Understanding the energy consumption of blockchain technologies : a focus on smart contracts

Dimitri Saingre

DOCTORAT BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THÈSE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPÉRIEURE
MINES-TÉLÉCOM ATLANTIQUE BRETAGNE
PAYS-DE-LA-LOIRE - IMT ATLANTIQUE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Dimitri SAINGRE

Understanding the energy consumption of blockchain technologies: a focus on smart contracts

Thèse présentée et soutenue à Nantes, le 13 Décembre 2021
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes (LS2N)
Thèse N° : 2021IMTA0280

Rapporteurs avant soutenance :

Maria POTOP-BUTUCARU	Professeur (HDR), Sorbonne Université
Romain ROUVOY	Professeur (HDR), Université de Lille

Composition du Jury :

Président :	Jean-Marc PIERSON	Professeur (HDR), Université de Toulouse
Examinateurs :	Maria POTOP-BUTUCARU	Professeur (HDR), Sorbonne Université
	Romain ROUVOY	Professeur (HDR), Université de Lille
	Anthony SIMONET-BOULOGNE	Responsable des projets Scientifique, IExec
Dir. de thèse :	Jean-Marc MENAUD	Professeur (HDR), IMT Atlantique
Co-dir. de thèse :	Thomas LEDOUX	Professeur (HDR), IMT Atlantique

Remerciements

Dans cette section, je tiens à remercier les personnes m'ayant accompagnés durant ces trois années de doctorat.

Je remercie tout d'abord M. Jean-Marc Menaud et M. Thomas Ledoux pour m'avoir donné l'opportunité d'effectuer ce doctorat. Ces trois années, riches en émotions, m'ont beaucoup apporté autant sur le plan professionnel que personnel. Leurs conseils et remarques avisés m'ont permit de développer mon esprit scientifique et de prendre du recul sur le sujet abordé dans ce manuscrit.

Je remercie aussi Mme Maria Potop-Butucaru, M. Romain Rouvoy, M. Anthony Simonet-Boulogne et M. Jean-Marc Pierson d'avoir accepté d'intégrer mon jury de thèse. Je suis reconnaissant pour toutes les questions et remarques relatives aux manuscrit et à la soutenance qui ont conduits à des échanges riches.

J'adresse une pensée aux membres de l'équipe STACK et plus globalement à toutes les personnes que j'ai pu côtoyer à l'IMT Atlantique durant ces trois années. J'ai notamment eu beaucoup de plaisir à apprendre à connaître les personnes que j'avais d'abord rencontré comme enseignant durant mes études d'ingénieur.

Enfin, je remercie chaleureusement ma famille et mes amis, qui m'ont soutenus pendant cette période. Merci Papa, Maman et Jérémie.

Merci enfin à toi Lucie, pour tout le soutiens et l'écoute. Tu es maintenant aussi experte que moi sur le sujet à force de m'entendre en parler ! Je t'aime de tout mon coeur.

Merci à tous et à toutes.

Abstract

Blockchains are distributed ledgers that store information in a peer-to-peer network. This technology was introduced in 2008 with *Bitcoin*. Since the launch of the Bitcoin network, the blockchain ecosystem has grown both in terms of actors and complexity. Many blockchain projects have been created in the last decade, each with its own technical features. Blockchains are now used in several domains like cloud computing, energy grids, or supply chain.

With the rise of the debates on climate change management, the rapid growth of the information and communication technologies (ICT) sector raises many ethical questions. Indeed, in 2020 the European Commission estimated an increase in energy consumption of data centers in Europe of 42% between 2010 and 2018. In the meanwhile, blockchains have been highly criticised for their energy consumption. Indeed, many blockchains (including Bitcoin) use a highly computational intensive consensus algorithm called *proof-of-work*. The high energy cost of proof-of-work has been well studied and established.

In this thesis, we propose to study another aspect of blockchains: smart contracts. Smart contracts are scripts deployed and executed on a blockchain. They enable the development of decentralised applications. Based on the analysis of transaction history and experiments on real hardware, we contribute to understanding the energy cost of smart contracts on the Ethereum blockchain. First, we propose a novel framework to deploy and analyse the performances and footprint of blockchains. Many existing tools lacked crucial features for blockchains performances (e.g., deployment, network emulation...). Based on this framework, we develop a model to estimate the energy consumption of Ethereum smart contract. We combined this model with a one-year Ethereum transaction history to give rough estimations of the energy consumption of blockchain-based applications. Finally, we proposed a novel protocol to identify and delete unused Ethereum smart contract. Our studies highlighted that most deployed contracts lead to little or no use. As contracts are meant to be available forever, we have shown that unused contracts impacted the performances of each Ethereum node. Our solution challenges the "immortality" aspect of smart contracts to provide "lighter" blockchains.

Résumé

Les *chaînes de blocs* sont des registres distribués permettant le stockage d'informations dans un réseau pairs à pairs. Chaque membre de ce réseau peut posséder une copie de ce registre. La modification de ce registre peut se faire par le biais de *transactions* diffusées sur le réseau. Ces transactions sont alors traitées sous forme de lots appelés *blocs*, par des membres appelés *mineurs* ou *validateurs*. Un lien entre chaque bloc validé et son précédent, garantis l'intégrité de l'historique des transactions validées. Par ce lien, une "chaîne" est formée : d'où le nom *chaîne de blocs*. Les chaînes de blocs sont souvent considérées comme fondations pour le développement d'applications dites *décentralisées*.

Depuis le lancement du réseau *Bitcoin*, de nombreuses activités de recherche et de développement ont mené à la création d'un écosystème complexe. Les chaînes de blocs sont maintenant étudiées et utilisées dans des domaines métiers variés, tels que la finance, l'informatique en nuage, les grilles énergétiques, les chaînes logistiques... Dans chacun de ces secteurs, la mise en place de *registres distribués* par le biais des chaînes de blocs vise à améliorer la collaboration entre différents acteurs (potentiellement malicieux), l'automatisation de processus (par le biais de *contrats intelligents*), la sécurisation des données et la transparence. Le développement des cryptomonnaies (et plus largement de la *finance décentralisée*) illustre bien ce potentiel en rendant possible le transfert de valeur financière à travers le monde sans le biais de tiers de confiance tels que les banques. Bien plus que de simples prototypes, les applications basées sur les chaînes de blocs semblent aujourd'hui occuper une réelle place dans notre économie. À titre d'exemple, les cryptomonnaies capitalisent aujourd'hui plus de deux billions d'euros (source : `https://coinmarketcap.com`).

Avec le développement des débats sur la gestion du changement climatique, la croissance rapide du secteur des *technologies de l'information et de la communication* (TIC) soulève de nombreuses questions éthiques. En effet, en 2020 la commission Européenne estimait une augmentation de la consommation énergétique des centres de données en Europe de 42% entre 2010 et 2018.

Les technologies de chaînes de blocs sont loin de faire exception face à l'inquiétude liée au coût environnemental des TICs. En effet, les chaînes de blocs telles que Bitcoin

requièrent l'utilisation de coûteuses ressources de calculs dans le cadre de leur algorithme de consensus : la *preuve de travail*. La preuve de travail repose sur une compétition, appelée *minage*, entre les différents membres du réseau, dans le but de pouvoir proposer de nouveaux blocs. En 2014, O'Dwyer et Malone [78] ont estimé que la consommation énergétique liée au minage de Bitcoin était comparable à celle de l'Irelande. Depuis, de nombreuses études confirment ont confirmé l'important coût énergétique du minage.

Dans le but de proposer de nouvelles chaînes de blocs moins coûteuses et plus performantes, de nouveaux algorithmes de consensus (tels que la *preuve d'enjeux*) ont émergés. De nombreuses chaînes de bloc, proposant ces algorithmes de consensus sans minage, montrent qu'une partie de cette communauté cherche à s'émanciper de la preuve de travail. A titre d'exemple la seconde version d'Ethereum, une des chaîne de blocs publique les plus utilisé, embarquera un algorithme de consensus basé sur une preuve d'enjeux.

Il ne faudrait cependant pas réduire la consommation énergétique des chaînes de blocs à celle de leur algorithme de consensus. En effet, l'émergence des *contrats intelligents*, en proposant le déploiement et l'exécution de programmes complets sur les chaînes de blocs, vient complexifier ces technologies. Si ces contrats intelligents rendent possible de nombreux nouveaux cas d'usage, ils viennent néanmoins alourdir l'empreinte énergétique des chaînes de blocs.

Au fur et à mesure que la consommation de la phase de consensus des chaînes de blocs se réduit, l'empreinte énergétique des contrats intelligent devient proportionnellement plus importante. Nous estimons alors dans cette thèse qu'il devient nécessaire de s'attarder à l'étude de ces aspects.

Contributions

Cette thèse vise à contribuer à l'étude et à la réduction du coût des contrats intelligent sur les plateformes de chaînes de blocs. En effet, cette problématique est souvent délaissée des études portant sur l'empreinte énergétique et environnementale des chaînes de blocs. Pour cela, nous avons concentré nos travaux sur la plateforme de chaîne de blocs publique *Ethereum*, une des principales plateformes proposant un système de contrat intelligent. Cette thèse s'articule autour de trois questions centrales :

Les contrats intelligents occupent-ils une place réellement importante dans les réseaux de chaînes de blocs ? Même s'il est clair que les activités académiques et industrielles autour des contrats intelligents semblent montrer une forte adoption dans

des domaines variés, peu de travaux académiques étudient leur utilisation réelle.

Peut-on estimer et modéliser le coût énergétique de ces contrats ? Mieux comprendre le coût énergétique engendré par l'exécution de ces contrats semble être nécessaire à mesure que certaines plateformes de chaîne de blocs s'émancipent peu à peu du *minage* et de la *preuve de travail*. Le coût de l'exécution des applicatifs logiciels devients alors, proportionnellement, plus important.

Quel est le coût engendré par l'immutabilité des contrats déployés, principe au cœur des technologies de chaînes de blocs ? Les plateformes de chaîne de blocs publiques permettent à chacun de déployer de nouveaux contrats intelligents. Une fois déployés, ceux-ci sont disponibles *ad vitam æternam*. Il se pose alors la question du coût de cette propriété. Au-delà d'un espace de stockage requis grandissant pour les membres du réseau, la mise à disposition, pour toujours, de tous les contrats déployés impacte-t-il le réseau de chaîne de bloc ?

Par le biais d'analyse d'historiques de transactions stockées sur Ethereum et d'expérimentations *in situ* sur la plateforme de recherche Grid'5000, nous proposons de répondre à ces questions par le biais de trois contributions :

— **Un nouveau cadriciel pour l'évaluation comparative des technologies blockchain : BCTMark**. Ce cadriciel répond aux défis du déploiement et du test des blockchains dans un contexte expérimental. Construit à partir de composants *open-source*, BCTMark aborde les phases de gestion des ressources de déploiment, de montée en charge du système évalué et de sauvegarde des résultats. Ce nouveau cadriciel permet alors une gestion complète du cycle de vie des experiences portant sur les performances des chaîne de bloc, en intégrant des capacités d'émulation du réseau (latence, perte de paquets...). Les capacités de BCTMark ont été illustrées par le biais de déploiements effectués sur deux plateformes de recherche (Grid5000 et un cluster local de Raspberry Pi) et sur trois chaînes de blocs (Ethereum Clique, Ethereum Ethash, Hyperledger Fabric). Ces travaux ont été présentés dans les conférences *Compas* et *AICCSA* et dans le journal *Cluster Computing*.

— **Une analyse de l'usage et de la consommation énergétique des contrats intelligent Ethereum**. Sur la base d'une année de transactions Ethereum, nous analysons l'importance des contrats intelligents dans son trafic. Plus précisément, nous donnons un aperçu du nombre d'appels de contrats intelligents, du déploiement de nouveaux contrats et de la complexité de ces contrats. Nous proposons ensuite un modèle permettant d'estimer la consommation énergétique des

contrats Ethereum en fonction de leur consommation de *gaz*. Ce modèle, combiné avec l'historique de transactions Ethereum, nous permet de donner une estimation du coût énergétique engendré par l'execution de contrats intelligent sur un ans. Une partie de ce travail a été présentée dans le journal *Cluster Computing*.

— **Un nouveau protocole qui peut être utilisé pour réduire le nombre de contrats inutilisés dans Ethereum**. Comme nous le montrons dans ce manuscrit, une grande partie des contrats Ethereum déployés ne seront que peu ou pas utilisés. Nous démontrons ici que le nombre croissant de contrats a un impact sur les performances de chaque nœud du réseau en augmentant le temps de traitement de chaque nouveau bloc reçus. Comme solution potentielle à ce problème, nous proposons un protocole pour inciter les mineurs à identifier et détruire les contrats inutilisés. Ce protocole vient ajouter une durée de vie pour chaque contrat. Cette durée de vie, augmentée à chaque nouvelle interaction avec le contrat, permet de supprimer les contrats qui cesseraient d'être utilisé. Sur la base d'historique de transactions, nous montrons que notre proposition pourrait conduire à une réduction significative du nombre de contrats stockés dans la blockchain. Ce travail a été présenté à la conférence *ISCC* et a été récompensé par le prix du "meilleur article étudiant".

Table of Contents

1 Introduction **17**

1.1 Motivations 17

1.2 Contributions 18

1.3 Publications 20

1.4 Thesis overview 20

2 Background **23**

2.1 Questioning the environmental impact of Blockchain technologies 23

2.1.1 The energy consumption of ICT 23

2.1.2 The development of blockchain technologies 25

2.1.3 Why study the energy footprint of blockchains? 29

2.2 Blockchains 29

2.2.1 An overview on blockchain technologies 30

2.2.2 A note on Ethereum's data storage 33

2.2.3 Chain fork choice 34

2.2.4 Differences between public and private Blockchains 35

2.2.5 Blockchain consensus algorithms 35

2.2.6 Decentralised applications with Smart contracts 38

2.3 Conclusion 44

3 State of the art **45**

3.1 A literature review on the energy consumption of blockchain technologies . 45

3.1.1 The energy footprint of Proof-of-Work mining 46

3.1.2 Alternatives to mining 51

3.2 Evaluate the performances of blockchain systems 52

3.3 Analysing the uses and energy consumption of smart contracts 53

3.4 Data suppression on blockchains 55

3.5 Conclusion 55

4 A new framework for benchmarking blockchain technologies 57
4.1 Introduction . . . 58
4.2 Contribution to state of the art . . . 60
4.3 BCTMark - Technical architecture & usage . . . 61
4.3.1 Usage . . . 62
4.3.2 Architecture . . . 64
4.4 Validation experiments . . . 66
4.4.1 Deployment of blockchains on two different testbeds . . . 67
4.4.2 Comparison of CPU usage of three blockchain systems . . . 68
4.4.3 Experiments Reproducibility . . . 70
4.4.4 Performance analysis of Smart contracts . . . 71
4.5 Conclusion . . . 73

5 Understanding the usage and energy consumption of Ethereum smart contracts 75
5.1 Introduction . . . 76
5.2 Understanding the current usage of Ethereum smart contracts . . . 77
5.2.1 Data extraction protocol . . . 78
5.2.2 Smart contracts calls in Ethereum traffic . . . 79
5.2.3 Gas consumption of Ethereum smart contracts . . . 80
5.2.4 New smart contract deployment . . . 83
5.2.5 Quantifying the number of unused smart contracts . . . 84
5.3 Measuring and modeling the energy consumption of Ethereum smart contracts . . . 87
5.3.1 Smart contracts footprint on non-Proof-of-Work systems . . . 87
5.3.2 Deriving energy consumption from gas consumption . . . 88
5.3.3 Ethereum smart contract execution model . . . 91
5.3.4 The impact of replication on smart contracts execution cost . . . 93
5.3.5 Limitations . . . 94
5.4 Conclusion . . . 95

6 Reducing the amount of unused smart contracts in Ethereum 97
6.1 Introduction . . . 98
6.2 The impact of unused smart contract on Ethereum . . . 99
6.2.1 Evaluation protocol . . . 99

6.2.2 The impact of unused smart contract on contract calls processing time . . . 100
6.2.3 Current Ethereum state size . . . 102
6.3 A Time to live protocol for smart contracts . . . 102
6.3.1 Overview . . . 102
6.3.2 Details . . . 103
6.3.3 Contract destruction and data retrieval . . . 105
6.3.4 Discussion on determining parameters value . . . 105
6.3.5 Discussions . . . 106
6.3.6 Protocol impact depending on TTL duration . . . 106
6.4 Conclusion . . . 107

7 Conclusion **109**
7.1 Achievements . . . 109
7.2 Perspectives . . . 111
7.2.1 Impact of layer-two solutions on blockchains footprint . . . 111
7.2.2 Modelling the energy consumption of smart contracts . . . 112
7.2.3 Designing blockchains with temporary data . . . 113

Bibliography **115**

List of Figures

2.1 Energy consumed by European Data Centers (data source: [71]) 24
2.2 The carbon footprint of the ICT sector in 2019 - breakdown between production and usage (data source: [97]) 25
2.3 Hierarchy between databases, distributed ledger technologies, and blockchains 27
2.4 Evolution the number of daily transactions on Ethereum (source: `etherscan.io`) 28
2.5 A schematic view of a classical blockchain "data-structure" 30
2.6 The use of a Merkel tree to store transactions in a block 32
2.7 Ethereum's data storage 34
2.8 Chain split management 34
2.9 How PoW works 36
2.10 Evolution of Ethereum mining difficulty 37
2.11 An Ethereum contract's lifecycle 39

3.1 Evolution of Bitcoin difficulty through time (data source: `https://www.blockchain.com`) 47
3.2 Estimation of the energy consumption per transactions, by Tasca et al. [105] 52

4.1 The experiment workflow of BCTMark 62
4.2 BCTMark architecture 64
4.3 Comparison of Power Usage for different loads 68
4.4 Comparison of CPU Usage for different loads 69
4.5 Evolution of Ethash CPU usage for 200 Txs 70
4.6 Cost in gas of three smart contracts depending on provided input 72

5.1 Calls to smart contracts over time 79
5.2 Calls to smart contracts over time, by types 80
5.3 Distribution gas usage per transactions (logarithmic scale) 81
5.4 Distribution gas usage per smart contract calls (logarithmic scale) 82
5.5 Evolution of gas price and number of processed transactions during the year 83

5.6 Number of contracts created over time 84
5.7 Distribution of the number of calls per smart contracts - Lowest 90 percentiles 85
5.8 Distribution of the number of calls per smart contracts - Highest 10 percentiles - Logarithmic scale 86
5.9 Evolution of the percentage of active contracts depending on the number of days since their creation (Contracts deployed that year only) 86
5.10 Comparison power usage with and without contract execution on Ethereum PoW 88
5.11 Comparison power usage with and without contract execution on Ethereum PoS 89
5.12 Quicksort - Gas consumption and average power depending on array size input 90
5.13 Power depending on gas consumption 91
5.14 Impact of smart contract calls replication on Ethereum network energy usage 92
5.15 Evolution of the number of Ethereum nodes listed by `ethernodes.org`, from October 2020 to June 2021 (source: `ethernodes.org`) 93
5.16 Global energy consumed by a smart contract call depending on the number of nodes in the network 95

6.1 Evolution of the number of contracts depending on TE value 107

List of Tables

2.1 Examples of fields included in an Ethereum block 31
2.2 Comparison between different blockchain smart contract platforms (sources: [49], [121], [94]) 40
2.3 Samples of Gas cost definition from Ethereum Yellow Paper[44] 43

3.1 Estimation of Bitcoin network energy consumption in literature 47
3.2 Differences in energy efficiency between different mining hardware (Data source: `https://en.bitcoin.it/wiki/Mining_hardware_comparison`) . . 49

4.1 Comparison of functionalities with the state of the art 61
4.2 Reproducibility across six runs 71

5.1 Proportion of ERC20 and ERC721 among active contracts 80
5.2 Estimated power usage for Ethereum processed smart contracts and transaction over the year 92

6.1 Block processing time (in milliseconds) compared to state trie size on three platforms (standard deviation between parenthesis) 101
6.2 Details of variables used in our protocol 103
6.3 Number of contracts called after an inactivity period 108

Chapter 1

Introduction

Contents

1.1 Motivations 17
1.2 Contributions 18
1.3 Publications 20
1.4 Thesis overview 20

1.1 Motivations

Blockchains are distributed ledgers that store information in a peer-to-peer network. Each member of this network can own a copy of this register. The modification of this register can be done through *transactions* diffused on the network. These transactions are then processed in batches called *blocks*, by members called *miners* or *validators*. A link between each validated block and its previous one, guarantees the integrity of the history of validated transactions. By this link, a chain is formed: hence the name *blockchain*. Blockchains are often considered as foundations for the development of a new generation of so-called decentralised applications.

Since the launch of Bitcoin (the first public blockchain network), numerous research and development activities have led to the creation of a complex ecosystem. Blockchains are now being studied and used in various business domains, such as finance, cloud computing, energy grids, supply chain, etc. In each of these sectors, the implementation of distributed ledgers through blockchains aims at improving collaboration between different (potentially malicious) actors, process automation (through smart contracts), data security and transparency. The development of cryptocurrencies (and more broadly of decentralised finance) illustrates this potential by making it possible to transfer financial value across the world without the need for trusted third parties such as banks. Much more than just prototypes, blockchain-based applications seem to have a real place in our

economy today. As an example, cryptocurrencies are now worth more than two trillion euros (source: `https://coinmarketcap.com`).

With the development of debates on the management of climate change, the rapid growth of the information and communication technologies (ICT) sector raises many ethical questions. Indeed, in 2020 the European Commission estimated an increase in energy consumption of data centers in Europe of 42% between 2010 and 2018.

Blockchain technologies are far from being an exception to the concern about the environmental cost of ICT. Indeed, blockchains such as Bitcoin require the use of expensive computational resources as part of their consensus algorithm: the proof-of-work. The proof of work relies on a competition, called *mining*, between different members of the network, in order to propose new blocks. In 2014, O'Dwyer & Malone [78] estimated that Bitcoin mining consumed 3GW (about as much as Ireland at the time). To design new blockchains that are less costly and more efficient, new consensus algorithms (such as the *proof of stake*) have emerged. The energy consumption of those new consensus algorithms are often several orders of magnitude inferior than PoW mining [105].

However, we can't reduce the energy consumption of blockchains to that of their consensus algorithm. Indeed, the emergence of *smart contracts*, by proposing the deployment and execution of complete programs on blockchains, makes these technologies more complex. While these smart contracts make many new use cases possible, they also increase the energy footprint of blockchains.

As the power consumption of the consensus phase of blockchains is reduced, we believe that it is important to study the cost of the deployed applications themselves.

1.2 Contributions

This thesis aims to contribute to the study and reduction of the cost of smart contracts on blockchain platforms. Indeed, the energy consumption of Bitcoin mining [78, 68, 66, 111, 72] and alternatives such as proof of stake [105, 65, 14] has been already well studied. However, the energy consumption of smart contracts themselves have been far less studied. Therefore, we have focused our work on the public blockchain platform *Ethereum*, one of the main platforms offering a smart contract system. As large blockchain communities like Ethereum change their consensus algorithms from Proof of work to non-mining algorithms (Proof of Stake in the case of Ethereum), studying the energy consumption of applications built on top of blockchains becomes more relevant. This thesis is structured around three

central questions:

How important are smart contracts in blockchain networks? Although it is clear that academic and industrial activities around smart contracts seem to show strong adoption in various domains, few academic works study their actual use.

Can we estimate and model the energy cost of these contracts? A better understanding of the energy cost incurred by the execution of these contracts seems to be necessary as some blockchain platforms gradually emancipate themselves from *mining* and *proof of work*. The cost of running software applications will then become proportionally more important.

What is the cost generated by the immutability of deployed contracts, a principle at the heart of blockchain technologies? Public blockchain platforms allow anyone to deploy new smart contracts. Once deployed, they are available *ad vitam aeternam*. The question then arises of the cost of this property. Beyond a growing storage space requirement for the network members, does the forever availability, of all the deployed contracts, impact the performances and cost of the blockchain network?

Through analysis of transaction histories stored on Ethereum and experiments *in situ* on the Grid'5000 research platform, we propose to answer these questions through three contributions:

— **A novel framework for benchmarking blockchain technologies: BCTMark**. This framework addresses the challenges of deploying and testing blockchains in an experimental context. Built from *open-source* components, BCTMark addresses the resources management, deployment benchmarking and result backup phases in the performances analysis of blockchains. This work has been presented in *Compas* and *AICCSA* conferences and in the *Cluster Computing* journal.

— **An analysis of the usage and energy consumption of Ethereum smart contracts**. Based on one year of real-world data, we analyze the importance of smart contracts in Ethereum traffic. More specifically, we give insights on the number of smart contract calls, new contracts deployment and the complexity of those contracts. We then propose a model to estimate the energy consumption of Ethereum contracts based on their *gas consumption*. A part of this work has been presented in *Cluster Computing* journal.

— **A novel protocol that can be used to reduce the number of unused contracts in Ethereum**. As shown in this manuscript, a large part of deployed Ethereum contracts will lead to little or no usage. We demonstrate here that the

growing number of contracts has an impact on the performances of each node in the network. As a potential solution to this matter, we propose a protocol to incentive miners to identify and destroy unused contracts. Based on real-world data, we show that our proposition could lead to a significant reduction of the number of contracts stored in the blockchain. This work has been presented in *ISCC* conference and as been awarded as "Best student paper".

1.3 Publications

Research activities presented in this manuscript has led to the publication of several papers.

Papers accepted in journals

Saingre, D., Ledoux, T., & Menaud, J. M. Measuring performances and footprint of blockchains with BCTMark: a case study on Ethereum smart contracts energy consumption. In *Cluster Computing*, 2021 (to appear)

Papers accepted or published in international conferences

Saingre, D., Ledoux, T., & Menaud, J. M. (2021, September). The Cost of Immortality: A Time to Live for Smart Contracts. In *2021 IEEE Symposium on Computers and Communications* (**ISCC**) - **Best student paper award** (to appear)

Saingre, D., Ledoux, T., & Menaud, J. M. (2020, November). BCTMark: a framework for benchmarking blockchain technologies. In *2020 IEEE/ACS 17th International Conference on Computer Systems and Applications* (**AICCSA**). IEEE.

Paper published in national conferences

Saingre, D., Ledoux, T., & Menaud, J. M. (2019, June). BCTMark-Vers un outil pour l'évaluation des performances et du coût énergétique des technologies blockchain. In **COMPAS 2019**-*Conférence d'informatique en Parallélisme, Architecture et Système*.

1.4 Thesis overview

This section presents the content of the following chapters.

Chapter 2 presents the research context behind this manuscript. Section 2.1 motivates this thesis by trying to answer the following question: *why should one study the energy footprint of blockchains?* Section 2.2 aims to give readers a broad understanding

of how blockchains work. The notions required to understand research presented in this manuscript are detailed here.

Chapter 3 presents the current literature related to the research questions presented in section 1.2. Section 3.1 will present a review of the existing literature regarding the energy consumption of blockchain systems. In section 3.2 we will present the existing tools that can be used to evaluate the performances of blockchain systems. Section 3.3 will present existing research that focuses on the analysis of the usage and performances of smart contracts. Finally, section 3.4 will present research work on the deletion of data on blockchains.

The next three chapters are then dedicated to our contributions.

Chapter 4 presents BCTMark, a novel framework to deploy and test blockchains. This framework has then been used to conduct experiments in the following chapters. BCTMark consists in a new tool to deploy and conduct experiments on blockchains and their performances. A set of experiments are finally presented to illustrate the possibilities of BCTMark.

Chapter 5 aims to provide a model to estimate the energy consumption of Ethereum smart contracts, based on real-world data. This chapter is composed of two parts. In section 5.2, we detail how important are smart contract in Ethereum traffic. In section 5.3, we provide a novel way to model the energy consumption of Ethereum contract. Based on this model, we provide estimations on the energy consumed by Ethereum contract over the course of one year.

Chapter 6 presents a simple protocol to identify and remove unused contracts in Ethereum. Section 6.2 identifies the impact of unused contracts on an individual Ethereum node performances. To reduce this growing impact by removing unused contracts from the Ethereum blockchain, we propose in section 6.3 a novel protocol that adds a *time to live* for smart contracts. We quantify the impact on the number of Ethereum contracts, based on historical data, as an evaluation.

Finally, **chapter 7** will conclude this manuscript. We will present there a summary of our contributions and potential perspectives.

Chapter 2

Background

This chapter introduces the fundamental concepts involved in this thesis, motivate our research.

Contents

2.1 Questioning the environmental impact of Blockchain technologies 23
2.1.1 The energy consumption of ICT 23
2.1.2 The development of blockchain technologies 25
2.1.3 Why study the energy footprint of blockchains? 29
2.2 Blockchains 29
2.2.1 An overview on blockchain technologies 30
2.2.2 A note on Ethereum's data storage 33
2.2.3 Chain fork choice 34
2.2.4 Differences between public and private Blockchains 35
2.2.5 Blockchain consensus algorithms 35
2.2.6 Decentralised applications with Smart contracts 38
2.3 Conclusion 44

2.1 Questioning the environmental impact of Blockchain technologies

2.1.1 The energy consumption of ICT

With the advent of the Internet, the *Information and Communication Technology* (ICT) sector has boomed in the last decades. Alongside with the development of an awareness of climate change, the growing environmental impact of ICT equipments raises ethical questions.

Definition of ICT

***I**nformation and **C**ommunication **T**echnology: the use of computers and other electronic equipment and systems to collect, store, use, and send data electronically* - Cambridge Business English Dictionnary

The European commission [71] has estimated that the energy consumption of data centers in Europe increased by 42% between 2010 and 2018, going from 53.9TWh/yr to 76.8TWh/yr. In 2018, 2.7% of the European energy demand was intended for data centers. They estimated a growth of this demand by 21% by 2025, reaching 92.6TWh/yr (see Figure 2.1).

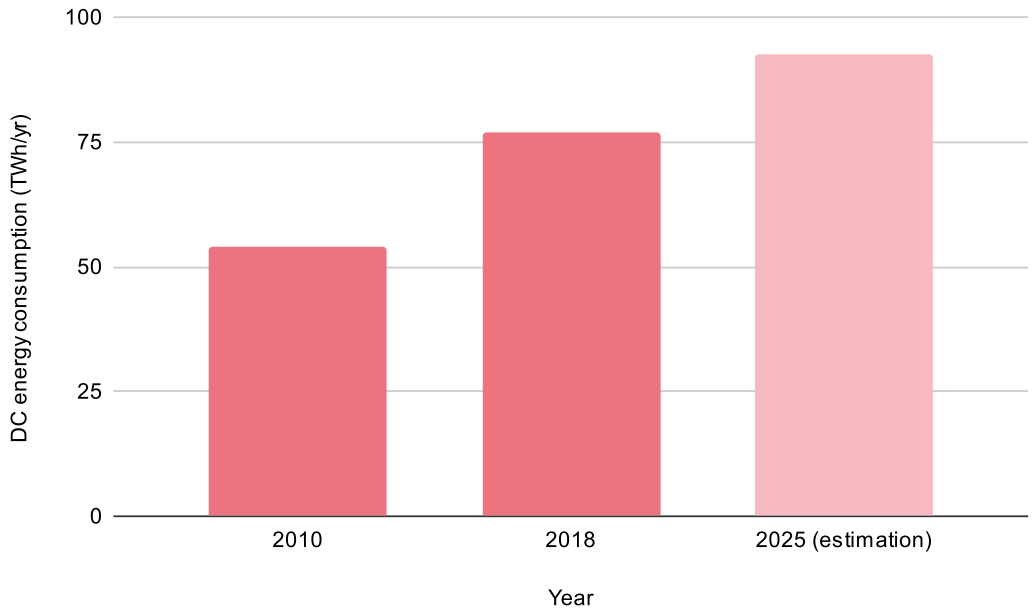

Figure 2.1 – Energy consumed by European Data Centers (data source: [71])

However, data centers only represent a part of the global energy consumption of ICT (around 14% [97]). Between 2015 and 2018, the ICT sector's energy consumption rose from 3289 to 4181 TWh [97]. This represent the equivalent of an increase of 6.2% every year. Taking into account the carbon intensity of the world electricity production, the ICT sector has been responsible for the emission of 1.8 Gt of CO2e in 2019. This represented 3.5% of

the year's overall greenhouse gas emission. Several factors can explain this increase in the carbon footprint: the development of the internet of thing (IoT), a growth in computation needs (e.g. for machine learning and data analytic), the development of video streaming...

Figure 2.2 illustrates a breakdown of the carbon footprint of the ICT sector. The CO2e emission produced by the production of the ICT infrastructures (servers, computers, smartphones...) only represent a third of the sector carbon footprint.

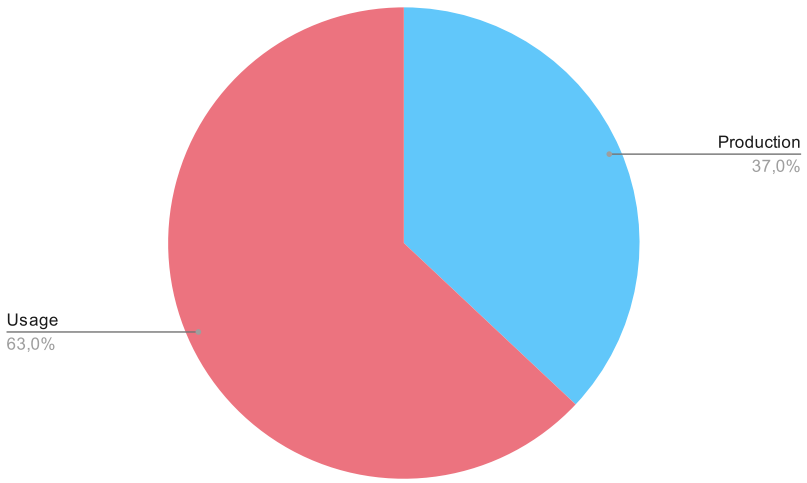

Figure 2.2 – The carbon footprint of the ICT sector in 2019 - breakdown between production and usage (data source: [97])

As two thirds of the greenhouse gas emission of the ICT sector come from its actual usage, it is crucial to understand how to reduce the energy consumption of infrastructures and software.

2.1.2 The development of blockchain technologies

Blockchains appeared in 2008 with the introduction of *Bitcoin* by Satoshi Nakamoto. Since then, a large and complex ecosystem (composed of different blockchain platforms, libraries and APIs, services...) has emerged.

The National Institute of Standards and technology (NIST) defines blockchains as a *tamper evident and tamper resistant digital ledgers implemented in a distributed fashion (i.e., without a central repository) and usually without a central authority (i.e., a bank, company or government)* [117]. Put another way, blockchains are a specific kind of *distributed ledger technology* (DLT). Rauchs et al. [91] discussed the concept of *distributed*

ledger technology and defined it as a "*multi-party system in which participants reach agreement over a set of shared data and its validity, in the absence of a central coordinator*". As explained in [12], not all DLT are implemented as a chain of blocks, although both terms are often used interchangeably. Figure 2.3 illustrates the relationship between databases, DLT and blockchains as sets. DLT can be seen as a specific kind of database whereas blockchains are an example of DLT. In the case of cryptocurrencies, the blockchain serves as a ledger that keeps track of all the financial exchanges between users without the need of a trusted bank.

The NIST described four key characteristics of what blockchains are [117]:

— **Ledger** - Blockchains are append-only ledgers. Each transaction modifying the blockchain state is stored so that the entire history can be read at any time.

— **Secure** - Using cryptographic securing mechanism (e.g. hash references between a block and the previous one, Merkel trees to store transactions), blockchains provide a tamper-proof ledger

— **Shared** - The ledger's data is shared among the peers, ensuring transparency of its data

— **Distributed** - the blockchain is distributed among peers, making it more resilient to malicious actors.

Blockchains, in particular those including smart contracts, seek to enable the development of decentralised applications. Hoffman et al.[50] discussed the concept of *decentralised applications* and its differences with *distributed computing*. They define distributed blockchain platforms as "*multi-stakeholder Web ecosystems acting as sophisticated support networks enabling peer-to-peer transfers of value and information, including goods and services, in a coordinated manner*".

Ethereum - A decentralised application platform

In this thesis, we focused our research and experimentation on Ethereum, one of the leading open-source Blockchain. First announced by *Vitalik Buterin* in a white paper in 2013, the main Ethereum network (called *mainnet*) has been live since 2015. As illustrated in Figure 2.4, every day more than 1 million transactions are processed by the Ethereum network. In October 2021, Ethereum market capitalisation peaked at nearly 500 billion dollars, making it second after Bitcoin.

Ethereum [44] has been the first blockchain to implement smart contracts. Ethereum is a Proof-of-Work based blockchain. However, the soon-to-be coming version two will

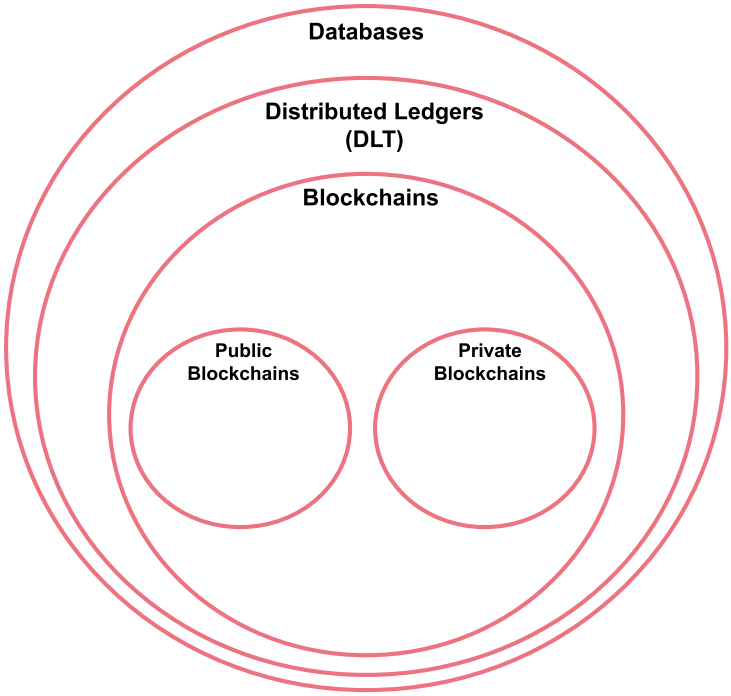

Figure 2.3 – Hierarchy between databases, distributed ledger technologies, and blockchains

move to a Proof-of-Stake based consensus engine, removing any mining. The genesis (first block) of Ethereum 2 has been deployed on December 1st 2020. The merge between the first version (Proof-of-work based) and second version (Proof-of-Stake based) of Ethereum is expected to happen in 2022. Since then, both networks exist in parallel.

Examples of blockchain use-cases

Blockchain technologies have been used in several domains, including but also going beyond cryptocurrencies. They are often used to build decentralised alternatives to existing services and products. We can cite several application domains from the literature:

— **Finance** - Finance has been the first use case of Blockchains, with the development of cryptocurrencies like Bitcoin or Ethereum. The Euro Banking Association stated that blockchains could bring several benefits to the finance sector such as offering real-time transparency on trade transactions [3]

— **Cloud computing** - Decentralised, blockchain-based, cloud platform have recently emerged. Uriarte et al. [110] have studied and compared three different platforms: IExec [51], Golem [119] and SOMN [100]. Both three projects aim to build a Cloud platform where individuals can rent their unused computational power, so oth-

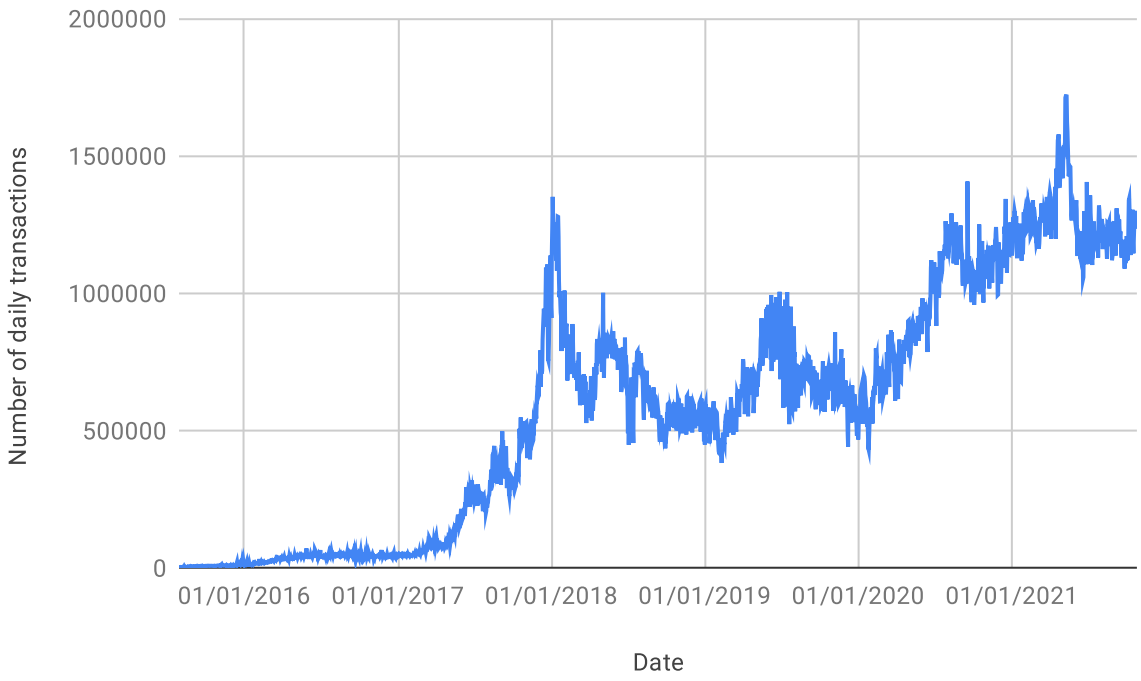

Figure 2.4 – Evolution the number of daily transactions on Ethereum (source: `etherscan.io`)

ers can run their applications. Uriarte et al. concluded that, in this setting, one challenge is to verify the computation one paid for. Moreover, decentralised cloud solutions often suffer from a lack of standards, making interoperability (e.g. in services definition, execution workflow) and comparison difficult.

— **Energy systems** - Blockchains have been used to build a decentralised peer-to-peer energy market. For example, the Brooklyn Microgrid (BMG) projects, run by the *LO3 Energy* company, has built a microgrid energy market in Brooklyn (New York). In a literature review paper, Mengelkamp et al. [69] have concluded that blockchains could be used to manage a microgrid energy market. They pointed out that the BMG project enabled a local energy trading at varying price where financial profits would benefit the local community. Similar projects have been also built by LO3 Energy in South Australia, Germany and in Texas (USA) [63].

— **Supply Chain** - Blockchains have been used in the supply chain sector to increase transparency, traceability and efficiency [24]. For example, IBM, Walmart and Tsinghua University have collaborating to leverage blockchains for food products

traceability [112]. A blockchain-based platform could be used to store a product's processing data (e.g. farm origination, expiration date, storage temperature) across several actors in the supply chain.

2.1.3 Why study the energy footprint of blockchains?

The energy demand of the ICT sector keeps increasing as new technologies and usages emerge. Worries on the growing environmental footprint of ICT leads to research activities on reducing this impact. As we saw, two thirds of the greenhouse emitted by the ICT sector comes from its actual usage. Understanding how to build more frugal infrastructures and software could help to reduce this impact.

Since its creation a few years ago, the blockchain ecosystem has largely developed. Although research activities on blockchains keep growing, these technologies still suffer from a negative image of high energy-consuming systems. This image is mainly induced by one of its leading representatives: Bitcoin and its Proof-of-Work consensus algorithm. In 2014, O'Dwyer and Malone [78] estimated that 3GW was necessary to power Bitcoin miners. Four years later, Mora et al. [72] stated that in a few decades, Bitcoin mining could have an impact on global warming on its own. In parallel, communities like Ethereum tries to develop and use new consensus engines like Proof-of-Stake in order to increase the performances of their system and limit its energy cost. As the blockchain ecosystem grows and become more complex, we believe that the blockchain community would benefit from research on blockchain's energy consumption.

The energy consumption of blockchains is not a novel research topic. Bitcoin mining [78, 68, 66, 111, 72] and alternatives such as proof of stake [105, 65, 14] has been already well studied. However, blockchains are not only a consensus system. With smart contracts, users can deploy and execute complex programs on a blockchain network to address more complex use cases. Unfortunately, the energy consumption of smart contracts themselves have been far less studied than that of blockchain's consensus systems.

2.2 Blockchains

This section aim to give readers a broad understanding of blockchains internals.

2.2.1 An overview on blockchain technologies

A blockchain is a distributed database that allows facts (called *transactions*) to be recorded as blocks (A batch of transactions). Each block has a link to the previous one (making a "chain of block," or *blockchain*). This data structure is distributed among all participants in a peer-to-peer network. This network is maintained by some peers called *miners* (for Proof-of-work based blockchains) or *validators* (for others), who are in charge of transaction validations.

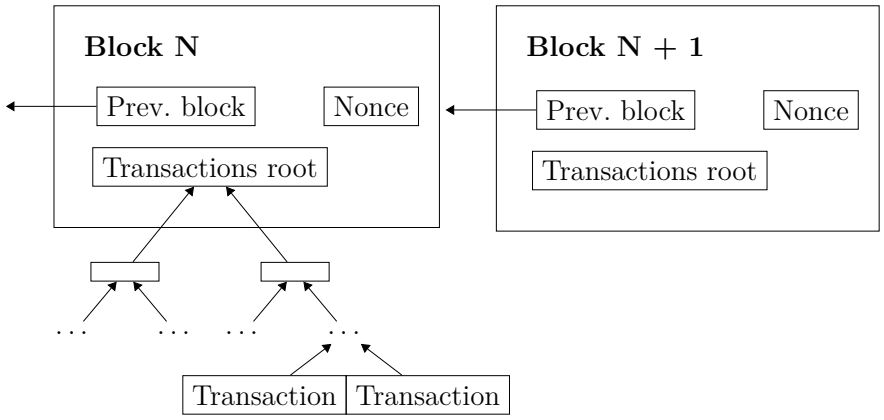

Figure 2.5 – A schematic view of a classical blockchain "data-structure"

A block in itself is a structure containing several fields. Table 2.1 present several fields contained in an Ethereum block.

To prevent the modification of previously validated transactions, blockchains make use of *hash functions* (e.g. Keccak256 for Ethereum [44]). Hash functions are a family of function that takes arbitrary data an input to produce fixed-size values. One key property of hash functions is their resistance to *collisions*: given an input and its hash value, it is costly (in terms of computation) to find another input that matches the same hash value. As presented in Figure 2.5, each block in the blockchain contains a reference to the previous one. This reference is the hash value of the previous block header. Therefore, modifying a given block requires modifying all subsequent blocks. Hash functions can also be used for transaction storage inside a block. For instance, Ethereum uses Merkel trees to store transactions in a block. As illustrated in Figure 2.6, Merkel trees are a type of tree data structure where data are stored in leafs and each node contain the hash value of its children. The use of Merkel trees to store transactions implies that peers can only store the root node of the transaction tree: the modification of a transaction will imply the

Field name	**Description**
number	The length of the blockchain once this block is added. Also known as block height
hash	The hash of the block header
parentHash	The hash of the previous block header
Nonce	Value used to resolve this block's proof of work.
transactionsRoot	The root of the corresponding transaction trie (the structure containing transactions)
stateRoot	The root of the corresponding state trie (the structure containing the state (i.e. data related accounts like balances and addresses)
receiptsRoot	The root of the corresponding receipt trie (the structure containing transaction receipts (i.e. transaction "effects")
miner	The address of the account that mined the block
difficulty	The effort required to mine this block. Evolves over time to maintain a constant block emission rate.
totalDifficulty	The sum of each block's difficulty up to this block.
extraData	Arbitrary data that can be included in this block by the miner.
gasLimit	The gas limit of this block. gasUsed cannot exceed this value.
gasUsed	The sum of gas used by each transaction included in this block.
timestamp	The date on which this block has been mined.
transactions	The transactions included in this block.
baseFeePerGas	The fees taken by miners on each transaction per unit of gas

Table 2.1 – Examples of fields included in an Ethereum block

modification of the transaction tree root's node and therefore will imply the modification of the block's hash.

The lifecycle of a transaction in a blockchain

Blockchains can be viewed as state machines, where each transaction will modify the existing state to produce a new one. In a given block, transactions are ordered and executed sequentially. Transactions are mostly used to send funds from an account to another (but can also be used for operations that are specific to a given blockchain, such as staking for Proof-of-Stake based system). In smart contract enabled blockchains, transactions can also be used to deploy and call contracts. The initial state is often called *genesis* and can contain, for instance, prefunded accounts.

Transactions are emitted from a blockchain client. The transaction message contains a collection of fields, including a recipient and an amount of tokens (Bitcoin, Ether...)

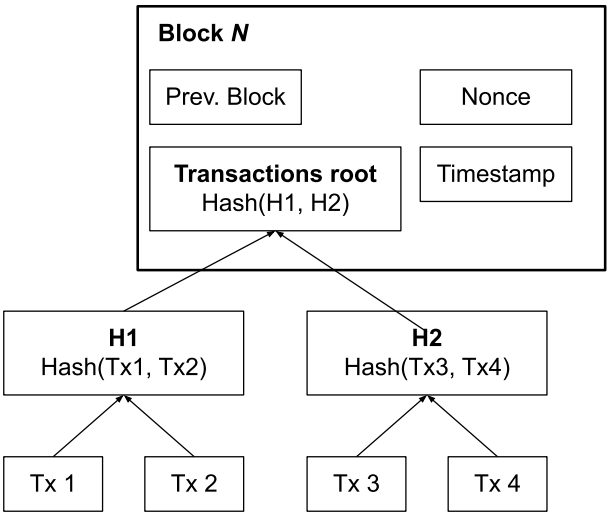

Figure 2.6 – The use of a Merkel tree to store transactions in a block

to transfer. It can often include arbitrary data that can be used, for instance, for smart contract calls. The transaction is then electronically signed by its sender. As all blockchain accounts are tied to a particular private key, no one but a key's owner can send funds from its related account.

Cryptographic signature of blockchain transactions

Transactions emitted over a blockchain network are signed by an asymmetric cryptographic scheme. Bitcoin and Ethereum, for example, uses an Elliptic Curve Digital Signature Algorithm (ECDSA) [53]. In order to sign a transaction, a blockchain user needs a private / public key pair. Using its private key, the user will sign its transaction before sending it across the network. The address of the account emitting the transaction is derived from its public key.

Once signed, the transaction is broadcasted across the network. Each blockchain peer that receives a transaction stores it in a *transaction pool*. The transaction pool is responsible for the storage of transactions to be included in a block. For each block creation, miners pick new transactions from the pool. The choice of which transaction to pick can depend on each miner, but it is reasonable to consider that transactions with the higher fees are picked first. Once the new block is mined, it is broadcasted across the network.

Peers receiving the block can choose to accept it and to include it in their blockchain. Transactions included in this block can then be considered as validated.

As an incentive for miners / validators to actively maintain the network, every block created include a financial reward for the block creator. This reward is often composed by:

— An intrinsic reward (the *block reward*), independent of the block content. This block reward is a way to create new tokens, increasing the total supply of the network. Several blockchains decrease this reward at a regular interval through a process called *halving*. For example, at its creation, Bitcoin offered 50BTC for every new block created. Since the last having (2020), the reward is 12.5BTC.

— Transaction fees, corresponding to the sum of the fees of each individual transaction included in the block. Fees values are often chosen by the transaction emitter and act like a *tip* for miners. Increasing the fees of a transaction can decrease its processing time as it incentive miners to include it in new blocks. In some blockchains, a minimum amount of fees is required. In Ethereum, this minimum amount is proportional to the amount of computation the transaction will induce for the network.

2.2.2 A note on Ethereum's data storage

In Ethereum, all blockchain data is stored in several modified Merkle Patricia Trees (called *tries*). The *State* trie stores all information related to accounts (including smart contracts). Each account in the State trie is a collection of information related to a given 20-Bytes hexadecimal address, like its balance (number of Wei [1] possessed by the account) and (for smart contracts) its *EVM* byte-code.

Each smart contract can store data in a *Storage* Trie. This Storage Trie is used to store contract variables that will persist across function calls. Contract's account, stored in the state trie, contains the 256-bit hash of its Storage root. The information in the state and storage tries are updated through transaction executions. Figure 2.7 illustrates the relationship between a given block, the state trie, and the storage trie.

Ethereum serializes its different Tries using a *Recursive Length Prefix* (RLP) encoding scheme (see Ethereum *yellow paper*[44]). Serialized data are then stored in a Key-Value database. For instance, the *Geth* Ethereum implementation uses LevelDB [46] to store its

1. The Smallest denomination of Ether. 1 Ether = 10^{18} Wei

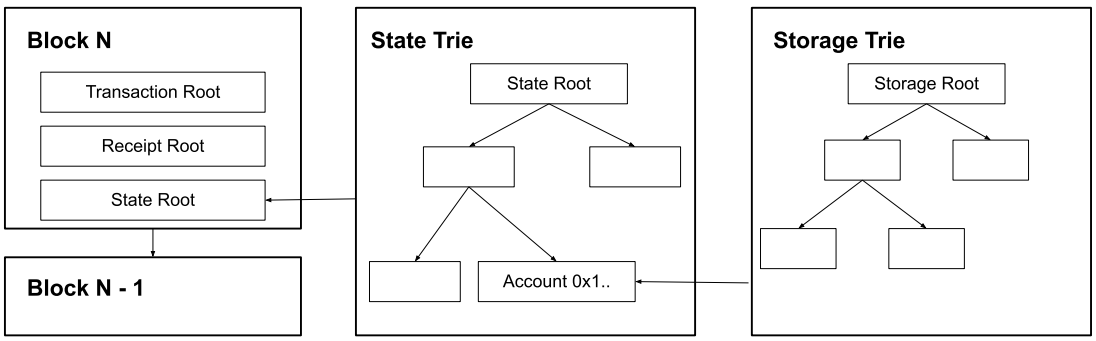

Figure 2.7 – Ethereum's data storage

data.

2.2.3 Chain fork choice

As many miners / validators can compete to produce blocks in parallel, two blocks can be emitted at the same time. When a peer receive two new blocks referencing the same parent block simultaneously, a chain fork (or chain split) happens. As illustrated in Figure 2.8, both blocks are stored.

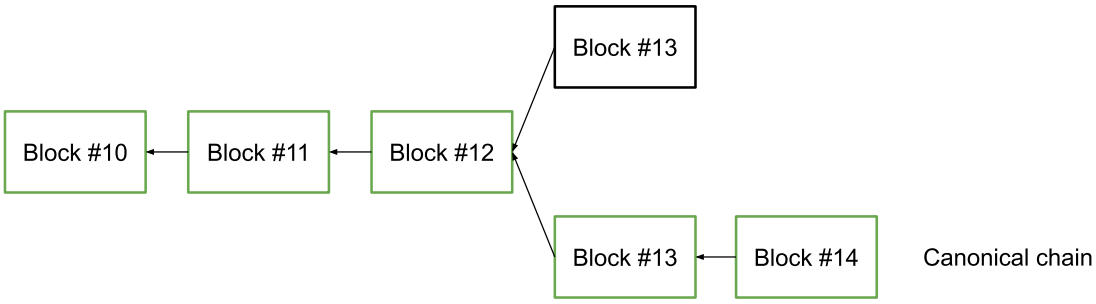

Figure 2.8 – Chain split management

With Bitcoin, the fork with the longest chain is considered as the "main" (canonical) one. However, not all blockchains have the same rules for choosing the fork in the chain. For instance, on Ethereum, the canonical chain is the one where the sum of its blocks' difficulty is the highest.

When the two forks have the same size or difficulty, miners will choose on which fork to mine until one fork outrun the other.

2.2.4 Differences between public and private Blockchains

Blockchain technologies can be divided into two categories: public / permissionless and private / permissioned blockchains.

Public blockchains, such as Bitcoin and Ethereum, have no identified users. One can join or leave the network at any time without the need for any authorisation. Security protocols of those public blockchains need to be enforced to face potential Byzantine faults. Proof-of-work is an example of a consensus system for public blockchains. Public blockchains are also called permissionless.

Private blockchains have different security models. They aim to identify participants, especially for the block validators. These are designated in the protocol so that no one else can validate the block. These blockchain systems, such as Ethereum (with its *Clique* engine) and Hyperledger Sawtooth (with its *Proof of Elapsed Time* system, based on the *Intel SGX* enclave), have different consensus engines. These engines have better performances (due to the different security models considered) but offer a lower degree of decentralisation. In the literature, a distinction can sometimes be made between *private* and *consortium* blockchains. In this case, *private* will refers to blockchains owned by a single authority, where *consortium* will refer to blockchains owned by several authorities. In this thesis, we will consider private and consortium blockchains as a whole, labelled as *private blockchains*.

2.2.5 Blockchain consensus algorithms

Peer-to-peer systems like blockchains consist in a large-scale, distributed network of entities. Blockchains admit no central entity to process transactions and deliver a correct state. Put differently, the whole network is decentralised: transactions can be broadcast, processed and stored by any peer. In this settings, blockchain networks are prone to a specific kind of failure: Byzantine fault tolerance (BFT) [60]. This kind of failure arise when malicious peers adopt arbitrary behaviours and spread deceptive messages to others. Ensuring that the network will converge to the same state is a challenging task. Therefore, the choice of a correct and efficient consensus algorithm is at the heart of blockchain projects.

Moreover, the network needs to ensure that transactions are validated by enough independent entities to reduce the risk of any censorship or bias in the choice of transactions to be processed. In [27], John R. Doucer first described the concept of *Sybil attacks*. A Sybil

attack consists in one single entity forging multiple identities. Without an identity validation protocol, it is impossible to ensure that the entities validating transactions do not belong to the same identity. However, having a single trusted entity to manage the identity of peers would however defeat the decentralisation aspect of public blockchains. "Proof" based consensus algorithms (like *Proof of work*) are designed to build public blockchains that are resistant to this kind of attack.

A lot of different consensus algorithms (some more popular than the others) are used for blockchains, for instance Ouroboros [54], Stellar [67], DPOS [28], Tendermint [108] ...

We present here three blockchain consensus algorithms mentioned in this manuscript.

Proof-of-Work

Proof-of-Work (abbr. PoW) has been the first consensus algorithms used by the blockchain community. PoW has been popularised by Bitcoin. The concept behind Proof-of-Work has been initially invented by Dwork and Naor [29] as a protocol to deter denial-of-service attack and spam on email services.

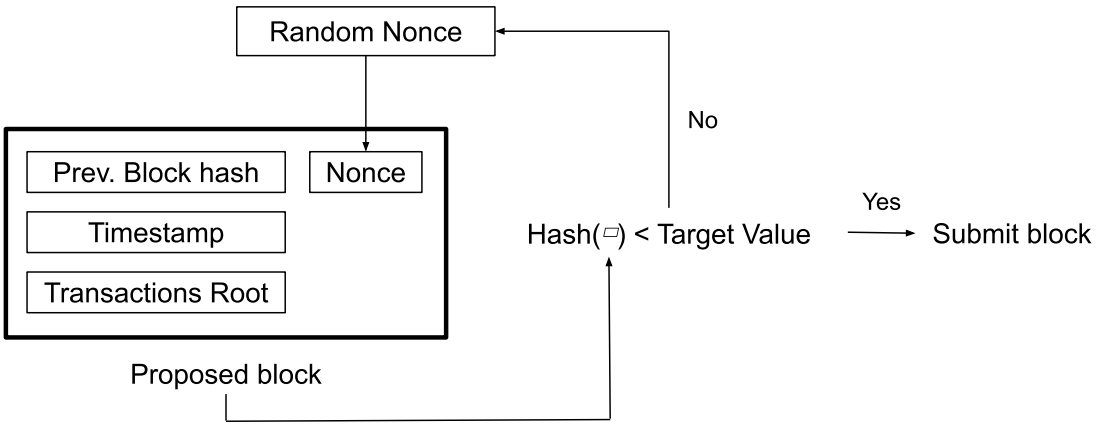

Figure 2.9 – How PoW works

With PoW, miners have to complete a resource-intensive computational task, often qualified as a "cryptographic puzzle". Every block here contains a value called a *nonce*. To be able to propose a valid block, a miner has to find a value for the nonce, such as the hash value of the whole block header is under a certain threshold called *target* (as illustrated in Equation 2.1). Once found, the miner can broadcast his block on the blockchain network.

$$\text{SHA256}(\text{SHA256}(block\_header) <= target \tag{2.1}$$

This threshold value evolves so that, even as the hardware becomes more powerful, the throughput of the entire network remains stable. For Bitcoin, this evolution aims to keep an emission rate of one block per 10 minutes. The *difficulty* is a variable often used to illustrate the amount of effort to do in order to mine a new block. An increase of this *difficulty* correspond to a decrease of the *target* value (a smaller target value implies fewer possible values for the nonce). Figure 2.10 illustrate the evolution of Ethereum mining difficulty from 2015.

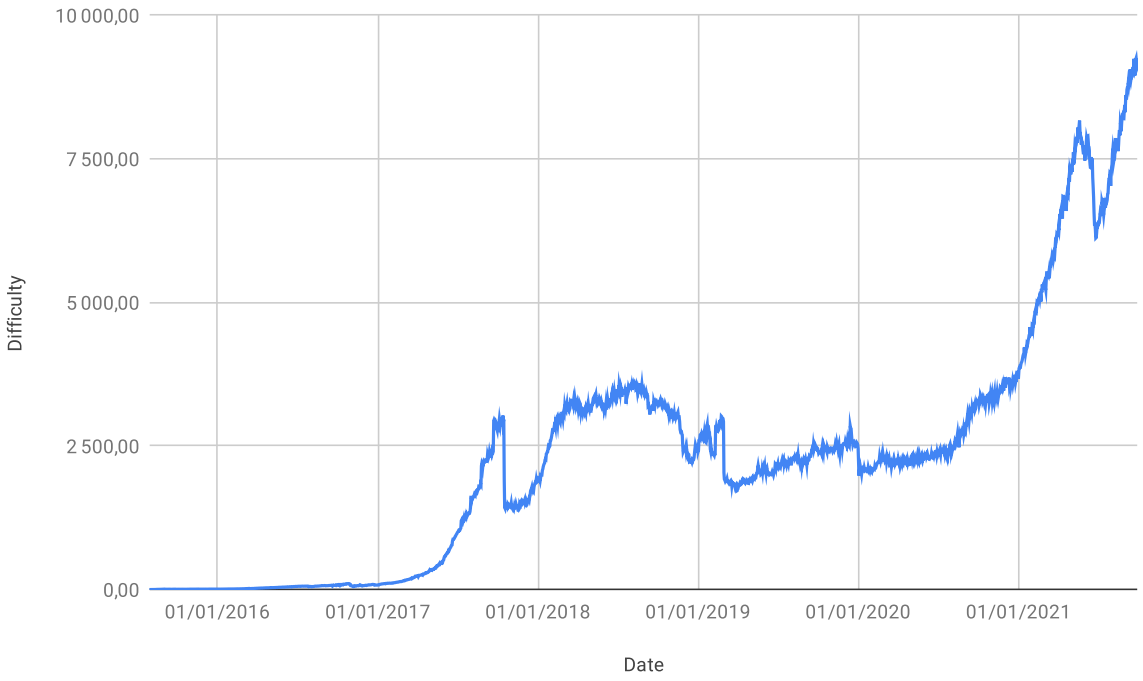

Figure 2.10 – Evolution of Ethereum mining difficulty

The overall goal with PoW is to create a system where "one CPU is equal to one vote". Although PoW has been widely used with Bitcoin, it has been quite criticised for its high-energy consumption [78]. Alternatives such as *proof-of-stake* [95] have emerged.

Proof-of-Stake

Instead of basing its security on computational power, *proof-of-stake* (abbr. PoS) systems rely on the distribution of wealth among validators. In order to be able to participate in block creation, a peer needs to prove the possession of a certain amount of tokens. Often, they will need to send tokens to an escrow accounts (certain networks impose a

minimum value). This phase is called putting tokens *at stake* (hence the name of the protocol). Then, for every block emission, participants will then get a chance to get elected as the leader / forger / minter / validator for this block. The chance to be elected as a block validator depends on the amount put at stake. PoS often comes with a punishing mechanism for validators that attempt to propagate malicious blocks.

In Ethereum 2.0, for example, users needs to put a fixed minimum of 32 ETH to be included into the *committee* (the validators pool). Once proposed, a block needs to be validated by $\frac{2}{3}$ of the validators to be finalised. If the block is rejected, the block author will lose its stake. This punishing mechanism ensures that any malicious behaviour from block validators induce a high financial cost.

At the core of these systems, a validator's voting power is proportional to the amount of wealth put at stake. To put it in another way, instead of voting "with their CPU power", stakeholders will "vote with their wealth".

Proof-of-Authority

Proof-of-Authority (abbr. PoA) is a consensus protocol used in private blockchains. As described in [18, 104], the goal behind PoA is to maintain a set of trusted signers (individuals in charge of block emission, equivalent to miners or validators). Ethereum's implementation of PoA (Ethereum *Clique* [104]) include a voting protocol to include or exclude an account in the signers' group. Usually, once a group of signers is formed, each individual will be designated in a *round-robin* fashion as the next block signer.

For a given set of S signers, PoA assumes at least $N/2 + 1$ honest signers[18].

2.2.6 Decentralised applications with Smart contracts

Generalities

The first generation of blockchains (like Bitcoin) had use-cases limited to token management and cryptocurrencies. Even if some of them featured scripting capacities (like *Bitcoin script* for Bitcoin), they lacked Turing complete programming environment required to develop more complex applications.

Ethereum [43] introduced the concept of *smart contracts* in the blockchain ecosystem. Smart contracts are Turing complete script that can be deployed and executed on the blockchain, through the same transaction system used by token exchange. By allowing arbitrary computation to be run on the blockchain, they extend its possible use-cases.

Table 2.2 compares several smart contract platforms [2]. As we can see, smart contracts' platform can either leverage existing programming languages and execution environment (e.g. Golang and Docker for Hyperledger Fabric) or implement their own (e.g. Solidity and EVM for Ethereum).

Once written, those smart contracts can be deployed on the network through a transaction containing their compiled code. Calls to the deployed contract usually use the same transaction mechanism used for token transfer. Contract execution is usually replicated across the network and its result will be stored on the blockchain.

Ethereum smart contract's life cycle

On Ethereum, smart contracts are executed on a specific virtual machine called the *Ethereum Virtual Machine* (EVM).

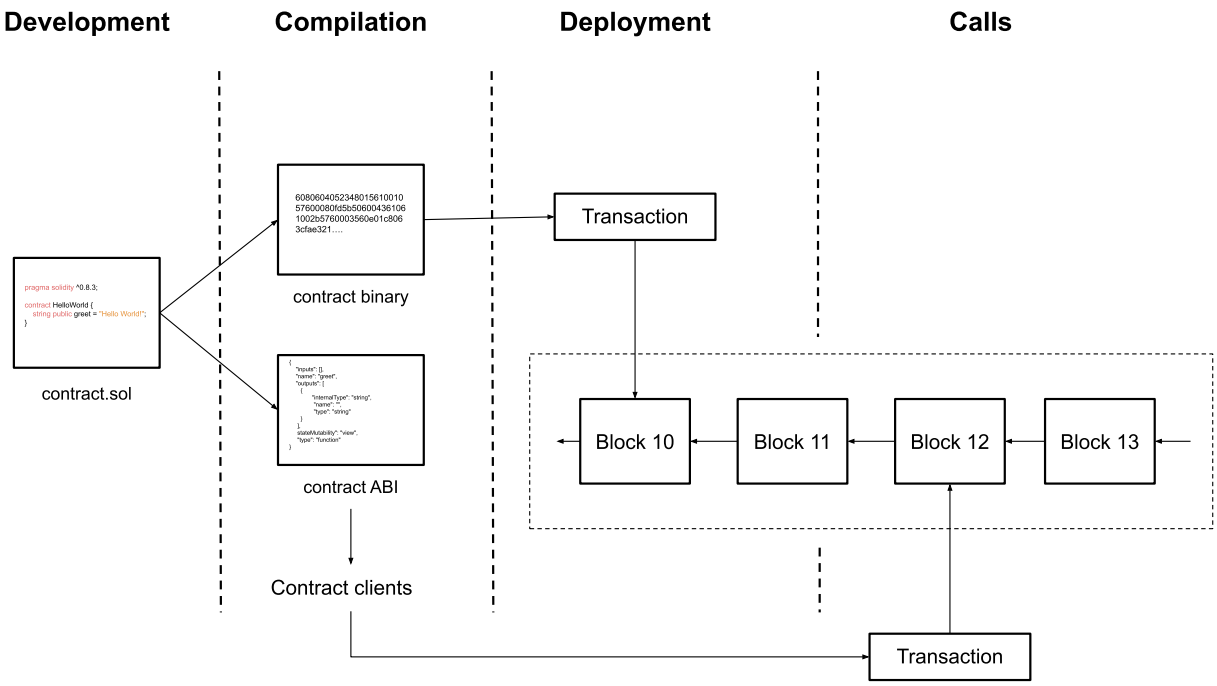

Figure 2.11 – An Ethereum contract's lifecycle

Figure 2.11 illustrate the lifecycle of an Ethereum smart contract, divided in four parts: **development**, **compilation**, **deployment** and **calls**.

2. We didn't include NEM, presented in [49], as we couldn't find any evidence of smart contract capacities in their documentation

	Execution environment	**Smart contract programming language**	**Consensus algorithm used by the blockchain**	**Permission model**
Ethereum [44]	EVM (Ethereum Virtual Machine)	Solidity	PoW	Public
Hyperledger Fabric [39]	Docker	Java, Golang	PBFT	Private
Neo [76]	NeoVM	Python, C#, Go, Typescript, Java	dBFT	Public
Corda [16]	JVM (Java Virtual Machine)	Java, Kotlin	Raft	Private
Quorum [90]	EVM	Solidity	Proof of Authority	Private
Cardano [9]	EVM	Solidity, Plutus	PoS	Public
Stellar [102]	Docker	Python, Javascript, Golang	Stellar consensus protocol	Private
Rootstock [93]	RVM (Rootstock Virtual Machine)	Solidity	PoW	Public
EOS [32]	WebAssembly	C++	BFT-DPOS	Public
Waves [114]	Custom Virtual Machine	RIDE	PoS	Public
Tendermint [108]	Custom Virtual Machine	RIDE	BFT	Private

Table 2.2 – Comparison between different blockchain smart contract platforms (sources: [49], [121], [94])

On Ethereum, contracts are usually developed in a dedicated programming language like Solidity. A smart contract in Ethereum can be seen as a class in *object-oriented programming*. It is composed of data and methods. A contract instance is *stateful*: data can be saved and used across multiple execution of a contract. For instance, a contract implementing a token-management system will need to save its user's balances. A contract's (public) methods will be its actual entry point: interacting with a contract comes down to calling its function, potentially with a given set of parameters.

Solidity

Listing 2.1 illustrate a simple *Hello World* in Solidity. Once compiled, it can be used to deploy a contract called *HelloWorld* that contains one function ("hello()") that will return "Hello, World". The *pragma* instruction specifies the compiler version required for this contract.

```
1 pragma solidity ^0.8.3;
2 contract HelloWorld {
3     string public greet = "Hello World!";
4     function hello() public view returns (string) {
5         return greet;
6     }
7 }
```

Listing 2.1 – Illustration of an *Hello World* in Solidity

Once developed, the contract can be compiled for the *Ethereum Virtual Machine* (EVM). From compilation will result two artefacts: the contract's byte code and the contract's *Application Binary Interface* (ABI). The contract's ABI is a JSON file that document the different methods' signatures specified in the contract. It can be used to generate clients to interact with the contract, once deployed. The contract's byte code, once deployed, will be stored in the blockchain and be used for contract execution.

On the EVM, transactions without any recipient are used for smart contract deployment. In order to deploy a compiled contract, a developer will emit a transaction containing the contact's byte code and without any recipient. Once processed and included in an accepted block, a new account will be created for the contract. Users can then send a transaction to the contract's account to execute it.

Ethereum contracts

Ethereum has two notions of accounts:

— Externaly owned accounts (EOA): accounts controlled by a private key,

— Contracts accounts: accounts controlled by a smart contract.

Both accounts possessed an address and can emit transactions to send ethers and call smart contracts. A key difference is the presence of bytecode for contracts accounts that can be executed once called by a transaction.

Contract calls on the EVM are done through transactions. To call a specific method M of a contract C, a user needs to encode the contract call in a transaction sent to the address of C. A contract call is composed of two parts: 1) the identifier of M (obtained by hashing M's signature) and 2) the list of methods parameters, if needed. The transaction will then be broadcasted into the Ethereum network and included in a block.

The notion of gas in Ethereum smart contracts

In Ethereum, transaction emission requires a fee paid by the caller. This fee will be collected by the miner that will include the corresponding transaction in its block. Therefore, it serves as an incentive for miners to include the transaction in their blocks. Transaction fees are calculated as such: $transaction\_fee = gas\_cost \times gas\_price$ where gas_price is defined by the caller and gas_cost equals to the sum of gas cost of each EVM instruction called during the smart contract function execution. As a result, the more computations a smart contract function does, the more expensive its call will be. Callers can freely define the price (gas_price) they are willing to pay for each gas unit the contract call will consume (gas_cost). This price depends on the market's state: a higher gas price will make the transaction more expensive for the caller to execute, but will serve as a higher incentive for miners to include the transaction in their blocks. Ethereum's Yellow Paper [44] defines the gas cost of each EVM instruction. A sample of the cost table can be found in Table 2.3. We can see that every transaction has an inherent cost of 21 000 gas ($G_{transaction}$). Deploying a new smart contract induces a minimal cost of 53 000 gas (21 000 gas for the transaction itself and 32 000 to create a new account). This represent, at the time of writing this thesis around 25 USD (source : `ethgasstation.info`).

Name	Value	Description
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{create}	32000	Paid for a CREATE operation.
G_{call}	700	Paid for a CALL operation.
$G_{transaction}$	21000	Paid for every transaction.

Table 2.3 – Samples of Gas cost definition from Ethereum Yellow Paper[44]

Interfaces and standardisation of Ethereum smart contracts

To standardise the behaviour of similar smart contracts, the Ethereum community has defined several standard interfaces (with a set of function signatures to be implemented). Standard, community-approved interfaces for contracts that implement similar use cases allow users and applications to interact more easily with those contracts. ERC20 [38] (ERC: ***E****thereum* ***R****equest for* ***C****hange*) and ERC721 [115] refer to smart contracts that implement the eponymous interfaces. Both implement a token management system: a fungible token for ERC20 and a non-fungible token for ERC721. The key differences between those two kind of tokens is that non-fungible tokens process a clear identity and owner whereas fungible tokens does not. For instance, a diploma can be considered as a non-fungible token (each diploma is unique and belong to one person). Money can be seen as a fungible token system as coins have no clear "identify" (Coins of the same value do not have any differences and can be exchanged indifferently).

Listing 2.2 details the interface of ERC20 tokens as specified by [38]. Getters (*totalSupply* and *balanceOf*) give values of total amount of the implemented token and the balance of a single account. Other functions such as *transfer*, *approve*, *transferFrom* are here to manage transfers of tokens between accounts.

```
interface IERC20 {
  function totalSupply() external view returns (uint256);
  function balanceOf(address account) external view returns (uint256);
  function allowance(address owner, address spender) external view returns (uint256);
  function transfer(address recipient, uint256 amount) external returns (bool);
  function approve(address spender, uint256 amount) external returns (bool);
  function transferFrom(address sender, address recipient, uint256 amount) external
      ↪ returns (bool);
}
```

Listing 2.2 – ERC20 Smart contract Solidity interface[38]

Studying whether a given contract implement a known interface can help to understand the purpose of this contract.

Managing side effects with oracles

As smart contracts need to be deterministic, they can't have side effects beyond the Blockchain (e.g. sending request to an external service exposed over the internet).

A common pattern to import off-chain data into a blockchain is to implement an oracle. Oracles are third-party services that serve as a bridge between a blockchain and the "outside world", either by pushing data into a blockchain or monitoring its state to produce side effects outside the blockchain. Mühlberger et al. [73] have characterised four oracle patterns, depending on the data flow direction (data going in or out of the blockchain) and the initiator of the data flow (push or pull from the blockchain's perspective). For instance, *Provable* [89] is an oracle service that integrate with several blockchains. It serves as an intermediate between smart contracts and different data sources such as IPFS.

2.3 Conclusion

This chapter was twofold. First, section 2.1 aimed to motivate research on the understanding and reduction of the environmental footprint of blockchain technologies. In the context of a global climate change and the growth of the footprint of the ICT sector, it appears crucial to ensure the design of energy-efficient technologies. Blockchain technologies, well-known for their energy consumption, are no exceptions. In section 2.2, we presented the technical aspects of blockchains required to appreciate the rest of this manuscript.

Chapter 3

State of the art

Contents

3.1 A literature review on the energy consumption of blockchain technologies 45

3.1.1 The energy footprint of Proof-of-Work mining 46

3.1.2 Alternatives to mining 51

3.2 Evaluate the performances of blockchain systems 52

3.3 Analysing the uses and energy consumption of smart contracts 53

3.4 Data suppression on blockchains 55

3.5 Conclusion 55

This chapter summarizes the existing scientific literature related to the research presented in this thesis. First, we will present a broad literature review on the energy consumption of blockchain systems. Then, we will present the existing work related to research work presented in this manuscript.

3.1 A literature review on the energy consumption of blockchain technologies

This section summarises current knowledge on the energy consumption of blockchain system, based on a literature review. The first part focuses on the energy footprint of blockchain mining. The second part focuses on studies made on other aspects of the blockchain ecosystem, such as alternatives to Proof-of-Work.

3.1.1 The energy footprint of Proof-of-Work mining

Evaluating the overall energy consumption of Bitcoin network.

Energy consumption is at the heart of proof of work. As mentioned in section 2.2.5, bitcoin mining consists in a competition between miners to find a value, called nonce. This competition is called *proof of work*. As this process is random, brute-force is the only tactic to use to find a correct node. As a consequence, high-end mining hardware give miners a competitive advantage in mining as they can provide a better hashrate (the number of hash computer per seconds). Optimised hardware also provide better performances (hashrate) per energy unit (joules) ratio. As the cost of electricity is the main expense required for mining, better hardware can increase the net profit of miners. Since the launch of the Bitcoin network, miners have shifted from CPU to more specific cards called ASIC (Application-specific integrated circuit).

O'Dwyer and Malone [78] modelled the overall energy consumption of Bitcoin mining. The energy consumed by the Bitcoin network for mining depends on block's difficulty. The higher the difficulty, the more hash the network needs to execute in order to find the good nonce value. Authors have determined that the number of hashes H needed to mine a block of difficulty D equals to:

$$H = D.2^{32} \tag{3.1}$$

As a result, the network energy consumption will depend on the number of hash generated per second (called the hash rate) that evolves with block difficulties.

Figure 3.1 illustrates the evolution of Bitcoin difficulty through time. We notice that its value has exploded since 2017. Since O'Dwyer and Malone published their paper (2014) mining difficulty has increased by a factor of over 16 000. Authors had estimated the energy required to processed transactions emitted in 2014. This estimation (P, in Watts) is based on Bitcoin network's hash rate (R, in Hash/s) and its mining efficiency (ϵ in Hash/Joules):

$$P = R/\epsilon \tag{3.2}$$

Depending on the hardware used by miners, authors claimed that the overall energy consumption induced by Bitcoin mining could be from 0.1 to 10GW. The authors concluded that this estimation was comparable to Ireland's national energy consumption (3GW at the time).

Since O'Dwyer and Malone [78], several estimations on the energy consumption of

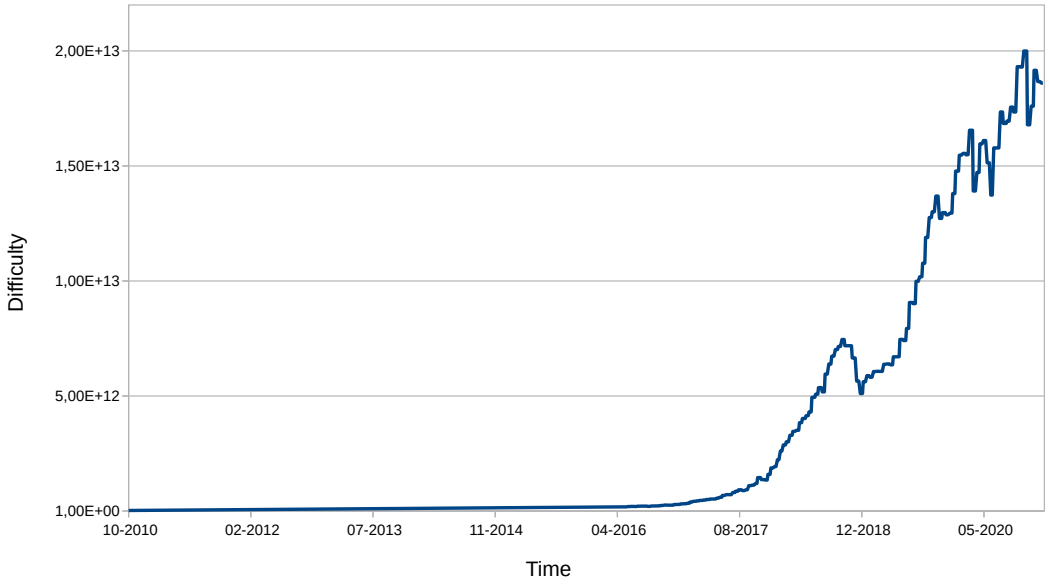

Figure 3.1 – Evolution of Bitcoin difficulty through time (data source: `https://www.blockchain.com`)

Paper	**Year of estimation**	**Estimation**
O'Dwyer & Malone [78]	2014	0.1 to 10 GW
McCook [68]	2015	120 MW
Magaki et al. [66]	2016	300 - 500 MW
Vranken [111]	2017	100 - 500 MW
Mora et al. [72]	2018	13GW
Digiconomist.net	2021	8.8GW

Table 3.1 – Estimation of Bitcoin network energy consumption in literature

Bitcoin network have been made. Table 3.1 illustrates these different evaluations. The key differences in these estimations are the date of estimation and the hypothesis made on the mining hardware used by Bitcoin miners. As seen before, the energy consumption of Bitcoin depends on its ever-growing mining difficulty and the energy efficiency of the mining hardware. Mora et al. [72], for instance, did not restrict their estimation to the latest mining hardware but also included older, less efficient hardware, resulting in a higher energy consumption estimation.

Regarding the impact of this energy usage, Mora *et al* [72] estimated that Bitcoin's mining competition could emit significant carbon emission. On its own, mining could push global warning above 2°C in just a few decades. They estimated that Bitcoin usage

led to the emission of 69 $MtCO_2e$ (Carbon Dioxide Equivalent) in 2017. To obtain this estimation, authors compiled a list of hardware suitable for mining and simulated the energy used for mining blocks emitted in 2017 (through O'Dwyer and Malone [78] equation between the number of hash needed to mine a block and its difficulty). The authors then inferred the CO_2e emission based on miner's location and the CO_2e emitted for electricity production in their country. By comparing the evolution of Bitcoin adoption with other technologies, they conclude that in a few decades the CO_2e emission of Bitcoin mining would be sufficient to induce a 2°C global warning. Those projections are however contested. For instance, Dittmar and Praktiknjo [23] highlight three issues: **1)** Mora *et al* have included legacy, energy-intensive hardware in their simulations; **2)** Estimations exceed others by a factor of 7 to 14; **3)** In the projection, the demand in energy implied in a few decades exceed the current evolution of world energy production.

Evolution of mining hardware and its energy efficiency

A core concept and the major energy consumption factor of Bitcoin is its Proof-of-Work mining mechanism. Financial rewards given to miners for each block validated give them an incentive to invest in powerful hardware. As electricity is the main cost for miners, the more energy-efficient their mining hardware is the more financially profitable it will be. As a result, the hardware used for Bitcoin mining as evolved over the year, from CPU to powerful ASIC cards.

This evolution of mining hardware as been detailed by Vranken [111] and Taylor [106]. Initially, CPU have been used for mining and could produce performances in the order of 10^{-1} / 10^{-2} MHash per Joule. In late 2010 came the first Bitcoin mining library compatible with GPU. GPU could offer better performances than CPU due to better optimisation and parallelism. As Taylor [106] highlights, the mining mechanism would typically be implemented with OpenCL whereas the Bitcoin protocol itself would be implemented in a high-level programming language such as Java. In mid 2011, miners switched to FPGAs. FPGA offers better performances in the nonce search (in the order of 10 MHash/Joules) as they are optimised on the hardware level. ASICs cards, arrived one year later, are the current hardware used for mining, offering much higher performance than FPGAs.

Table 3.2 highlights the evolution of the energy efficiency of mining hardware. Data presented in this table comes from a wiki that is a bit outdated [1] and lacks informations on the latest ASIC cards. For information, the ASIC S19 J Pro, currently produced by

1. At the time of writing, this article was last updated in early 2018.

	CPU	**GPU**	**FPGA**	**ASIC**
Min (MHash/J)	0.01	0.01	10.4	107
Max (MHash/J)	1.14	3.09	23.25	11 100
Average (MHash/J)	0.15	1.11	18.05	1854.19
Standard Deviation (MHash/J)	0.20	0.76	4.65	2694.54

Table 3.2 – Differences in energy efficiency between different mining hardware (Data source: `https://en.bitcoin.it/wiki/Mining_hardware_comparison`)

the Bitmain company, can compute 32 787 MHash per Joule.

As mining hardware become more and more energy-efficient, one can wonder if this evolution can stabilise or lower the rising Bitcoin energy consumption. Zade et al. [118] modelled future energy consumption of both Bitcoin and Ethereum depending on the underlying mining hardware. In their paper, they combined several scenarios on both the evolution of mining difficulty and evolution of mining hardware efficiency. The authors drew the following conclusions: between mining difficulty and hardware evolution, the main factor in the evolution of Bitcoin's energy consumption is its mining block difficulty. Hardware efficiency only have a limited impact in lowering Bitcoin's energy footprint: as it becomes more and more efficient, it keeps being outrun by an ever-increasing mining difficulty designed to maintain a constant block mining rate (around one block per ten minutes).

Making Proof-of-Work greener

As we have just seen, despite considerable improvement in mining hardware energy efficiency, Proof-of-Work still suffer from a massive energy consumption issue. Significant work has been made to make Proof-of-Work "greener". Contributions to this topic can be split into two categories:

— Improving performances of Proof-of-Work based blockchains,

— Replacing the mining puzzle by so-called "useful" work. In this case, the goal is not to reduce the energy consumption of the system, but to make use of this computation in order to produce so "useful" work.

Green-PoW [61] and Bitcoin-NG [37] are two examples of Proof-of-Work based blockchain designed for better performances.

Green-PoW aims to reduce the number of miners in competition. In this system, time is divided into epoch composed of two rounds (each round resulting in a block produced by

a miner). The difference between Green-PoW and Bitcoin is that, will Green-PoW, all the miners that managed to produce a block in the first round of each epoch will be allowed to compete in the second block. With Bitcoin, the miners who managed to mine a block but were not able to spread it faster than their competitors will see their effort wasted. With Green-PoW they gain access to the second mining-round. As a consequence, for each epoch, the second round only have a small subset of miners competing for a block mining. Authors estimate that this design can lead to a 50% reduction in energy consumption.

Bitcoin-NG takes a different approach in order to improve the performances of Proof-of-Work blockchains. Authors decoupled the transaction validations from the mining leader election scheme. Each time a miner validate a nonce, it can validate transactions until the next nonce validated by another miner. It results in a higher number of transactions validated.

As explained before, Proof-of-Work aims to provide a consensus system based on computational power. To spread their block and obtain its corresponding reward, miners compute hashes in order to solve a "puzzle". As it is, this puzzle can be seen as a massive loss of energy because it has no use outside the election of the leader. In the literature, several papers describe systems where the hash function at the heart of Proof-of-Work is replaced by computation considered "useful" [2].

Zhang et al [120] proposed REM (*Resource Efficient Mining*), a mining framework that aims to replace Proof-of-Work by a Proof-of-Useful-Work. In this consensus framework, clients submit computation tasks to blockchain agents. Blockchain agents increase their probability to mine a valid block by performing tasks. Here, nonce trials are replaced by tasks instructions. Blockchain agent's honesty is attested by a Trusted Execution Environment (TEE) called Intex Software Guard Extension (SGX). This model is only *partially decentralised* as it assumes trust into Intel (as the Hardware provider). REM can tolerate compromised SGX nodes and reject their blocks. The overall impact of tasks execution in an SGX environment is estimated to be at 5-15%.

Shoker [98] proposed to replace computation generated by Proof-of-Work by matrix multiplications through a consensus mechanism called Proof-of-Exercise. This consensus has a behaviour similar to REM [120] (restricted to matrix multiplications and without the use of a TEE): nodes (here called *employer*) submit matrices to be multiplied. Data need to be provided in a third-party hosting system. Nodes called *verifiers* are in charge of

2. In the following papers, computations are considered "useful" when their purpose is not restricted to the blockchain consensus mechanism.

result validation through a probabilistic verification scheme. A limitation of this system is that, once input data (matrix to be multiplied) are not available anymore (as matrix are stored in a third party storage), a proof cannot be verified. Moreover, with Proof-of-Work, nonce validation is trivial. With Proof-of-Exercise, tasks are submitted several times to different nodes to ensure solution validity. Finally, even if matrix multiplications can have a large range of use-cases (e.g., machine learning algorithms), Proof-of-Exercise is still restricted to a small subset of tasks.

Other examples of "useful" mining scheme exists such as Primecoin [56] that computes primer number instead of block hashes. In this case, "usefulness" can be seen as quite limited.

3.1.2 Alternatives to mining

We have just presented the current literature on the energy consumption of Proof-of-Work systems: its evaluation, the evolution of mining hardware and proposals for improvement. However, other blockchain consensus schemes exist (e.g., Proof-of-Stake).

Cole and Chang [14] modelled the energy consumption of different consensus algorithms in the context of IoT applications. Their study compared three algorithms: Proof-of-Work, the Ripple protocol and the Stellar protocol. As opposed to Proof-of-Work, neither Ripple and Stellar protocol embed any mining scheme. Their main observations are that transactions processed through the Ripple system consumed on average $5.71e^{-6}$kWh of electricity on each node, whereas those processed through the Stellar protocol consumed $1.91e^{-6}$kWh of electricity. Authors estimated that this cost could be linear with the number of nodes in the network but did not provide any arguments to support this hypothesis.

Tasca et al. [105] focused on modelling and comparing the energy consumption of five Proof-of-stake based blockchains: Ethereum 2.0, Algorand, Cardano, Polkadot, Tezos and Hedera. Whereas previous work often tried to estimate the energy consumption of whole blockchain networks, the authors aimed to give an estimation of the energy consumed by individual transactions. Their resulting model (see Figure 3.2) depends on the number of validators n_{val} in the network, their individual average energy consumption p and the overall throughput l of the network.

Based on historical data found online (for n_{val} and l) and on constructor data (for p), the authors have estimated and compared the average energy consumed by the processing of a transaction on each network. Their estimation have highlighted a certain disparity

$$f_{c_{tx}}(l) = \frac{n_{val} \times p}{l}$$

Figure 3.2 – Estimation of the energy consumption per transactions, by Tasca et al. [105]

between the different proof-of-stake network. For instance, considering a general purpose server (Dell PowerEdge R730), the authors found that the Polkadot network was 75 times more demanding than Algorand. Moreover, this study confirms that the energy consumption of Bitcoin far exceed that of studied proof-of-stake blockchains by several orders of magnitude. Finally, the authors highlighted that the underlying hardware had a significant impact on the energy consumed by blockchains.

Regarding hardware, Loghin et al. [65] measured the energy consumption of three different blockchain systems on different infrastructures in order to better understand blockchains' energy consumption on low-end nodes (such as Jetson TX2 and Raspberry Pi 3 boards). Their key conclusion is that high-end low-power nodes (such as Jetson TX2) offer the best energy/performances ratio. The high cost of Proof-of-Work blockchains makes them unusable on the cheapest low-power nodes, such as Raspberry Pi.

3.2 Evaluate the performances of blockchain systems

Behind the term *blockchain* lies a rich ecosystem of technologies. To ensure the proper evaluation of new contributions, researchers and developers needs experimentation tooling. We present and evaluate the functionalities of existing tools for the performances analysis of blockchains.

Blockbench [22] is an academic tool that aims to analyze private blockchains. Blockbench uses two workloads, YCSB [15] and Smallbank [7], to quantify the transaction rate, latency, scalability and failure resistance of three blockchain technologies (two implementations of Ethereum [45][81] and one of Hyperledger [39]). Deployment of blockchains to be tested is managed through bash scripts. Blockbench is able to collects metrics about performances (latency and throughput) of deployed blockchains but lacks metrics on the underlying system like CPU, memory or disk usage (important to consider the overall footprint of blockchain technologies).

Hyperledger Caliper [8] is a tool maintained by the Hyperledger Foundation. Hyperledger Caliper is well integrated with Hyperledger products and offers a complete exper-

imentation lifecycle. Hyperledger Caliper can monitor blockchain performances but also server metrics through Prometheus [88]. Unfortunately, it does not seem to include any network emulation, crucial for studies on the impact of network failure or latency on a blockchain. Moreover, Hyperledger Caliper does not seem to include any functionality for resources reservation on scientific testbeds like Grid'5000.

Boston Blockchain Benchmark (BBB) [80] is another benchmarking tool that enables the deployment and evaluation of blockchains on emulated hardware. BBB uses Mininet [70] to deploy the SUT. It enables a fine control on the hardware and network as both are emulated. BBB enables fine-grained network configuration with parameters like latency, packet loss and bandwidth limitations. BBB deploys the whole blockchain network on a single server: each peer is deployed on its own process. While this simplify experiments, hardware emulation can limit some experiments. For instance, evaluations on energy consumption can require physical energy monitoring. Having the whole blockchain network on a single server makes fine-grained energy monitoring more difficult.

DAGbench [25] is a benchmarking tool targeting *Directed Acyclic Graph* (DAG) based distributed ledgers. Authors have written their framework from scratch in Javascript. DAGBench currently support three DAG-based blockchains (but could be extended for more): IOTA [87], Nano [62] and Byteball [13]. Authors compare those three technologies on several criteria like latency, throughput and CPU consumption. The authors have implemented two workloads: one that transfer arbitrary data between two accounts and one that read data on the blockchain.

In chapter 4 we present BCTMark, a novel framework that aims to fill the gaps of previously mentioned tools. This framework enables researchers to manage the whole lifecycle of their experiments, from blockchain deployment on a physical infrastructure to load generation and metrics (e.g. performances-related metrics, server usage...) collections. BCTMark also enables network emulation (e.g. configurable latency between servers).

3.3 Analysing the uses and energy consumption of smart contracts

In this thesis, we focused most of our research efforts on smart contracts. In chapter 5 we present a study on the use and energy consumption of Ethereum contracts. A few existing paper aimed to better understand those two topics.

Pinna et al. [84] proposed an empirical study on 10 000 Ethereum smart contracts.

This study is made from a software engineering point of view, detailing metrics like EVM compiler version, number of lines of codes, number of declared contracts, and functions... The authors have reported eight key observations. For instance, newly deployed contracts often tend to use the last version of their programming language. This study provides information on how contracts are developed but does not include any energy footprint analysis.

Chen et al. [11] discuss the financial cost of under-optimized smart contracts in Ethereum. They identify several gas-costly patterns (e.g., the presence of "dead code") and quantify, in a set of smart contracts, the presence of those patterns. Based on those patterns, they proposed a tool (*GASPER*; **GAS**-costly **P**atterns check**ER)** that can discover those gas-costly patterns. We believe that such an approach could help to reduce the energy consumption of smart contract calls. Unfortunately, the authors do not quantify the overhead in gas of those patterns, preventing the estimation of potential benefits based on our model.

Feist et al. [41] proposed a static analysis framework for smart contracts named *Slither*. The proposed framework converts smart contracts written in Solidity to an intermediate representation better suited for analysis. Slither has been conceived for four use cases: 1) vulnerabilities detection, 2) code optimisation, 3) code understanding for developers, 4) assistance with code review through a provided API. The authors have compared the performances of each component of Slither with state-of-the-art tools, demonstrating good performances. Slither does not include any notion related to energy consumption at the moment.

A more comprehensive list of smart contract analysis tools has been compiled by Di Angelo and Salzer [21]. The authors have illustrated the key features of several academic and non-academic tools and their differences. The lack of any notion of energy seems to indicate that this issue has not been tackled before.

As mentioned, the energy cost of smart contracts have often been neglected. In section 5.3 we propose to measure and model the energy consumption of smart contracts. Regarding the usage made of smart contracts, a few studies picked a set of contracts and characterised their usage. The approach presented in section 5.2 is different, as we aimed to provide metrics on a year-long transaction dataset.

3.4 Data suppression on blockchains

In public blockchains like Ethereum, everyone can deploy new smart contracts. As the number of contracts on Ethereum keeps increasing, in chapter 6 we propose to study the consequences of such a growth Ethereum's network performances. As a potential solution to limit this impact, we propose a novel protocol to identify and delete unused contracts.

To the best of our knowledge, very few academic papers have addressed the issue of removing data on blockchains. Dennis et al. [20] proposed a formal analysis of a temporal "rolling" blockchain, in order to maintain a stable blockchain size. In their proposed model, transactions are stored for a pre-set period (older transactions are removed). In their paper, Dennis et al. considered a Bitcoin-like blockchain. Bitcoin differs from Ethereum at least in two points: 1) it does not have smart contracts, 2) it uses a different model (called *UTXO*) for transactions. With UTXO, coins have a transaction history: the balance of a given account is the sum of coins addressed to this account minus coins emitted from this account. Ethereum uses a different approach by updating different data structures (as detailed in subsection 2.2.2) through each transaction. The use of those different structures makes it more difficult to determine which data to delete with a "rolling" blockchain. Our protocol focuses on state trie and smart contract management. We consider both approaches to be complementary: the approach proposed by Dennis et al. could be used to manage transaction history, whereas ours could be used to manage smart contracts accounts.

We also find a few exchanges on related topics outside academia. In an article [3], Vitalik Buterin (co-founder of Ethereum) discuss the issue of state management in Ethereum [6]. He discusses several potential strategies for state expiry. Our approach can be seen as a mix between approaches here called *rent via time-to-live* (where users can pay to extend the lifetime of a state object) and *refresh by touching* (where a state object's lifetime is extended through interactions). Unfortunately, the article does not include any evaluation of any discussed approaches.

3.5 Conclusion

This section aimed to review the existing literature regarding the energy footprint of blockchains and smart contracts. While the energy consumption of different consensus

3. We decided to include the following reference for exhaustiveness and as its content helped us in our thoughts, but we remind readers that this does not constitute any peer-reviewed content.

algorithms has been investigated in several studies, that induced by smart contracts has often been neglected. We propose in the following chapter to contribute to the understanding of smart contract usage and footprint. To carry our experiments, we developed a novel framework that can be used to deploy and run benchmark on blockchains. Using this framework, we measured the energy consumption of smart contracts on Ethereum. Then, based on real historical data, we highlighted that a large part of Ethereum contracts were actually little used. To design lighter blockchains, we finally propose a protocol to identify and destroy unused contracts. As blockchains are designed to store data forever, the design of a mechanism that would enable the removal of existing data has (to the best of our knowledge) often been neglected.

Chapter 4

A new framework for benchmarking blockchain technologies

Over the last years, research activities on blockchain technologies have fairly increased. Since the first release of Bitcoin, many projects have emerged to create or improve blockchain features like privacy, while others propose to overcome technical limitations such as scalability and energy consumption. New proposals are often evaluated with ad hoc tools and experimental environments. Reproducibility and comparison of these new contributions with the state of the art of the blockchain technologies are therefore complicated. Only a few tools partially address the design of a generic benchmarking of blockchain technologies (e.g., load generation). In this chapter, we introduce BCTMark, a generic framework for benchmarking blockchain technologies on an emulated network in a reproducible way. To illustrate the portability of experiments using BCTMark, we have conducted some experiments on two different testbeds: a cluster of Dell PowerEdge R630 servers (Grid'5000) and one of Raspberry Pi 3+. Experiments have been conducted on three different blockchain systems (Ethereum Clique/Ethash and Hyperledger Fabric) to measure their CPU consumption and energy footprint for different numbers of clients.

Research presented in this chapter have resulted in a publication at the *17th International Conference on Computer Systems and Applications (AICCSA 2020).*

Contents

4.1 Introduction 58
4.2 Contribution to state of the art 60
4.3 BCTMark - Technical architecture & usage 61
4.3.1 Usage 62
4.3.2 Architecture 64
4.4 Validation experiments 66
4.4.1 Deployment of blockchains on two different testbeds 67
4.4.2 Comparison of CPU usage of three blockchain systems 68
4.4.3 Experiments Reproducibility 70
4.4.4 Performance analysis of Smart contracts 71
4.5 Conclusion 73

4.1 Introduction

Since their introduction in 2008 with Bitcoin [75], blockchain technologies have been widely developed. First used for crypto-currencies, blockchains are now being implemented in many cases: sharing of computing resources [119], decentralised social networks [101], government services [30], storage solutions [59, 103], energy trading [74, 69], ...

Despite the potential of blockchain technologies in many areas, technical limitations slow their development as a possible alternative to centralised services. For example, several issues dealing with their scalability [37, 19] or energy cost [111, 78] have been identified.

Several improvement proposals have been recently made to face those issues. We can cite the examples of the *reparameterisation*[1] proposals in the Bitcoin community (see BIP[2] 100 to 107), new consensus systems such as [95, 92] or the introduction of off-chain transactions systems like [86].

Those proposals have been mostly evaluated through debates (e.g., in the case of the BIPs) or have used ad hoc evaluations that are often not reproducible (i.e., cannot be run on systems other than the one they have been designed for). We argue that, to properly

1. Evolution of parameters like block size and emission rate
2. Bitcoin Improvement Proposal

compare the performances of several blockchain systems and quantify the contribution of new proposals regarding performance issues or functionality (e.g., fault tolerance), the blockchain community needs proper tooling for reproducible experiments.

This chapter presents a framework enabling reproducible research on the performances (latency, throughput, energy consumption, ...) of blockchain technologies. BCTMark (**B**lock**C**hain **T**echnologies Bench**mark**ing) is intended to be a framework which can be used to deploy, compare, and evaluate (through various scenarios) any blockchain on a large number of different infrastructures. This framework provides an abstraction of the underlying physical infrastructure and can, therefore, be used to deploy easily on any platform that supports the SSH protocol. To demonstrate this flexibility, we have deployed experiments on both a public research cluster (Grid'5000 [5]) with "classical servers" (Dell PowerEdge R630 servers) and a private "low-power" Raspberry-Pi cluster.

The author of [99] defines several criteria to define a "good" benchmark. We argue that BCTMark has features that cover each defined criteria:

— **Repeatable**: BCTMark can manage the whole lifecycle of experiments (resources reservation, deployment, load generation, metrics collection,...) and can be used to deploy the same experiment on different infrastructures. Experiments results are consistent across different run (see subsection 4.4.3).
— **Observable**: BCTMark embeds several components to observe both performances and impact of the system under test (CPU consumption, disk and memory usage, ...)
— **Portable**: BCTMark can be used to compare different blockchain systems or different versions of the same blockchains. Users can write a driver to use this solution to compare their new system to existing ones.
— **Easily presented**: BCTMark embeds a Grafana dashboard [48] that can be used to present the results. Metrics are also stored in a time-series database.
— **Realistic**: Network capacities (bandwidth, latency, packet loss, ...) can be described in the deployment topology to emulate real-world deployment.
— **Runnable**: BCTMark can manage the whole lifecycle of the experiment (from the resources reservations on a given testbed to the metrics collection of the system under test). It makes them easier to run: the same configuration deployment can be shared with other scientists, even on different testbeds. The deployment topology itself (number of peers, network partition and capacities, ...) can be easily described in YAML, a language commonly used for configuration.

To the best of our knowledge, only a few tools address the issue of blockchains benchmarking (see section 3.2). These existing frameworks, while promising, do not manage aspects necessary for rigorous benchmarking like environment deployment (improving reproducibility), collection of resources usage (e.g., CPU and memory consumption) and network emulation (crucial as, for blockchains, network issues have an impact on the diffusion of new blocks).

To sum up, our contribution results in the design and the development of a framework that can be used to create experiments on performance and functionality evaluation of blockchains systems. Thanks to the design and features provided by BCTMark, these experiments can be repeated in different environments (and therefore can be shared with the scientific community for peer evaluation) and can be run in a realistic environment thanks to network emulation functionalities.

4.2 Contribution to state of the art

We mentioned in section 3.2 existing tools regarding the performances analysis of blockchains. We will now present the differences between BCTMark and the existing state of the art.

With Blockbench [22], the deployment of blockchains to be tested is managed through bash scripts that do not offer abstractions over the targeted testbed. On the contrary, playbooks written in Ansible and deployed with BCTMark can be used to deploy an arbitrary number of peers on any testbed that supports SSH connections. BCTMark also provides the same abstraction over network, enabling scientists to express easily network constraints and topology. While Blockbench collects metrics about performances (latency and throughput), BCTMark can also collect both system metrics like CPU, memory or disk usage (important to consider the overall footprint of blockchain technologies) and functional metrics (e.g., number of connected peers). Finally, Blockbench only targets private blockchain whereas BCTMark also target public blockchains (as demonstrated in the experiments).

Hyperledger Caliper [8] offers a complete lifecycle similar to the one we introduced in subsection 4.3.1. Unlike BCTMark, it does not however include any network emulation nor functionality for resources reservation on scientific testbeds like Grid'5000.

The main difference between BCTMark and BBB [80] is that BBB deploys the whole blockchain network on a single server, with one process per peer. The experiment we

present in subsection 4.4.1 needs proper energy monitoring that would have been difficult to produce with a whole blockchain network on a single server.

BCTMark seeks to be a general-purpose blockchain benchmarking network that allows experiments to be deployed on real hardware and emulated network to be as close as possible to a real-world deployment. Main differences in terms of functionalities between those tools and BCTMark are summarized in Table 4.1 [3].

	Blockbench	**Hyperledger Caliper**	**BBB**	**DAGBench**	**BCTMark**
Targeted systems	Private Blockchains	Mainly Hyperledger systems	Private blockchains	DAG-Based blockchains	Every blockchain
Deployment management	No	Yes	Yes (but on a single server only)	No	Yes
Network emulation abstraction	No	No	Yes	No	Yes
Portability to new testbeds	N/A (do not manage deployment)	Yes (but no management of resources reservation)	Yes, as hardware is emulated (as long as Mininet can be installed)	N/A (do not manage deployment)	Yes (as long as testbed has SSH)
Metric collections	Yes (SUT)	Yes (SUT + testbed)	Yes (SUT)	Yes (SUT + testbed)	Yes (SUT + testbed)

Table 4.1 – Comparison of functionalities with the state of the art

4.3 BCTMark - Technical architecture & usage

This section presents BCTMark, our solution for benchmarking blockchain technologies. We first introduce how BCTMark can be used to run existing experiments and how developers/scientists can integrate new blockchain systems to be tested. Then, we detail its architecture and underlying components.

3. *SUT = System Under Test*

4.3.1 Usage

From a user's point of view, the workflow of an experiment performed with BCTMark proceeds as described in Figure 4.1.

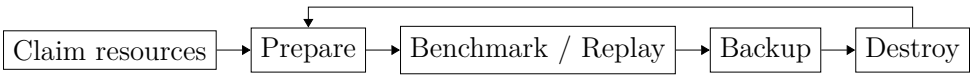

Figure 4.1 – The experiment workflow of BCTMark

The first step is to *claim* resources on which to deploy the experiment. BCTMark is intended to be portable to manage repeatable experiments. Experiments can be deployed on any infrastructure that supports SSH connections. Some research testbeds (like Grid'5000) require users to book resources before using them. This reservation phase can be addressed by BCTMark. As shown in Listing 4.1, the deployment topology can be described in a YAML file. This provided example can be used to deploy on a local device 1) an Ethereum network with one bootnode and two peers, 2) one benchmark worker (used to generate loads), 3) a "dashboard" server that hosts both the monitoring stack and the load generator master (that coordinate workers, see subsection 4.3.2 for details on load generation). In this case, the *claim* phase will only start the required virtual machines.

Once the infrastructure resources claimed, BCTMark can *prepare* the experiment by deploying the required components (i.e., download and install dependencies, copy configuration files...). For each *role* (see Listing 4.1), there is corresponding component to be deployed. The monitoring stack (*dashboard* role) and the benchmarking workers (*bench_worker* role) are common to many experiments. Users can define their roles to deploy their blockchain network. In the example in Listing 4.1, we need two roles to deploy an Ethereum network: bootnodes [4] and peers.

After deployment, users can run the benchmark themselves. BCTMark provides two possibilities to do this: an ad hoc load generation and a one based on previous traces (for more details on the implementation choices, readers may refer to subsection 4.3.2). Once the benchmark has ran, the results can be backed-up, and the environment destroyed/-cleaned for another experiment.

```
1 deployment:
2   vagrant:
3     backend: virtualbox
```

4. Bootnodes are peers that have an address known by everyone in the network. New peers can connect to those bootnodes to get the address of other peers in the network

```
4     box: generic/debian10
5     resources:
6       machines:
7         - roles: ["dashboard"]
8           flavour: tiny
9           number: 1
10        - roles: ["ethgethclique:bootnode"]
11          flavour: tiny
12          number: 1
13        - roles: ["ethgethclique:peer"]
14          flavour: tiny
15          number: 2
16        - roles: ["bench_worker"]
17          flavour: tiny
18          number: 1
```

Listing 4.1 – Configuration example for local deployment with Vagrant

From a developer's point of view, all the following necessary actions must be implemented to integrate a new blockchain to be tested:

— *Deployment*: write a new Ansible playbook (cf. subsection 4.3.2) that specify how to deploy, backup and delete the system;

— *Metric collection*: write a Telegraf plugin (cf. subsection 4.3.2) to gather system-specific metrics (e.g., block emission rate) if not already available through HTTP web services (BCTMark can collect metrics exposed at given HTTP endpoint);

— *Adhoc Load generation*: write functions that correspond to an interaction one can have with the system (e.g., how to send a transaction, how to call a smart contract, ...);

— *Reproducible Load generation*: implement functions to backup transactions (and serialize those) and functions to replay a given serialized transaction.

A developer/researcher would benefit from the design of BCTMark as a framework to easily integrate its new blockchain technology to be tested. Indeed, BCTMark already provides:

— *Deployment*: portability of deployment on several testbeds that support SSH;

— *Network emulation*: latency, bandwidth limits, ...;

— *Metric collection*: collection of metrics related to the infrastructure (e.g., CPU usage);

— *Load generation*: distribution of the load to generate among workers.

Only specific interactions with the blockchain to be tested need to be implemented.

4.3.2 Architecture

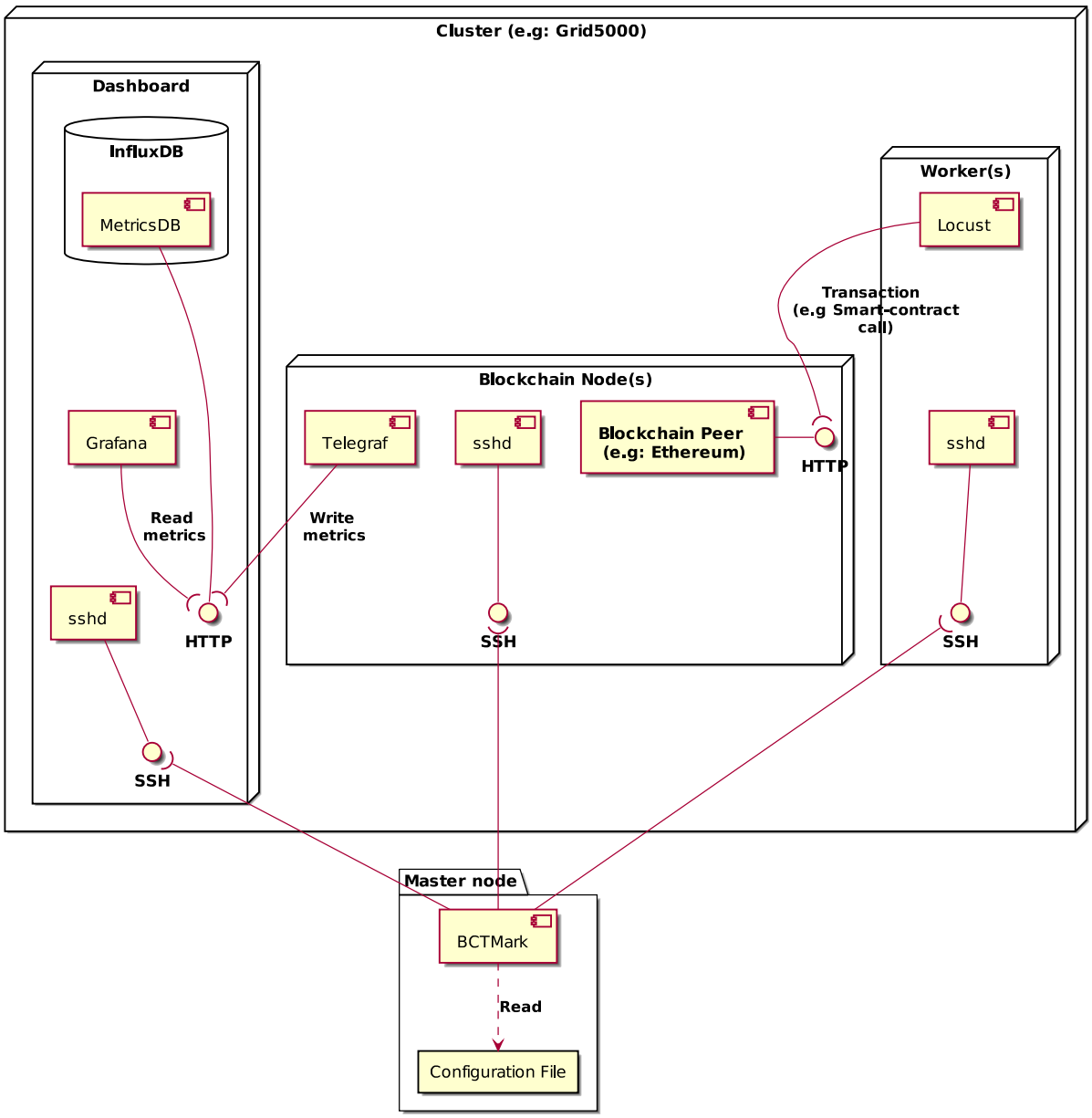

Figure 4.2 – BCTMark architecture

To avoid reinventing the wheel, BCTMark is based on the state-of-the-art, industry-proven tools. Altogether they empower researchers, allowing them to provision computing

resources, deploy blockchain peers, generate load (based on an history to reproduce or according to a given scenario), and collect metrics relating to peers' performance and energy consumption. The architecture of BCTMark is illustrated in Figure 4.2.

Deployment. BCTMark can deploy the entire experiment stack: system under test, monitoring system, and load generators.

Deployment does not require any agent installation on the machines. They are managed through SSH. A *playbook* defines the configuration to be deployed, which takes the form of configuration files in YAML format. Those configuration files make it possible to specify the desired deployment in a explicit, documented, and repeatable way. BCTMark also provides an abstraction layer of the underlying infrastructure. The deployment topology can be described in a high-level point of view, portable on different testbeds. That makes experiments portable on various infrastructures such as Vagrant (local deployment), Grid'5000 and Chameleon.

To manage deployment, BCTMark uses EnosLib [31] (an open-source library to build experimental frameworks) and Ansible [1] (a software that allows to manage deployment of configuration on a cluster). These two components enable self-describing, reproducible deployments.

Metrics management. Metrics about the server (CPU, memory consumption, HDD usage, ...) and blockchains (number of blocks produced, hashrate, ...) are collected, stored in time-series and displayed by Telegraf [107], InfluxDB [52] and Grafana [48].

Telegraf natively allows the collection of server metrics through many plugins written in the Go programming language. New ones can be developed to manage the collection of data on deployed blockchain peers. Current experiments on Ethereum deployment use the HTTP plugin from Telegraf to collect metrics through the Ethereum HTTP API.

Network Emulation. One strength of BCTMark is its ability to describe simply the desired network to emulate. Users can describe in the YAML deployment configuration file several groups of peers and emulate any desired network condition between them. The current characteristics of the network that can be emulated are the percentage of packet loss, network delay, and network rate (i.e., bandwidth). A use case of this feature could be to study the effect of a sudden network partitioning or merge on a blockchain system. Under the hood, BCTMark uses EnosLib that applies the desired network rules using the Linux command TC.

Load generation. BCTMark supports two ways to generate workloads [5]: an *ad hoc*

5. By *workload*, we mean *transactions* to be processed by the system under test

load generation (based on Python scripts) and a load generation based on an history. The first one uses Locust [64], a load generator written in Python. The user needs to specify, through Python methods, any interaction a user can have with the system under test (e.g., sending a transaction to someone or deploying/calling a smart contract). Locust will then use those methods to generate random loads.

The second way to generate load is based on a provided history. BCTMark can extract the history of a peer in the system and serialize it in a YAML file containing all the transactions. To reproduce the history, it can split the transactions between different workers, create the number of accounts needed to replay it, and let the workers re-run the transactions. This way, we can aim to replay transactions issued from the *mainnet* of a targeted blockchain system.

Energy consumption. BCTMark does not embed any energy monitoring tools. However, as it enables the deployment of experiments on any kind of testbeds, it can be used to deploy systems on clusters where the energy consumption is monitored. We have already tried this by deploying experiments on the SeDuCe [82] cluster (see subsection 4.4.1). It is part of the Grid'5000 testbed and is monitored with both energy and thermal sensors.

4.4 Validation experiments

In this section, we illustrate BCTMark's capabilities through three experiments. The first one demonstrates its capacity to deploy experiments on different testbeds, the second one its capacity to compare two blockchain systems and the third one, its usage for smart contract performance evaluation. Those experiments use two different testbeds (both having power measuring capacities):

1. A Raspberry-pi 3+ cluster. Each node has a quadcore Cortex-A53 ARMv7 CPU and 1GB of RAM.
2. Grid'5000 [5] Ecotype: A Dell PowerEdge R630 cluster. Each node has two Intel Xeon E5-2630L v4 (Broadwell, 1.80GHz, 10 cores/CPU) CPU and 128 GiB of RAM. Grid'5000 is a large scale public research testbed containing several clusters. Ecotype is one of those clusters, located in Nantes (France).

We evaluated three blockchain systems:

1. Ethereum Ethash, an implementation of the Proof of Work (PoW) system of Ethereum. It is the default implementation of Ethereum, used in the context of

a public blockchain. In this system, every peer can actively participate to block mining.

2. Ethereum Clique, an implementation of the Proof of Authority (PoA) system of Ethereum. PoA is used in the context of a private blockchain. In this system, pre-selected and identified peers can validate blocks one at a time. It does not involve any mining.
3. Hyperledger Fabric. It is also intended for private blockchain. Peers submit transactions to special peers called *orderers*. Orderers are in charge of the ordering process of transactions. Hyperledger Fabric uses a voting-based consensus protocol.

4.4.1 Deployment of blockchains on two different testbeds

This experiment illustrates the capabilities of BCTMark to deploy blockchains on different testbeds. We have deployed Ethereum Clique on both Raspberry Pi and Ecotype cluster under three scenarios. The IDLE scenario does not include any load generation. Peers just generate and share empty blocks. The two other scenarios include a load generation of 5 and 50 transactions per second. Load is generated by separated workers and spread randomly across peers. For both experiments, we deployed 12 peers and 6 load generator workers.

Results are presented in Figure 4.3. The bar plotted on the graph corresponds to the average power usage of every machines in the cluster. The error bar illustrates the standard deviation of power usage.

Those two platforms have different power draw. Power usage on the Dell servers goes from 130.4 to 131.54 watts (0.7% increase) whereas power usage on the Raspberry Pi platform goes from 3.4 to 5.2 watts (44% increase). This result was expected as Raspberry Pi are much more limited than classical "high performances" Dell servers. This experiment however illustrates that non-mining chains can be installed on low-power platforms like Raspberry Pi. This can be useful in the context of the development of blockchains in IoT / Edge computing. In the context of research on energy consumption, low-power platforms can be useful to illustrate subtle differences in the consumption.

We can, however, note that this conclusion may not be the same for mining systems such as Ethereum Ethash. Indeed, we could not install Ethash on our Raspberry Pi platform due to shortage in memory. The algorithm used by Ethereum Ethash for mining is memory intensive and therefore not suited for low-power platforms with not enough

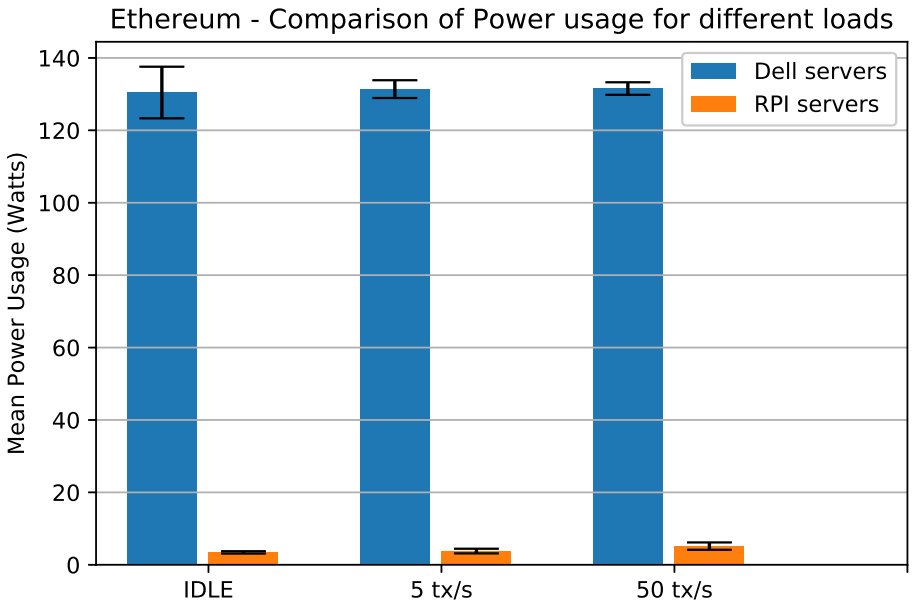

Figure 4.3 – Comparison of Power Usage for different loads

RAM. A solution for this issue could be to set-up both high-performance nodes dedicated to mining and low-power nodes that would only broadcast transactions to the miner's network.

4.4.2 Comparison of CPU usage of three blockchain systems

This experiment aims to illustrate the capabilities of BCTMark to deploy different blockchain systems. We deployed Hyperledger Fabric, Ethereum Ethash, and Ethereum Clique on the Ecotype cluster under four scenarios: IDLE (no-load generation) and load generation of 5, 50, and 200 transactions per second. The deployed network is composed of a network of 39 peers and three load generator workers. Figure 4.4 illustrates this experiment. The bar corresponds to the average CPU usage across all machines, whereas the error bar goes from the 10th quartile to the 90th quartile.

We can first notice that the CPU consumption of the Ethash system exceeds the CPU usage of the two others. Moreover, in this deployment, peers only mine blocks using one thread. It could be possible to dedicate more resources for mining, increasing the CPU consumption furthermore. The other two systems have non-mining consensus systems, decreasing the amount of computation needed to secure the network.

The CPU usage of non-mining systems are also more stable than the Ethash sys-

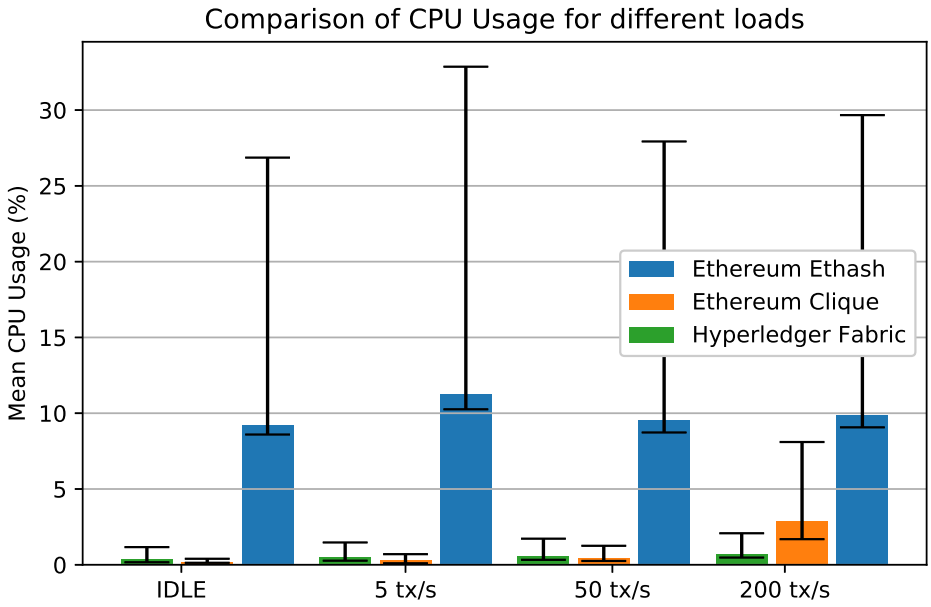

Figure 4.4 – Comparison of CPU Usage for different loads

tem. Figure 4.5 illustrates the evolution of the CPU usage for Ethash peers during the "200 transactions per second" scenario [6]. The spike at the beginning of the experiment, reaching almost 100% CPU, is due to the construction of the data structure needed by peers to start mining. We can also see that, after this spike, the CPU usage increases over time. This increase may be due to the evolution of the difficulty in mining resulting from the mining competition between peers.

On the other hand, in the first three scenarios, the CPU consumption of the two private blockchains is roughly the same. However, at 200 transactions per second, the CPU consumption of the Ethereum Clique network increases from 0.3% to 2.9%. This increase suggests that Hyperledger Fabric could have better performances in the context of a private blockchain. These results about private blockchains are consistent with those shown in the Blockbench. paper [22].

6. CPU usage values seem to differ from ones in Figure 4.4 but this visual effect is due to 1) high variance in data and 2) high density in data points that hides lowest values. We can notice the high variance on Figure 4.4.

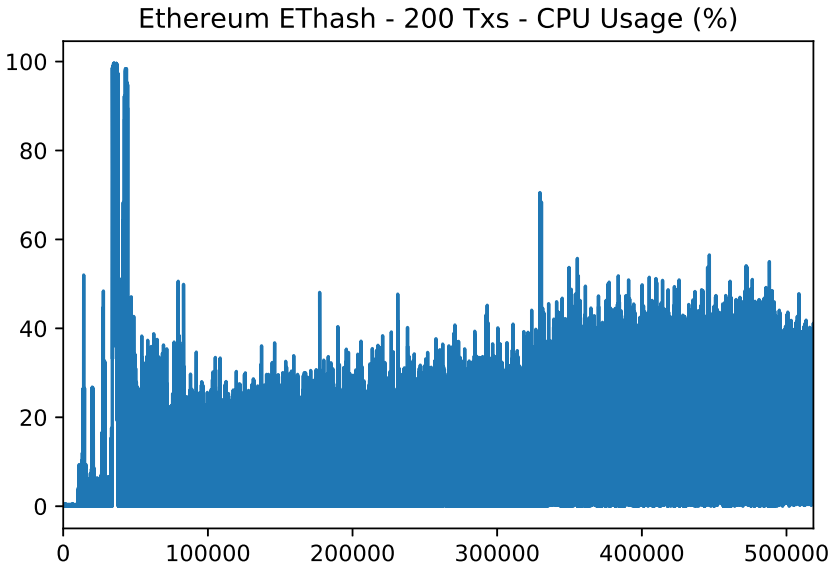

Figure 4.5 – Evolution of Ethash CPU usage for 200 Txs

4.4.3 Experiments Reproducibility

One of the goals behind BCTMark was to enforce reproducibility on blockchain experiments. Reproducibility means that running experiments several times (in similar conditions) should give coherent results. The goal of this section is to illustrate how experiments made with BCTMark can be reproduced.

We reproduced the experiments done in subsection 4.4.2 on Ethereum Clique and Ethash to have data on both public and private blockchain systems (readers can refer to this section to read about the infrastructure used and the deployment topology). Each of the four scenarios has been run six times. For every run, we have recorded the average CPU usage across all machines. The data presented in Table 4.2 illustrate the differences in the results we obtained. For instance, the min column illustrates the min CPU average across the six runs. Experiments should show consistent results to be considered reproducible.

These results show that we obtained few differences between the six runs. The standard deviation (column 'Std') remains low across all scenarios. This small difference in results leads us to believe that experiments with BCTMark should produce consistent results. Having exactly the same deployment topology with the same configuration is, in our opinion, the main factor explaining these consistent results. BCTMark allows researchers to share experiments that can be run in the same way by other peers in their community.

Deployment	Min	Max	Mean	Std
Ethereum Clique IDLE	0.113	0.117	0.115	0.002
Ethereum Clique 5 Txs	0.138	0.155	0.145	0.007
Ethereum Clique 50 Txs	0.463	0.526	0.494	0.028
Ethereum Clique 200 Txs	1.682	3.686	2.185	0.843
Ethereum Ethash IDLE	8.751	9.574	9.158	0.304
Ethereum Ethash 5 Txs	9.137	10.169	9.542	0.410
Ethereum Ethash 50 Txs	9.192	11.012	9.945	0.821
Ethereum Ethash 200 Txs	8.934	10.621	9.584	0.630

Table 4.2 – Reproducibility across six runs

4.4.4 Performance analysis of Smart contracts

The experiment, presented in Figure 4.6, is intended to illustrate the capabilities of BCTMark for performance analysis of software developed for blockchains. As explained in subsection 2.2.6, blockchains like Ethereum enable developers to write applications through smart contracts. On Ethereum, each call to a smart contract requires a "fee" related to its cost in gas. Gas is a unit related to the computational cost of each instruction in a contract. The more computation there is in a contract, the more expensive for end-users it will be. Moreover, in each mined block, there can be a limit to the sum of each transaction's cost in gas. As a result, there is an incentive for smart contract developers to control their contract's cost in gas.

To illustrate how implementation design and details can impact the cost of a smart contract, we implemented three classical sorting algorithms: Quicksort, Bubblesort, and Mergesort. Listing 4.2 illustrates the implementation of Quicksort.

We then have deployed those contracts on a four nodes Ethereum network and generated calls to those contracts. We have measured the cost in gas for each call. For each algorithm, we have generated calls to its sorting function with a random array of integers.

Figure 4.6 illustrates the evolution of gas consumption depending on the size of the input array. We can see that the evolution of gas requirements are coherent with the complexity of those three algorithms. Quicksort and Mergesort have the same average complexity of $O(nlog(n))$, whereas Bubblesort has an average complexity of $O(n^2)$. This statement is reassuring as it tends to show that the EVM [7] has been correctly implemented. This simple example demonstrates that BCTMark can be used to study smart contracts' costs and that its implementation has an impact on smart contracts possibilities. For the

7. Ethereum Virtual Machine, the VM running smart contracts

same iso-functionality (here, sorting), the cost of calling the contract will differ depending on the underlying algorithm. For information, at the time this article was written, the Mergesort algorithm would have cost around $0.55 to sort 100 items (cost of ∼101659 gas). In contrast, the Bubblesort algorithm would have cost around $11.77 (cost of ∼2168761 gas) [8].

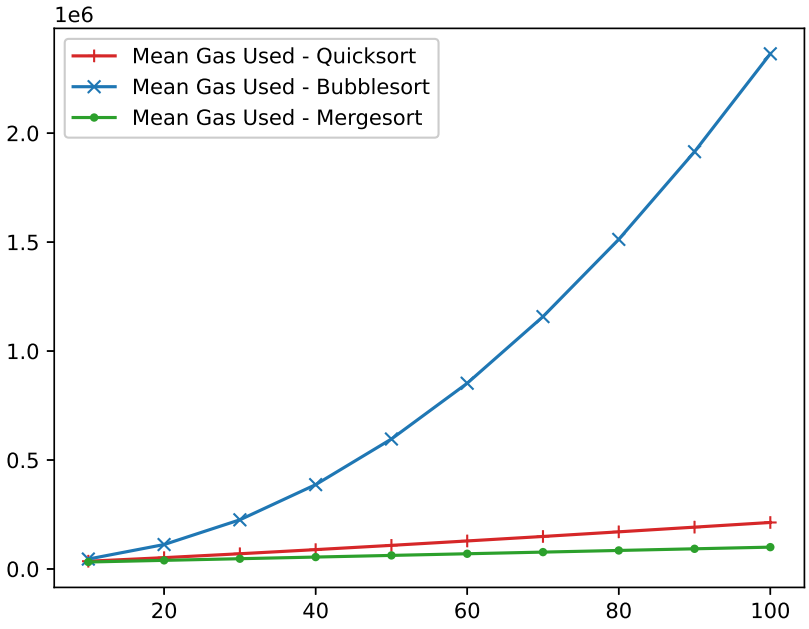

Figure 4.6 – Cost in gas of three smart contracts depending on provided input

8. Calculated on `https://ethgasstation.info` with average gas price

```
pragma solidity ^0.7.3;

contract quicksort {

    function sort(uint[] memory data) public returns(uint[] memory) {
       quickSort(data, int(0), int(data.length - 1));
       return data;
    }

    function quickSort(uint[] memory arr, int left, int right) internal{
        int i = left;
        int j = right;
        if(i==j) return;
        uint pivot = arr[uint(left + (right - left) / 2)];
        while (i <= j) {
            while (arr[uint(i)] < pivot) i++;
            while (pivot < arr[uint(j)]) j--;
            if (i <= j) {
                (arr[uint(i)], arr[uint(j)]) = (arr[uint(j)], arr[uint(i)]);
                i++;
                j--;
            }
        }
        if (left < j)
            quickSort(arr, left, j);
        if (i < right)
            quickSort(arr, i, right);
    }
}
```

Listing 4.2 – Implementation of Quicksort in Solidity

4.5 Conclusion

In this chapter, we introduced BCTMark, a framework for benchmarking blockchains. Existing tools, while promising, do not include important aspects of reproducible experiments on blockchain systems like network emulation or reproducible deployment. BCTMark aims to empower developers and researchers to create reproducible experiments on blockchain performances. For this purpose, BCTMark provide abstractions over testbeds

and network. To facilitate the development of benchmarks, BCTMark includes functionalities like load generation and metrics collection. To illustrate BCTMark's functionalities, we have run three experiments on three blockchains (Ethereum Ethash *vs* Clique and Hyperledger Fabric) and two testbeds (one Grid'5000 cluster and one Raspberry Pi cluster).

BCTMark has been used for *in situ* experiments in the following chapters. In chapter 5, we used BCTMark to evaluate the energy consumption of smart contracts depending on their gas input. In chapter 6, we used BCTMark to deploy a minimal Ethereum network in order to measure performances of individual nodes depending on the network *state* size.

BCTMark's code is open-source and accessible here: `https://gitlab.inria.fr/dsaingre/bctmark`.

Chapter 5

Understanding the usage and energy consumption of Ethereum smart contracts

Smart contracts play a fundamental role in recent blockchain-based decentralised applications. By allowing developers to write and deploy their own applications, they extend the possible use-cases of blockchains. This chapter is twofold. First, we analyse one-year of real-world Ethereum transactions to better understand how important are smart contracts in this community. Then, we focus on the energy footprint of smart contracts. Through *in situ* experiments, we build a model of the energy consumption of smart contracts depending on their cost in *gas*. Using this model, we try to estimate the energy consumed by all contracts ran on Ethereum during one year.

With our work, we aim to initiate studies that go beyond the measure of the energy consumption of blockchain consensus algorithm and take into account the complexity of modern blockchain-based decentralised applications.

Part of this chapter has been accepted for publication in *Cluster Computing* journal (to appear).

Contents

5.1 Introduction 76
5.2 Understanding the current usage of Ethereum smart contracts 77
5.2.1 Data extraction protocol 78
5.2.2 Smart contracts calls in Ethereum traffic 79
5.2.3 Gas consumption of Ethereum smart contracts 80
5.2.4 New smart contract deployment 83
5.2.5 Quantifying the number of unused smart contracts 84
5.3 Measuring and modeling the energy consumption of Ethereum smart contracts 87
5.3.1 Smart contracts footprint on non-Proof-of-Work systems 87
5.3.2 Deriving energy consumption from gas consumption 88
5.3.3 Ethereum smart contract execution model 91
5.3.4 The impact of replication on smart contracts execution cost . . 93
5.3.5 Limitations 94
5.4 Conclusion 95

5.1 Introduction

In 2008 Bitcoin [75] became the first large-scale currency that did not rely on any central authority. Seeing the potential of blockchain and the limitation of a cryptocurrency-only technology, Ethereum [43] introduced the concept of *smart contracts*. Those scripts, deployed and executed on the blockchain, enable the development of blockchain-based decentralised applications (Dapps). Since then, researchers and companies have proposed a wide range of use cases for Dapps like sharing of computing resources [119, 51], decentralised social networks [101], government services [30], storage solutions [59, 103] and energy trading [74, 69].

This chapter is twofold. First, we investigate how important smart contracts are in the Ethereum community five years after its first release. Based on the open data generated by Ethereum, this study analyses more than 247 million transactions emitted over one year to give researchers insights into the current usage of smart contracts. This analysis gives insights:

— On the proportion of smart contract calls compared to classical Ether (Ethereum's cryptocurrency) transfer, to understand if smart contracts (as a whole) are really used, or they are only tied up to experimental projects;
— On some functionalities of smart contracts implement. The goal is to understand not only if smart contracts are used, but also for what usage;
— On the amount of computation made by smart contracts, in order to quantify their complexity.

Second, leveraging BCTMark (described in chapter 4), we study in the energy consumption of decentralised, smart contract-based applications. Indeed, even if the energy consumption of Proof-of-Work have been well studied [78], novel consensus algorithms have emerged since. As those new algorithms grow in adoption, it becomes crucial to understand the energy consumption of other key aspects of modern blockchains systems. Therefore, section 5.3 proposes a novel method to measure and model the energy consumption of smart contracts deployed on Ethereum. Based on this model, we give insights into the energy consumed by real-world smart contracts over one year. Our key finding is that even if smart contract calls are relatively cheap on their own, their replication across the whole network generates an important energy cost.

5.2 Understanding the current usage of Ethereum smart contracts

This section aims to give insights on the current usage made of Ethereum's smart contracts. After an explanation of the data extraction protocol, we analyse one year of Ethereum transactions data. First, we quantify the importance of smart contracts in Ethereum's traffic to understand whether smart contracts are actually used. Second, we determine for how many computation smart contracts are used. This analysis will be based on the gas consumption of smart contracts. Third, we focus our analysis on smart contracts created that year in order to determine if new contracts are still being created and what volume of traffic is generated by these new contracts. Finally, we determine the number of deployed, yet unused, smart contract. In this last part, we illustrate that a large part of contracts deployed on Ethereum will lead to little or no usage. Chapter 6 will then investigate how this can impact Ethereum's individual nodes performances and a potential solution.

5.2.1 Data extraction protocol

As stated in section 2.1.2, Ethereum is a public blockchain. Everyone can become a participant in the network and gain access to all data in the blockchain.

To query these data (blocks, transactions, accounts...) in a convenient way, we extracted them in a relational database using *Ethereum-ETL* [36]. Ethereum-ETL is an open-source ETL (Extract-Transform-Load) developed to extract data from one Ethereum node (through its JSON-RPC API) to different outputs (including Google Cloud, CSV files or PostgreSQL). In our case, we stored the extracted data in an open-source Database Management System (DMS) called PostgreSQL. The data extraction was split in two phases.

First, we installed an Ethereum Node on a Dell PowerEdge R630 server (Intel Xeon E5-2630 v3 (Haswell, 2.40GHz, 2 CPUs/node, 8 cores/CPU), 128 GiB of RAM, 600GB HDD SATA Seagate ST600MM0006 storage). We chose one of the main open-source Ethereum implementations: OpenEthereum (v3.0.1-stable) [1]. We fully synchronized this node with the rest of the Ethereum Mainnet (The main/"production" Ethereum network). Processing took several days and resulted in the downloading of more than 10 million blocks. The second step was to extract and analyze recent data from this synchronized node, with Ethereum-ETL. Ethereum-ETL includes, in its extraction process, ERC20 and ERC721 smart contracts detection. To detect if a given contract implements one of those interfaces, Ethereum-ETL checks if each function signature defined in ERC20/ERC721 interfaces is present in the contract's binary code. This detection presents a limitation: contracts that would have modified these function signatures (e.g., by changing parameters or function names) cannot be counted as ERC20/ERC721 even if their behavior is similar. Also, smart contracts that would implement tokens without using one of those two standard interfaces cannot be detected. In conclusion, in the rest of this chapter, numbers given on the number of ERC20/ERC721 contracts can be considered as an underestimation on the real number of smart contracts implementing a token management mechanism.

We extracted a year of data, from September 2019 to August 2020. The resulting PostgreSQL database represented about 300 GB of data.

1. `https://github.com/openethereum`

5.2.2 Smart contracts calls in Ethereum traffic

Over the year, the Ethereum network managed an average of 24 million transactions per month, with values going from 17 millions transactions in January 2020 to 36 millions transactions in August 2020. The goal of this section is to quantify the importance of smart contract usage in Ethereum's traffic. In order to identify transactions that have been addressed to smart contracts, we check if each transaction's recipient address correspond to a smart contract address. The smart contract address list have been created during the Ethereum-ETL's extraction phase. Figure 5.1 compares the number of smart contracts calls and the number of "normal" Ether transfer transactions through time. Over the year, 140 million calls to smart contracts where emitted, representing on average 56% of Ethereum traffic.

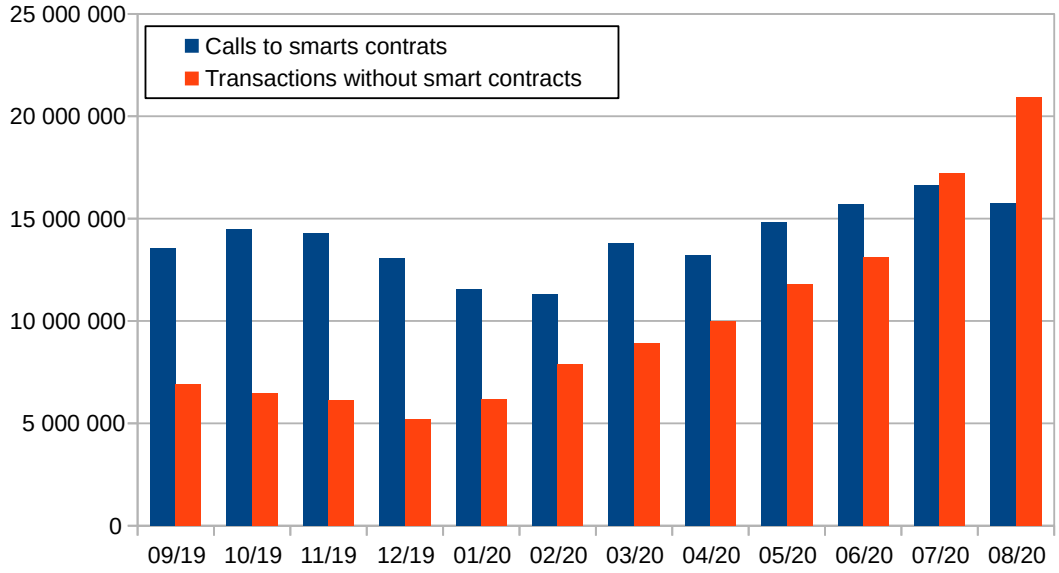

Figure 5.1 – Calls to smart contracts over time

As stated in section 2.2.6, one way to estimate the functionalities implemented by smart contracts is to detect whether they implement a known code interface. Two well-known and used interfaces are ERC20 and ERC721, that are both used to implement token management systems. Table 5.1 illustrates the proportion of ERC20 and ERC721 contracts in smart contracts that have been called over the year. Only a small minority of active smart contracts implements those interfaces. However, they represent a large part in smart contracts calls. Figure 5.2 compares the number of calls to ERC20 and ERC721 contracts and the number of calls to other smart contracts. ERC20 contracts represent

	Number	**% of Total**
ERC20	28 347	4.8
ERC721	680	0.1
Others	558 234	95.1
TOTAL	**587 261**	**100**

Table 5.1 – Proportion of ERC20 and ERC721 among active contracts

55% of smart contracts calls. ERC721 contracts represent 2% of smart contracts calls. At least 74.64% of Ethereum traffic is dedicated to token exchange, composed of 1) 43.41% of Ether, "classical", transactions, 2) 30.73% of ERC20 transactions (through smart contract calls) and 0.50% of ERC721 transactions (through smart contract calls). The remaining 25.35% are used to interact with smart contracts that does not implement ERC20 nor ERC721 interfaces.

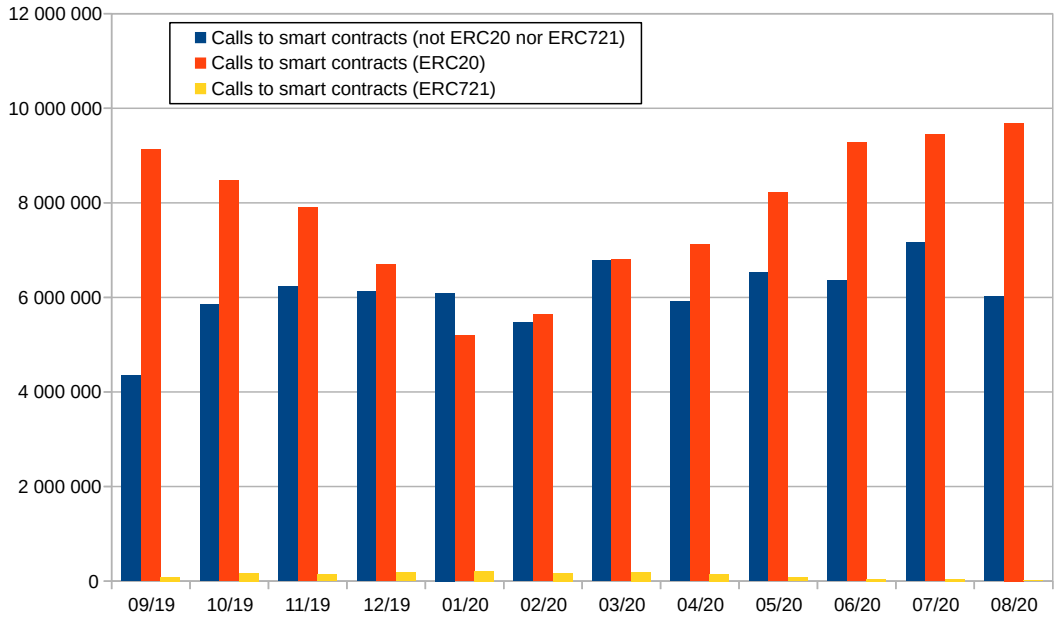

Figure 5.2 – Calls to smart contracts over time, by types

5.2.3 Gas consumption of Ethereum smart contracts

As explained in section 2.2.6, gas corresponds to the cost instructions in the Ethereum Virtual Machine. The gas cost of a smart contract function call will be equal to the sum of each instruction used during the function execution. The more complex a function is,

the more expensive its call will be for the caller. As gas cost is correlated to CPU usage, we can gain insight into how computational intensive this contract is by studying the gas consumption of an Ethereum smart contract.

Figure 5.3 displays the distribution of gas usage in all transactions in Ethereum. The minimum cost for an Ethereum transactions is 21 000 gas. Around 32% of all transactions consumes this amount of gas. 65% of all transactions consume at most twice this amount of gas. 90% of all transactions consume 128 852 gas or less, i.e., six times the amount of a "normal" transaction. That year, the maximum amount of gas used by one transaction was 12 million, i.e., 580 times more than a "normal" transaction. These high quantities of gas appear to be the result of errors in the execution of smart contracts (in Ethereum, a smart contract execution that raises an exception consumes all gas and result in a failed transaction). Indeed, of the 5% costliest transactions, 4% have been reverted due to an execution error. Among the other 95% transactions, the error rate is 3.7%, meaning that there is only a 0.3% difference between the most expensive transactions and the others. Execution error can be detected by looking at the execution status stored in the blockchain in the transaction receipt.

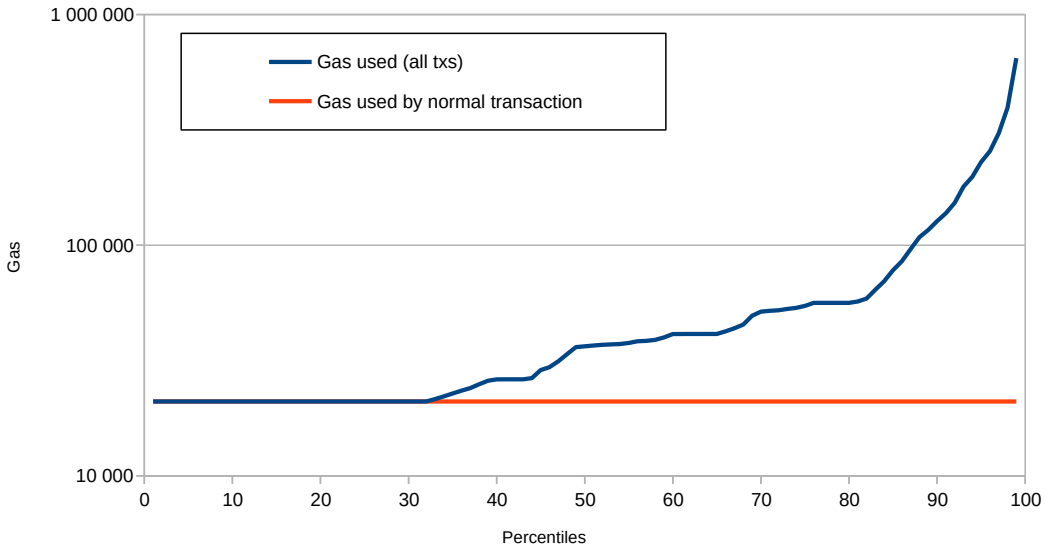

Figure 5.3 – Distribution gas usage per transactions (logarithmic scale)

Figure 5.4 illustrates the gas usage of smart contracts, depending on their type (ERC20, ERC721, other smart contracts). We have seen in subsection 5.2.2 that 75% of the Ethereum traffic were dedicated to tokens exchange and that 55% of smart contract calls

were addressed to ERC20 contracts. Smart contracts calls always have a gas cost superior to "normal" transactions. However, ERC20 contracts seem to have stable gas consumption. 98% of ERC20 contract calls cost at most 64 000 gas, i.e., three times more than a normal transactions. 63% of ERC20 calls did not cost more than twice the amount of gas of "normal" transactions. On the non-token smart contracts side, 78% of smart contracts calls did not cost more than the 98% ERC20 contracts calls mentioned above.

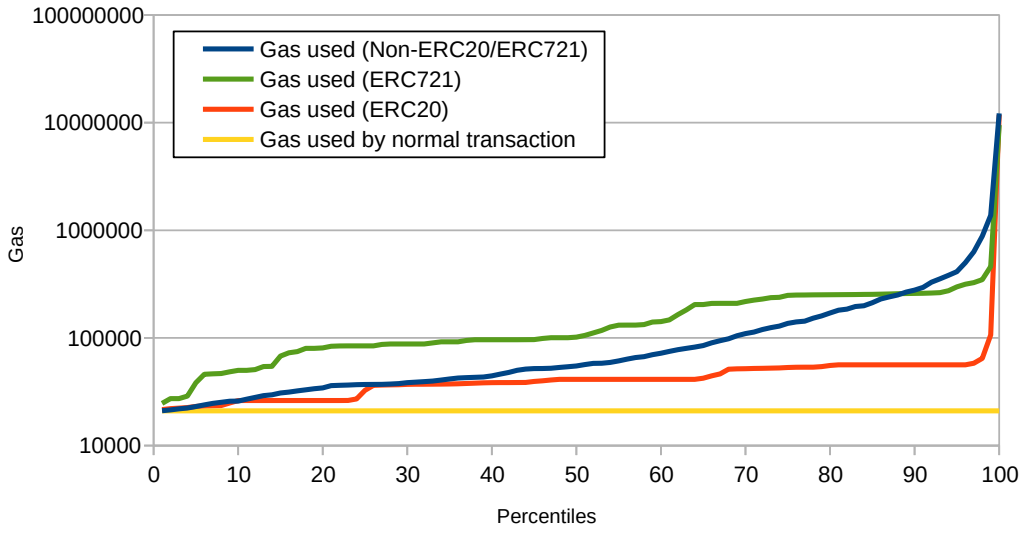

Figure 5.4 – Distribution gas usage per smart contract calls (logarithmic scale)

As detailed in section 2.2.6, the transaction fee paid by the smart contract caller is equal to the function's gas cost multiplied by the declared gas price. The caller can choose the gas price depending on the state of the market. A higher gas price will lead to a more expensive transaction fee but will serve as a better incentive for miners to include the transaction in their block (this implies that the transaction will be processed quicker). Figure 5.5 illustrates the evolution of gas price and the number of processed transactions over the year. Gas price is specified in Wei: one Wei is equal to 10^{-6} Ether.

We can first notice that gas prices have nearly always increased over the year. Its value has been multiplied by 5.4 in one year. This means that the same transaction would have cost 5.4 times more in August 2020 than in September 2019. Comparing the gas price with the number of transactions processed could partially explain its evolution. Pearson correlation coefficient gives us a correlation value of 0.9, meaning that gas price and the number of processed transactions may be linearly correlated. An explanation behind this is that, as the number of pending transactions increases, one should pay more fees to have

its transactions processed in a reasonable time.

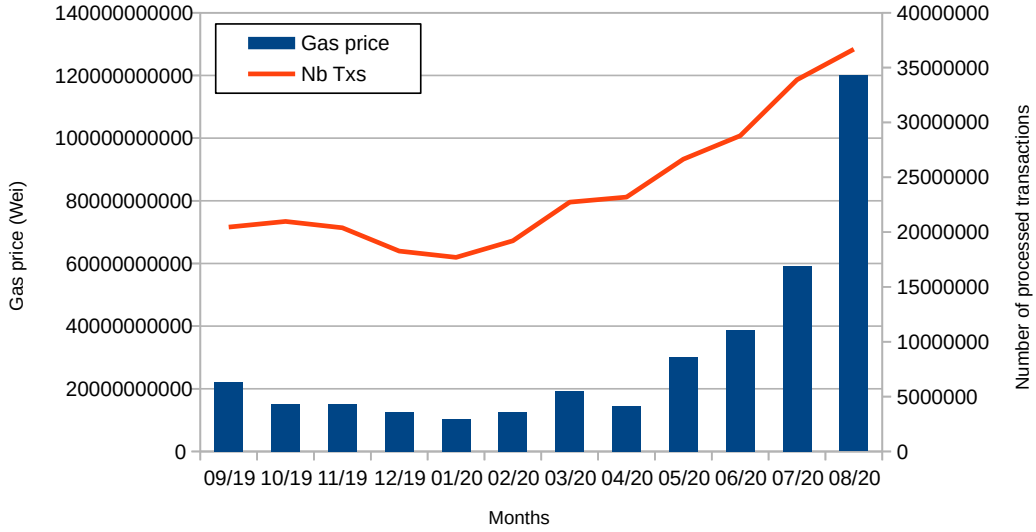

Figure 5.5 – Evolution of gas price and number of processed transactions during the year

Gas price represents a real incentive for developers to optimize their smart contracts in order for them to be called with reasonable fees. If we consider an Ether price of 596 USD (current price on the 1st December 2020), a gas price of 119 965 865 711 Wei, and a median gas cost specified in Figure 5.3, we obtain a median transaction fee of 1.5 USD for normal transactions, 2.9 USD for ERC20 contracts, 7.2 USD for ERC721 contracts and 3.9 USD for other contracts.

5.2.4 New smart contract deployment

As seen before, half of Ethereum traffic involves smart contract function calls, and half of Ethereum smart contract calls are used to manage tokens. This section focuses the analysis on the smart contracts deployed that year, to understand if any new contracts are still being deployed.

Figure 5.6 illustrates the number of smart contracts deployed over the year. In Ethereum, the deployment of a new smart contract is made by emitting a transaction embedding the contract byte-code and with no recipient. The cost of a smart contract deployment is roughly similar to the cost of a classical, Ether transfer, transaction. The receipt of this transaction will contain the new smart contract address. By checking the number of transactions used to deploy smart contracts we can see that, on average, 73 000 new con-

tracts have been deployed each month. Smart contract deployment represents only 0.3% of the total of transactions processed each month. Although it may seem impressive, this number must be put into perspective with the actual usage of these new smart contracts.

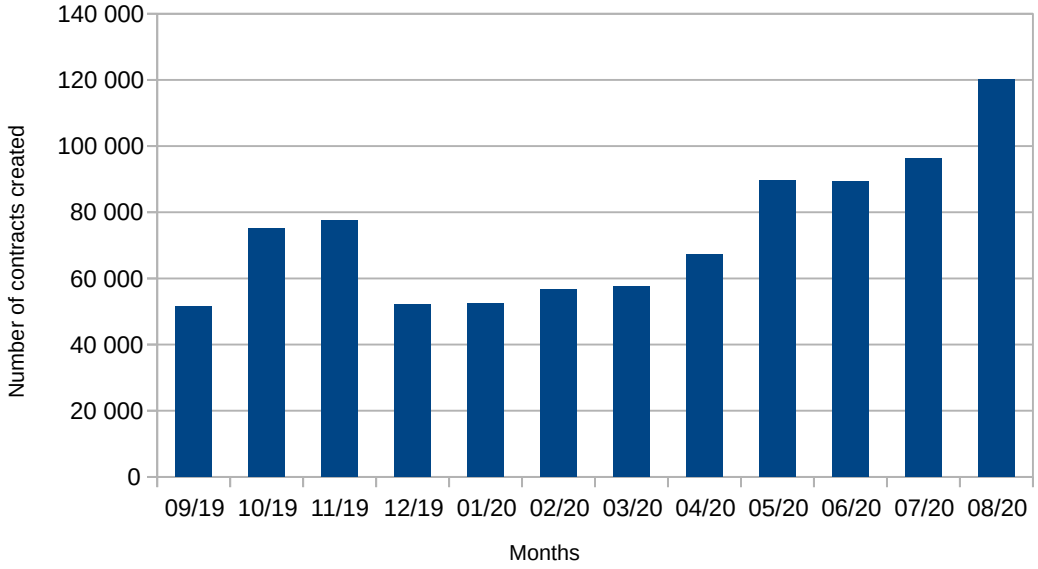

Figure 5.6 – Number of contracts created over time

5.2.5 Quantifying the number of unused smart contracts

Figure 5.7 and Figure 5.8 illustrate the distribution of the number of calls per "active" smart contracts [2], through percentiles (data has been divided into two figures for readability). Percentiles are used to illustrate frequency distribution. Among contracts that have been deployed that year, we can see that 48% have been called only once [3] and 88% have been called less than 10 times. Only a few contracts are responsible for most of the traffic. Indeed, 3236 contracts (1.2% of a total of 259 472 "active" contracts deployed that year) are the recipients of 95% contracts calls. If we do not limit the analysis to contracts that were deployed during the year and if we consider calls to every smart contract, the trends remain the same. 2851 contracts (0.7% of a total of 412 362 "active" contracts) are the recipients of 95% of all the smart contracts calls. Among those 2851 contracts,

2. We define "active" contracts as contracts that have been called at least once.
3. This value correspond to the 48th percentile in Figure 5.7

1300 (46%) are ERC20 contracts. The most called contract, Tether [4] (an ERC20 cryptocurrency), accumulates 29% of smart contracts calls. The second most used contracts is responsible for 2% of smart contracts calls.

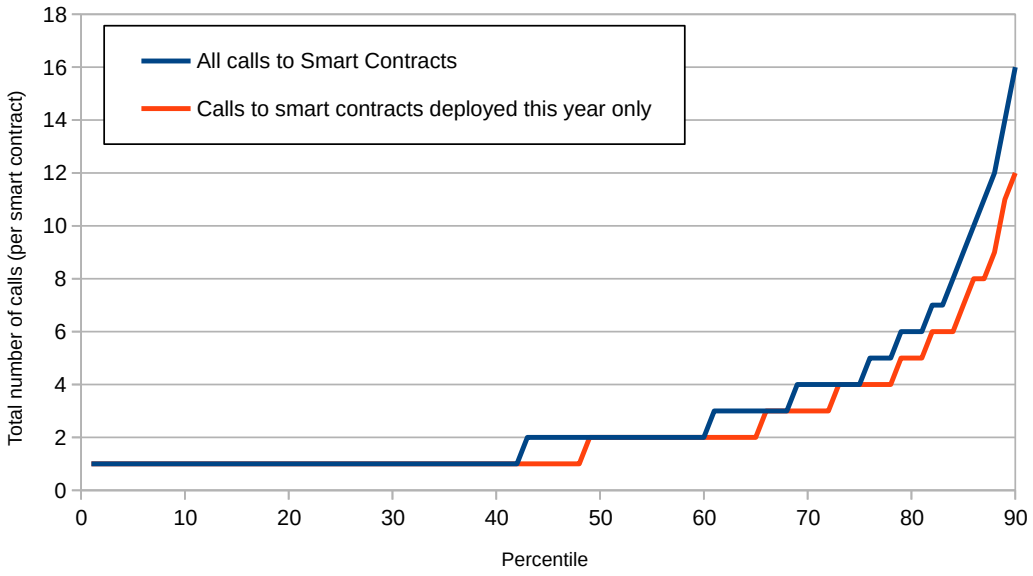

Figure 5.7 – Distribution of the number of calls per smart contracts - Lowest 90 percentiles

As we have seen, only a small minority of contracts are responsible for the traffic of Ethereum smart contracts. The majority of smart contracts will also stop to be used only a few days after their deployment. Figure 5.9 illustrates the evolution of the proportion of active contracts depending on the number of days since their deployment. We only consider contracts that have been deployed the first 8 months of our dataset, and we consider the first 4 months of "existence" of each contract. We can see that 50% of considered smart contracts won't be called again a week after their deployment. Only 16.6% of deployed smart contracts will be called after 4 months.

This section aimed to analyze the life cycle of newly deployed smart contracts (number of contracts deployed, number of calls per contracts, number of days until a contract stop being used...). However, "active" smart contracts only represent a minority of all contracts. Among all contracts deployed that year, only 259 472 (30%) of a total of 866 508 have been called at least one since their deployment. 607 036 (70%) contracts have been deployed without ever been called. Even if they have never been used, their bytecode is always stored into the blockchain!

4. https://tether.to/

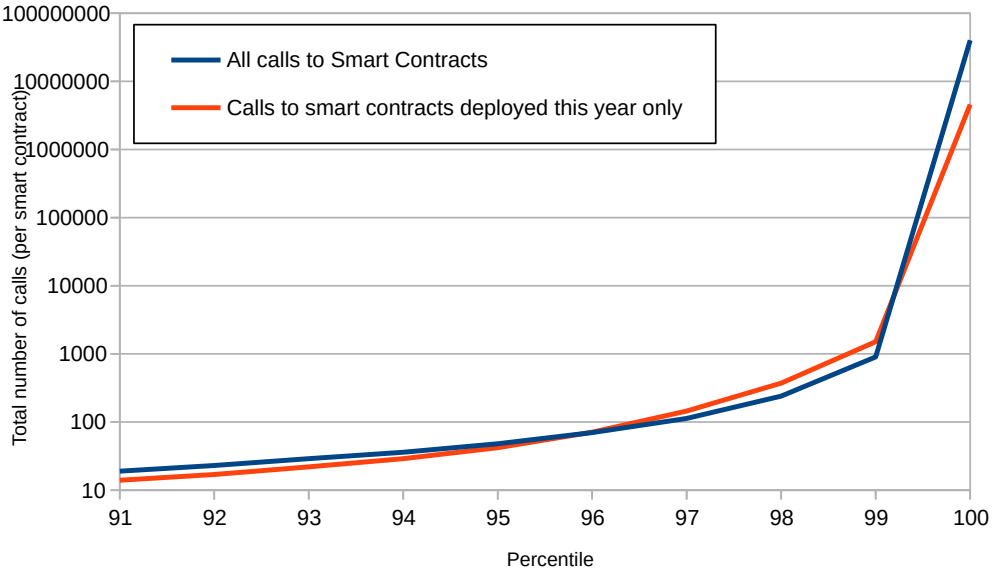

Figure 5.8 – Distribution of the number of calls per smart contracts - Highest 10 percentiles - Logarithmic scale

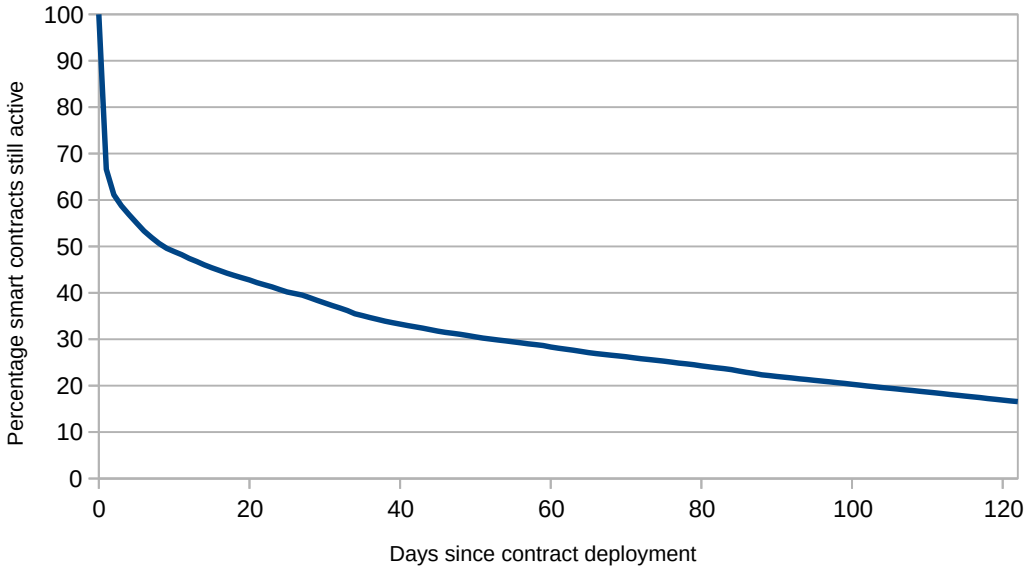

Figure 5.9 – Evolution of the percentage of active contracts depending on the number of days since their creation (Contracts deployed that year only)

5.3 Measuring and modeling the energy consumption of Ethereum smart contracts

5.3.1 Smart contracts footprint on non-Proof-of-Work systems

The high-energy footprint of Proof-of-Work-based blockchain systems has been established several times [78, 68, 111]. However, we did not find any studies focused on the energy consumption of smart contracts. We aim to illustrate here, through experimentation, how the energy consumption of smart contracts execution will become more significant in non-Proof-of-Work systems.

To do this, we deployed, with BCTMark, a private Ethereum network on a cluster of power-monitored servers [5]. This cluster of six Dell PowerEdge R640 servers, is equipped with Intel Xeon Gold 5220 18 cores CPU, 96 GiB of memory, 480 GB SSD SATA Micron MTFDDAK480TDN, and 25 Gbps Ethernet connection. Ethereum's peers have been deployed on five servers. The remaining server was used to host the monitoring dashboard and database. To illustrate the difference in energy consumption between running a smart contracts on a Proof-of-Work system and running the same contract on a Proof-of-Authority system, we deployed the Quicksort contract detailed in subsection 4.4.4. For each system (Ethereum Proof-of-Work and Proof-of-Authority), we studied the energy consumption on two cases. In the first, called IDLE, peers are mining empty blocks (no transactions are emitted). In the second, we add a constant load generation consisting of contract calls with a random array of size 1000 to be sorted.

Figure 5.10 compares the Ethereum Proof-of-Work system's power usage with and without (IDLE) contract calls. In both cases, we can notice a succession of plateaus that corresponds to the mining process's energy consumption. The IDLE network consumes on average 156 Watts, whereas the network executing contract calls consumes on average 160 Watts. The constant execution of contracts represents a 2% increase in its energy consumption.

Figure 5.11 illustrates the same deployment on an Ethereum network using Proof-of-Authority. The average power usage on the IDLE system is 63.9 Watts, which represents 40% of the power usage of our IDLE Proof-of-Work system. Running calls of our smart contract raise the power usage of our system to an average of 72.1 Watts. This represents an increase of 13.9%. We observe spikes in the power consumption in the deployment with

5. Each server is equiped with a sensor that measure its energy consumption

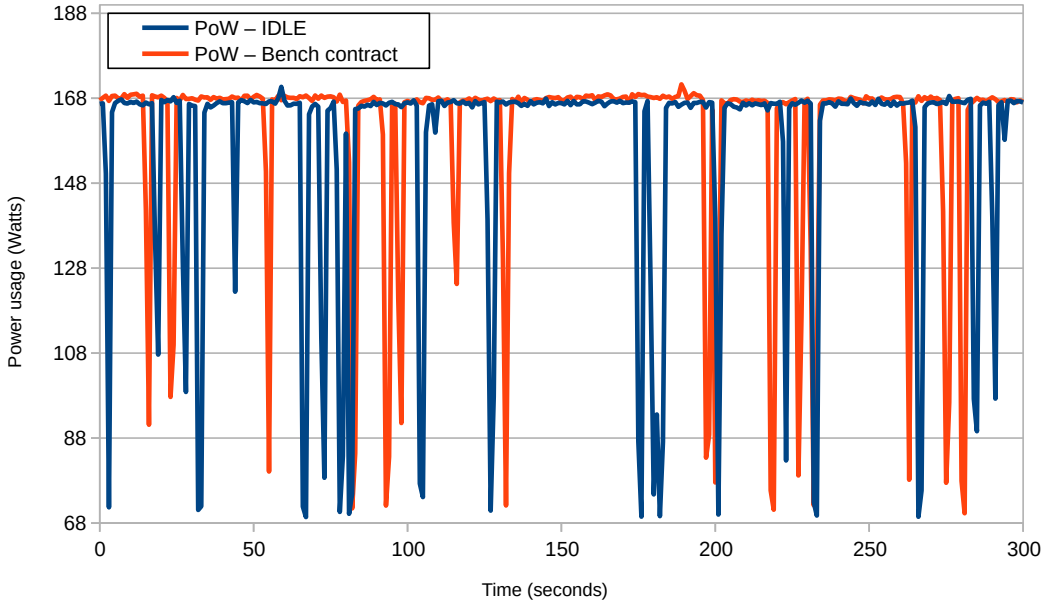

Figure 5.10 – Comparison power usage with and without contract execution on Ethereum PoW

contract executions. These spikes are due to the processing of those smart contract calls. No spikes are visible with the Proof-of-Work engine, as the overhead in power consumption is too small there. In addition, no plateau is visible in the Proof-of-Authority deployments because this mechanism does not involve any mining for securing the blockchain.

From these two experiments, we can conclude that smart contracts execution implies more overhead, in proportion, in systems without Proof-of-Work. As the adoption of non-mining blockchain systems will rise [6], one of the main energy-consuming parts of blockchains will become the actual processing of smart contract calls and transactions. That is why we estimate that research will focus on reducing the energy footprint of transactions and smart contract execution. Once we have stated this fact, we should now wonder what the actual energy consumed by smart contracts is.

5.3.2 Deriving energy consumption from gas consumption

We believe that research on smart contracts' energy efficiency presents two benefits: reducing overall blockchain energy consumption and reducing the financial cost implied by

6. In fact, the second version of Ethereum, in development, will include a Proof-of-Stake engine that does not include any mining.

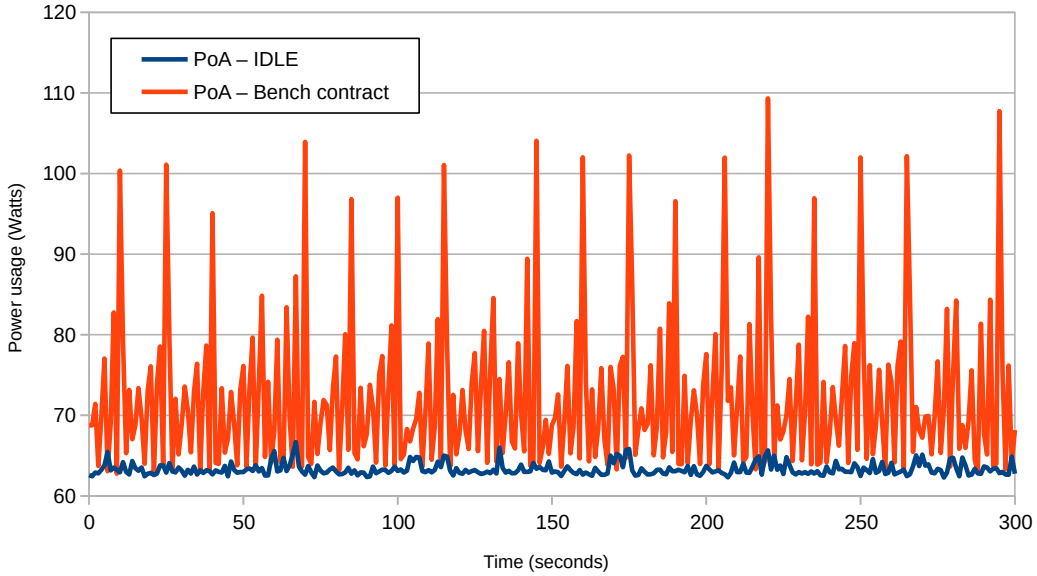

Figure 5.11 – Comparison power usage with and without contract execution on Ethereum PoS

smart contracts transaction fees. Indeed, the gas consumption of a smart contract function call depends on its complexity. The more computation it will do, the more (financially) costly it will be. This section presents a simple way to estimate a smart contract's energy consumption based on its gas consumption. This estimation is a two-step process. First, we need to calibrate a "gas to power" model on a given infrastructure. Then, we can use this model to deduce the energy needed to execute smart contracts.

In order to use this model on real Ethereum data and obtain an estimation of the energy used by real-world smart contract executions, we deployed an Ethereum node on a server and connected this node to the public Ethereum network. We were able to download all transactions emitted and validated during one year, from September 2019 to August 2020. The resulting dataset contains over 247 million transactions, including 140 million smart contract calls.

Our experiments use the same cluster as the one described in subsection 5.3.1. On this cluster, two smart contracts were deployed and executed to gather metrics to calibrate our model. These two smart contracts consist implement a Quicksort algorithm and a Mergesort algorithm (both are the same used in subsection 4.4.4). We evaluated the energy consumption of these two smart contracts depending on the size of the input array. The evaluation was conducted *in situ*, by deploying the smart contract on the

Ethereum network and generating calls to the sorting algorithm with a random integer array of a given size. We collected the average power consumption of all servers during the experiment. For the Quicksort algorithm, the results are illustrated in Figure 5.12.

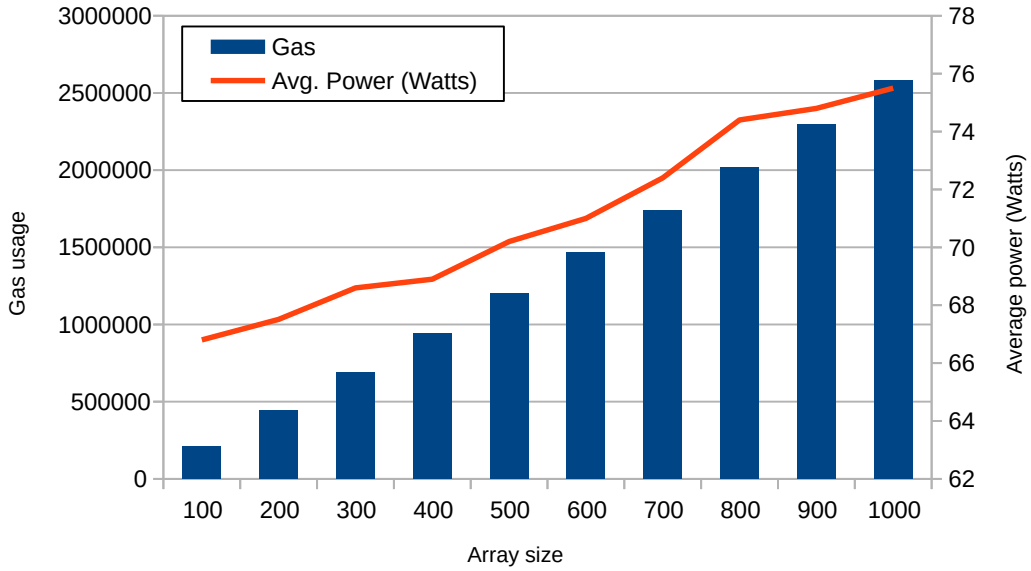

Figure 5.12 – Quicksort - Gas consumption and average power depending on array size input

We can notice that power and gas consumption are both correlated. Those two variables get a Pearson correlation coefficient of 0.99, indicating a linear correlation. Experiments on Mergesort lead to the same conclusion, with a coefficient of 0.93.

Using this data, we generate a linear regression model to infer the energy consumption of smart contracts. The estimation given by the model will be tied to the infrastructure on which the model has been calibrated. However, this can give us an idea of what it would cost to run smart contracts on this infrastructure. The model resulting from our experiments is illustrated in Figure 5.13. On our infrastructure, the power needed to run a smart contract consuming X gas is given by the following function:

$$Power(X) = X * 0,0000036756792144 + 66,1136 \tag{5.1}$$

In our real-world year long dataset, the median value of X (the gas consumed) for smart contract calls is 54 842, with values going from 21 051 to 12 188 440. Based on this power usage $Power(X)$ (in Watts) and the time T (in seconds) needed to run

a given contract, we can compute its energy consumption (in Joules) with a function $Energy(X) = Power(X) * T$. We recall that 1 Watt equals 1 Joule per second. In our example, we run an average of 4164 smart contracts calls in 30 minutes for each experiment for both algorithms, resulting in a 0.43 seconds contract call.

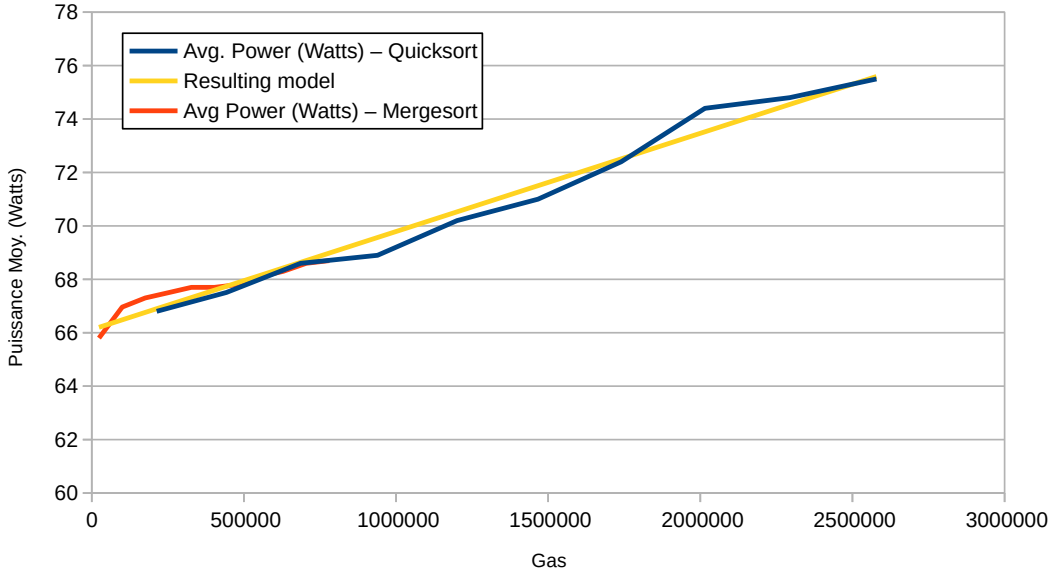

Figure 5.13 – Power depending on gas consumption

If we consider the difference in smart contract calls' running time as negligible, we can have a rough estimation of the energy needed to run smart contracts on our infrastructure. Estimation results are illustrated in Table 5.2. We estimate the average energy consumption of smart contract calls to be around 29.47 Joules. These values correspond to the energy consumed on a single Ethereum peer. As we will see in the rest of the paper, smart contracts are executed by the entire network on each call, by each peer, which multiplies its energy consumption.

5.3.3 Ethereum smart contract execution model

We have just seen that we could model Ethereum smart contracts' energy consumption according to their gas usage. Estimations in Table 5.2 only correspond to the energy consumption of smart contracts on a single peer. As blockchain networks consist of large networks of peers distributed across the world, we cannot model smart contracts' energy consumption by reasoning on a single node.

	Smart contracts	Simple Txs
Min (Joules)	28,66	28,66
Max (Joules)	73,38	28,66
Average (Joules)	29,47	28,66
Standard deviation	4,49	0

Table 5.2 – Estimated power usage for Ethereum processed smart contracts and transaction over the year

Indeed, even if transaction fees are paid only once by the caller, transactions and smart contract calls are executed across the whole network. Figure 5.14 illustrates the replication of smart contracts calls on a three nodes Ethereum Proof-of-Authority network [7]. Each of the three curves illustrates the power usage of each three servers. We emitted a call to the smart contract sorting function. This contract call has been sent to Gros-42 only. The first spike in energy usage at 163 seconds corresponds to the execution of this call on Gros-42. This call is then transmitted to the two other nodes that will execute it. This

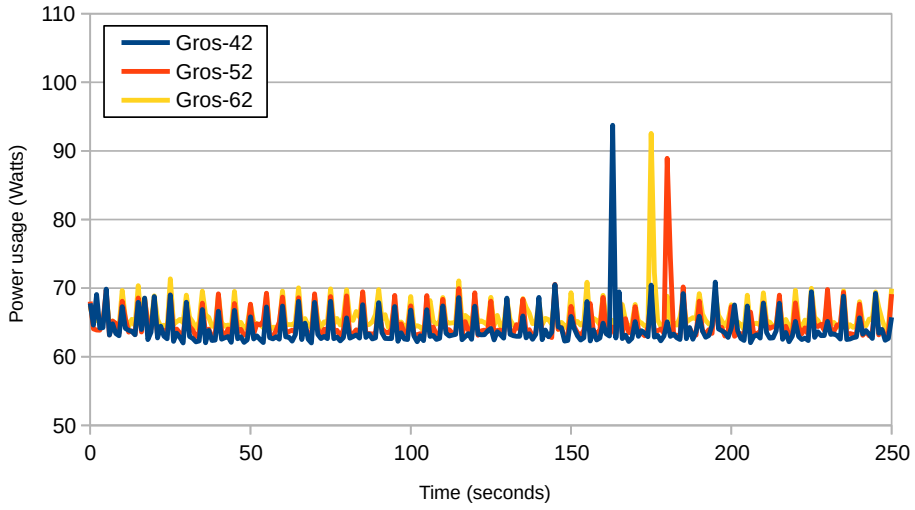

Figure 5.14 – Impact of smart contract calls replication on Ethereum network energy usage

experiment shows us that three smart contract executions have resulted from a single call. As blockchains assume an untrusted environment, each node has to execute transactions

7. Infrastructure and deployment protocol is the same as the one described in subsection 5.3.1. Smart contract deployed is Quicksort.

to verify their output. As a result, the bigger the network, the more expensive each transaction and call will be.

5.3.4 The impact of replication on smart contracts execution cost

Previous work in [55] has been done in order to estimate the number of peers in the Ethereum network. When this study has been written (early 2018), the authors measured 15 454 peers in a single day. To obtain this estimation, the authors have modified an existing implementation of Ethereum to build a tool named *NodeFinder*. Main modifications are that NodeFinder doesn't have a maximum peer limit (it seeks to connect to as many peers as possible) and it disconnects from peers right after the initial Ethereum handshake (to limit the number of active connections). NodeFinder will also periodically reconnect to known peers to ensure that those are still active. At the same time, the authors reported that `ethernodes.org` (a website that present statistics on the Ethereum network) listed 4717 peers, meaning that NodeFinder outperformed existing methods by a factor of 2.3.

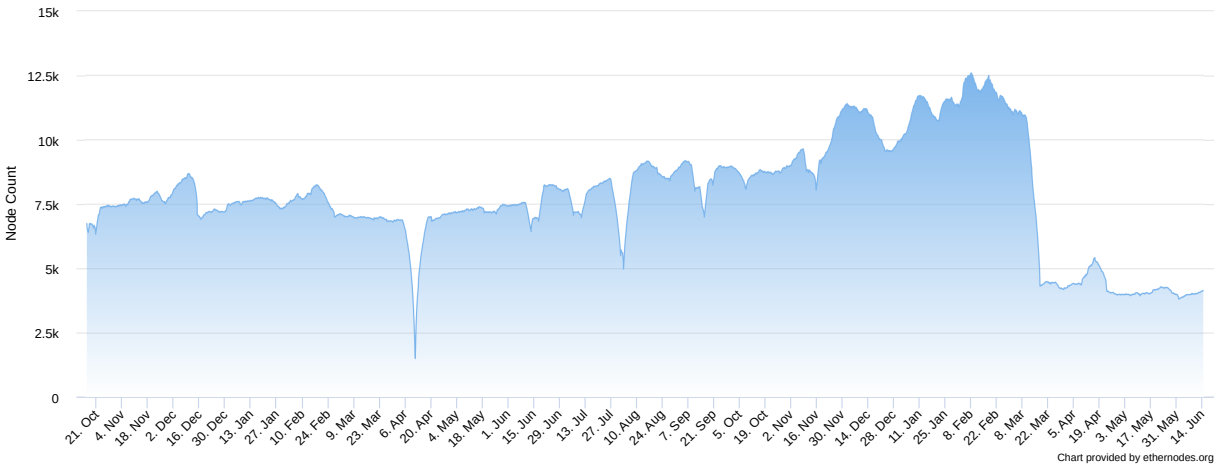

Figure 5.15 – Evolution of the number of Ethereum nodes listed by `ethernodes.org`, from October 2020 to June 2021 (source: `ethernodes.org`)

Figure 5.15 present the recent evolution of the number of active Ethereum nodes counted by `ethernodes.org`. On June 11th 2021 (at the time of writing this study), the number of peers presented on `ethernodes.org` was 4036. However, we can see a sudden drop in the number of active peers in early March 2021, with a number of listed nodes going from eleven thousand to less than five thousands. We estimate that this drop may

be linked with Ethereum *Berlin* update, which occurred at the same time. We can't be sure if this update has affected the number of active peers (e.g.: nodes that refused the update went to another network or were shut down by their owners) or if it has affected the number of peers listed by the website (e.g.: due to a change in the node discovery protocol). However, we can estimate that the real number of active peers may either be equal or higher than 4036.

Figure 5.16 gives a rough estimation on the energy consumed by an average contract call, depending on the number of nodes in the network. As each contract call is executed on every node in the network (see subsection 5.3.3), we projected an energy consumption what would be linear with the number of nodes.

Considering that the number of peers in Ethereum network was around four thousand (number of nodes declared on `ethernodes.org`) – and with the strong assumption that the smart contracts are executed in the same type of infrastructure as ours – we can consider that the energy consumed by an average smart contract call ranges from 29.47 (estimation on single node) to 1×10^5 Joules (replication of the contract call on each node). Multiplied by the 140 143 091 contract calls emitted over the year, it results in a global energy consumption of 1.67×10^{13} Joules. This amount is in the same order of magnitude that the energy of the maximum fuel a Airbus A380 can carry (source: `https://en.wikipedia.org/wiki/Orders_of_magnitude_(energy)`). Prior to the sudden drop in early March 2021, we can estimate the energy consumption of a single average contract call to be around 3×10^5 Joules (replication on eleven thousand nodes).

5.3.5 Limitations

Our model is subject to two limitations. First, it only includes the execution cost of smart contract calls, excluding the network communication energy cost. Second, it does not take into account the long-lasting energy cost of blockchain transactions. As new nodes will join the blockchain network, some may replay past transactions during their synchronization to ensure that the data they download from other peers is correct. To stay simple, our model only consider the execution of new transactions in active nodes. However, more complex models to be helpful to give more accurate estimation on the energy consumption of smart contracts.

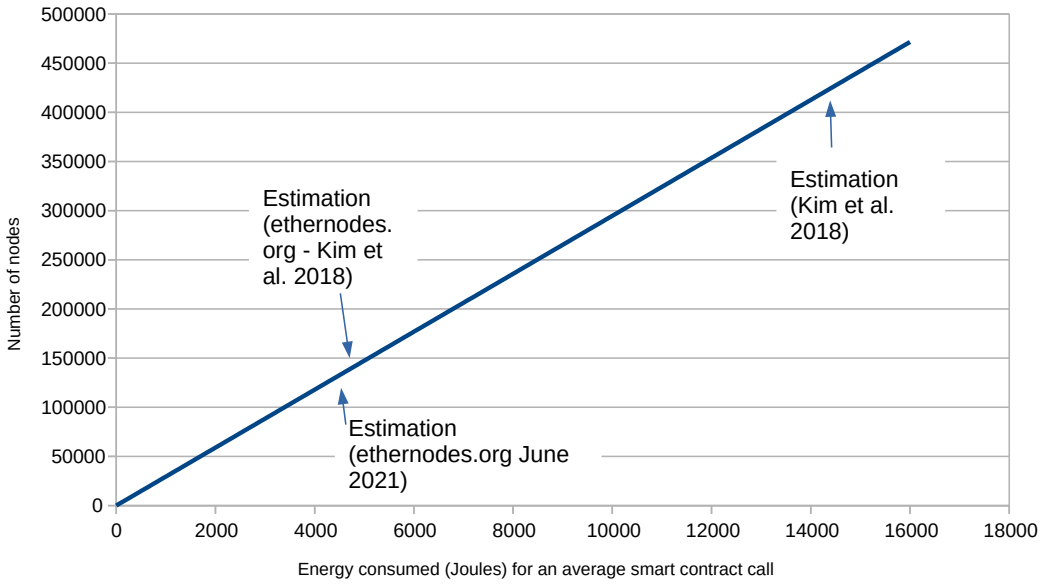

Figure 5.16 – Global energy consumed by a smart contract call depending on the number of nodes in the network

5.4 Conclusion

This chapter focused on the usage and energy consumption of Ethereum smart contracts.

In a first part, we studied one year of Ethereum transaction to better understand how important were smart contract in Ethereum's traffic. This analysis resulted in several observations. Firstly, smart contracts are at the heart of the Ethereum community: around half of Ethereum traffic corresponds to contract calls, and tens of thousands new contracts are created each month. A large part of those smart contract seems to be related to *decentralised finance* and token management, as half of contract calls are destined to ERC20 contracts. Secondly, even if many smart contracts are deployed each month, most of those contracts will lead to little or no usage. On the basis of this fact, we studied in chapter 6 the impact of having such a growing number of contracts and potential solutions.

In a second part, we measured and modelled the energy consumption of Ethereum smart contracts based on their cost in gas. Taking into accounts the replication of smart contracts execution, we provided rough estimations on the energy consumed by a contract call on the Ethereum network depending on the number of nodes.

Chapter 6

Reducing the amount of unused smart contracts in Ethereum

Smart contracts, scripts at the heart of blockchain-based applications, are meant to be available forever once deployed. However, this property has a price. The amount of space required to store new contracts keeps increasing. This increase impacts each participating node's performance and makes it inconvenient for low-end devices to participate in the network. Among all contracts deployed in the blockchain, a vast majority will lead to little if any usage. We demonstrated in subsection 5.2.5 that, in the course of one year, 70% of deployed contracts lead to no use. Unfortunately, unused contracts keep occupying space on the blockchain. To tackle this issue, we propose a new protocol to identify and delete unused contracts. Through simulation, based on Ethereum historical data, we show that deletion of smart contracts after an inactivity period of 90 days could lead to a 66% reduction in the number of contracts stored over a year.

Research presented in this chapter have resulted in a publication at the *26th IEEE Symposium on Computers and Communications (ISCC 2021)*. This publication received a "best student paper" award.

Contents

6.1 Introduction 98
6.2 The impact of unused smart contract on Ethereum 99
6.2.1 Evaluation protocol 99
6.2.2 The impact of unused smart contract on contract calls processing time 100
6.2.3 Current Ethereum state size 102
6.3 A Time to live protocol for smart contracts 102
6.3.1 Overview 102
6.3.2 Details 103
6.3.3 Contract destruction and data retrieval 105
6.3.4 Discussion on determining parameters value 105
6.3.5 Discussions 106
6.3.6 Protocol impact depending on TTL duration 106
6.4 Conclusion 107

6.1 Introduction

Smart contracts are at the heart of the Ethereum community. Tens of new thousands of contracts are deployed each month. However, the majority of those deployed contracts will rarely, if ever, be used (see subsection 5.2.5). Unfortunately, even with no usage, every contract have a continuous cost on the blockchain. Every contract deployment makes the Ethereum State (the data structure storing every Ethereum account) bigger. As this state is read and updated for every new transactions, its growth has an impact on block processing time (see subsection 6.2.2). Therefore, it is necessary to specify protocols to limit the growth of Ethereum State.

In this chapter, we challenge the "immortality" aspect of smart contracts by lifespan mechanism to every deployed contracts. Through a new protocol (presented in subsection 6.3.1) applied to smart contracts, we provide an incentive for miners to identify and remove unused contracts without altering the transaction history. This will ensure that only active contracts will be stored in Ethereum State. Through simulation, based on one year of Ethereum transactions data (see subsection 6.3.6), we illustrate potential

significant reduction in the number of contracts stored in the blockchain. For instance, removing every contract after an inactivity period of 90 days could lead to a 66% reduction of deployed contracts at the end of our one-year dataset.

To sum up, we present in this chapter the following contributions:

— A Time-To-Tive (TTL) protocol for smart contracts. After presenting this new protocol, we discuss the impact of different protocol parameters and security considerations. Eventually, we evaluate the potential impact of our proposition on the number of active contracts on Ethereum, based on one year of real-world transactions.

— Motivate research on managing the impact of unused smart contracts by giving insight on the number of unused smart contracts and their impact on the network. This analysis is based of transactions emitted on the Ethereum network over one year.

6.2 The impact of unused smart contract on Ethereum

6.2.1 Evaluation protocol

After establishing in subsection 5.2.5 that a large part of deployed smart contracts will rarely (if ever) be used, we aim to evaluate the impact of those unused contracts on Ethereum block processing time (a key point in the overall network performances). To measure this impact, we deployed a minimal two nodes Ethereum network (using Geth v1.10.3, the primary open-source Ethereum implementation) on three testbeds:

— A two-nodes network hosted on a Grid5000 [5] server equipped with 2 x Intel Xeon E5-2630 v3 CPU (8 cores/CPU), 128 GB of RAM and a 600 GB HDD, here called *G5K HDD*

— A two-nodes network hosted on the same Grid5000 network but with a 200 GB SSD, here called *G5K SSD*

— A two-nodes network hosted on a Raspberry Pi 4B equipped with a Quad-core Cortex-A72 (ARM v8) CPU, 8 GB of RAM, here called *RPi*

Our experiment consists of measuring block processing time depending on the state trie size. We define four collected metrics: 1) *execution time* (time spent to modify current state with transactions in the received block); 2) *validation time* (time spent to verify if the newly produced state is valid); 3) *write time* (time spent to write the new state

on disk); 4) *insert time* (the overall time spent to insert the received block, includes the three others). We collected those metrics through the standard metrics collection of *Geth* (the *Geth* Ethereum client emits performance metrics to a third-party database). In order to isolate different kind of transactions, we define three loads: 1) *transactions load* (generation of random Ether transfer transactions); 2) *contract creations load* (generation of contract deployment transactions); 3) *contract calls load* (generation of contract calls). The contract used for the two last load generations is an implementation of a basic Counter manager contract (illustrated in Listing 6.1).

```
contract Counter {
    uint256 public counter;
    constructor() {
        counter = 0;
    }
    function increment() public returns(uint256){
        counter = counter + 1;
        return counter;
    }}
```

Listing 6.1 – A simple Counter contract for analysis purpose

6.2.2 The impact of unused smart contract on contract calls processing time

Table 6.1 illustrates the impact of State size in Ethereum block processing time. In nearly all cases, an increase in Ethereum State size leads to a rise in block processing time (this increase is noted $+X.XX\%$ in the table). This increase is more significant for the low-end Raspberry Pi platform (because of their lower performances). Among all four collected metrics, an increase in Ethereum state size seems to impact block writing time the most. Out of the three loads, the Ether transfer appears to be the "cheapest" one for the "high performances" Grid5000 nodes. An Ether transfer only updates two balances (creating a new account in the State Trie if the recipient is an unknown address). On the contrary, contract deployments require more time to execute than a simple Ether transfer, as it needs to create a new account in the State Trie, copy the contract's bytecode and execute the constructor function.

		Ether transfer		Contract deployment		Contract call	
Platform	Metric	0GB	10GB	0GB	10GB	0GB	10GB
G5K HDD	Execution	4.22 (0.001)	4.41 (0.001) (+4.5%)	6.19 (0.002)	6.14 (0.002) (-0.8%)	7.17 (0.0008)	7.45 (0.001) (+3.91%)
	Validation	1.24 (0.0003)	1.31 (0.001) (+5.65%)	0.95 (0.0008)	0.96 (0.0002) (+1.05%)	1.24 (0.0004)	1.29 (0.0001) (+4.03%)
	Write	1.71 (0.002)	2.14 (0.0005) (+25.15%)	8.67 (0.018)	13.05 (0.006) (+50.52%)	1.88 (0.0008)	2.25 (0.003) (+19.68%)
	Insert	7.38 (0.002)	9.00 (0.004) (+21.95%)	21.19 (0.028)	28.26 (0.009) (+33.36%)	10.55 (0.001)	11.43 (0.005) (+8.34%)
G5K SSD	Execution	4.13 (0.001)	4.44 (0.002) (+7.51%)	6.05 (0.004)	6.46 (0.003) (+6.78%)	6.51 (0.005)	7.18 (0.003) (+10.29%)
	Validation	1.19 (0.0004)	1.32 (0.0009) (+10.92%)	0.92 (0.0001)	0.97 (0.0002) (+5.43%)	1.11 (0.0006)	1.26 (0.0008) (+13.51%)
	Write	1.87 (0.002)	2.41 (0.002) (+28.88%)	8.98 (0.009)	13.37 (0.005) (+48.89%)	1.69 (0.001)	2.27 (0.003) (+34.32%)
	Insert	7.43 (0.004)	8.56 (0.003) (+15,21%)	21.66 (0.019)	29.15 (0.005) (+34.58%)	9.59 (0.007)	11.08 (0.003) (+15,54%)
RPi	Execution	13.11 (0.008)	13.09 (0.012) (-0.15%)	11.00 (0.039)	12.22 (0.005) (+11.09%)	12.74 (0.004)	14.54 (0.004) (+14.13%)
	Validation	1.14 (0.0006)	1.59 (0.0002) (+39,47%)	0.89 (0.0006)	0.94 (0.0002) (+5.62%)	1.04 (0.0003)	1.12 (0.0004) (+7.69%)
	Write	1.91 (0.0003)	14,21 (0.002) (+643,98%)	8.41 (0.023)	23.12 (0.014) (+174.91%)	1.82 (0.0009)	5.08 (0.003) (+179.12%)
	Insert	16.89 (0.008)	49.43 (0.014) (+192,66%)	27.53 (0.081)	49.33 (0.009) (+79.19%)	16.37 (0.0004)	21.95 (0.001) (+34.09%)

Table 6.1 – Block processing time (in milliseconds) compared to state trie size on three platforms (standard deviation between parenthesis)

6.2.3 Current Ethereum state size

When writing the paper corresponding to research presented here (Late May 2021), contract codes occupy 1.95 GiB of space and State trie occupies 79.50 GiB on an Ethereum node synchronized using Geth v1.10.3. Apart from impacting block processing time, state growth limits the use of low-end devices as participating nodes. Indeed, to process transactions, a node needs to store and process at least the state and recent blocs. As the State grows because of new account creations, it becomes less convenient for low-end nodes to host an Ethereum node.

6.3 A Time to live protocol for smart contracts

Due to a smart contract deployment's relatively low financial cost [1], tens of thousands of contracts are deployed each month (see Figure 5.6). Among all those contracts, only a small portion will generate significant and long-term use. All the remaining unused contracts, still stored in the state forever, participate in increasing the hardware performances required to host an Ethereum node. This section proposes a potential solution to limit the footprint caused by unused smart contracts by restricting their lifetime (otherwise unlimited).

6.3.1 Overview

In order to limit the amount of deployed, yet unused, smart contracts on Ethereum, we propose to add a notion of *lifetime* to every contracts. Once deployed, a smart contracts can exist for a given time. This default lifetime, here called *Time To Live* is fixed by the blockchain community.

After this default time has passed, any miner can request the contract destruction and be rewarded for its action. A contract TTL can be extended in two ways: through interaction (function call by transactions) or through payment.

The overall goal behind this protocol is to add a continuous financial cost for a smart contract's users that would reflect its hosting cost reality. This cost is manifested through direct payment to maintain potentially inactive contracts or through transactions fees for

1. We estimate the average cost of contract deployment in our one-year dataset to be around 38 euros with an average gas cost of 520 249 and a gas price of 32gwei (average gas cost from `ethgasstation.info`)

active contracts. Unused contracts with no users willing to pay for its maintenance would be deleted in order to remove the overhead generated otherwise.

An existing way to remove contracts on Ethereum

Smart contracts are initially designed to be stored on the blockchain forever. However, in Ethereum, contracts can implement a call to *selfdestruct(address)* function that, once called, will destroy the contract and transfer funds in its balance to the provided address. Contract destruction removes its account (address, balance, bytecode) from the state trie and all stored variables from the storage trie. The contract can't be called anymore, and its address can be reassigned to a new contract. However, all transactions related to the destructed contract still exist in the blockchain transaction history.
Chen et al. [10] investigated the use of *selfdestruct* in Ethereum smart contracts. They found out that only 5.1% out of 54,739 analyzed contracts contain a *selfdestruct* function. Through surveys addressed to smart contract developers, they defined six reasons for why most contracts does not include *selfdestruct*. Those reasons include *security* and *trust* issues. Indeed, an incorrect contract implementation could lead to an accidental or *malicious* use of *selfdestruct*. Chen et al. also define five reasons for the inclusion of *selfdestruct* (e.g. to destroy a contract when security vulnerabilities are detected). However, numbers show that those reasons are not enough to incentive developers to implement *selfdestruct* in their contracts.

6.3.2 Details

We introduce the notations used in this section in Table 6.2

Notation	**Details**
TTL	Time after which a contract's next rent can be redeemed.
TE	Time Extension - Duration in seconds for which a contract TTL is extended at *interaction* or *extension payment*.
P	Price to pay to extend TTL by TE.
R	Reward gave to miner for unused contract destruction.

Table 6.2 – Details of variables used in our protocol

At deployment time, a smart contract has an initial expiration date of $TTL = TE$. Without any extension, miners will be able to request its destruction at time TTL and be financially rewarded by R tokens. Those R tokens come from a fixed deposit made during contract deployment by the account requesting the deployment. This new *destruction deposit* will serve as an incentive for miners to purge unused contracts.

If any user interacts with this contract by sending a transaction to its address, the contract's TTL will be extended if its remaining lifetime is shorter than TE (see algorithm 1). In other words, a contract is considered active TE seconds after its last call.

Algorithm 1: Function called for every interaction with a contract

Function `ExtendTTL(`*addr*`):`
  if $CurrentDate >= TTL(addr) - TE$ **then**
    $TTL(addr) = CurrentDate + TE$
  end

To ensure that a contract will still be deployed on a certain date, regardless of its activity, any participant can pay in advance for its hosting fees (see algorithm 2). By paying $N \times P$ tokens, the participant will increase the contract's TTL by $N \times TE$.

Algorithm 2: Function called to increment a contract TTL in advance

Function `PayForTTL(`*addr, tokens*`):`
  $TTL(addr) = TTL(addr) + TE \times (tokens \bmod P)$

Once a contract's expiration date has been reached, it become eligible for destruction. Any miner will be able to emit a *call for destruction* transaction. This new kind of transaction will destroy a given contract if and only if it's TTL is greater or equal to current date. As for blocs' timestamp, we allow here a certain lag between current date and TTL as blockchain networks does not include any central clock. A *call for destruction* has two effect: 1) destroy the specified contract and 2) financially reward the caller. This reward of R tokens correspond to the *destruction deposit* made during the contract's deployment. In theory, anyone can send a *call for destruction* but in practice we consider that, as miners have power on which transaction they include in new blocs, miners have a financial incentive to manage contract destruction themselves.

In order not to flood the network with *call for destruction* transactions, we define its input as a list of contracts to be deleted. This way, a miner can request the destruction of unused contracts through batches. For a list of contracts to be destroyed, only contracts

available for destruction will actually be destroyed (others will be ignored). The reward granted to the miner will correspond to the sum of each individual contract reward.

6.3.3 Contract destruction and data retrieval

Through a *call for destruction* transaction, a miner can destroy an unused contract. In Ethereum, destructing a contract is equivalent to removing its account (address, balance, bytecode) and related storage from the state trie. However, a contract destruction does not impact a blockchain transaction history. Transactions related to a contract could still be retrieved and replayed after its destruction for later use (e.g. for later analysis). The impact of our proposed protocol lie in State trie management.

6.3.4 Discussion on determining parameters value

As we described our protocol for smart contracts with Time To Live, we have presented several protocol parameters (detailed in Table 6.2). We did not specified any fixed values as we wanted to describe a general protocol that could be adapted to any blockchain network. We will now discuss on some considerations regarding the choice of specific values.

Setting TE

TE represents the unit of time used for contract lifetime and its payment. A value too high would reduce the gains on state management, taking longer to purge unused contracts. On the other hand, a value too low could be too intrusive and lead to the destruction of contract that could be still used. We detail this trade-off in subsection 6.3.6.

Setting P

We believe that paying in advance should be roughly equal to basic transaction fees. If set too high, this could create an incentive for the creation of undesired "uptime services" that could maintain contracts through interactions for cheaper ($< P$) fees.

Setting R

A higher *destruction deposit* serves as a better incentive for miners to purge unused contracts but makes it more expensive for developers to deploy new contracts. However,

too high a value could be an incentive for undesired behaviour (see section 6.3.5).

6.3.5 Discussions

Potential attacks

With this proposition, we enable the destruction of deployed (unused) contracts by miners once their TTL has been outdated. The main potential attack we conceive would be the censorship of some contract-related transactions by miners in order to reach a contract's TTL, destroy this contract and redeem the destruction reward. We believe that the plausibility of such an attack lies in the byzantine fault tolerance of the blockchain consensus algorithm. Indeed, as long as enough miners stay honest, contract-related transaction will be still processed. Moreover, this potentially undesired behaviour could be limited by avoiding setting the destruction reward R too high.

Implementing the protocol

Integrating our protocol to existing blockchains like Ethereum would require modifications of the blockchain itself, resulting in a *hardfork* (a new version of the blockchain protocol that won't be compatible with the previous one).

In the case of Ethereum, the existing *selfdestruct* operation that enable a contract to be deleted from the state has to be called from the contract's code. To leverage this functionality, the Ethereum protocol has to be modified to add a new type of transactions that enable the call of *selfdestruct* on a contract (even if the function was not included in the contract's code) if it's lifespan is ended. Information's regarding the remaining living time of a contract could be stored in the state along with existing contract's data.

6.3.6 Protocol impact depending on TTL duration

As stated in subsection 6.3.4, TE is the principal protocol parameter that will determine the duration of an inactivity period before determining that a contract is eligible for destruction.

Figure 6.1 illustrates the evolution of the number of deployed contracts during our one-year dataset and potential gains depending on TE value. In this evaluation, we consider that a contract will be destructed right after an inactivity period of TE days. We do not consider any payment in advance in order to extend contract lifespan. As expected, a

small TE (e.g 10 days) leads to significant reduction in the number of deployed contracts (90% reduction, from 945 116 contracts at the end of the year to 85 485). An larger value of TE still brings significant impact (77% reduction for $TE = 60$ days) while limiting overhead for contract users.

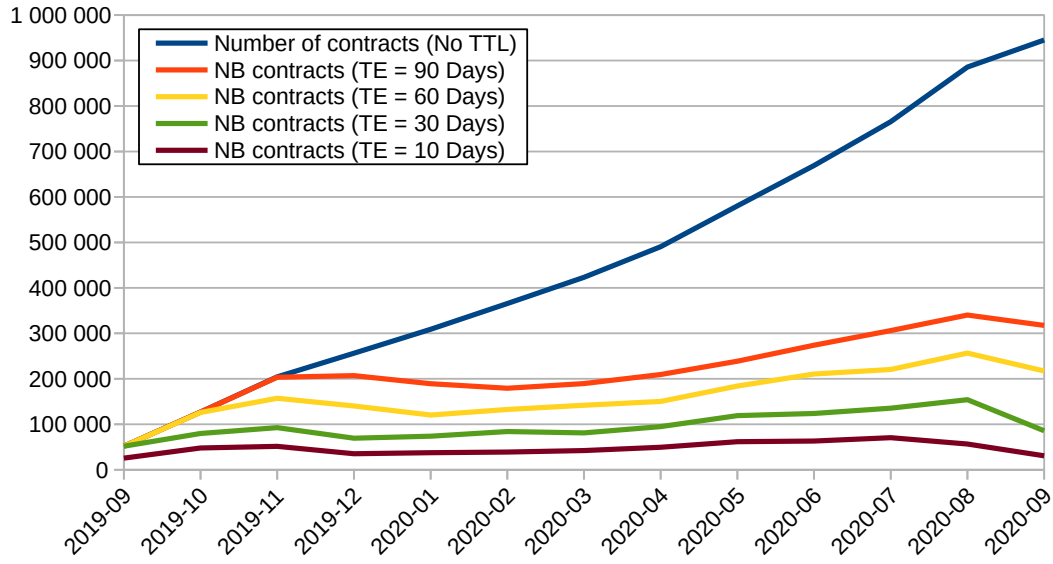

Figure 6.1 – Evolution of the number of contracts depending on TE value

Concerning the potential overhead of this proposed protocol, Table 6.3 illustrates the number of contracts called after an inactivity period of TE days. We see that, in this one year dataset, only 7.1% (66 709) of all 945 116 contracts have been used after a 10 days inactivity period. This percentage drops to 1.1% (10 236) for a 90 days inactivity period. Among all contracts that have been called after a certain period of inactivity, only a small minority have lead to significant usage. For instance, 90% of contracts used after an inactivity period of 10 days have been called at most 10 times after this inactivity period. Users of such contracts that could be used after an inactivity period greater than TE could pay in advance to ensure that their contract will be still deployed for later use.

6.4 Conclusion

This chapter tackles the issue of the ever-growing space required by unused smart contracts. As we saw in subsection 5.2.5, a large majority of deployed smart contracts will lead to limited usage. In subsection 5.2.5, we determined that 70% of deployed contracts

TE **(in Days)**	**10**	**30**	**60**	**90**
Number contract called after inactivity period of TE days	66 709	36 731	18 226	10 236
% Total (945 116)	7.1	3.9	1.9	1.1
Number of calls after inactivity period of TE days (max)	358	274	254	218
Number of calls after inactivity period of TE days (average)	6	4	3.5	3
Number of calls after inactivity period of TE days (median)	2	2	1	1
Number of calls after inactivity period of TE days (90th percentile)	10	7	6	5

Table 6.3 – Number of contracts called after an inactivity period

were never called. Half of the remaining 30% won't be called a week after their deployment. This ever-growing amount of unused contracts is not without consequences for the Ethereum community. Indeed, each contract deployment participates in the increase of Ethereum state size. This increase impacts the performances of the network and makes impractical the use of low-end devices as participating nodes.

In order to identify and remove unused smart contracts, we propose a novel "Time-To-Live" protocol for smart contracts. This protocol incentives miners to identify and remove unused contracts from the blockchain. We simulated that, over the course of one year, removing contracts after an inactivity period of 90 days could lead to the reduction of the number of contracts in Ethereum state by 66%.

Chapter 7

Conclusion

This chapter first summarise the contributions presented in this thesis, providing answer to the three questions presented in the introduction. Then, we discuss potential leads for future work.

Contents

7.1 Achievements 109

7.2 Perspectives 111

7.2.1 Impact of layer-two solutions on blockchains footprint 111

7.2.2 Modelling the energy consumption of smart contracts 112

7.2.3 Designing blockchains with temporary data 113

7.1 Achievements

This thesis aimed to better understand the cost of smart contracts deployed on blockchain platforms. To this end, we focused our research work on the Ethereum platform. We provided analysis based on real Ethereum history and experiments deployed on real infrastructure, using the Grid'5000 research testbed and a local Raspberry Pi cluster. To conduct our experiments, we proposed a novel framework for the deployment and performances analysis of blockchains called *BCTMark*. Research presented in this manuscript permit us to answer the questions proposed in the introduction.

How important are smart contracts in blockchain networks?

Although the academic literature and industrial activity would tend to show the use of smart contracts in a lot of different domains, only a few papers analysed the *real* usage of smart contracts on a public blockchain network. In section 5.2, by downloading and analysing one year of Ethereum transaction history, we provided readers insights on the importance of smart contracts in the Ethereum traffic. Our investigations shown that smart contracts calls indeed occupy a significant part in Ethereum traffic. In our dataset,

half of the processed transactions were intended for contract calls. A large part of those calls were targeting contracts implementing an ERC20 fungible token interface. This fact tends to show how important token management is for the Ethereum community. Our investigations also illustrated an important fact: a majority of deployed contracts will lead to little or no usage. We have shown in subsection 5.2.5 that 70% of contracts deployed in our dataset were never called. Among the remaining, only a few induced significant usage. Starting from this point, we proposed in chapter 6 to investigate the consequences of storing so many unused contracts.

Can we estimate and model the energy cost of these contracts? We have seen a general tendency for blockchain platforms to quit Proof-of-Work based consensus system, often for performances reasons. As a consequence, we estimate that it is now important for researcher to focus on the energy aspect of decentralised applications themselves (as discussed in subsection 5.3.1). A first step is to study and model the energy consumption of smart contracts. In section 5.3, we proposed a model to estimate the energy consumption of Ethereum smart contracts based on their cost in gas. To this end, we deployed and measured the energy cost of smart contracts on physical infrastructure. Based on this model and on several estimations on the size of the Ethereum network, we estimated the rough energy cost of smart contracts calls on Ethereum.

What is the cost generated by the immutability of deployed contracts? A key promise of Ethereum is the immutability of data inserted in blocs. Once deployed, smart contracts are supposed to be available forever on the platform. However, as the popular adage says, "*There is no such thing as a free lunch*". We studied in section 6.2 the performances impact induced by the growing number of contracts in Ethereum. Our experiments show an increase in the block processing time as the Ethereum *state* grows. This state growth is induced by the creation of new accounts, including contract-based accounts. Therefore, the identification and suppression of unused contracts would lighten the block processing phase for all peers. We proposed in section 6.3 a protocol to add a lifetime to all deployed contracts. The idea is simple: each new contract come with a defined lifespan. Each calls to a contract will increase its lifespan. Once the lifespan of a contract is exceeded due to a period of no use, miners are financially incentive to call for the destruction of this contract. This proposition seeks to challenge the "immortality" aspect of smart contracts to produce a "lighter" blockchain.

To conduct our experiments on real testbeds, we also proposed in chapter 4 a novel framework for the deployment and performances analysis of blockchains. Indeed, as the

blockchain ecosystem grows in complexity, the need for standard, performances analysis tools become more important. A few existing tools addresses this issue, but each of those were lacking important aspects required for scientific experiments. To this end, we proposed *BCTMark*, a framework that addresses the several phases in the performances' analysis of blockchains: deployment, load generation, metrics management and network emulation. To illustrate the capacities of BCTMark, we deployed and analysed the performances of three blockchain networks on different testbeds.

Research presented in this thesis lead to the publication of papers in a national conference, two international conferences and in a journal.

7.2 Perspectives

Contributions presented in this thesis could be extended in several directions.

7.2.1 Impact of layer-two solutions on blockchains footprint

As mentioned in this manuscript, blockchain solutions often suffer from performances and footprint issues like low throughput, high energy cost due to consensus algorithm choice or a high replication factor, an increasing need for storage... As a potential solution to those issues, an ecosystem of protocols called *layer two* has emerged. Layer two protocols are built on top of existing blockchains (now considered as layer one). The goal of such solutions is to move computation and / or data off-chain and only store and process critical data on the blockchain.

Payment channels (e.g., the Lightning network [86]) are an example of layer two solutions. When a set of users want to exchange a high number of transactions, payment channels can be used to process transactions while generating a minimal amounts of traffic on the blockchain. Payment channels typically generates only two transactions on the blockchain: one at the opening of the channel (freezing a certain amount of money for each user to generate initial balances) and one when closing the channel (redistributed the funds to each users, depending on their final balance). All exchanges between those two transactions are processed by the payment channel without any transactions on the blockchain. In this case, both data and computations are stored off-chain. Payment channels are often used for micropayments: as they generate a minimal amount of blockchain transactions, they minimise the cost of transaction fees.

Although those protocols are often presented as solutions to reduce the footprint and increase the performances of blockchains, we didn't find any experimentations measuring their impact. Using BCTMark, we could deploy more complex scenarios to evaluate the different impact (in terms of performances and energy cost) and trade-off induced by the introduction of layer two systems in blockchains. For instance, we could deploy a set of experiments with a blockchain network and a growing set of payment channels to measure how this layer two system can affect the overall blockchain energy consumption. Moreover, payment channels are far from being the only layer two system for blockchains. Comparing how different proposal affect the energy consumption of blockchains could help communities to build efficient blockchain networks and researcher to improve existing proposals.

7.2.2 Modelling the energy consumption of smart contracts

Section 5.3 presented a model to estimate the energy consumption of Ethereum smart contracts depending on their cost in gas. We inferred a global energy cost by multiplying the energy consumption of an individual node by the number of nodes (as every node will execute each contract calls).

However, our intuition is that this produces a low estimate of a contract's call energy cost. As new nodes will enter the blockchain network, they can repeat its history. Therefore, a contract call may not be executed only once on each nodes present in the network at his emission but it can also be executed in the future when new nodes will replay history to ensure the integrity of the blockchain data they are downloading.

Future work on modelling the energy cost of Ethereum contract calls would be threefold. First, we could verify the intuition detailed in the previous paragraph. Then we could model the execution of Ethereum contracts and integrate this model into existing blockchain simulators like Blocksim [40] or Simblock [2]. Both simulators could benefit from energy models that could help in the design of more frugal blockchain protocols. Finally, we could model the impact of different layer two protocols to better understand the impact of different network architectures (e.g. a blockchain on its own or a blockchain using payment channels).

7.2.3 Designing blockchains with temporary data

Regarding the evaluation of the impact of the growing number of Ethereum contracts, we could measure the energy footprint potentially induced by state growth. As illustrated in section 6.2, the creation of new contracts leads to performance's impact on Ethereum's block processing time. Due to a lack of time, we didn't include any energy considerations in our experiments. Quantifying the impact of the state growth on the energy consumption of Ethereum nodes could help to better understand the cost of "immortal" contracts.

The protocol proposed in section 6.3 still has room for improvement. First, we could investigate the possibility to create "immortal" contracts through on chain governance. Indeed, our proposition does not let the possibility for a blockchain community to define core contracts that would stay available regardless their usage. Blockchains like Tezos[47] include a mechanism to let the community decide and vote on next features and protocol improvement. Seeking inspiration on such "on-chain" governance mechanism could help define a proper balance between the current "immortal contracts" situation and our proposition. Then, we could study the possibility to dynamically determine the TTL extension value TE, based on account creation rate, in order to limit contract suppression if state size stabilises. Finally, we could investigate how this protocol could be adapted to *Externally Owned Accounts* (accounts that does not belong to smart contracts). It appears hazardous to delete accounts that could belong to human users, as they could be using their account for long-time investment.

Such advances could help to build the foundation of a new framework for low-footprint blockchains.

BIBLIOGRAPHY

[1] Ansible, *Ansible is Simple IT Automation*, https://www.ansible.com/.

[2] Yusuke Aoki et al., « Simblock: A blockchain network simulator », *in*: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, 2019, pp. 325–329.

[3] *Applying cryptotechnologies to Trade Finance*, https://www.abe-eba.eu/media/azure/production/1549/applying-cryptotechnologies-to-trade-finance.pdf, Accessed: 2021-09-16.

[4] Adam Back et al., « Hashcash-a denial of service counter-measure », *in*: (2002).

[5] Daniel Balouek et al., « Adding Virtualization Capabilities to the Grid'5000 Testbed », *in*: *Cloud Computing and Services Science*, ed. by Ivan I. Ivanov et al., vol. 367, Communications in Computer and Information Science, Springer International Publishing, 2013, pp. 3–20, ISBN: 978-3-319-04518-4, DOI: 10.1007/978-3-319-04519-1_1.

[6] Vitalik Buterin, *A Theory of Ethereum State Size Management*, https://hackmd.io/@vbuterin/state_size_management, Accessed: 2021-10-15.

[7] Michael J Cahill, Uwe Röhm, and Alan D Fekete, « Serializable isolation for snapshot databases », *in*: *ACM Transactions on Database Systems (TODS)* 34.4 (2009), p. 20.

[8] *Hyperledger Caliper*, https://www.hyperledger.org/projects/caliper, Accessed: 2019-12-6.

[9] *Cardano*, cardano.org, Accessed: 2021-10-15.

[10] Jiachi Chen et al., « Why do smart contracts self-destruct? investigating the self-destruct function on ethereum », *in*: *arXiv preprint arXiv:2005.07908* (2020).

[11] Ting Chen et al., « Under-optimized smart contracts devour your money », *in*: *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, 2017, pp. 442–446.

[12] Mohammad Jabed Morshed Chowdhury et al., « A comparative analysis of distributed ledger technology platforms », *in*: *IEEE Access* 7 (2019), pp. 167930–167943.

[13] Anton Churyumov, « Byteball: A decentralized system for storage and transfer of value », *in*: *URL https://byteball. org/Byteball. pdf* (2016).

[14] Ryan Cole and Liang Cheng, « Modeling the Energy Consumption of Blockchain Consensus Algorithms », *in*: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, IEEE, 2018, pp. 1691–1696.

[15] Brian F Cooper et al., « Benchmarking cloud serving systems with YCSB », *in*: *Proceedings of the 1st ACM symposium on Cloud computing*, ACM, 2010, pp. 143–154.

[16] *Corda - Open-source blockchain platform for business*, `corda.net`, Accessed: 2021-10-15.

[17] *Cryptokitties*, `https://www.cryptokitties.co/`.

[18] Stefano De Angelis et al., « PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain », *in*: (2018).

[19] Christian Decker and Roger Wattenhofer, « A fast and scalable payment network with bitcoin duplex micropayment channels », *in*: *Symposium on Self-Stabilizing Systems*, Springer, 2015, pp. 3–18.

[20] Richard Dennis, Gareth Owenson, and Benjamin Aziz, « A temporal blockchain: a formal analysis », *in*: *2016 International Conference on Collaboration Technologies and Systems (CTS)*, IEEE, 2016, pp. 430–437.

[21] Monika Di Angelo and Gernot Salzer, « A survey of tools for analyzing Ethereum smart contracts », *in*: *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, IEEE, 2019, pp. 69–78.

[22] Tien Tuan Anh Dinh et al., « Blockbench: A framework for analyzing private blockchains », *in*: *Proceedings of the 2017 ACM International Conference on Management of Data*, ACM, 2017, pp. 1085–1100.

[23] Lars Dittmar and Aaron Praktiknjo, « Could Bitcoin emissions push global warming above 2° C? », *in*: *Nature Climate Change* 9.9 (2019), pp. 656–657.

[24] Alexandre Dolgui et al., « Blockchain-oriented dynamic modelling of smart contract design and execution in the supply chain », *in*: *International Journal of Production Research* 58.7 (2020), pp. 2184–2199.

[25] Zhongli Dong et al., « Dagbench: A performance evaluation framework for dag distributed ledgers », *in*: *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, IEEE, 2019, pp. 264–271.

[26] John R Douceur, « The Sybil Attack », *in*: *Peer-to-Peer Systems*, ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron, Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260, ISBN: 978-3-540-45748-0.

[27] John R Douceur, « The sybil attack », *in*: *International workshop on peer-to-peer systems*, Springer, 2002, pp. 251–260.

[28] *DPOS Consensus Algorithm - The Missing White Paper*, `https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper`, Accessed: 2021-10-15.

[29] Cynthia Dwork and Moni Naor, « Pricing via processing or combatting junk mail », *in*: *Annual international cryptology conference*, Springer, 1992, pp. 139–147.

[30] *E-Estonia*, `https://e-estonia.com/`.

[31] Enoslib, *Enoslib : A framework to build experimental frameworks on various platforms*, `https://github.com/BeyondTheClouds/enoslib`, Github repository.

[32] *EOS*, `eos.io`, Accessed: 2021-10-15.

[33] *Ethereum Wiki - Ethash*, `https://github.com/ethereum/wiki/wiki/Ethash`.

[34] *Ethereum Wiki - Ethash - DAG*, `https://github.com/ethereum/wiki/wiki/Ethash-DAG`.

[35] *Ethereum Wiki - Ethash Design Rationale*, `https://github.com/ethereum/wiki/wiki/Ethash-Design-Rationale`.

[36] Medvedev Evgeny, *Ethereum-ETL*, `https://github.com/blockchain-etl/ethereum-etl`, 2020.

[37] Ittay Eyal et al., « Bitcoin-ng: A scalable blockchain protocol », *in*: *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 45–59.

[38] Vogelsteller Fabian and Buterin Vitalik, *EIP-20: ERC-20 Token Standard*, `https://eips.ethereum.org/EIPS/eip-20`, Accessed: 2020-12-04.

[39] *Hyperledger Fabric*, `https://github.com/hyperledger/fabric`, Accessed: 2019-12-6.

[40] Carlos Faria and Miguel Correia, « BlockSim: blockchain simulator », *in*: *2019 IEEE International Conference on Blockchain (Blockchain)*, IEEE, 2019, pp. 439–446.

[41] Josselin Feist, Gustavo Grieco, and Alex Groce, « Slither: a static analysis framework for smart contracts », *in*: *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, IEEE, 2019, pp. 8–15.

[42] Eduardo Castelló Ferrer, « The blockchain: a new framework for robotic swarm systems », *in*: *Proceedings of the Future Technologies Conference*, Springer, 2018, pp. 1037–1058.

[43] Ethereum Foundation, *A Next-Generation Smart Contract and Decentralized Application Platform*, `https://github.com/ethereum/wiki/wiki/White-Paper`, whitepaper, 2013.

[44] Wood Gavin, *Ethereum: a secure decentralised generalised transaction ledger*, `https://ethereum.github.io/yellowpaper/paper.pdf`, yellowpaper, 2020.

[45] *Go-Ethereum - Official Go implementation of the Ethereum protocol*, `https://github.com/ethereum/go-ethereum`, Accessed: 2019-12-6.

[46] Sanjay Ghemawat and Jeff Dean, *LevelDB is a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values.* `https://github.com/google/leveldb`, 2021.

[47] LM Goodman, « Tezos: A self-amending crypto-ledger position paper », *in*: *Aug* 3 (2014), p. 2014.

[48] *Grafana - The open observable platform*, `https://grafana.com/`, Accessed: 2019-12-6.

[49] Tharaka Hewa, Mika Ylianttila, and Madhusanka Liyanage, « Survey on blockchain based smart contracts: Applications, opportunities and challenges », *in*: *Journal of Network and Computer Applications* (2020), p. 102857.

[50] Michał R Hoffman, Luis-Daniel Ibáñez, and Elena Simperl, « Toward a Formal Scholarly Understanding of Blockchain-Mediated Decentralization: A Systematic Review and a Framework », *in*: *Frontiers in Blockchain* 3 (2020), p. 35.

[51] Iexec, *IExec: Blockchain-Based Decentralized Cloud Computing*, `https://iex.ec/wp-content/uploads/pdf/iExec-WPv3.0-English.pdf`, whitepaper, 2018.

[52] *InfluxDB - Scalable datastore for metrics, events, and real-time analytics*, `https://github.com/influxdata/telegraf`, Accessed: 2019-12-6.

[53] Don Johnson, Alfred Menezes, and Scott Vanstone, « The elliptic curve digital signature algorithm (ECDSA) », *in*: *International journal of information security* 1.*1* (2001), pp. 36–63.

[54] Aggelos Kiayias et al., « Ouroboros: A provably secure proof-of-stake blockchain protocol », *in*: *Annual International Cryptology Conference*, Springer, 2017, pp. 357–388.

[55] Seoung Kyun Kim et al., « Measuring ethereum network peers », *in*: *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 91–104.

[56] Sunny King, « Primecoin: Cryptocurrency with prime number proof-of-work », *in*: *July 7th* 1.*6* (2013).

[57] Sunny King and Scott Nadal, « Ppcoin: Peer-to-peer crypto-currency with proof-of-stake », *in*: *self-published paper, August* 19 (2012), p. 1.

[58] Nir Kshetri, « Can blockchain strengthen the internet of things? », *in*: *IT professional* 19.*4* (2017), pp. 68–72.

[59] Protocol Labs, *Filecoin: A Decentralized Storage Network*, `https://filecoin.io/filecoin.pdf`, whitepaper, 2017.

[60] Leslie Lamport, Robert Shostak, and Marshall Pease, « The Byzantine Generals Problem », *in*: *ACM Transactions on Programming Languages and Systems* 4.*3* (1982), pp. 382–401, ISSN: 01640925, DOI: `10.1145/357172.357176`, URL: `http://portal.acm.org/citation.cfm?doid=357172.357176`.

[61] Noureddine Lasla et al., « Green-PoW: An Energy-Efficient Blockchain Proof-of-Work Consensus Algorithm », *in*: *arXiv preprint arXiv:2007.04086* (2020).

[62] Colin LeMahieu, « Nano: A feeless distributed cryptocurrency network », *in*: *Nano [Online resource]. URL: https://nano. org/en/whitepaper (date of access: 24.03. 2018)* (2018).

[63] *LO3 Energy - Empowering communities through localized energy solutions*, `https://lo3energy.com/innovations/`, Accessed: 2021-09-16.

[64] *Locust - A modern load testing framework*, `https://locust.io/`, Accessed: 2019-12-6.

[65] Dumitrel Loghin et al., « Blockchain goes green? An analysis of blockchain on low-power nodes », *in*: *arXiv preprint arXiv:1905.06520* (2019).

[66] Ikuo Magaki et al., « Asic clouds: Specializing the datacenter », *in*: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2016, pp. 178–190.

[67] David Mazieres, « The stellar consensus protocol: A federated model for internet-level consensus », *in*: *Stellar Development Foundation* 32 (2015).

[68] Hass McCook, « An order-of-magnitude estimate of the relative sustainability of the Bitcoin network », *in*: *A critical assessment of the Bitcoin mining industry, gold production industry, the legacy banking system, and the production of physical currency* 2 (2014), p. 25.

[69] Esther Mengelkamp et al., « Designing microgrid energy markets: A case study: The Brooklyn Microgrid », *in*: *Applied Energy* 210 (2018), pp. 870–880.

[70] *Mininet: An instant Virtual Network on your laptop (or other PC)*, `http://mininet.org/`, Accessed: 2021-06-15.

[71] F Montevecchi et al., « Energy-efficient cloud computing technologies and policies for an eco-friendly cloud market », *in*: *European Commission, final study report, Vienna, report commissioned by the Directorate-General for Communications Networks, Content and Technology* (2020).

[72] Camilo Mora et al., « Bitcoin emissions alone could push global warming above 2 C », *in*: *Nature Climate Change* 8.*11* (2018), pp. 931–933.

[73] Roman Mühlberger et al., « Foundational oracle patterns: Connecting blockchain to the off-chain world », *in*: *International Conference on Business Process Management*, Springer, 2020, pp. 35–51.

[74] Michael Mylrea and Sri Nikhil Gupta Gourisetti, « Blockchain for smart grid resilience: Exchanging distributed energy at speed, scale and security », *in*: *2017 Resilience Week (RWS)*, IEEE, 2017, pp. 18–23.

[75] Satoshi Nakamoto, « Bitcoin: A Peer-to-Peer Electronic Cash System », *in*: *Journal for General Philosophy of Science* 39.*1* (2008), pp. 53–67, ISSN: 09254560, DOI: 10.1007/s10838-008-9062-0, arXiv: 43543534534v343453.

[76] *Neo Smart Economy*, neo.org, Accessed: 2021-10-15.

[77] Jiacheng Ni and Xuelian Bai, « A review of air conditioning energy performance in data centers », *in*: *Renewable and sustainable energy reviews* 67 (2017), pp. 625–640.

[78] Karl J O'Dwyer and David Malone, *Bitcoin mining and its energy footprint*, 2014.

[79] Openethereum, *Openethereum*, https://github.com/openethereum/openethereum, 2020.

[80] Haochen Pan et al., « BBB: A Lightweight Approach to Evaluate Private Blockchains in Clouds », *in*: *GLOBECOM 2020-2020 IEEE Global Communications Conference*, IEEE, 2020, pp. 1–6.

[81] *Parity - Fast and feature-rich multi-network Ethereum client.* https://github.com/paritytech/parity-ethereum, Accessed: 2019-12-6.

[82] Jonathan Pastor and Jean Marc Menaud, « SeDuCe: a Testbed for Research on Thermal and Power Management in Datacenters », *in*: *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, IEEE, 2018, pp. 1–6.

[83] Gareth W Peters and Efstathios Panayi, « Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money », *in*: *Banking beyond banks and money*, Springer, 2016, pp. 239–278.

[84] Andrea Pinna et al., « A massive analysis of ethereum smart contracts empirical study and code metrics », *in*: *IEEE Access* 7 (2019), pp. 78194–78213.

[85] *PoET 1.0 Specification*, https://sawtooth.hyperledger.org/docs.

[86] Joseph Poon and Thaddeus Dryja, *The bitcoin lightning network: Scalable off-chain instant payments*, 2016.

[87] Serguei Popov, « The tangle », *in*: ().

[88] *Prometheus - From metrics to insight*, `https://prometheus.io/`, Accessed: 2019-12-6.

[89] *Provable - blockchain oracle service, enabling data-rich smart contracts*, `https://provable.xyz/`, Accessed: 2021-09-29.

[90] *Consensys Quorum*, `consensys.net/quorum/`, Accessed: 2021-10-15.

[91] Michel Rauchs et al., « Distributed ledger technology systems: A conceptual framework », *in*: *Available at SSRN 3230013* (2018).

[92] Team Rocket, *Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies*, 2018.

[93] *Rootstock*, `rsk.co`, Accessed: 2021-10-15.

[94] Sara Rouhani and Ralph Deters, « Security, performance, and applications of smart contracts: A systematic survey », *in*: *IEEE Access* 7 (2019), pp. 50759–50779.

[95] Fahad Saleh, « Blockchain without waste: Proof-of-stake », *in*: *Available at SSRN 3183935* (2019).

[96] Vikram Saraph and Maurice Herlihy, « An Empirical Study of Speculative Concurrency in Ethereum Smart Contracts », *in*: *arXiv preprint arXiv:1901.01376* (2019).

[97] The shift project, *IMPACT ENVIRONNEMENTAL DU NUMÉRIQUE : TENDANCES À 5 ANS ET GOUVERNANCE DE LA 5G*, `https://theshiftproject.org/wp-content/uploads/2021/03/Note-danalyse_Numerique-et-5G_30-mars-2021.pdf`, Accessed: 2021-10-15.

[98] Ali Shoker, « Sustainable blockchain through proof of exercise », *in*: *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, IEEE, 2017, pp. 1–9.

[99] Bart Smaalders, « Performance anti-patterns », *in*: *ACM Queue* 4.*1* (2006), pp. 44–50.

[100] *SOMN - Decentralized Fog Computing Platform*, `https://sonm.com`, Accessed: 2021-09-16.

[101] Steem, *Steem - An incentivized, blockchain-based, public content platform*, `https://steem.com/steem-whitepaper.pdf`, whitepaper, 2016.

[102] *Stellar - An open network for money*, stellar.org, Accessed: 2021-10-15.

[103] Inc Storj Labs, *Storj: A Decentralized cloud storage network framework*, https://storj.io/storjv3.pdf, whitepaper, 2018.

[104] Péter Szilágyi, *EIP-225: Clique proof-of-authority consensus protocol*, https://eips.ethereum.org/EIPS/eip-225, 2017.

[105] Paolo Tasca et al., « Energy Footprint of Blockchain Consensus Mechanisms Beyond Proof-of-Work », *in*: *arXiv preprint arXiv:2109.03667* (2021).

[106] Michael Bedford Taylor, « The evolution of bitcoin hardware », *in*: *Computer* 50.*9* (2017), pp. 58–66.

[107] *Telegraf - The plugin-driven server agent for collecting and reporting metrics.* https://github.com/influxdata/influxdb, Accessed: 2019-12-6.

[108] *Tendermint*, tendermint.com, Accessed: 2021-10-15.

[109] « Untangling Blockchain: A Data Processing View of Blockchain Systems », *in*: *IEEE Transactions on Knowledge and Data Engineering* 30.*7* (2018), pp. 1366–1385, ISSN: 10414347, DOI: 10.1109/TKDE.2017.2781227.

[110] Rafael Brundo Uriarte and Rocco DeNicola, « Blockchain-based decentralized cloud/-fog solutions: Challenges, opportunities, and standards », *in*: *IEEE Communications Standards Magazine* 2.*3* (2018), pp. 22–28.

[111] Harald Vranken, « Sustainability of bitcoin and blockchains », *in*: *Current opinion in environmental sustainability* 28 (2017), pp. 1–9.

[112] *Walmart, IBM and Tsinghua University Explore the Use of Blockchain to Help Bring Safer Food to Dinner Tables Across China*, https://newsroom.ibm.com/2016-10-19-Walmart-IBM-and-Tsinghua-University-Explore-the-Use-of-Blockchain-to-Help-Bring-Safer-Food-to-Dinner-Tables-Across-China, Accessed: 2021-09-16.

[113] Qin Wang et al., « SoK: Diving into DAG-based blockchain systems », *in*: *arXiv preprint arXiv:2012.06128* (2020).

[114] *Waves*, waves.tech, Accessed: 2021-10-15.

[115] Entriken William et al., *EIP-721: ERC-721 Non-Fungible Token Standard*, https://eips.ethereum.org/EIPS/eip-721, Accessed: 2020-12-04.

[116] Karl Wüst and Arthur Gervais, « Do you need a blockchain? », *in*: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, IEEE, 2018, pp. 45–54.

[117] Dylan Yaga et al., « Blockchain technology overview », *in*: *arXiv preprint arXiv:1906.11078* (2019).

[118] Michel Zade et al., « Is bitcoin the only problem? a scenario model for the power demand of blockchains », *in*: *Frontiers in Energy Research* 7 (2019), p. 21.

[119] Julian Zawistowski et al., *The Golem Project*, `https://golem.network/crowdfunding/Golemwhitepaper.pdf`, whitepaper, 2016.

[120] Fan Zhang et al., « {REM}: Resource-efficient mining for blockchains », *in*: *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 1427–1444.

[121] Zibin Zheng et al., « An overview on smart contracts: Challenges, advances and platforms », *in*: *Future Generation Computer Systems* 105 (2020), pp. 475–491.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Comprendre la consommation énergétique des blockchains : un regard sur les contrats intelligents

Mot clés : Chaînes de blocs, contrats intelligents, consommation énergétique, système distribués, systèmes pairs à pairs

Résumé : Les systèmes de chaînes de blocs sont des registres répliqués dans un réseau pair à pair. Elles ont connu un développement rapide depuis quelques années en s'illustrant dans de nombreux domaines d'activités. En permettant le traitement et la sauvegarde de données dans un contexte distribué et Byzantin, ces technologies ont le potentiel de modifier de nombreux secteurs. Par exemple, dans le cadre de la *finance décentralisée*, les cryptomonnaies se développement comme une alternative aux monnaies fiduciaires en proposant un système de paiement dépourvu de tiers de confiance. Cependant, une certaine inquiétude vis-à-vis de l'impact environnemental des chaînes de blocs a émergé en parallèle de leur développement. En particulier, de nombreuses recherches ont démontré le coût énergétique important des chaînes basées sur les preuves de travail. Dans cette thèse, nous proposons de contribuer à l'étude expérimentale du coût énergétique des solutions logicielles basées sur les chaînes de blocs. Face à l'enrichissement progressif de l'écosystème lié aux chaînes de blocs, nous proposons *BCTMark*, un nouvel outil de déploiement et d'évaluation des performances des chaînes de blocs. Partant de cet outil, nous concentrons notre étude sur l'impact des contrats intelligents sur la chaîne de blocs *Ethereum*. D'une part, nous proposons un modèle pour l'estimation du coût énergétique des contrats intelligents développé pour *Ethereum*. D'autre part, nous proposons un nouveau protocole pour l'identification et l'élimination des contrats non utilisés dans le but de proposer des chaînes de blocs plus frugales en calculs et espaces de stockages.

Title: Understanding the energy consumption of blockchains: a focus on smart contracts

Keywords: blockchain, smart contracts, energy consumption, distributed systems, peer to peer systems

Abstract: Blockchain systems are ledgers distributed in a peer-to-peer network. They have been developing rapidly over the past few years and have been used in many domains. By enabling the processing and storage of data in a distributed and Byzantine context, these technologies have the potential to change many sectors. For instance, in the context of decentralized finance, cryptocurrencies are developing as an alternative to fiat currencies by offering a payment system without relying on a trusted third party. However, some concerns about the environmental impacts of blockchains have emerged in parallel with their development. In particular, many studies have demonstrated the high energy cost of proof-of-work based blockchains. In this thesis, we propose to contribute to the experimental study of the energy cost of blockchain-based software solutions. Facing the progressive enrichment of the blockchain ecosystem, we propose *BCTMark*, a new tool for deploying and evaluating the performance of blockchains. Based on this tool, we focus our study on the impact of smart contracts on the blockchain *Ethereum*. On the one hand, we propose a model for estimating the energy cost of smart contracts developed for *Ethereum*. On the other hand, we propose a new protocol for the identification and elimination of unused contracts in order to propose blockchains that are more frugal in computation and storage space.