there exists a single total order of append operations that yields a given final convergence state. Second, Algorithm 1 guarantees that the size of the successive sequences read by a node can only increase and that read operations always reflect the writes made on the same node. Consequently, in order to have a sequentially consistent execution, it is necessary and sufficient to remove all the read operations that return a sequence that is not a prefix of the sequence read after convergence. These read operations constitute the minimal set of temporary inconsistencies TI_{min}.

3.5 Experimental results

We perform the evaluation of UPS via PeerSim [129], a well-known Peer-to-Peer simulator. A repository containing the code and the results is available online [2]. To assess the trade-offs between consistency level and latency as well as the overhead of UPS, we focus the evaluation on three metrics:

— the level of consistency of the replicated object;
— the latency of messages;
— the overhead in number of messages.

3.5.1 Methodology

Network settings

We use a network of 1 million nodes, a fanout of 10 and an RPS view size of 100. These parameters yield a broadcast reliability (i.e., the probability that a node receives a message) that is above 99.9%. Reliability could be further increased with a higher fanout [104], but these parameters, because they are all powers of 10, make it convenient to understand our experimental results.

We use density values of Primary nodes of: $10^{-1}, 10^{-2}$ and 10^{-3}. According to Equation 3.5 in Section 3.3, we expect to see a latency gain for Primary nodes of 1, 2, and 3 rounds respectively. We evaluate four protocol configurations: one for Uniform Gossip (baseline), and three for UPS with the three above densities, then run each configuration 25 times and record the resulting distribution.

Scenario

We consider a scenario where all nodes share an instance of an update-consistent append-only queue, similar to the one used as an example in Section 3.1. Following the definition of update consistency, nodes converge into a strongly consistent state once they stop modifying the queue. We opt for a scenario where 10 $\mathrm{append}(x)$ operations, with $x \in \mathbb{Z}$, are performed on the queue by 10 random nodes, over the first 10 rounds of the simulations at the frequency of one update per round. In addition, all nodes repeatedly read their local copy of the queue in each round.

We expect the system to experience two periods: first, a temporary situation during which updates are issued and disseminated (simulating a system continuously performing updates), followed by a stabilized state after updates have finished propagating and most nodes have converged to a strongly consistent state.

2. `https://gitlab.inria.fr/WIDE/gps-exp`

Consistency metric

Our experimental setting enables us to further refine the generic consistency metric we introduced in Section 3.4. Since nodes perform a finite number of update operations and each node reads the state of the queue at each round, we define a per-round metric to compare the evolution of the inconsistency of Primary and Secondary nodes through time. For each round r, we define the sets $R_P(r)$ and $R_S(r)$ as the sets of all the read operations performed at round r by the Primary and Secondary nodes respectively. Then we define per-round inconsistency, $Incons_P(r)$ and $Incons_S(r)$, as the proportion of Primary and Secondary nodes that see an inconsistent read at round r (i.e., the corresponding reads are in TI_{min}).

$$Incons_P(r) = \frac{|TI_{min} \cap R_P(r)|}{d \times |N|}$$

$$Incons_S(r) = \frac{|TI_{min} \cap R_S(r)|}{(1-d) \times |N|}$$

$$Incons_{P+S}(r) = d \times Incons_P(r) + (1-d) \times Incons_S(r)$$

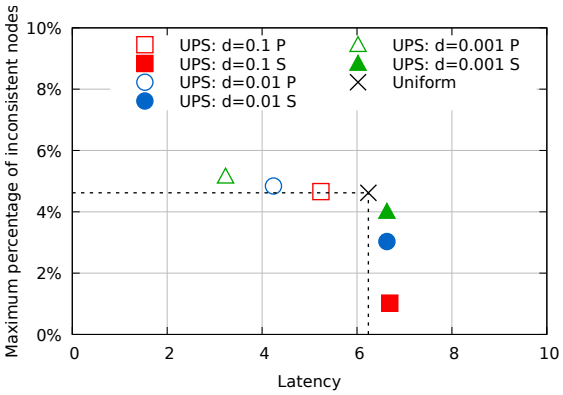
In the following, we focus on these *instantaneous* per-round metrics, which provide an equivalent but more accurate view than the simple relative inconsistency of an experimental run, $RI(Ex)$. We can in fact compute $RI(Ex)$ as follows.

$$RI(Ex) = |N| \times \sum_{r=0}^{\infty} (Incons_{P+S}(r)).$$

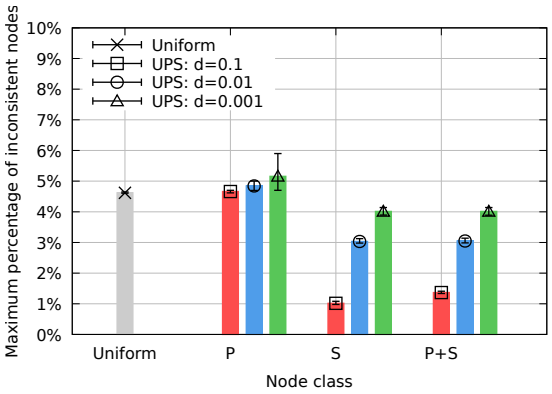
Plots

Unless stated otherwise, plots use boxes and whiskers to represent the distribution of measures obtained for the represented metric. For the inconsistency measures, the distribution is over all runs. For the latency measures, the distribution is over all nodes within all runs. The end of the boxes show the first and third quartiles, the end of the whiskers represent the minimum and maximum values, while the horizontal bar inside the boxes is the mean. In some cases, the low variability of the results makes it difficult to see the boxes and whiskers.

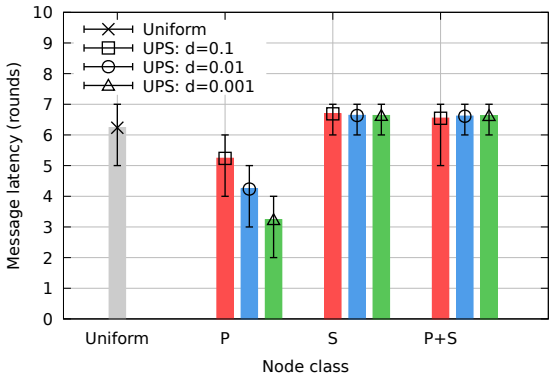
For clarity purposes, curves are slightly shifted to the right to avoid overlap between them. All the points between rounds r and $r+1$ belong to round r. In the following plots, Primary nodes are noted P, Secondary nodes are noted S and the system as a whole is noted $P+S$. Since only a small fraction of nodes are Primary nodes, the results of $P+S$ are naturally close to those of S.



(a) Experimental trade-offs between consistency and latency (closer to the bottom left corner is better). A big gain on one side of the balance incurs a small penalty on the other side of the balance for a class.



(b) Consistency trade-off in UPS (lower is better). Secondary nodes are much more consistent than nodes in the baseline while Primary nodes only experience a small consistency penalty. The consistency of both Primary and Secondary nodes improves as the density of Primary nodes increases.



(c) Latency trade-off in UPS (lower is better). Primary nodes latency is lower than that of the baseline; the less Primary nodes, the lower their latency. Secondary nodes remain half a round slower than the baseline, regardless of the density of Primary nodes in the network.

Figure 3.5 – Consistency and latency trade-off in UPS. Primary nodes are faster and a bit less consistent, while Secondary are more consistent and a bit slower. The top of the bars is the mean while the ends of the error bars are the 5th and 95th percentiles.

3.5.2 Overall results

Figure 3.5a mirrors Figure 3.2 discussed in Section 3.1 and provides an overview of our experimental results in terms of consistency/latency trade-offs for the different sets of nodes involved in our scenario. Each UPS configuration is shown as a pair of points representing Primary and Secondary nodes: Primary nodes are depicted with hollow shapes, while Secondary nodes use solid symbols. Uniform Gossip is represented by a single black cross. The position on the x-axis charts shows the average update latency experienced by each set of nodes, and the y-axis their perceived level of inconsistency, taken as the maximum $Incons_X(r)$ value measured over all runs.

The figure clearly shows that UPS delivers the differentiated consistency/latency trade-offs we set out to achieve in our introduction: Secondary nodes enjoy higher consistency levels than they would in an uniform update-query consistency protocol, while paying only a small cost in terms of latency. The consistency boost strongly depends on the density of Primary nodes in the network, while the cost in latency does not, reflecting our analysis of Section 3.3. Primary nodes present the reverse behavior, with the latency gains of Primary nodes evolving in the reverse direction of the consistency gains of Secondary nodes. We discuss both aspects in more details in the rest of this section.

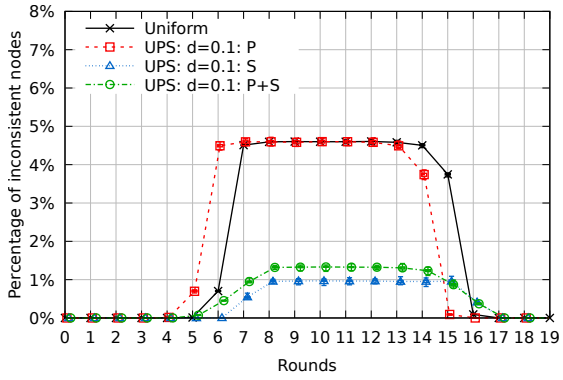
3.5.3 Consistency level

Figure 3.5b details the consistency levels provided by UPS by showing the worst consistency that nodes experience over all simulations. We note an evident improvement of the maximum inconsistency level of Secondary nodes over the baseline and a slight decrease for Primary nodes. In addition, this figure clearly shows the impact of the density of Primary nodes over the consistency of Secondary nodes, and to a lesser extent over the consistency of Primary nodes.

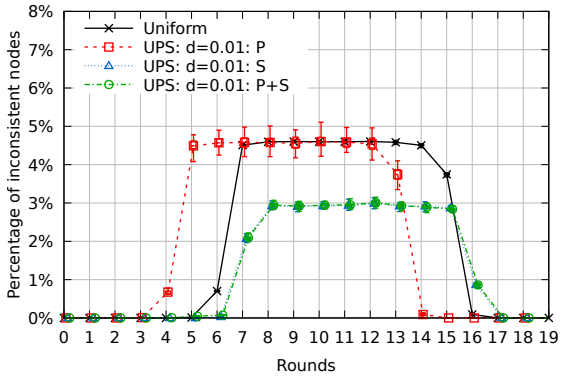
Figures 3.6a, 3.6b and 3.6c show the evolution over time of the inconsistency measures $Incons_P(r)$, $Incons_S(r)$ and $Incons_{P+S}(r)$ defined in Section 3.5.1. We can observe an increase of inconsistencies during the temporary phase for all configurations and a return to a consistent state once every node has received every update.

During the temporary phase, the inconsistency level of Primary nodes is equivalent to that of nodes in Uniform Gossip. In that phase, around 4.6% of Primary nodes are inconsistent with a higher variability for lower densities.

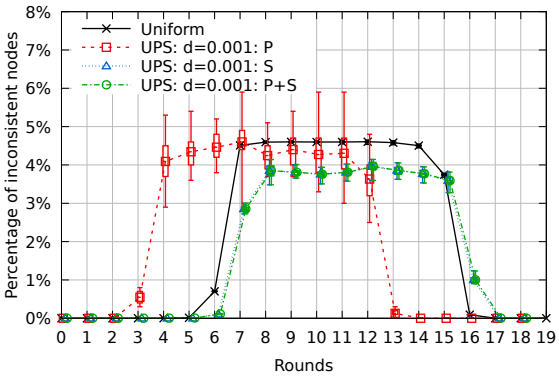
Meanwhile, the inconsistency level of Secondary nodes is much lower than that of nodes in Uniform Gossip. The density of Primary nodes plays a key role in this difference; the higher the density, the lower the inconsistency level of Secondary nodes. This level remains under 1.0% for the highest density but goes up to 4.0% for the lowest density.



(a) Density $= 10^{-1}$



(b) Density $= 10^{-2}$

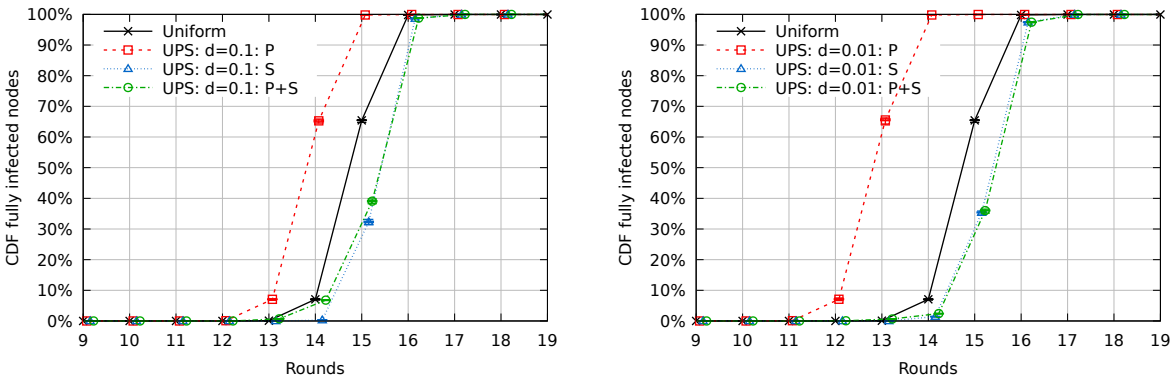


(c) Density $= 10^{-3}$

Figure 3.6 – While updates are being disseminated, Secondary nodes are more consistent than the baseline and Primary nodes are on average as consistent as the baseline. The more Primary nodes are in the system, the more Secondary nodes are consistent. Once updates stop being performed, Primary nodes converge faster to a strongly consistent state.

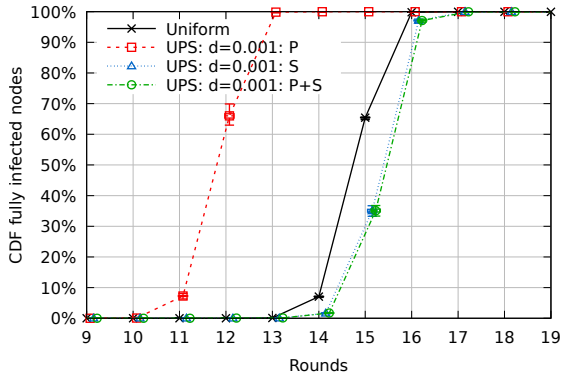
The jitter, as defined in Section 3.2.2, is a good metric to compare the consistency of different sets of nodes: a lower jitter implies a lower probability for a node to receive updates in the wrong order, which in turn leads to a higher consistency. The error bars in Figure 3.5c provide an approximation of the jitter of a set of nodes since they represent the bounds in latency of 90% of the nodes in a set. These error bars show that 90% of Secondary nodes receive updates within 1 round of margin, while the same proportion of Primary nodes and nodes in Uniform Gossip receive updates within 2 rounds of margin. This difference in jitter explains why Secondary nodes are more consistent than Primary nodes and nodes in Uniform Gossip.

Once the dissemination reaches a critical mass of Primary nodes, the infection of Secondary nodes occurs quickly. If the density of Primary nodes is high enough, then it becomes possible for a majority of Secondary nodes to receive the same update at the same time. Since



(a) Density $= 10^{-1}$

(b) Density $= 10^{-2}$



(c) Density $= 10^{-3}$

Figure 3.7 – Primary nodes receive all the updates faster than the baseline: they gain 1, 2 and 3 rounds for densities of $10^{-1}, 10^{-2}$ and 10^{-3} respectively. Meanwhile Secondary nodes receive all the updates only half a round later on average compared to nodes in the baseline.

it takes fewer rounds for Secondary nodes to be fully infected compared to Primary nodes, Secondary nodes turn out to be more consistent.

3.5.4 Message latency

Figure 3.5c represents the distribution of all the values of message latency over all the simulation runs. The three $P + S$ bars and the *Uniform* bar contain each 250 million message latency values (25 runs $\times$ 1 million nodes $\times$ 10 sources with a reliability above 99.9%).

This figure shows that UPS infects Primary nodes faster than Uniform Gossip would. Specifically, Primary nodes obtain a latency gain of 1, 2 and 3 rounds with densities of 10^{-1}, 10^{-2} and 10^{-3} respectively. Secondary nodes, on the other hand, are infected half a round slower than nodes in Uniform Gossip for all density values.

Figures 3.7a, 3.7b and 3.7c compare the evolution of the dissemination of updates between

Uniform Gossip and UPS with different densities. Again, Primary nodes have a 1, 2 and 3 rounds latency advantage over the baseline while Secondary nodes are no more than a round slower.

We can also observe this effect on Figure 3.6 by looking at how fast all the nodes of a class return to a consistent state. We notice similar latency gain for Primary nodes and loss for Secondary nodes.

Overall, the simulation results match the analysis in Section 3.3 and confirm the latency advantage of Primary nodes over Uniform Gossip (Equation 3.5) and the small latency penalty of Secondary nodes (Equation 3.6).

3.5.5 Network overhead

The number of messages exchanged in the simulated system confirms Equation 3.3 in Section 3.3. Considering the experienced reliability, we observe in UPS an increase in the number of messages of 10^{-1}, 10^{-2} and 10^{-3} compared to Uniform Gossip for all three densities of 10^{-1}, 10^{-2} and 10^{-3} respectively.

3.6 Related work on consistency metrics

Most of the work related to UPS/GPS has been covered in Chapter 2. In the following we rapidly discuss earlier work seeking to evaluate a system's overall consistency, in order to position the metric we have proposed.

The approach of Zellag and Kemme [126] detects inconsistencies in cloud services by finding cycles in a *dependency graph* composed of transactions (nodes) and conflicts between them (edges). Counting cycles in the dependency graph yields a measure of consistency that is formally grounded. It requires however a global knowledge of the system, which makes it difficult to use in practice in large-scale systems. Golab et al. introduced first Δ-atomicity [123], [130] and then Γ [124], two metrics that quantify data staleness against Lamport-atomicity [131] in key-value store traces. These metrics are not suitable for our problem since they do not take into account the ordering of update operations.

More practical works [16], [125] evaluate consistency by relying on system specific information such as the similarity between different cache levels or the read-after-write latency (the first time a node reads the value that was last written).

Finally, CRDTs [46] remove the need to measure consistency by only supporting operations that cannot create conflicts. This naturally leads to eventual consistency without additional ordering requirements on communication protocols.

3.7 Conclusion

In this chapter, we have presented Update-Query consistency Primary-Secondary (UPS), a novel eventual consistency mechanism that offers heterogeneous properties in terms of data consistency and delivery latency. Primary nodes can deliver updates faster at the cost of a small consistency penalty, while Secondary nodes experience stronger consistency but with a slightly higher latency. Both sets of nodes observe a consistent state with high probability once dissemination completes.

To measure the different levels of consistency observed by Primary and Secondary nodes, we have proposed a novel and scalable consistency metric grounded in theory. This metric detects when a node performs read operations that conflict with the sequential specification of its associated object.

We formally analyzed the latency incurred by nodes as well as the overhead in message complexity in GPS, the underlying two-phase epidemic broadcast protocol. We then evaluated both the consistency and latency properties of UPS by simulating a 1 million node network. Results show how the density (fraction) of Primary nodes influences the trade-off between consistency and latency: lower densities favor fast dissemination to Primary nodes, while higher densities favor higher consistency for Secondary nodes.

Our future plans include deploying UPS in a real system and performing experiments to confront the algorithm to real-life conditions. We also plan to investigate how UPS could be combined with a complementary anti-entropy protocol [101] to reach the last few susceptible nodes and further improve its reliability.

We assumed in this chapter honest behaviors from the nodes of the system, but other threat models take into consideration the potential maliciousness of the nodes. We weaken our assumption in the next chapter by considering a system resilient to Byzantine nodes. In our next contribution, we propose a different take on heterogeneity in decentralized systems by offering adaptable trade-offs between security and resource consumption in blockchains.

CHAPTER 4

SHORTCUTTING THE BITCOIN VERIFICATION PROCESS

One of the limitations of blockchains is their storage scalability; their size is unbounded and their growth endless. The size of the Bitcoin blockchain increased from 75 GiB in August 2016 to 125 GiB in August 2017, taking 50 GiB in just a year. To fully benefit from the security model of Bitcoin, a bootstrapping node has to download and verify the entirety of the 125 GiB. Low-resource devices such as smartphones can choose the less costly option of only performing block difficulty verification where a mere 35 MiB of block headers, as of August 2017, must be downloaded and verified. However, such a drastic decrease in resource consumption comes with a security drawback: these nodes only verify the difficulty of blocks without verifying their correctness.

In this chapter, we propose our second contribution in the form of Dietcoin [1] and its diet nodes that can verify the correctness of blocks without having to pay the onerous price of a full chain download and verification. Diet nodes can verify the correctness of blocks by downloading shards of the UTXO set from other nodes for a few additional MiB of bandwidth on top of the base 35 MiB needed for block headers. Additionally, diet nodes can build upon the trust they have in a block to verify the next one and are thus able to verify entire subchains.

We first present in Section 4.1 the impact of the blockchain size on the cost to bootstrap a Bitcoin node. We then introduce in Section 4.2 Dietcoin and diet nodes designed to overcome this scalability issue. We present in Section 4.3 Dietcoin-specific related work, and we finally conclude and open interesting perspectives for this work in Section 4.4.

1. Not to be mistaken for Escobar's Dietbitcoin.

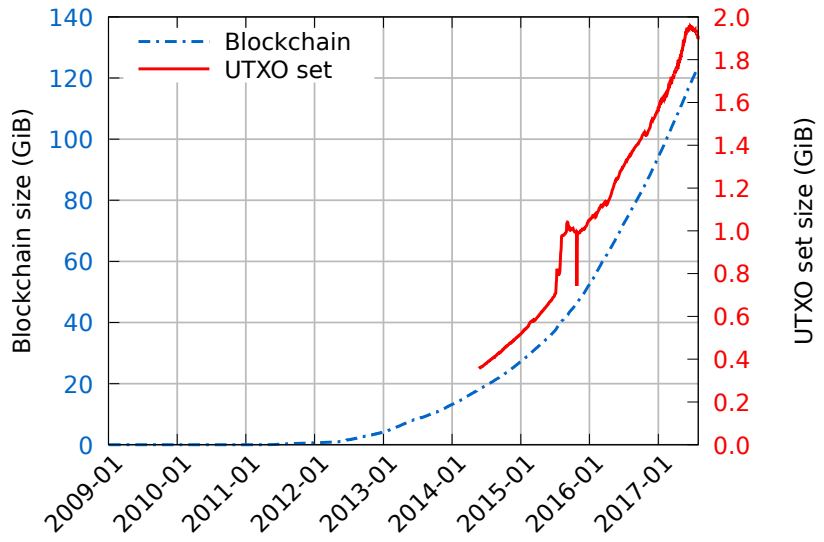


Figure 4.1 – Both the Bitcoin blockchain and the UTXO set have almost tripled in size over the past two years.

4.1 Bitcoin's costly bootstrap

As we have seen in Chapter 2.4, Bitcoin is the first successful and today most prominent cryptocurrency. The backbone mechanism behind Bitcoin is its blockchain that implements a tamper-proof decentralized ledger in which Bitcoin transactions are stored, secured by a series of computationally costly Proof-of-Work.

A core property of blockchains is the ability for everyone to verify their entire content. Verifying a blockchain remains, however, a particularly costly process. The verifying node must first download the entire blockchain, which in many cases has reached a size beyond the practical communication capabilities of many mobile devices. The Bitcoin blockchain, for instance, had grown to 125 GiB as of August 2017 (Figure 4.1), and historically follows an exponential growth despite having a capped block size (which is a subject of vigorous debates). As blockchains are ever growing structures, the problem can only become more acute.

Once the blockchain has been downloaded, the verifying node must then check its consistency block by block, a linear process that can take hours on high-end machines and much longer on smartphones. The exorbitant price of a full chain verification makes it unrealistic for low-resource devices to fully implement a blockchain protocol. Some blockchain systems, such as Bitcoin, therefore enable nodes to perform varying degrees of verification: *full nodes* verify everything while lightweight nodes only verify a small fraction of the data.

In the case of Bitcoin, this lightweight verification is known as *Simplified Payment Verification* (SPV). SPV nodes only download and verify a much reduced version of the Bitcoin

blockchain, comprised only of its block headers, which weights about 35 MiB (a reduction by three orders of magnitude compared to the full blockchain size). This summary version however only contains the chaining information making up the blockchain, not the recorded transactions. This information is sufficient for SPV nodes to verify the validity of the chain's structure and the Proof-of-Work difficulty of each block, but they cannot verify the correctness of each block's transactions. SPV nodes therefore have to assume a block correct solely based on its difficulty. They can therefore be tricked into trusting a difficult-enough block that contains malicious transactions that for instance are spending non-existing coins.

To protect themselves against malicious transactions, full nodes keep track of unspent funds in a structure known as the set of *Unspent TransaCTion Outputs* (UTXO set). The UTXO set is aggregated by linearly parsing the entire blockchain which makes it unfortunately costly to construct, to exchange (with a size of 1.9 GiB as of August 2017, see Figure 4.1), and to maintain, which explains why low-resource devices do not use it. Some devices such as smartphones can be used as Bitcoin payment devices, especially in regions with fragile infrastructures, but their security is constrained by the current limitations of the Bitcoin protocol.

In this contribution, we propose to bridge the gap between full nodes and SPV nodes by introducing *diet nodes*, and their associated protocol *Dietcoin*. Dietcoin strengthens the security guarantees of SPV nodes by bringing them close to those of full nodes. Dietcoin enables low-resource nodes to verify the transactions contained in a block, thus verifying the correctness of the block, without constructing a full-fledged UTXO set. In our protocol, diet nodes securely download from full nodes only the parts of the UTXO set they need in order to verify a block of interest. This selective download mechanism must, however, be realized with care. Diet nodes must be able to detect any tampering of the UTXO set itself, at a cost that remains affordable for low-resource devices, both in terms of communication and computing overhead.

In addition to verifying just one block, diet nodes can also verify the correctness of a subchain of arbitrary length by building upon the trust they have in a block to verify the next one. Diet nodes can shift the needed trust from the genesis block into the block of their choice, and start verifying the blockchain from this trusted block. In essence, Dietcoin enables low-resource devices to shortcut the linear verification process of Bitcoin.

4.2 The Dietcoin system

To address the vulnerabilities of SPV nodes and to improve the confidence mobile users can have in recent transactions, we propose *Dietcoin*, an extension to Bitcoin-like blockchains. Although our proposal can be applied to other existing Proof-of-Work blockchains using the UTXO model for coins, we describe Dietcoin in the context of the Bitcoin system as presented in Chapter 2.4.

The core of Dietcoin consists of a novel class of nodes, called diet nodes, which provide low-power devices with the ability to perform full block verification with minimal bandwidth and storage requirements. Instead of having to download and process the entire blockchain to build their own copy of the UTXO set, diet nodes query the UTXO set of full nodes and use it to verify the validity of the transactions they are interested in and the correctness of the blocks that contain them. This provides diet nodes with security guarantees that are close to those of full nodes, while incurring only a small communication and CPU overhead compared to those born by SPV clients

Since downloading the entire UTXO set would result in prohibitive bandwidth overhead, as shown in Figure 4.1, Dietcoin-enabled full nodes split their UTXO set into small shards to let diet nodes download only the shards that are relevant to the transactions in the block of interest.

To prevent diet nodes from trusting maliciously forged shards, the shard hashes are used as leaves of a Merkle tree, whose root is stored in each block. Having the Merkle root stored in blocks enables the UTXO shards to benefit from the same Proof-of-Work protection as transactions and allows nodes to verify the correctness of the Merkle root when they receive blocks. The Merkle root stored in block B_{k-1} secures the shards needed to verify B_k, which forces attackers to counterfeit at least these two blocks to include a forged transaction in B_k, as we demonstrate in Section 4.2.2.

In addition to the full verification done on the block of their interest B_k, diet nodes can increase their trust in B_k by fully verifying the subchain of length ℓ leading to B_k: $(B_{k-\ell+1}, ..., B_k)$. By doing so, diet nodes can verify that none of the ℓ blocks that lead to the transaction contain any illegal transactions or erroneous UTXO Merkle root, and thus form a valid postfix of the blockchain. To make a diet node trust a forged transaction in block B_k, an attacker has to counterfeit the subchain of $\ell + 1$ blocks $(B_{k-\ell}, ..., B_k)$ as we show in Section 4.2.3. Thanks to the Proof-of-Work protection, the cost of this attack increases exponentially as ℓ increases linearly.

In the following, we detail the operation of Dietcoin by first describing how full nodes can provide diet nodes with verifiable UTXO shards. Secondly we discuss how miners link blocks to the state of their UTXO set. We then explain how diet nodes can extend the verification process from one block to a subchain of any length. Finally, we detail the operation of diet nodes when verifying transactions.

4.2.1 Sharding the UTXO set

To enable the operation of diet nodes, Dietcoin-enabled full nodes need (i) to provide diet nodes with shards of the UTXO set, while (ii) enabling them to verify that these shards are authentic.

To satisfy (i), Dietcoin-enabled full nodes store the UTXO set resulting from the application

of the transactions in each block in the form of shards. The size of shards is capped such that the average over all shard sizes must not cross a predefined limit $\mathcal{M}$. The use of shards enables diet nodes to download only the relevant parts of the UTXO set and also limits the storage requirements at full nodes, which only need to store the modified shards for each block to let diet nodes query older versions of the shards. Similarly, capping the averaged maximum size for a shard to $\mathcal{M}$ limits the bandwidth employed by diet nodes in the verification process.

To satisfy (ii), full nodes also maintain a Merkle tree that indexes all the shards of the UTXO set. Using shards also proves advantageous with respect to this Merkle tree. If nodes were to index UTXO entries directly, the continuous changes in the UTXO set would cause the Merkle tree to become quickly unbalanced, leading to performance problems or requiring a potentially costly self-balancing tree. The use of shards, combined with the right sharding strategy, gives the UTXO Merkle tree a relatively constant structure, enabling shards to be updated in place most of the time. Moreover, it makes it possible to predict the size of the UTXO Merkle tree and its incurred overhead, which enables us to better control and balance the storage overhead for full nodes and the bandwidth requirements of diet nodes.

A number of sharding strategies satisfy the requirement of a rarely changing number of shards. In this contribution, we use the simplest approach consisting of indexing UTXO entries by their first k bits. This strategy resembles a random approach since the first bits of an UTXO are the transaction hash it references, which value is expected to be random due to the uniformity property of the SHA-256 hash function. This strategy comes with the added advantages of obtaining shards of homogeneous size and a full binary Merkle tree with 2^k leaves.

Keeping in mind the shard size cap, nodes locally adapt the value of k to cope with the growth of the UTXO set. When the average shard size reaches above $\mathcal{M}$, nodes increment k by one thereby *(i)* halving the average shard size, and *(ii)* adding one layer to the Merkle tree, doubling its storage footprint in the process.

4.2.2 Linking blocks with the UTXO set

With reference to Figure 4.2, consider a Dietcoin-enabled miner that is mining block B_k, and let UTXO$_k$ be the state of the UTXO set after applying all the transactions in B_k. The miner stores the root of the UTXO Merkle tree associated with UTXO$_k$ as an unspendable output in the first transaction of block B_k before trying to solve the Proof-of-Work as shown in Figure 4.2. Storing the root of the UTXO Merkle tree in the first transaction of the block does not require any modification to the structure of Bitcoin's blocks. Thus it is possible for Dietcoin-enabled full nodes and miners to co-exist with their legacy Bitcoin counterparts, which prevents conflicts in the case of a network upgrade to Dietcoin. Dietcoin-enabled full nodes verify the value of the UTXO Merkle root against their own local copy when they verify a block, while legacy Bitcoin nodes simply ignore it.

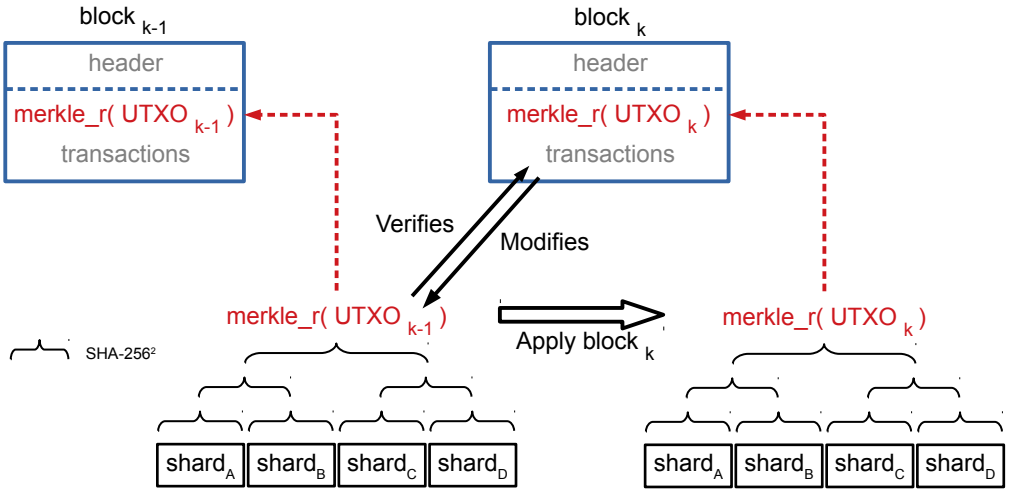


Figure 4.2 – The UTXO set is updated every time a block is validated. For a counterfeited block to be validated by diet nodes, a malicious node has to forge at least two consecutive blocks: the first block B_{k-1} containing a fake Merkle root of UTXO$_{k-1}$, and the second block B_k spending fake coins validated by the fake UTXO$_{k-1}$.

The UTXO Merkle tree provides a computationally efficient way for diet nodes to verify whether the shards they download during the verification process are legitimate and correspond to the current state of the ledger. Still referring to Figure 4.2, let us consider a diet node d that wishes to verify a transaction in block B_k. Node d needs to obtain: block B_k, the UTXO Merkle root in block B_{k-1}, the shards of the UTXO set between the two blocks, and the elements of the associated Merkle tree that are required to verify their legitimacy. It can then verify the shards using the root stored in block B_{k-1}'s first transaction, and use them to verify the correctness of the transactions in B_k.

We observe that storing the Merkle-root referring to the state after the block into the block itself forces diet nodes to download two blocks to verify transactions. This makes it inherently harder for an attacker to provide a diet node with a fake block B_k because it would need to forge not only block B_k but also block B_{k-1}.

4.2.3 Extended verification

Diet nodes have the ability to extend their confidence in a block by iterating the verification process towards its previous blocks. By doing so, diet nodes ensure the correctness of the UTXO Merkle root present in block B_{k-1} used to verify the correctness of block B_k. The extended verification can be performed on a subchain of any length ℓ provided that the verifying diet node can query UTXO shards of any age. A diet node fully verifying the subchain $(B_{k-\ell+1}, ..., B_k)$ can only be tricked into trusting a malicious transaction in block B_k if the at-

tacker manages to counterfeit the $\ell + 1$ blocks between from $B_{k-\ell}$ and B_k, which becomes exponentially more costly as ℓ increases linearly.

Block $B_{k-\ell}$ contains the UTXO Merkle root that serves as a basis for the verification of the subsequent blocks. Since the block $B_{k-\ell}$ is not verified, it is thus trusted by the diet node. A comparison can be made with full nodes where the first block of the chain, the genesis block, is hard-coded and is therefore trusted. By shifting the trust from the genesis block to the block $B_{k-\ell}$ for diet nodes, Dietcoin effectively shortcuts the verification process.

Picking the value of ℓ exhibits a trade-off between security and verification cost. On the one hand choosing a large ℓ draws the behavior of the diet node closer to that of a full node, while on the other hand choosing a small ℓ draws it closer to that of an SPV node. The user can make her decision based on which block depth ℓ is, in her opinion, large enough to render all blocks prior to $B_{k-\ell}$ unlikely to be counterfeited. For instance, a diet node user can choose ℓ such that the trusted block has a block depth of 6 or greater since it is the de facto standard in Bitcoin to consider blocks of depth 6 or greater as secured [132].

In the case depicted in Figure 4.2, assuming $\ell = 2$, a diet node downloads block B_{k-1} to verify the transactions and the UTXO Merkle root in B_{k-1} to further increase its confidence in B_k. To verify B_{k-1}, the diet node uses the UTXO Merkle root in B_{k-2}.

4.2.4 Detailed operation

Equipped with the knowledge of Dietcoin's basic mechanisms, we can now detail the verification process carried out by diet nodes. Algorithm 4 depicts the actions taken by a diet node when its user starts the application using Dietcoin and compares it with those taken by legacy SPV clients. Black dotted lines [•] are specific to diet nodes, while hollow dotted lines [○] are common to both diet and SPV nodes.

The algorithm begins when the application using Dietcoin starts and updates its view of the blockchain. In this first part of the algorithm, a diet node behaves exactly in the same manner as an SPV node. First, it issues a query containing the latest known block hash and an obfuscated representation of its own public keys in the form of a bloom filter (lines 4-5). A full node responds to this query by sending a list of all the block headers that are still unknown to the SPV/diet node, together with the transactions matching the SPV/diet node's bloom filter, and the information from the transaction Merkle tree that is needed to confirm their presence in their blocks. Using this information, the SPV/diet node verifies the received headers (including Proof-of-Work verification) and updates its view of the blockchain (line 6).

Since the response from the full node might contain false positives due to the use of a bloom filter for public key obfuscation, the next verification step consists in ensuring that the received transactions match one of the user's public keys (lines 8-9). Once false positives are discarded, the SPV/diet node verifies that the received transactions are in the blocks (line 10-12).

Algorithm 4 – SPV [○] and Diet [○●] node p (compacted)

```
 1: parameters: ○ pubKeys_p, ● maxDepth_p, ● maxLength_p(ℓ)
 2: global variables: ○ headerStore_p, ○ tipId_p, ● highestVerified_p
○3: procedure UPDATECHAIN( )                                   ▷ App interface
○4:     filter ← bloomFilter(pubKeys_p)
○5:     ({header, txMTree, txs}) ← send QUERYMERKLEBLOCKS(tipId_p, filter)
○6:     tipId_p ← verifyHeaders((headers))
○7:     for all {header, txMTree, txs} ∈ ({header, txMTree, txs}) do
○8:         if ∀k ∈ pubKeys_p : k ∉ {tx.inputs ∪ tx.outputs} then
○9:             continue                          ▷ Ignore bloom filter false positives
○10:        assert(∀tx ∈ txs : HASH(tx) ∈ txMTree)
○11:        builtTxMRoot ← buildMRoot(txMTree)
○12:        assert(builtTxMRoot = header.txMRoot)
●13:        VERIFYBLOCKSUPTO(height(header))
○14:        callback(txs, header)                          ▷ Callback to app
●15: procedure VERIFYBLOCKSUPTO(last)
●16:     ▷ Do not verify blocks twice or below maxDepth_p
●17:     first ← max(highestVerified_p, height(tipId_p) − maxDepth_p)
●18:     ▷ Verify up to maxLength_p blocks
●19:     first ← max(first, last − maxLength_p)
●20:     if first ≥ last then
●21:         return                                     ▷ Fallback to SPV mode
●22:     ▷ The first UTXO Merkle root is not verified
●23:     utxoMRoot ← send QUERYUTXOMROOT(HASH(headerStore_p[first]))
●24:     for all blockId of height ∈ [first + 1, last] do
●25:         block ← send QUERYBLOCK(blockId)
●26:         {shards, utxoMTree} ← send QUERYUTXOS(blockId)
●27:         assert(∀shard ∈ shards : HASH(shard) ∈ utxoMTree)
●28:         builtUtxoMRoot ← buildMRoot(utxoMTree)
●29:         assert(builtUtxoMRoot = utxoMRoot)
●30:         for all tx ∈ block.transactions do
●31:             for all i ∈ tx.inputs do
●32:                 shard ← shards[getShardKey(i)]
●33:                 assert(i ∈ shard ∧ valid proof of ownership of i)
●34:                 shard.remove(i)
●35:             for all o ∈ tx.outputs do
●36:                 shards[getShardKey(o)].add(o)
●37:         utxoMTree ← updateMTreeInPlace(utxoMTree, shards)
●38:         utxoMRoot ← buildMRoot(utxoMTree)
●39:         assert(utxoMRoot = block.utxoMRoot)
●40:         highestVerified_p ← height(blockId)
```

At this point, a standard SPV node simply returns the received transactions to the application (line 14). A diet node, on the other hand, continues the verification process. To this end, the diet node first computes which blocks to fully verify to ensure that *(i)* no block is fully verified twice (line 17), *(ii)* old blocks considered by the user as secured enough are not fully verified (parameter $\mathsf{maxDepth}_p$, line 17) and *(iii)* only a subchain of limited length ℓ is fully verified (parameter $\mathsf{maxLength}_p$, line 19). If no block is selected for full verification, the diet node falls back to SPV mode (lines 20-21).

To bootstrap the full verification process, the diet node must first download the UTXO Merkle root present in the block prior to the first block to verify (line 23). For each of the selected blocks B_k, the diet node downloads from Dietcoin-enabled full nodes *(i)* the block B_k itself (line 25), *(ii)* the state, before B_k, of the UTXO shards associated with both the block's transactions' inputs and outputs (line 26), and *(iii)* the partial UTXO Merkle tree required to prove the integrity of the downloaded shards (line 26).

Once it has all the data, the diet node proceeds with the verification process. For each block B_k, it first verifies that the downloaded UTXO shards match the UTXO Merkle root from the previous block B_{k-1} (lines 27-29). Then it verifies that the transactions in each block use only available inputs from these shards (lines 32-33), and computes the new state of these shards based on the transactions in B_k (lines 34, 36). Finally it verifies that the updated shards lead to the UTXO Merkle root contained in B_k (lines 37-39). Once the verification process has terminated, the diet node returns the transactions associated with the local user to the application (line 14).

4.3 Related work on UTXO commitment

Making the UTXO set available for queries between nodes has been discussed several times in the Bitcoin community over the past few years. Bryan Bishop published a comprehensive list of such proposals [133] that share some of the following goals: *(i)* enabling faster node bootstrap, *(ii)* strengthening the security guarantees of lightweight nodes, and *(iii)* scaling the UTXO set to reduce its storage cost. The primary goal of Dietcoin is to strengthen the security of lightweight nodes. Dietcoin's strongest feature is the ability for diet nodes to efficiently perform subchain verification, as described in Section 4.2.3, which offers stronger security guarantees than the referenced proposals made by members of the community. Moreover, even though this chapter focuses on the security of lightweight nodes, it is also possible to bootstrap full nodes faster with Dietcoin.

Sharing similar goals with Dietcoin, Andrew Miller suggests to store in blocks the root of a self-balancing Merkle tree built on top of the UTXO set [134]. In such a system, lightweight nodes only download the UTXOs they need, which results in a lower bandwidth consumption

than with shards as we propose it, but at the cost of a greater storage overhead since the stored Merkle tree is larger. Moreover, Dietcoin combines a full, and thus always balanced, Merkle tree built on top of 2^k shards that can each be updated in place as blocks are appended to the chain. This combination of tree stability and updatable shards enables efficient subchain verification, as described in Section 4.2.3, and adapting this feature to a system using a self-balancing Merkle tree does not seem trivial.

Vault [135] also proposes to use Merkle trees to securely record the state of the distributed ledger in recent blocks, and shards this state across nodes, to reduce storage costs. Contrarily to Dietcoin, however, Vault targets balance-based schemes, such as introduced by Ethereum, in blockchains relying on Proof-of-Stake consensus. Vault further stores individual accounts in the Merkle trees, rather than UTXO shards as we do. This represents a different and to some extent orthogonal trade-off to that of Dietcoin, in that Vault chooses to increase the size of Merkle tree witnesses that must be included in transactions, but removes the need for lightweight nodes to download UTXO shards.

4.4 Conclusion

We have presented the design of Dietcoin, an improvement over Bitcoin that strengthens the security guarantees of light nodes by bringing them closer to those of full nodes. The Dietcoin protocol enables low-resource devices to become diet nodes which can verify the transactions contained in blocks without constructing a full-fledged UTXO set. Despite focusing on Bitcoin in this chapter, the features of Dietcoin can be adapted to fit other Proof-of-Work blockchains using the UTXO model (e.g., Litecoin, Zcash) or the account model (e.g., Ethereum).

Diet nodes can download the shards of the UTXO set from full nodes that they need in order to verify a block, or a subchain of blocks, of interest. Diet nodes are able to detect any tampering of the UTXO set itself, at a cost that remains affordable for low-resource devices, both in terms of communication and computing overhead. In our approach, Dietcoin-enabled full nodes split their UTXO set into small shards, and enable diet nodes to download only the shards that are relevant to the transactions in the block, while verifying that these shards do indeed corresponds to the state of the UTXO set for the block they are verifying.

In addition to the scenario we advanced, the mechanisms powering Dietcoin can also be used to (i) quickly bootstrap a full node by rendering the full UTXO set downloadable and (ii) cheaply verifying the reception of a coin (a transaction output) long after the fact by ensuring that it is in the expected UTXO shard.

Future work

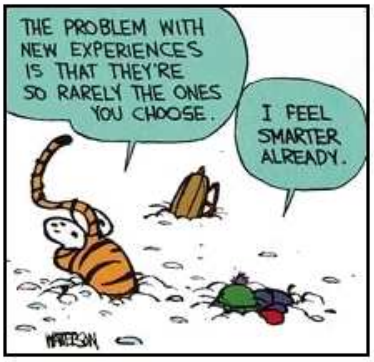
The next milestone for Dietcoin is the completion of its evaluation designed to obtain results from exploiting a real dataset in a realistic environment powered by the de facto standard bitcoin client library.

We plan to evaluate the performances of Dietcoin by implementing its features in the bitcoinj library [2] and have a Dietcoin full node replay the Bitcoin blockchain to a diet node running on an Android smartphone. In our setup, the full node replays the existing Bitcoin blockchain, keeps track of the UTXO shards and enriches the blocks with the UTXO Merkle roots. The diet node running on Android verifies the correctness of the enriched block it receives and is being monitored for us to evaluate its bandwidth, storage, CPU and battery consumption.

The implementation is well underway and the code can be adapted to other Bitcoin improvement proposals to test their claims on a real dataset in a realistic environment powered by the de facto standard bitcoin client library.

Once the initial evaluation complete, we plan to compress UTXO shards to further diminish the bandwidth requirements of diet nodes. We also want to experience with other UTXO sharding policies such as sharding by the first k bits of a UTXO owner, or sharding by UTXO age.

2. `https://github.com/bitcoinj/bitcoinj`



CHAPTER 5

CONCLUSION

Ensuring the reliability of distributed services has become increasingly difficult and expensive in our era of a rapidly growing Internet. Distributed services must scale to match the growing demand and must do so in an affordable manner. To avoid the high cost of tightly coupled coordination, service providers favor data replication protocols that offer more flexible consistency guarantees. In the quest for scalability, decentralized systems are particularly interesting for large-scale systems as their loose coupling makes them scalable by design. However, services tend to offer uniform quality of service to their users, thus completely ignoring the uniqueness of each node's requirements.

We have proposed in this thesis two contributions on data replication protocols that harness the heterogeneity requirement in large-scale systems to offer personalized quality of service to users.

Gossip Primary-Secondary

The first contribution that we presented in this thesis is the couple formed by *Gossip Primary-Secondary* (GPS) and *Update consistency Primary-Secondary* (UPS). GPS is a novel priority gossip protocol that enables the UPS consistency criterion to offer heterogeneity in consistency and update delivery latency for data replication.

The network in GPS is split into two subsets with a minority of primary nodes and a majority of secondary nodes. Thanks to UPS, primary nodes are able to deliver updates faster than secondary nodes but in a less orderly fashion. On the other hand, UPS provides secondary nodes with stronger consistency at the cost of a small increase in the latency of update delivery.

The provided formal analysis displays the difference in latency between primary and secondary nodes, as well as the increased number of messages incurred by GPS. The simulations performed on a 1-million-node network shows the claimed differentiated guarantees in consistency and latency, and the impact of the density of primary nodes in the network.

Dietcoin

The second contribution displayed in this thesis brings personalization to security and resource requirements in Bitcoin. Existing Bitcoin light nodes can check the difficulty of blocks

but are unable to check their correctness; they have to assume blocks are correct and are thus less secured than full nodes which do verify the correctness of each block.

By modifying the storage, protocol, and consensus layers of Bitcoin, we propose in *Dietcoin* our *diet node*, whose security and resource consumption lie in between those of existing light nodes and full nodes. Diet nodes can verify the correctness of cherry-picked blocks by querying shards of the UTXO set of full nodes at a low bandwidth cost. Diet nodes can build upon the trust they have in a verified block to verify its ℓ successive blocks in the chain. By verifying the correctness of an entire subchain, diet nodes increase the trust in the most recent block of the subchain by ensuring that it is appended to a correct subchain.

Perspectives

Pushing back the boundaries of the CAP theorem

We propose in GPS a model with two classes of nodes, primary and secondary nodes, with opposing consistency and update delivery latency trade-offs. The personalization of each node quality of service could be extended further by proposing per-device personalization instead of having to fit nodes in one of the two pre-defined classes. Each node could have its own tailor-fitted consistency criterion.

While GPS does offer different qualities of service to different nodes in large-scale systems, each node benefits from the same quality of service throughout its existence. Time-based trade-offs could be explored in such systems. For example, one could offer in a similar system strong consistency with low latency whenever possible but decrease the latency guarantees during peaks of updates to limit inconsistencies.

Lowering the verification and storage cost of blockchains

Dietcoin enables low-resource nodes to verify the correctness of sub-chains of blocks once they have verified the difficulty of the blockchain. Recent work on Non-Interactive Proofs of Proof-of-Work (NIPoPoWs [136]) suggests that the linear process of chain difficulty verification can be drastically reduced into a process of logarithmic cost. It would be interesting to explore how NIPoPoWs could be combined with Dietcoin to greatly reduce both chain difficulty verification and block correctness verification.

Since Dietcoin decouples the storage from the linear chain verification and shards the UTXO set, we could deploy a Distributed Hash Table (DHT) to store blocks and UTXO shards, and mutualize the storage requirements of both diet and full nodes. In addition, a DHT would make it easier for diet nodes to find old UTXO shards that might only be stored by a small number of nodes.

While Dietcoin as it was presented in this thesis relies on the UTXO model for coins, it could be adapted to the account model used for instance in Ethereum. Ethereum in particular plans to introduce strong consistency, or finality as they call it, to old-enough blocks via the Casper upgrade [137] by transitioning from Proof-of-Work to a mix of Proof-of-Work and Proof-of-Stake. Since old-enough blocks are strongly consistent, nodes do not have to verify them according to the security model of post-Casper Ethereum. Therefore, a diet node in such a system could choose a strongly consistent block as its trusted block, that is needed to start the subchain verification, and obtain the exact same security guarantees as a full nodes in a post-Casper Ethereum. This would completely bridge the gap between light nodes and full nodes.

On large-scale decentralized coordination

As we have shown in this thesis, designing efficient large-scale coordination protocols still remains a challenging task. Unfortunately, using deterministic protocols to coordinate nodes in large-scale decentralized systems is extremely costly, and only probabilistic protocols are available for open-membership systems.

Researchers in the distributed computing field have been exploring consensus for decades now, but always under the prism of closed-membership networks. With the advent of the open-membership Nakamoto consensus, a key and yet unanswered question in distributed computing therefore arises: Is deterministic consensus with an unbounded and dynamic number of nodes possible? And if yes, under which synchrony assumptions?

On a similar but more practical note, as Proof-of-Work membership is the only open-membership protocol to date, the existence of other Sybil-proof open-membership protocols remains an open question. Tackling this question could provide worthy results able to replace Proof-of-Work membership and its too expensive energy and hardware consumption coopetition (cooperative competition). Moreover, designing a Sybil-proof open-membership protocol that is fair and deterministic would be a first step to solve our first question.

APPENDIX A

RÉSUMÉ EN FRANÇAIS

L'imprimerie a permis au peuple de lire.
Internet va lui permettre d'écrire.

Benjamin Bayart

Si vous avez déjà perdu vos données personnelles suite à une panne d'ordinateur, vous savez à quel point les sauvegardes sont importantes. Pour les fournisseurs de services, les sauvegardes sont essentielles à la robustesse de leurs services, car la perte des données de leurs utilisateurs n'est pas une option envisageable. Les données sont donc généralement répliquées dans plusieurs régions géo-réparties pour éviter les pannes catastrophiques affectant des régions entières. Toutefois, dans cette situation, chaque mise à jour doit également être répliquée par chaque instance pour conserver des réplicas cohérents, ce qui devient de plus en plus difficile à mesure que le nombre d'utilisateurs, et donc de mises à jour, augmente. La croissance spectaculaire du nombre de personnes connectées à Internet et le temps de plus en plus long qu'elles consacrent à Internet constituent clairement un défi pour la réplication de données. Ce défi et ses ramifications sont au cœur de cette thèse.

Démocratisation de l'accès à Internet

Selon l'*Union internationale des télécommunications* (UIT), le taux d'adoption d'Internet dans le monde est passé de 22 % en 2007 à 48 % en 2017 [1], [2], ce qui représente 3,6 milliards de personnes connectées. La baisse combinée du prix de la connexion haut débit — en particulier dans les pays les moins avancés où il a diminué d'un facteur 2,5 entre 2013 et 2016 [2] — et la baisse du prix des appareils donnant accès à Internet, tels que les smartphones, a rendu l'accès à Internet beaucoup plus abordable et a fortement accru son utilisation.

Parallèlement à l'augmentation du taux d'adoption, le taux d'engagement a également augmenté. Selon Flurry Analytics [3], les citoyens américains sont passés de 2h40 par jour devant leur écran de smartphone en 2013 à 5 heures par jour en 2016, dont un tiers est consacré aux réseaux sociaux en ligne.

L'effet de rebond sur le taux d'adoption peut être vu à travers la croissance du nombre d'utilisateurs de réseaux sociaux populaires et d'applications telles que Facebook ou Pokémon Go.

Il a fallu un peu plus de 4 ans à Facebook, le plus grand réseau social en ligne, pour passer d'un milliard d'utilisateurs actifs en octobre 2012 à deux milliards en janvier 2017 [4], permettant ainsi des centaines de milliers de mises à jour et de publications de contenu par minute [5] et des dizaines de millions de messages par minute via leur plate-forme de messagerie WhatsApp [6]. Dans un autre genre, Pokémon Go a battu tous les records de taux d'adoption [7] : Avec plus de 100 millions de téléchargements de jeux au cours du premier mois suivant son lancement et 400 millions supplémentaires [8] au cours du second mois, Pokémon Go est devenu le plus grand lancement de jeux pour mobile de l'histoire.

Comme Internet est à la fois un moyen de liberté d'expression fondamentale [9] et un moyen de développement sociétal et économique, il est plus que probable qu'une fraction toujours croissante de la population mondiale ait accès à Internet. Les régions très peuplées telles que l'Afrique subsaharienne, l'Asie du Sud et l'Asie du Sud-Est n'ont pas encore un fort taux d'accès à Internet. Toutefois, selon l'UIT, les pays les moins avancés, principalement présents dans ces régions, enregistrent la plus forte croissance d'abonnements au haut débit fixe et mobile [2] et rattraperont potentiellement les taux d'adoption des pays développés au cours de la prochaine décennie.

Les appareils personnels actuels ont de nombreuses capacités et prennent en charge un large éventail d'applications. Ils se retrouvent presque partout aujourd'hui dans nos vies sous différentes formes. Alors que les gens ne possédaient qu'un seul ordinateur de bureau ou ordinateur portable dans les années 1980-1990, ils ont commencé à posséder des smartphones et des tablettes dans les années 2000. Ces appareils de poche peuvent être rapidement consultés n'importe où et n'importe quand, simplifiant ainsi l'accès à Internet. Sur le même plan, l'émergence de l'Internet des objets (*Internet of Things*, IoT) est une autre raison de s'attendre à une augmentation du nombre d'appareils connectés à Internet par habitant. Dans l'approche IoT, les appareils n'ont plus de capacité générique mais sont conçus pour être adaptés à leurs tâches. Ces dispositifs peuvent être conçus pour le divertissement, tels que les téléviseurs et les consoles, pour améliorer le confort des foyers, tels que les systèmes de vidéosurveillance et de gestion de divers capteurs, et peuvent même être portés sur soi, tels que les montres et les assistants de remise en forme. En plus des appareils personnels, il existe d'autres sous-domaines de l'IoT adaptés par exemple à la gestion urbaine (*Smart city*) ou à des tâches liées à l'entreprise (*Industrie 4.0*).

L'augmentation spectaculaire du nombre d'appareils connectés à Internet, combinée à la multiplication de leurs utilisations, pèse de manière prévisible sur la disponibilité des services auxquels l'utilisateur a accès et doit être gérée par les fournisseurs de services. Dans le cas de Pokémon Go, le lancement du jeu attendu était parsemé de pannes de serveur [10]-[12] empêchant une utilisation fluide pour ses utilisateurs et générant par la suite une mauvaise réputation pour l'application. Même le cloud de Google qui prend en charge l'application a eu

des problèmes de dimensionnement en raison de cette croissance époustouflante unique [13].

Étant donné que les utilisateurs évitent les services non fiables, la capacité à être extensible est un impératif non seulement pour la croissance d'un service, mais également pour son succès et sa survie.

Architectures pour systèmes extensibles

Les fournisseurs doivent multiplier le nombre de serveurs qu'ils déploient afin d'améliorer les performances et la fiabilité de leurs services, et afin de fournir des services hautement extensibles capables de satisfaire un nombre croissant d'utilisateurs [14].

L'ajout de serveurs augmente la résilience aux pannes matérielles et empêche l'épuisement des ressources à mesure que le nombre d'utilisateurs augmente. Les serveurs sont généralement répartis sur différentes régions géographiques [15], [16]. Cela empêche les pannes catastrophiques locales à une région d'affecter le système entier, et permet de réduire le temps de latence moyen des clients naturellement géo-répartis en rapprochant les serveurs des clients. Cependant, le déploiement d'un système géo-réparti à grande échelle contraint l'utilisation de réseaux à grande distance qui sont intrinsèquement moins fiables et qui présentent des latences plus élevées que les réseaux locaux aux centres de données. Ainsi, la réplication de données dans un système géo-réparti doit faire face à la probabilité accrue de détérioration des communications en plus de délais plus longs. Ce problème est encore aggravé par le fait que ces architectures sont souvent étroitement couplées, ce qui facilite la coordination entre les serveurs mais empêche l'extensibilité globale des systèmes.

Une autre approche de la conception de systèmes vastes consiste à utiliser des architectures faiblement couplées, extensibles par nature. Les systèmes décentralisés sont une instance de tels systèmes [17] ; ils offrent une grande robustesse face aux pannes et à la dégradation du réseau, ainsi qu'une répartition efficace de la charge [18]. Les nœuds de ces systèmes étant moins dépendants les uns des autres, (i) leurs utilisateurs bénéficient d'un contrôle plus strict sur les ressources qu'ils choisissent de contribuer (réalisant de fait une décentralisation du contrôle) et (ii) les nœuds honnêtes sont moins affectés par des comportements malveillants d'autres nœuds. La coordination dans les systèmes décentralisés est toutefois plus difficile à réaliser que dans les systèmes à couplage étroit. Les protocoles de coordination classiques tels que Paxos [19] coûtent très cher en nombre de messages et reposent généralement sur un leader pour réduire ce coût. Etant donné qu'un leader a tendance à resserrer le couplage d'un réseau, les protocoles de coordination récents se passant de leader offrent une perspective remarquable pour les architectures visant à maximiser la décentralisation. Les blockchains sont un exemple de système de coordination sans leader, sur lesquels nous reviendrons plus loin dans cette introduction en raison de leur importance pour le travail que nous présentons.

Les systèmes à couplage étroit sont plus simples et plus intuitifs à gérer que les systèmes à couplage lâche et sont donc plus répandus dans les systèmes opérationnels. Cependant, à mesure que la charge d'un service augmente, ses développeurs doivent souvent échanger la simplicité de la centralisation et du couplage étroit contre l'efficacité de la géo-répartition et du couplage lâche.

Dans le contexte de la réplication de données, le couplage étroit offre des propriétés de cohérence des données facile à utiliser pour les développeurs, mais cette simplicité est payée par des performances inférieures à celles des systèmes à réplication géographique hautement disponibles.

Approches pour la réplication de données

Plus précisément, les algorithmes de réplication de données traditionnels fournissent une *cohérence forte* des données en mettant la sûreté de fonctionnement au premier rang. Ces algorithmes se concentrent sur le maintien de répliques exactes à tout moment sur chaque nœud, ce qui est souvent effectué à l'aide d'un coordinateur unique, mais tournant, pour assurer un consensus. En règle générale, un nœud donné agit en tant que chef de file [19], [20] parmi les appareils pour dicter l'ordre des événements aux autres appareils afin de garantir un ordre total des événements et des répliques exactes (comme dans Apache ZooKeeper [21] et Apache Kafka [22]).

Bien que simple et intuitive, cette méthode s'adapte mal à une montée rapide en charge en raison des goulots d'étranglement créés par la présence d'un coordinateur unique. Pire encore, les consensus à base de quorum peuvent nécessiter une consommation en ressources encore plus importante en cas de panne ou de disparition du chef. De plus, la latence due à la géo-distribution des systèmes nuit considérablement à l'efficacité en communication de ces algorithmes. La nécessité de s'appuyer sur des mécanismes de quorum en cas de défaillance empêche les systèmes à cohérence forte d'offrir une haute disponibilité lorsqu'ils sont construits sur des réseaux à grande échelle sujets à des retards de messages et à des partitions transitoires du réseau [23], [24].

Tenant compte des changements d'échelle des systèmes, l'approche opposée de *cohérence à terme* privilégie la disponibilité par rapport à la sécurité lors de la réplication de données. Certains systèmes middleware majeurs tels que Apache Cassandra [25] et Riak [26] fournissent une cohérence à terme par construction. Comme la cohérence à terme nécessite moins de coordination entre les nœuds, la latence et les pannes ont moins d'impact sur le fonctionnement du système. La cohérence à terme souffre malheureusement de ses propres inconvénients. Sa principale faiblesse est la probabilité pour les utilisateurs de faire l'expérience d'incohérences transitoires ou d'observer un état obsolète, ce qui va à l'encontre d'une expé-

rience utilisateur fluide et en apparence naturelle. De nos jours, la cohérence forte n'est pas de facto utilisée lors de la conception d'un système, mais correspond plutôt à une propriété coûteuse qui ne devrait être atteinte que lorsque cela est nécessaire.

L'émergence récente de registres répartis tolérants aux pannes a toutefois remis en cause notre compréhension du compromis entre extensibilité et cohérence : des systèmes tels que Bitcoin [27] offrent des solutions à cohérence forte avec une forte probabilité, tout en acceptant en principe un nombre arbitraire et dynamique de participants.

Données permanentes avec des registres répartis tolérants aux pannes

Les registres répartis sont apparus il y a 10 ans avec Bitcoin [27] et ont depuis laissé leur marque dans les médias et les institutions [28]. Derrière l'hyper médiatisation alimentée par la spéculation financière, Bitcoin et sa technologie sous-jacente, la blockchain, permettent la création de services radicalement nouveaux. Les exemples les plus connus de ces services sont les plates-formes décentralisées pour les transferts de valeur, également appelées cryptomonnaies (comme Dogecoin [29] et Monero [30]) et les plates-formes décentralisées pour le calcul informatique (comme Ethereum [31]) qui permettent aux applications décentralisées de s'exécuter en fournissant des services tels que des contrats automatisés et des organisations autonomes décentralisées.

D'un point de vue système, les blockchains constituent une nouvelle famille de systèmes décentralisés qui gèrent un registre approuvé par tous avec des accès en lecture et en écriture pouvant être publiques. Une des propriétés clés des blockchains est leur *vérifiabilité* : tout membre y accédant peut s'assurer de l'exactitude du registre en téléchargeant et en vérifiant tout son historique. La principale nouveauté de Bitcoin réside dans son consensus publique, éponymement appelé consensus de Nakamoto, qui garantit la cohérence du registre. Contrairement aux techniques classiques de consensus tolérant les comportements Byzantin [32], le consensus de Nakamoto n'a pas besoin de savoir combien de participants se trouvent dans le système, ce qui en fait le premier protocole de consensus à adhésion ouverte. Le consensus de Nakamoto offre de nouvelles perspectives aux systèmes décentralisés en rendant accessible au public l'abstraction la plus puissante de l'informatique répartie qu'est le consensus.

Cependant, le consensus de Nakamoto n'est pas non plus une solution parfaite. Son principal inconvénient en rapport avec cette thèse est le manque de déterminisme dans ce consensus. Le consensus de Nakamoto ne fournit que des garanties probabilistes de cohérence au registre qu'il protège, en ce sens que les décisions prises par consensus récent ont une probabilité non nulle d'être annulées. Une autre question très controversée concerne le mécanisme principal utilisé par les blockchains inspirés par Bitcoin, qui nécessite d'énormes ressources de

calcul et qui donne plus de pouvoir de décision aux participants disposant de plus de puissance de calcul. Ce mécanisme encourage les participants à participer à une course dans laquelle la puissance de calcul la plus élevée, et donc le plus grand consommateur d'énergie, l'emporte.

Grâce à leur consensus décentralisé et au fait que tout le monde peut vérifier leur exactitude, les blockchains sont décrites comme des systèmes ne nécessitant pas de confiance. Aucun membre en particulier ne doit être considéré comme fiable pour que le système reste correct, ce qui suit la devise "*Ne faîtes pas confiance, vérifiez*.".

Nous avons choisi dans cette thèse d'établir une distinction claire entre les registres décentralisés et les registres centralisés — au sens de centralité du contrôle — et de ne désigner les blockchains que par des registres décentralisés. Les registres centralisés, également appelés blockchains à autorisations (par exemple, HyperLedger Fabric [33], R3 Corda [34]) fonctionnent avec un nombre connu de participants et peuvent donc bénéficier des décennies de travaux théoriques et pratiques sur le consensus avec un nombre de participants connu. Les registres décentralisés et centralisés fournissent tous deux un service similaire, mais si les registres décentralisés avaient besoin d'un nouveau type de consensus pour exister, les registres centralisés auraient pu apparaître indépendamment de cette nouveauté. Du point de vue des systèmes répartis, les registres décentralisés constituent une percée technique, tandis que les registres centralisés sont le résultat d'optimisations successives.

Défis et contributions de la thèse

Le défi général que nous explorons dans cette thèse est la nécessité de renforcer la coordination dans un système décentralisé sans pour autant déléguer les décisions à un coordinateur central qui pourrait limiter l'extensibilité du système. Interprété dans le contexte de la réplication des données, ce défi se réduit à l'obtention des meilleures garanties de cohérence possibles tout en restant dans un modèle de cohérence faible imposé par le manque de centralité.

De plus, nous défendons l'argument selon lequel l'augmentation de l'hétérogénéité des appareils et de la taille des systèmes permet des systèmes offrant des qualités de service hétérogènes. Notre vision est que nous devrions pouvoir construire des systèmes décentralisés avec des garanties personnalisées pour chaque appareil.

Pour concrétiser cette vision, nous exploitons la tension qui existe entre la cohérence et la disponibilité des données dans la réplication des données à des fins de personnalisation. Nous devons également considérer que, à l'ère de l'informatique ubiquitaire, les appareils ont des capacités limitées et bénéficieraient d'une personnalisation limitant l'utilisation de leurs ressources telles que le calcul et la bande passante.

L'approche globale de notre travail consiste à développer de nouveaux mécanismes offrant des compromis personnalisables aux algorithmes et systèmes déjà bien connus. À cette fin,

nous proposons deux contributions. La première contribution porte sur la personnalisation de la cohérence et de la latence dans les protocoles de diffusion épidémique, alors que la seconde porte sur la personnalisation des exigences en matière de sécurité et de consommation de ressources dans les blockchains. Chaque contribution est détaillée ci-dessous.

Gossip Primary-Secondary

Notre première contribution consiste en une combinaison de deux nouveaux protocoles, baptisés *Gossip Primary-Secondary* (GPS, ou *Rumeur Primaire-Secondaire* en français) et *Update consistency Primary-Secondary* (UPS, ou *Cohérence d'écritures Primaire-Secondaire* en français), qui reposent sur une dissémination épidémique pour offrir des garanties différenciées de cohérence des données et de latence de livraison des écritures pour les participants d'un système.

Pour permettre la personnalisation, le réseau est divisé en deux sous-ensembles distincts aux performances opposées : un petit sous-ensemble de nœuds primaires (par exemple 1 %) observe une latence de messages plus faible, tandis que le plus grand sous-ensemble de nœuds secondaires observe un meilleur ordre de messages permettant une meilleure cohérence des données. Expliqué brièvement, le protocole GPS fait en sorte que les nœuds primaires reçoivent les messages d'abord, puis qu'ils dirigent la diffusion vers les nœuds secondaires. Le meilleur ordre des messages est obtenu en faisant en sorte que l'ensemble des nœuds principaux agisse tel un séquenceur souple de messages pour les nœuds secondaires : en recevant les messages d'abord et en les conservant brièvement en mémoire avant de les renvoyer, les nœuds principaux suppriment une partie du caractère aléatoire généré par la nature chaotique des protocoles épidémiques. Au lieu d'utiliser un coordinateur central, nous utilisons tout un sous-ensemble de nœuds en tant que coordinateur souple décentralisé et obtenons des gains de cohérence clairs pour les nœuds secondaires.

Dietcoin

Notre seconde contribution considère également un problème découlant de la capacité des systèmes hétérogènes de réplication de données, mais dans le contexte de Bitcoin. Le réseau Bitcoin est composé de deux types de nœuds : des nœuds complets qui vérifient la validité de la blockchain en entier lorsqu'ils intègrent le réseau, et des nœuds légers (également appelés nœuds SPV dans Bitcoin) qui vérifient uniquement la difficulté de chaque bloc de la chaîne lors de leur intégration au réseau et supposent qu'un bloc assez difficile est un bloc valide. Cette supposition est expliquée par le coût prohibitif d'avoir un nœud complet sur des appareils disposant de peu de ressources. Ce coût est représenté par le téléchargement et la vérification de plus de 150 Gio de données (courant 2018).

Pour remédier à la limitation inhérente des vérifications effectuées par les nœuds légers, nous proposons *Dietcoin* et ses *nœuds diet* qui étendent les capacités des nœuds légers de Bitcoin en leur permettant de vérifier la validité d'un bloc ou d'une sous-chaîne de blocs. Un nœud diet peut demander en toute sécurité des parties de l'ensemble des UTXO (l'état agrégé de la chaîne) pour effectuer ladite vérification tout en maintenant une consommation réduite en bande passante. Par exemple, un nœud diet désirant augmenter sa confiance dans le dernier bloc de la chaîne peut choisir de vérifier les ℓ derniers blocs pour s'assurer que le bloc qui l'intéresse est construit sur une sous-chaîne correcte et qu'il est donc plus susceptible de faire partie de la chaîne correcte de Bitcoin. Les mécanismes utilisés dans Dietcoin permettent également une intégration rapide des nœuds complets et des contrôles efficaces des doubles dépenses par les nœuds légers.

Perspectives

Repousser les limites du théorème CAP

Nous proposons dans GPS un modèle avec deux classes de nœuds, les nœuds primaires et secondaires, avec des compromis opposés en termes de cohérence et de latence des écritures. La personnalisation de la qualité de service pour chaque nœud pourrait être étendue encore plus en proposant une personnalisation par appareil au lieu d'avoir à classer les nœuds dans l'une des deux classes prédéfinies. Chaque nœud pourrait avoir son propre critère de cohérence personnalisé.

Bien que GPS offre différentes qualités de service aux différents nœuds dans des systèmes à grande échelle, chaque nœud bénéficie de la même qualité de service tout au long de son existence. Des compromis temporels pourraient être explorés dans de tels systèmes. Par exemple, une forte cohérence combinée à une latence faible pourrait être offerte dans de tels systèmes quand elle est possible, mais les garanties de latence seraient diminuées pendant les pics d'écritures pour limiter les incohérences.

Réduction des coûts de vérification et de stockage des blockchains

Dietcoin permet aux nœuds disposant de peu de ressources de vérifier la validité de sous-chaînes de blocs après avoir vérifié la difficulté de la chaîne. Des travaux récents sur des techniques cryptographiques de Preuves Non-Interactives de Preuve de Travail (Non-Interactive Proofs of Proof-of-Work, NIPoPoWs [136]) suggèrent que le processus linéaire de la vérification des difficultés de la chaîne peut être radicalement réduit à un processus au coût logarithmique. Il serait intéressant d'explorer comment NIPoPoW pourrait être combiné à Dietcoin afin de réduire considérablement et la vérification des difficultés de la chaîne et la vérification de sa validité.

Puisque Dietcoin dissocie le stockage de la vérification linéaire de la chaîne et qu'il fragmente l'ensemble des UTXO, nous pourrions déployer une Table de Hachage Répartie (Distributed Hash Table, DHT) pour stocker les blocs et les fragments UTXO, et mutualiser les besoins de stockage des nœuds complet et nœuds diet. De plus, une DHT permettrait aux nœuds diet de trouver plus facilement d'anciens fragments UTXO susceptibles de n'être stockés que par un petit nombre de nœuds.

Bien que Dietcoin, tel qu'il est dans cette thèse, repose sur le modèle UTXO pour la monnaie, il pourrait être adapté au modèle de compte utilisé par exemple dans Ethereum. Ethereum en particulier prévoit d'introduire la cohérence forte, appelée finalité dans leur terminologie, pour les blocs suffisamment anciens via la mise à jour Casper [137] en passant de la preuve de travail à une combinaison de preuve de travail et de preuve d'enjeu. Étant donné que les anciens blocs sont fortement cohérents, les nœuds ne doivent pas les vérifier conformément au modèle de sécurité d'Ethereum post-Casper. Par conséquent, un nœud diet dans un tel système pourrait choisir un bloc fortement cohérent comme bloc de confiance, nécessaire pour lancer la vérification d'une sous-chaîne, et obtenir exactement les mêmes garanties de sécurité qu'un nœud complet dans Ethereum post-Casper. Cela permettrait de combler complètement le fossé entre les nœuds légers et les nœuds complets.

Sur la coordination décentralisée à grande échelle

Comme le montre cette thèse, la conception de protocoles de coordination efficaces à grande échelle reste une tâche complexe. Malheureusement, l'utilisation de protocoles déterministes pour coordonner les nœuds dans les systèmes décentralisés à grande échelle est extrêmement coûteuse et seuls les protocoles probabilistes sont faisables pour les systèmes à adhésion ouverte.

Les chercheurs du domaine de l'informatique répartie ont exploré le consensus depuis des décennies, mais toujours sous le prisme des réseaux à adhésion fermée. Avec l'avènement du consensus de Nakamoto à adhésion ouverte, une question clé et pour l'instant restée sans réponse de l'informatique répartie se pose alors : Un consensus déterministe avec un nombre de nœuds non-limité et dynamique est-il possible ? Et si oui, sous quelles hypothèses de synchronie ?

Dans une optique similaire mais plus pratique, le protocole d'adhésion à base de Proof-of-Work est l'unique protocole d'adhésion ouvert à ce jour, l'existence d'autres protocoles d'adhésion ouverte résilients aux attaques Sybil reste une question ouverte. S'attaquer à cette question pourrait fournir des résultats utiles pouvant remplacer le protocole d'adhésion à base de Proof-of-Work et sa coopétition (i.e., compétition coopérative) trop coûteuse en énergie et en matériel. En outre, concevoir un protocole d'adhésion ouverte résilient aux attaques Sybil et à la fois juste et déterministe serait un premier pas pour résoudre notre première question.

APPENDIX B

LIST OF PUBLICATIONS

Peer-reviewed conferences

[138] **Speed for the elite, consistency for the masses: differentiating eventual consistency in large-scale distributed systems.**
Davide Frey, Achour Mostefaoui, Matthieu Perrin, Pierre-Louis Roman, François Taïani. The 35th IEEE Symposium on Reliable Distributed Systems (SRDS 2016), pp. 197-206 (10 pages), Sep 2016, Budapest, Hungary.

[139] **Similitude: Decentralised Adaptation in Large-Scale P2P Recommenders.**
Davide Frey, Anne-Marie Kermarrec, Christopher Maddock, Andreas Mauthe, Pierre-Louis Roman, François Taïani. The 15th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2015), pp. 51-65 (15 pages), Jun 2015, Grenoble, France.

Workshops

[140] **Bringing secure Bitcoin transactions to your smartphone.**
Davide Frey, Marc X. Makkes, Pierre-Louis Roman, François Taïani, Spyros Voulgaris. The 15th Workshop on Adaptive and Reflective Middleware (ARM 2016), pp. 3:1-3:6 (6 pages), Dec 2016, Trento, Italy.

Research reports

[141] **Dietcoin: shortcutting the Bitcoin verification process for your smartphone.**
Davide Frey, Marc X. Makkes, Pierre-Louis Roman, François Taïani, Spyros Voulgaris. Research report RR-9162, Mar 2018.

BIBLIOGRAPHY

[1] International Telecommunication Union, *ICT Data and Statistics 2007*, https://www.itu.int/ITU-D/ict/statistics/ict/, 2007.

[2] ——, *ICT Facts and Figures 2017*, https://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2017.pdf, Jul. 2017.

[3] Flurry Analytics, *U.S. Consumers Time-Spent on Mobile Crosses 5 Hours a Day*, http://flurrymobile.tumblr.com/post/157921590345/us-consumers-time-spent-on-mobile-crosses-5, Mar. 2017.

[4] J. Constine, *Facebook now has 2 billion monthly users... and responsibility*, https://techcrunch.com/2017/06/27/facebook-2-billion-users/, Jun. 2017.

[5] M. Osman, *28 Powerful Facebook Stats Your Brand Can't Ignore in 2018*, https://sproutsocial.com/insights/facebook-stats-for-marketers/#usagestats, Feb. 2018.

[6] Smart Insights, *What happens online in 60 seconds?*, https://www.smartinsights.com/internet-marketing-statistics/happens-online-60-seconds/, Feb. 2017.

[7] R. Swatman, *Pokémon Go catches five new world records*, http://www.guinnessworldrecords.com/news/2016/8/pokemon-go-catches-five-world-records-439327, Aug. 2016.

[8] M. Lynley, *With 500M downloads, Pokémon Go is coming to the Apple Watch*, https://techcrunch.com/2016/09/07/pokemon-go-the-hottest-game-on-the-planet-is-coming-to-the-apple-watch/, Sep. 2016.

[9] United Nations Human Rights Council, *UN Resolution L 20 The promotion, protection and enjoyment of human rights on the Internet*, http://www.icsft.net/6332/, Jun. 2016.

[10] P. Hernandez, *Pokémon Go's Launch Has Been Terrible*, https://kotaku.com/pokemon-gos-launch-has-been-terrible-1783336449, Aug. 2016.

[11] D. Thier, *Pokémon Go Servers Down As Game Launches In Canada*, https://www.forbes.com/sites/davidthier/2016/07/17/pokemon-go-servers-down-as-game-launches-in-canada/amp/, Jul. 2016.

[12] A. Frank, *Pokémon Go's server issues have been driving people wild all day*, https://www.polygon.com/2016/7/7/12123750/pokemon-go-server-issues-ios-android-fix, Jul. 2016.

[13] M. Rothenberg, *Cloud Bottlenecks: How Pokémon Go (and other game dev teams) caught them all*, https://arstechnica.com/information-technology/2017/01/what-pokemon-go-and-other-games-can-teach-developers-about-scaling-up-in-the-cloud/, Jan. 2017.

[14] A. S. Tanenbaum and M. v. Steen, *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006, ISBN: 0132392275.

[15] *Google Data Centers, Data center locations*, https://www.google.com/about/datacenters/inside/locations/index.html, accessed Sep. 10 2015.

[16] H. Lu, K. Veeraraghavan, P. Ajoux, J. Hunt, Y. J. Song, W. Tobagus, S. Kumar, and W. Lloyd, "Existential Consistency: Measuring and Understanding Consistency at Facebook", *in SOSP*, ACM, 2015.

[17] A.-M. Kermarrec and F. Taïani, "Want to scale in centralized systems? Think P2P", en, *Journal of Internet Services and Applications*, vol. 6, no. 1, p. 16, Aug. 2015, ISSN: 1869-0238. DOI: 10.1186/s13174-015-0029-1.

[18] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric Computing: Vision and Challenges", *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sep. 2015, ISSN: 0146-4833. DOI: 10.1145/2831347.2831354.

[19] L. Lamport, "The Part-time Parliament", *ACM Trans. Comput. Syst.*, vol. 16, no. 2, May 1998, ISSN: 0734-2071. DOI: 10.1145/279227.279229.

[20] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance and Proactive Recovery", *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002, ISSN: 0734-2071. DOI: 10.1145/571637.571640.

[21] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free Coordination for Internet-scale Systems", *in Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIX ATC'10, USENIX Association, 2010.

[22] J. Kreps, N. Narkhede, J. Rao, *et al.*, "Kafka: A distributed messaging system for log processing", *in Proceedings of the NetDB*, 2011.

[23] E. A. Brewer, "Towards robust distributed systems", *in PODC*, vol. 7, ACM, 2000.

[24] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", *ACM SIGACT News*, vol. 33, no. 2, 2002.

[25] A. Lakshman and P. Malik, "Cassandra: A Decentralized Structured Storage System", *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, Apr. 2010.

[26] R. Klophaus, “Riak Core: Building Distributed Applications Without Shared State”, *in ACM SIGPLAN Commercial Users of Functional Programming*, ser. CUFP ’10, ACM, 2010.

[27] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2008. [Online]. Available: http://www.cryptovest.co.uk/resources/Bitcoin%20paper%20Original.pdf.

[28] R. Ali, J. Barrdear, R. Clews, and J. Southgate, “Innovations in Payment Technologies and the Emergence of Digital Currencies”, Social Science Research Network, Rochester, NY, SSRN Scholarly Paper ID 2499397, Sep. 2014.

[29] M. Billy, *Dogecoin*, https://dogecoin.com.

[30] *Monero*, https://getmonero.org.

[31] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger”, *Ethereum Project Yellow Paper*, 2014. [Online]. Available: http://bitcoinaffiliatelist.com/wp-content/uploads/ethereum.pdf.

[32] L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem”, *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982, ISSN: 0164-0925. DOI: 10.1145/357172.357176.

[33] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains”, *in Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys ’18, New York, NY, USA: ACM, 2018, 30:1–30:15, ISBN: 978-1-4503-5584-1. DOI: 10.1145/3190508.3190538.

[34] R3, *Corda*, https://www.corda.net.

[35] L. Lamport, “How to make a multiprocessor computer that correctly executes multiprocess programs”, *IEEE ToC*, vol. 100, no. 9, 1979.

[36] M. P. Herlihy and J. M. Wing, “Linearizability: A Correctness Condition for Concurrent Objects”, *ACM Trans. Program. Lang. Syst.*, vol. 12, no. 3, pp. 463–492, Jul. 1990, ISSN: 0164-0925. DOI: 10.1145/78969.78972.

[37] M. Ahamad, G. Neiger, J. E. Burns, P. Kohli, and P. W. Hutto, “Causal memory: definitions, implementation, and programming”, *Distributed Computing*, vol. 9, no. 1, pp. 37–49, Mar. 1995, ISSN: 1432-0452. DOI: 10.1007/BF01784241.

[38] L. Lamport, “Time, Clocks, and the Ordering of Events in a Distributed System”, *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978, ISSN: 0001-0782. DOI: 10.1145/359545.359563.

[39] P. Bailis, A. Ghodsi, J. M. Hellerstein, and I. Stoica, “Bolt-on causal consistency”, *in ACM SIGMOD Int. Conf. on Man. of Data*, 2013.

[40] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, “Don’t settle for eventual: scalable causal consistency for wide-area storage with COPS”, *in SOSP*, ACM, 2011.

[41] S. Almeida, J. Leitão, and L. Rodrigues, “ChainReaction: a causal+ consistent datastore based on chain replication”, *in EuroSys*, ACM, 2013.

[42] M. Zawirski, N. Preguiça, S. Duarte, A. Bieniusa, V. Balegas, and M. Shapiro, “Write Fast, Read in the Past: Causal Consistency for Client-Side Applications”, *in Proceedings of the 16th Annual Middleware Conference*, ser. Middleware ’15, New York, NY, USA: ACM, 2015, pp. 75–87, ISBN: 978-1-4503-3618-5. DOI: 10.1145/2814576.2814733.

[43] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser, “Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System”, *in Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, ser. SOSP ’95, Copper Mountain, Colorado, USA: ACM, 1995, pp. 172–182, ISBN: 0-89791-715-4. DOI: 10.1145/224056.224070.

[44] W. Vogels, “Eventually Consistent”, *CACM*, vol. 52, no. 1, Jan. 2009, ISSN: 0001-0782. DOI: 10.1145/1435417.1435432.

[45] M. Perrin, A. Mostefaoui, and C. Jard, “Update Consistency for Wait-free Concurrent Objects”, *in International Parallel and Distributed Processing Symposium*, Hyderabad, India: IEEE, 2015.

[46] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, “Conflict-Free Replicated Data Types”, *in SSS*, 2011.

[47] S. Burckhardt, A. Gotsman, H. Yang, and M. Zawirski, “Replicated Data Types: Specification, Verification, Optimality”, *SIGPLAN Not.*, vol. 49, no. 1, 2014.

[48] G. Oster, P. Urso, P. Molli, and A. Imine, “Data Consistency for P2P Collaborative Editing”, *in Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, ser. CSCW ’06, New York, NY, USA: ACM, 2006, pp. 259–268, ISBN: 978-1-59593-249-5. DOI: 10.1145/1180875.1180916.

[49] S. Weiss, P. Urso, and P. Molli, “Logoot-Undo: Distributed Collaborative Editing System on P2P Networks”, *IEEE TPDS*, vol. 21, no. 8, 2010.

[50] R. Friedman, “Implementing hybrid consistency with high-level synchronization operations”, *Dist. Comp.*, vol. 9, no. 3, 1995.

[51] H. Attiya and R. Friedman, “Limitations of fast consistency conditions for distributed shared memories”, *Inf. Proc. Letters*, vol. 57, no. 5, 1996.

[52] P. Keleher, A. L. Cox, and W. Zwaenepoel, “Lazy Release Consistency for Software Distributed Shared Memory”, *in ISCA*, ACM, 1992.

[53] R. Friedman, M. Raynal, and F. Taïani, “Fisheye Consistency: Keeping Data in Synch in a Georeplicated World”, *in NETYS*, 2015.

[54] C. Xie, C. Su, M. Kapritsos, Y. Wang, N. Yaghmazadeh, L. Alvisi, and P. Mahajan, “Salt: Combining ACID and BASE in a distributed database”, *in OSDI*, USENIX, 2014.

[55] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh, “Consistency-based service level agreements for cloud storage”, *in SOSP*, ACM, 2013.

[56] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues, “Making Geo-Replicated Systems Fast as Possible, Consistent when Necessary”, *in OSDI*, USENIX, 2012.

[57] R. Guerraoui, M. Pavlovic, and D.-A. Seredinschi, “Incremental Consistency Guarantees for Replicated Objects”, *in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA: USENIX Association, 2016, pp. 169–184, ISBN: 978-1-931971-33-1.

[58] M. Raynal, *Concurrent Programming: Algorithms, Principles, and Foundations*, en. Berlin Heidelberg: Springer-Verlag, 2013, ISBN: 978-3-642-32026-2.

[59] C. Dwork, N. A. Lynch, and L. J. Stockmeyer, “Consensus in the presence of partial synchrony”, *J. ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[60] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, “Consensus with Byzantine Failures and Little System Synchrony”, *in 2006 International Conference on Dependable Systems and Networks (DSN 2006), 25-28 June 2006, Philadelphia, Pennsylvania, USA, Proceedings*, 2006, pp. 147–155. DOI: 10.1109/DSN.2006.22.

[61] D. Ongaro and J. Ousterhout, “In Search of an Understandable Consensus Algorithm”, *in 2014 USENIX Annual Technical Conference (USENIX ATC 14)*, Philadelphia, PA: USENIX Association, 2014, pp. 305–319, ISBN: 978-1-931971-10-2.

[62] T. Crain, V. Gramoli, M. Larrea, and M. Raynal, “(Leader/Randomization/Signature)-free Byzantine Consensus for Consortium Blockchains”, Feb. 2017, arXiv: 1702.03068. [Online]. Available: http://arxiv.org/abs/1702.03068.

[63] M. Raynal, *Fault-Tolerant Message-Passing Distributed Systems - An Algorithmic Approach*. Springer-Verlag, 2018, ISBN: 978-3-319-94140-0. DOI: 10.1007/978-3-319-94141-7.

[64] C. Dwork and M. Naor, “Pricing via Processing or Combatting Junk Mail”, en, *in Advances in Cryptology — CRYPTO’ 92*, ser. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, Aug. 1992, pp. 139–147, ISBN: 978-3-540-57340-1 978-3-540-48071-6. DOI: 10.1007/3-540-48071-4_10.

[65] A. Back, “Hashcash - A Denial of Service Counter-Measure”, Tech. Rep., 2002.

[66] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse, “Bitcoin-NG: A Scalable Blockchain Protocol”, *in 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016, ISBN: 978-1-931971-29-4.

[67] C. Decker, J. Seidel, and R. Wattenhofer, “Bitcoin Meets Strong Consistency”, *in Proceedings of the 17th International Conference on Distributed Computing and Networking*, ser. ICDCN ’16, New York, NY, USA: ACM, 2016, 13:1–13:10, ISBN: 978-1-4503-4032-8. DOI: 10.1145/2833312.2833321.

[68] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, “Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing”, *in 25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX: USENIX Association, 2016, pp. 279–296, ISBN: 978-1-931971-32-4.

[69] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, “OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding”, *in 2018 IEEE Symposium on Security and Privacy (SP)*, vol. 00, San Francisco, May 2018, pp. 19–34, ISBN: 978-1-5386-4353-2. DOI: 10.1109/SP.2018.000-5.

[70] M. Zamani, M. Movahedi, and M. Raykova, “RapidChain: Scaling Blockchain via Full Sharding”, *in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’18, Toronto, Canada: ACM, 2018, pp. 931–948, ISBN: 978-1-4503-5693-0. DOI: 10.1145/3243734.3243853.

[71] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, “Algorand: Scaling Byzantine Agreements for Cryptocurrencies”, *in Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP ’17, New York, NY, USA: ACM, 2017, pp. 51–68, ISBN: 978-1-4503-5085-3. DOI: 10.1145/3132747.3132757.

[72] A. Kiayias, A. Russell, B. David, and R. Oliynykov, “Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol”, en, *in Advances in Cryptology – CRYPTO 2017*, ser. Lecture Notes in Computer Science, Springer, Cham, Aug. 2017, pp. 357–388, ISBN: 978-3-319-63687-0 978-3-319-63688-7. DOI: 10.1007/978-3-319-63688-7_12.

[73] B. David, P. Gaži, A. Kiayias, and A. Russell, “Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain”, *in Advances in Cryptology – EUROCRYPT 2018*, J. B. Nielsen and V. Rijmen, Eds., Cham: Springer International Publishing, 2018, pp. 66–98, ISBN: 978-3-319-78375-8.

[74] E. Buchman, J. Kwon, and Z. Milosevic, *The latest gossip on BFT consensus*, 2018. eprint: arXiv:1807.04938.

[75] C. Cachin and M. Vukolić, “Blockchain Consensus Protocols in the Wild”, *arXiv:1707.01873 [cs]*, Jul. 2017, arXiv: 1707.01873. [Online]. Available: http://arxiv.org/abs/1707.01873.

[76] M. Vukolić, “The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication”, en, *in Open Problems in Network Security*, ser. Lecture Notes in Computer Science 9591, J. Camenisch and D. Kesdoğan, Eds., Springer International Publishing, Oct. 2015, pp. 112–125, ISBN: 978-3-319-39027-7 978-3-319-39028-4. DOI: 10.1007/978-3-319-39028-4_9.

[77] I. Abraham, D. Malkhi, and T. D. C. C. b. S. Schmid, “The Blockchain Consensus Layer and BFT”, en, *Bulletin of EATCS*, vol. 3, no. 123, Oct. 2017.

[78] M. Blum, P. Feldman, and S. Micali, “Non-interactive Zero-knowledge and Its Applications”, *in Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC ’88, Chicago, Illinois, USA: ACM, 1988, pp. 103–112, ISBN: 0-89791-264-0. DOI: 10.1145/62212.62222.

[79] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, “Bulletproofs: Short Proofs for Confidential Transactions and More”, *in 2018 IEEE Symposium on Security and Privacy (SP)*, vol. 00, May 2018, pp. 319–338. DOI: 10.1109/SP.2018.00020.

[80] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, *Scalable, transparent, and post-quantum secure computational integrity*, Cryptology ePrint Archive, Report 2018/046, https://eprint.iacr.org/2018/046, 2018.

[81] N. Narula, W. Vasquez, and M. Virza, “zkLedger: Privacy-Preserving Auditing for Distributed Ledgers”, *in 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, Renton, WA: USENIX Association, 2018, pp. 65–80, ISBN: 978-1-931971-43-0.

[82] J. A. Garay, A. Kiayias, and N. Leonardos, “The Bitcoin Backbone Protocol: Analysis and Applications”, *in EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 9057, Springer, 2015, pp. 281–310.

[83] I. Eyal and E. G. Sirer, “Majority Is Not Enough: Bitcoin Mining Is Vulnerable”, en, *in Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science 8437, N. Christin and R. Safavi-Naini, Eds., Springer Berlin Heidelberg, Mar. 2014, pp. 436–454, ISBN: 978-3-662-45471-8 978-3-662-45472-5. DOI: 10.1007/978-3-662-45472-5_28.

[84] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, “Optimal Selfish Mining Strategies in Bitcoin”, en, *in Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, Feb. 2016, pp. 515–532, ISBN: 978-3-662-54969-8 978-3-662-54970-4. DOI: 10.1007/978-3-662-54970-4_30.

[85] K. Nayak, S. Kumar, A. Miller, and E. Shi, “Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack”, *in 2016 IEEE European Symposium on Security and Privacy (EuroS P)*, Mar. 2016, pp. 305–320. DOI: 10.1109/EuroSP.2016.32.

[86] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, “Eclipse Attacks on Bitcoin’s Peer-to-Peer Network”, *in 24th USENIX Security Symposium (USENIX Security 15)*, Washington, D.C.: USENIX Association, 2015, pp. 129–144, ISBN: 978-1-931971-23-2.

[87] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, “On the Security and Performance of Proof of Work Blockchains”, *in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’16, New York, NY, USA: ACM, 2016, pp. 3–16, ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978341.

[88] G. O. Karame, E. Androulaki, and S. Capkun, “Double-spending Fast Payments in Bitcoin”, *in Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS ’12, Raleigh, North Carolina, USA: ACM, 2012, pp. 906–917, ISBN: 978-1-4503-1651-4. DOI: 10.1145/2382196.2382292.

[89] A. Gervais, S. Capkun, G. O. Karame, and D. Gruber, “On the Privacy Provisions of Bloom Filters in Lightweight Bitcoin Clients”, *in Proceedings of the 30th Annual Computer Security Applications Conference*, ser. ACSAC ’14, New York, NY, USA: ACM, 2014, pp. 326–335, ISBN: 978-1-4503-3005-3. DOI: 10.1145/2664243.2664267.

[90] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer, “On Scaling Decentralized Blockchains”, *in Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, Feb. 2016, pp. 106–125, ISBN: 978-3-662-53356-7 978-3-662-53357-4. DOI: 10.1007/978-3-662-53357-4_8.

[91] B. Bishop, “Review of Bitcoin Scaling Proposals”, *in Scaling Bitcoin Workshop Phase 1*, Sep. 2015. [Online]. Available: https://scalingbitcoin.org/montreal2015#workshop.

[92] S. Barber, X. Boyen, E. Shi, and E. Uzun, “Bitter to Better — How to Make Bitcoin a Better Currency”, en, *in Financial Cryptography and Data Security*, Springer, Berlin, Heidelberg, Feb. 2012, pp. 399–414. DOI: 10.1007/978-3-642-32946-3_29.

[93] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, “SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies”, *in 2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 104–121. DOI: 10.1109/SP.2015.14.

[94] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, “Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning”, *in 2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 526–545. DOI: 10.1109/SP.2016.38.

[95] E. Anceaume, T. Lajoie-Mazenc, R. Ludinard, and B. Sericola, “Safety analysis of Bitcoin improvement proposals”, *in 2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, Oct. 2016, pp. 318–325. DOI: 10.1109/NCA.2016.7778636.

[96] K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, N. Gailly, L. Gasser, I. Khoffi, J. Cappos, and B. Ford, “CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds”, *in 26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC: USENIX Association, 2017, pp. 1271–1287, ISBN: 978-1-931971-40-9.

[97] P. Otte, M. d. Vos, and J. Pouwelse, “TrustChain: A Sybil-resistant scalable blockchain”, *Future Generation Computer Systems*, 2017, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.08.048. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17318988.

[98] C. Decker and R. Wattenhofer, “A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels”, *in Stabilization, Safety, and Security of Distributed Systems*, A. Pelc and A. A. Schwarzmann, Eds., Cham: Springer International Publishing, 2015, pp. 3–18, ISBN: 978-3-319-21741-3.

[99] J. Lind, I. Eyal, F. Kelbert, O. Naor, P. Pietzuch, and E. G. Sirer, *Teechain: Scalable Blockchain Payments using Trusted Execution Environments*, 2017. eprint: arXiv:1707.05454.

[100] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, “A Secure Sharding Protocol For Open Blockchains”, *in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’16, New York, NY, USA: ACM, 2016, pp. 17–30, ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978389.

[101] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, “Epidemic Algorithms for Replicated Database Maintenance”, *in Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC ’87, New York, NY, USA: ACM, 1987, pp. 1–12, ISBN: 978-0-89791-239-6. DOI: 10.1145/41840.41841.

[102] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, “Bimodal Multicast”, *ACM Trans. Comput. Syst.*, vol. 17, no. 2, pp. 41–88, May 1999, ISSN: 0734-2071. DOI: 10.1145/312203.312207.

[103] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec, “Lightweight Probabilistic Broadcast”, *ACM Trans. Comput. Syst.*, vol. 21, no. 4, pp. 341–374, Nov. 2003, ISSN: 0734-2071. DOI: 10.1145/945506.945507.

[104] A.-M. Kermarrec, L. Massoulie, and A. Ganesh, “Probabilistic reliable dissemination in large-scale systems”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 3, pp. 248–258, Mar. 2003, ISSN: 1045-9219. DOI: 10.1109/TPDS.2003.1189583.

[105] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, “Gossip-based Peer Sampling”, *ACM Trans. Comput. Syst.*, vol. 25, no. 3, Aug. 2007, ISSN: 0734-2071. DOI: 10.1145/1275517.1275520.

[106] S. Voulgaris, D. Gavidia, and M. v. Steen, “CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays”, en, *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–217, Jun. 2005, ISSN: 1064-7570, 1573-7705. DOI: 10.1007/s10922-005-4441-x.

[107] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, “Scamp: Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication”, en, *in Networked Group Communication*, ser. Lecture Notes in Computer Science 2233, J. Crowcroft and M. Hofmann, Eds., Springer Berlin Heidelberg, Nov. 2001, pp. 44–55, ISBN: 978-3-540-42824-4 978-3-540-45546-2.

[108] P. Felber, A.-M. Kermarrec, L. Leonini, E. Rivière, and S. Voulgaris, “Pulp: An adaptive gossip-based dissemination protocol for multi-source message streams”, en, *Peer-to-Peer Networking and Applications*, vol. 5, no. 1, pp. 74–91, Mar. 2012, ISSN: 1936-6442, 1936-6450. DOI: 10.1007/s12083-011-0110-x.

[109] M. Jelasity, A. Montresor, and O. Babaoglu, “Gossip-based Aggregation in Large Dynamic Networks”, *ACM Trans. Comput. Syst.*, vol. 23, no. 3, pp. 219–252, Aug. 2005, ISSN: 0734-2071. DOI: 10.1145/1082469.1082470.

[110] R. Ormándi, I. Hegedus, and M. Jelasity, “Asynchronous Peer-to-peer Data Mining with Stochastic Gradient Descent”, *in Proceedings of the 17th International Conference on Parallel Processing - Volume Part I*, ser. Euro-Par’11, Bordeaux, France: Springer-Verlag, 2011, pp. 528–540, ISBN: 978-3-642-23399-9.

[111] G. Danner and M. Jelasity, “Fully Distributed Privacy Preserving Mini-batch Gradient Descent Learning”, en, *in Distributed Applications and Interoperable Systems*, ser. Lecture Notes in Computer Science 9038, A. Bessani and S. Bouchenak, Eds., Springer International Publishing, Jun. 2015, pp. 30–44, ISBN: 978-3-319-19128-7 978-3-319-19129-4.

[112] M. Jelasity, A. Montresor, and O. Babaoglu, “T-Man: Gossip-based Fast Overlay Topology Construction”, *Comput. Netw.*, vol. 53, no. 13, pp. 2321–2339, Aug. 2009, ISSN: 1389-1286. DOI: 10.1016/j.comnet.2009.03.013.

[113] S. Voulgaris and M. v. Steen, “VICINITY: A Pinch of Randomness Brings out the Structure”, en, *in Middleware 2013*, ser. Lecture Notes in Computer Science 8275, D. Eyers and K. Schwan, Eds., Springer Berlin Heidelberg, Jan. 2013, pp. 21–40, ISBN: 978-3-642-45064-8 978-3-642-45065-5.

[114] R. Guerraoui, R. R. Levy, B. Pochon, and V. Quéma, “Throughput Optimal Total Order Broadcast for Cluster Environments”, *ACM Trans. Comput. Syst.*, vol. 28, no. 2, 5:1–5:32, Jul. 2010, ISSN: 0734-2071. DOI: 10.1145/1813654.1813656.

[115] M. Matos, H. Mercier, P. Felber, R. Oliveira, and J. Pereira, “EpTO: An Epidemic Total Order Algorithm for Large-Scale Distributed Systems”, *in Middleware*, ACM, 2015, ISBN: 978-1-4503-3618-5.

[116] R. Baldoni, R. Guerraoui, R. R. Levy, V. Quéma, and S. T. Piergiovanni, “Unconscious Eventual Consistency with Gossips”, *in SSS*, 2006.

[117] A. Sousa, J. Pereira, F. Moura, and R. Oliveira, “Optimistic total order in wide area networks”, *in SRDS*, IEEE, 2002.

[118] M.-J. Lin and K. Marzullo, “Directional Gossip: Gossip in a Wide Area Network”, *in EDCC*, 1999, ISBN: 978-3-540-66483-3 978-3-540-48254-3.

[119] N. Carvalho, J. Pereira, R. Oliveira, and L. Rodrigues, “Emergent Structure in Unstructured Epidemic Multicast”, *in DSN*, 2007. DOI: 10.1109/DSN.2007.40.

[120] K. Hopkinson, K. Jenkins, K. Birman, J. Thorp, G. Toussaint, and M. Parashar, “Adaptive Gravitational Gossip: A Gossip-Based Communication Protocol with User-Selectable Rates”, *IEEE TPDS*, vol. 20, no. 12, 2009.

[121] I. Gupta, A.-M. Kermarrec, and A. Ganesh, “Efficient and adaptive epidemic-style protocols for reliable and scalable multicast”, *IEEE TPDS*, vol. 17, no. 7, 2006.

[122] D. Frey, R. Guerraoui, A.-M. Kermarrec, B. Koldehofe, M. Mogensen, M. Monod, and V. Quéma, “Heterogeneous Gossip”, *in Middleware*, ACM, 2009.

[123] W. Golab, X. Li, and M. A. Shah, “Analyzing consistency properties for fun and profit”, *in PODC*, ACM, 2011.

[124] W. Golab, M. R. Rahman, A. AuYoung, K. Keeton, and I. Gupta, “Client-centric benchmarking of eventual consistency for cloud storage systems”, *in ICDCS*, IEEE, 2014.

[125] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs, and B. Rinaldi, “YCSB++: benchmarking and performance debugging advanced features in scalable table stores”, *in Symp. on Cloud Comp. (SoCC)*, ACM, 2011.

[126] K. Zellag and B. Kemme, “How consistent is your cloud application?”, *in Symp. on Cloud Comp. (SoCC)*, ACM, 2012.

[127] F. Taïani, S. Lin, and G. S. Blair, “GossipKit: A Unified ComponentFramework for Gossip”, *IEEE TSE*, vol. 40, no. 2,

[128] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié, “From epidemics to distributed computing”, *IEEE computer*, vol. 37, no. LPD-ARTICLE-2006-004, pp. 60–67, 2004.

[129] A. Montresor and M. Jelasity, “PeerSim: A scalable P2P simulator”, *in P2P*, IEEE, 2009. DOI: 10.1109/P2P.2009.5284506.

[130] M. R. Rahman, W. Golab, A. AuYoung, K. Keeton, and J. J. Wylie, “Toward a Principled Framework for Benchmarking Consistency”, *in HotDep*, USENIX, 2012.

[131] L. Lamport, “On interprocess communication”, *Distributed Computing*, vol. 1, no. 2, 1986, ISSN: 0178-2770.

[132] *Bitcoin Developer Guide*. [Online]. Available: https://bitcoin.org/en/developer-guide#verifying-payment.

[133] B. Bryan, *[bitcoin-dev] Protocol-Level Pruning*, Nov. 2017. [Online]. Available: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2017-November/015297.html.

[134] A. Miller, *Storing UTXOs in a Balanced Merkle Tree (zero-trust nodes with O(1)-storage)*, https://bitcointalk.org/index.php?topic=101734.0, Aug. 2012.

[135] D. Leung, A. Suhl, Y. Gilad, and N. Zeldovich, “Vault: Fast Bootstrapping for Cryptocurrencies”, Tech. Rep. 269, 2018. [Online]. Available: https://eprint.iacr.org/2018/269.

[136] A. Kiayias, A. Miller, and D. Zindros, *Non-Interactive Proofs of Proof-of-Work*, Cryptology ePrint Archive, Report 2017/963, https://eprint.iacr.org/2017/963, 2017.

[137] D. Ryan and C.-C. Liang, *EIP 1011: Hybrid Casper FFG*, Apr. 2018. [Online]. Available: https://eips.ethereum.org/EIPS/eip-1011.

[138] D. Frey, A. Mostéfaoui, M. Perrin, P.-L. Roman, and F. Taïani, "Speed for the Elite, Consistency for the Masses: Differentiating Eventual Consistency in Large-Scale Distributed Systems", *in 35th IEEE Symposium on Reliable Distributed Systems, SRDS 2016, Budapest, Hungary, September 26-29, 2016*, Sep. 2016, pp. 197–206. DOI: 10.1109/SRDS.2016.032.

[139] D. Frey, A.-M. Kermarrec, C. Maddock, A. Mauthe, P.-L. Roman, and F. Taïani, "Similitude: Decentralised Adaptation in Large-Scale P2P Recommenders", *in Distributed Applications and Interoperable Systems - 15th IFIP WG 6.1 International Conference, DAIS 2015, Held as Part of the 10th International Federated Conference on Distributed Computing Techniques, DisCoTec 2015, Grenoble, France, June 2-4, 2015, Proceedings*, Jun. 2015, pp. 51–65. DOI: 10.1007/978-3-319-19129-4_5.

[140] D. Frey, M. X. Makkes, P.-L. Roman, F. Taïani, and S. Voulgaris, "Bringing secure Bitcoin transactions to your smartphone", *in Proceedings of the 15th International Workshop on Adaptive and Reflective Middleware, ARM@Middleware 2016, Trento, Italy, December 12-16, 2016*, Dec. 2016, 3:1–3:6. DOI: 10.1145/3008167.3008170.

[141] ——, "Dietcoin: shortcutting the Bitcoin verification process for your smartphone", Mar. 2018. [Online]. Available: http://arxiv.org/abs/1803.10494.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Explorer l'hétérogénéité dans la réplication de données décentralisées faiblement cohérentes

Mot clés : Système décentralisé, Cohérence à terme, Protocole épidémique, Blockchain

Resumé : Les systèmes décentralisés sont par nature extensibles mais sont également difficiles à coordonner en raison de leur faible couplage. La réplication de données dans ces systèmes géo-répartis est donc un défi inhérent à leur structure. Les deux contributions de cette thèse exploitent l'hétérogénéité des besoins des utilisateurs et permettent une qualité de service personnalisable pour la réplication de données dans les systèmes décentralisés. Notre première contribution *Gossip Primary-Secondary* étend le critère de cohérence *Update consistency Primary-Secondary* afin d'offrir des garanties différenciées de cohérence et de latence de messages pour la réplication de données à grande échelle. Notre seconde contribution *Dietcoin* enrichit Bitcoin avec des *nœuds diet* qui peuvent (i) vérifier la validité de sous-chaînes de blocs en évitant le coût exorbitant de la vérification initiale et (ii) choisir leurs propres garanties de sécurité et de consommation de ressources.

Title: Exploring heterogeneity in loosely consistent decentralized data replication

Keywords: Decentralized system, Eventual consistency, Epidemic protocol, Blockchain

Abstract: Decentralized systems are scalable by design but also difficult to coordinate due to their weak coupling. Replicating data in these geo-distributed systems is therefore a challenge inherent to their structure. The two contributions of this thesis exploit the heterogeneity of user requirements and enable personalizable quality of services for data replication in decentralized systems. Our first contribution *Gossip Primary-Secondary* enables the consistency criterion *Update consistency Primary-Secondary* to offer differentiated guarantees in terms of consistency and message delivery latency for large-scale data replication. Our second contribution *Dietcoin* enriches Bitcoin with *diet nodes* that can (i) verify the correctness of entire subchains of blocks while avoiding the exorbitant cost of bootstrap verification and (ii) personalize their own security and resource consumption guarantees.