Models and Simulation of Blockchain Systems

Maher Alharby

School of Computing
Newcastle University

A thesis submitted for the degree of
Doctor of Philosophy

November 2020

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, foot notes,tables and equations and has fewer than 150 figures.

Acknowledgements

First and foremost, I would like to thank Allah for providing me with the strength, patience, knowledge and opportunity to complete my research. Without his blessings, this would not have been possible.

My sincere praises and acknowledgement are to my supervisor, prof. Aad van Moorsel, for his extraordinary supervision and guidance throughout my PhD journey. He was always there, reviewing my papers very carefully and offering me critical and useful feedback every time I consulted him. Without his great support, I would not be able to complete this research at its current level. I do admire his work attitude and professional skills, and to me, he has been a more than perfect supervisor during my life at Newcastle.

Special thanks go to my parents for supporting and encouraging me all the time since I was a little child, for my lovely wife for giving me all the support I need during this time, and for my two small children, Oday and Almas, who bring me the joys and happiness.

I would also like to thank my home country Saudi Arabia, the Ministry of Education and Taibah University for granting me a full scholarship to pursue my doctoral study. Finally, thanks to my supportive colleagues, co-authors of my papers and my friends both in Newcastle and back home.

Abstract

In the literature, there are a variety of proposed blockchain systems (e.g., Bitcoin and Ethereum), each of which with its own design decisions. Both in the design and the deployment of blockchain systems, many configuration choices and design decisions need to be made. Investigating different implementation and design choices is neither feasible nor practical on real blockchain systems. Simulation models emerge as an excellent technique to study blockchains without either implementing a new system or interrupting an existing one. Despite some attempts in the literature to utilise simulation models to evaluate specific aspects of blockchain systems, there is a lack of a general-purpose, flexible, extensible and widely usable simulation tool for blockchains.

In this thesis, we contribute to the field of blockchain analysis by proposing BlockSim as a generic framework to build discrete-event dynamic system models for blockchain systems. BlockSim aims to provide flexible and extensible simulation constructs to study a variety of blockchains and a set of design and deployment questions. BlockSim is implemented as a publicly available simulation tool and thoroughly validated against real-life systems and measurement studies.

Another contribution of this thesis is an extensive analysis to estimate the distributions for Ethereum smart contract using data for over 300,000 real transactions. To run realistic simulation studies, we integrate these distributions into the simulator to generate representative transactions.

Furthermore, this thesis offers two extensive data-driven simulation studies related to Ethereum smart contracts that demonstrate the applicability and usefulness of BlockSim. The first study is the analysis of the Ethereum Verifier's Dilemma and the proposal of two approaches (parallelisation and active insertion of invalid blocks) to mitigate its implications. The second study is the analysis of the uncertainty that miners face about the fee and cost of transactions and its impact on the received profits.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Introduction

A blockchain is a distributed ledger that records all transactions that have ever occurred in the blockchain network. This ledger is replicated and shared among the network's nodes. The main feature of blockchain is that it allows non-trusting participants to communicate and send transactions between each other in a secure way without the need for a trusted third party.

Blockchain is an ordered list of blocks, where each block is identified by its cryptographic hash. Each block references the block that came before it, resulting in a chain of blocks. Each block consists of a set of transactions. Once a block is created and appended to the blockchain, the transactions in that block cannot be changed or reverted. This ensures the integrity of the transactions and prevents the double-spending problem.

Blockchain systems typically run a peer-to-peer network that comprises a number of nodes, where every node is connected to several other nodes or peers. A company or a person can run a node of the peer-to-peer network. The role of such nodes is either to create and send new transactions in the network or to maintain the blockchain state by executing unconfirmed transactions and appending new blocks to the blockchain ledger. Nodes who maintain the blockchain state are often referred to as *miners*, and they are usually motivated to behave honestly through some incentives.

Cryptocurrencies emerged as the first generation of blockchain technology. Cryptocurrencies are digital currencies that are based on cryptographic techniques and peer-to-peer networks. The first and most popular example of cryptocurrencies is Bitcoin. Other blockchain systems such as Ethereum emerged as the second generation of blockchain to allow building complex distributed applications beyond the

cryptocurrencies. Smart contracts are considered as the main element of this generation [100]. Smart contracts are computer programs that can be enforced and executed on a blockchain system if the specified conditions are met, without the involvement of a trusted third party. The Ethereum blockchain is the most popular blockchain for developing smart contracts.

Every blockchain system in these generations has its own design decisions. During the design as well as the development of blockchain systems, many architectural, configurations and design decisions need to be made. However, imperfect design decisions may negatively impact the performance of the blockchain systems, requiring changes to the running system. For example, Bitcoin has been forked many times since its release in 2009 to alter some performance configurations such as the block size and the block interval time. Litecoin and Dogecoin are two blockchains that result from those forks, where the block interval has been reduced to improve the throughput of the blockchain. In addition, Ethereum also has been forked several times. In 2016, the Ethereum network was flooded by a Denial of Service (DoS) attack. As a response to this attack, Ethereum was forked to improve its incentive model.

To eliminate the need of changing the running system to fix imperfect design decisions, it is crucial to adequately investigate the different implementation and design choices prior to and during the design and the deployment of a blockchain system. Relying on experimentation or trial-and-error may lead to costly forks or is not even feasible or practical on real blockchain systems. Simulation is an excellent approach to study blockchains without either implementing a new system or interrupting an existing one [19]. With simulations, it is feasible for designers and analysts to explore design trade-offs and configuration questions for blockchains in a reasonable time. Despite the importance of simulation to the field of blockchain analysis, there is a lack of a generic and extensible simulation framework for performance analysis of blockchain systems.

This research aims to advance the current state of the art of blockchain analysis by addressing several shortcomings of the current literature through the utilisation of discrete-event simulation models. That is, the goal of this thesis is to address the following limitations:

Lack of a generic and extensible simulation tool for blockchains. In the literature, there are some attempts to utilise simulation models to evaluate various aspects of blockchain systems. However, all of these attempts utilise simulation models for specific and limited purposes [7, 16, 50, 51, 75, 78, 91, 102]. Thus, there is a

lack of a general-purpose, flexible, extensible and widely usable simulation tool for blockchains, to assist in answering a variety of design and deployment questions.

The core aim of this research is to address this gap by proposing a simulation framework and tool for blockchain systems that can be applied to a large set of blockchain systems, easily manipulated to study a particular system while at the same time easy to use and extend. The intended user of the simulator can be blockchain designers, analysts and researchers who want to analyse a large variety of blockchain systems. The proposed simulator can provide means to rigorously analyse performance problems and study various solutions to mitigate their implications. Examples of such analysis problems are the Verifier's Dilemma and the profit uncertainty presented in Chapters 6 and 7 of this thesis, respectively.

Lack of rigorous analysis of the Ethereum Verifier's Dilemma. In proof-of-work based blockchains such as Ethereum, verification of blocks is an integral part of establishing consensus across nodes. However, in Ethereum, miners do not receive a reward for verifying. This implies that miners face the Verifier's Dilemma: use resources for verification, or use them for the more lucrative mining of new blocks?. This Verifier's Dilemma is well recognised in the literature, e.g., [71, 94], but has not been systematically analysed.

This research aims to fill this gap through a rigorous analysis of the implications of the Verifier's Dilemma with the help of the proposed simulator in addition to some closed-form expressions. The analysis considers both current and future implementations of Ethereum to derive conclusions of when this dilemma can become a serious problem. The aim is not only to analyse the implications of this dilemma, but also to propose some mitigation solutions. To conduct this analysis study in a realistic setting, we feed the simulator with real data about Ethereum smart contract transactions, that will be gathered from the Ethereum network and from controlled experiments.

Lack of analysis of the impact of the profit uncertainty in Ethereum. In Ethereum, miners are uncertain about the rewards and the cost of executing smart contract transactions. This makes miners unable to make informed decisions about which transactions to select and execute in their forthcoming blocks in order to maximise their profits. This is especially true with the presence of incompatible incentive model, as reported in the literature [1, 2, 29, 83, 101]. To the best of our knowledge, there is no work that investigates the impact of the uncertainty problem miners face when selecting transactions on the profit earned under the incompatible Ethereum incentive model.

This research aims to bridge this gap by providing an extensive analysis of the impact of the uncertainty miners face in Ethereum when selecting transactions with the help of the proposed simulator. To accomplish this aim, different transaction selection strategies must be designed to draw conclusions about the impact of such uncertainty. Similar to the analysis of the Verifier's Dilemma, this analysis aims to consider both current and future implementations of Ethereum after feeding the simulator with real data to draw realistic results and conclusions.

1.1 Contributions

The work carried out in this PhD research makes a number of contributions to the field of blockchain analysis. The main contributions of this research are as follows:

- Conducting a systematic mapping study to explore the current research on blockchain-based smart contracts. The study aims to provide a survey of the scientific literature, identify academic research trends and uptake and identify gaps for further research. We conducted the first study in 2017, then updating it in 2018 before revising it further in 2020. The results of the mapping study show a significant growth in research outputs related to smart contracts since its emergence in 2014, with an emphasis on smart contract applications. Also, the results suggest two research areas to be further explored, which we utilise to identify the case studies for the proposed simulator in this thesis **(Chapter 3)**.

- Designing and developing a generic blockchain simulation tool named `BlockSim` that is flexible enough to support the analysis of a large variety of blockchains and a wide set of analysis problems. BlockSim is designed to cross three different blockchain layers: the incentives layer, the consensus layer and the network layer [96]. BlockSim provides simulation constructs that are intuitive, hide unnecessary detail and extensible. At the core of BlockSim is a Base Model that contains a number of main functional blocks common across blockchains (e.g., blocks and nodes) that can be extended as required. We implement the Base Model in Python, extend it to support the implementation of Bitcoin and Ethereum blockchains and validate it against real-life systems and measurement studies from the literature. The code of the BlockSim simulator is publicly available[1] **(Chapter 4)**.

[1] https://github.com/maher243/BlockSim.

- Conducting an extensive analysis to estimate the distributions for Ethereum smart contract transactions, with respect to different attributes. To determine these distributions we use publicly available Ethereum smart contract information, augmented with experimental data for over 300,000 smart contract transactions obtained on a test bed. The estimated distributions are then fed as inputs to the BlockSim simulator to conduct data-driven simulation studies **(Chapter 5)**.

- Conducting an extensive analysis of the Ethereum Verifier's Dilemma, using a data-driven model-based approach that combines closed-form expressions and discrete-event simulation. We extend the BlockSim simulator with the functionality necessary for this analysis. We show that, indeed, it is often economically rational not to verify. We consider two approaches to mitigate the implications of the Verifier's Dilemma, namely parallelisation and active insertion of invalid blocks, both shown to be effective **(Chapter 6)**.

- Conducting an extensive analysis of the impact of the uncertainty miners perceive in Ethereum when selecting transactions, using data-driven and simulation approaches. We design different transaction selection strategies for scenarios with and without uncertainty. We conduct this analysis using the BlockSim simulator after extending some of its functionalities. We show that such uncertainty has a significant impact on the earned profits **(Chapter 7)**.

1.2 List of Publications

Chapters 3,4,5,6 and 7 have already been published in international conferences and journals. For papers 2 and 6, which have additional co-authors, we note that the majority of the work was done by the researcher, including the idea, running the experiments and writing the paper. Co-authors of these two papers contributed through some discussions and through revising the paper, editing and providing comments. The following is a list of these publications.

1. **Maher Alharby**, and Aad van Moorsel. **"Blockchain-based smart contracts: A systematic mapping study"**. In Proceedings of the 4th International Conference on Computer Science and Information Technology. AIRCC Publishing Corporation, 2017. (Chapter 3)

2. **Maher Alharby**, Amjad Aldweesh, and Aad van Moorsel. **"Blockchain-based smart contracts: A systematic mapping study of academic research (2018)"**. In Proceedings of the 2018 International Conference on Cloud Computing, Big Data and Blockchain. IEEE, 2018. (Chapter 3)

3. **Maher Alharby**, and Aad van Moorsel. **"The impact of profit uncertainty on miner decisions in blockchain systems"**. Electronic Notes in Theoretical Computer Science, 2018. (Chapter 7)

4. **Maher Alharby**, and Aad van Moorsel. **"BlockSim: A Simulation Framework for Blockchain Systems"**. ACM SIGMETRICS Performance Evaluation Review, 2019. (Chapter 4)

5. **Maher Alharby** and Aad van Moorsel. **"BlockSim: An Extensible Simulation Tool for Blockchain Systems"**. Frontiers in Blockchain, 2020. (Chapter 4)

6. **Maher Alharby**, Robin Lunardi, Amjad Aldweesh, and Aad van Moorsel. **"Data-Driven Model-Based Analysis of the Ethereum Verifier's Dilemma"**. In Proceedings of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. IEEE, 2020. (Chapter 6)

7. **Maher Alharby** and Aad van Moorsel. **"Fitting and Regression for Distributions of Ethereum Smart Contracts"**. In Proceedings of the 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services. IEEE, 2020. (Chapter 5)

In addition to these publications, I am a co-author of some research papers that do not contribute directly to this thesis, but they are relevant. The following is a list of these papers:

8. Amjad Aldweesh, **Maher Alharby**, and Aad van Moorsel. **"Performance benchmarking for smart contracts to assess miner incentives in Ethereum"**. In Proceedings of the 14th European Dependable Computing Conference. IEEE, 2018.

9. Amjad Aldweesh, **Maher Alharby**, Aad van Moorsel. **"Performance benchmarking for Ethereum opcodes"**. In Proceedings of the 15th International Conference on Computer Systems and Applications. IEEE, 2018.

10. Amjad Aldweesh, **Maher Alharby**, Maryam Mehrnezhad and Aad van Moorsel. **"OpBench: A CPU Performance Benchmark for Ethereum Smart Contract Operation Code"**. In Proceedings of the 2019 IEEE International Conference on Blockchain. IEEE, 2019.

1.3 Thesis Structure

- **Chapter 2. Background:** This chapter introduces background information related to blockchain technology, smart contracts, modelling and simulation. In addition, this chapter explains discrete-event simulations as well as the methodology for conducting a simulation study. This chapter also discusses related work on performance evaluation of blockchain systems.

- **Chapter 3. Blockchain-based smart contracts: a systematic mapping study of academic research:** This chapter applies the systematic mapping study approach to explore the current research on smart contracts and to identify gaps for further work.

- **Chapter 4. BlockSim: an extensible simulation tool for blockchain systems:** This chapter introduces BlockSim as a generic simulation model and tool for blockchain systems. This embraces the design, the implementation and the validation of the simulation tool. This chapter also provides a simulation study to show the applicability of the proposed simulator.

- **Chapter 5. Data collection and distributions of Ethereum smart contracts:** This chapter introduces the data collection exercise conducted to gather Ethereum smart contract transactions and the analysis performed to obtain distributions for this data. The distributions are meant to serve as inputs for the simulator.

- **Chapter 6. Analysis of the Ethereum Verifier's Dilemma:** This chapter introduces an extensive simulation study to investigate the Ethereum Verifier's Dilemma and two approaches to mitigate its implications. This embraces the description of the Verifier's Dilemma as well as some closed-form expressions to estimate its implications. This chapter also extends the simulator to support the analysis of the Verifier's Dilemma and the proposed mitigation approaches. In addition, it discusses the main results obtained and the validity threats to the study.

- **Chapter 7. Analysis of the profit uncertainty in Ethereum:** This chapter introduces a simulation study to investigate the uncertainty miners face when selecting transactions, and its impact on the received profits. This embraces the description of the uncertainty issue, the design of a simulation model as well as the design of different selection strategies. This chapter also extends the simulator to support this study. In addition, it presents the main insights gained and discusses the threats to the validity of the study.
- **Chapter 8. Conclusion:** This chapter summarises the main contributions and findings of our research. In addition, it presents the limitations of our research that can be considered in future work.

Chapter 2

Background

Summary

This chapter covers background information about blockchain technology and simulations. Sections 2.1-2.4 are meant to explain the blockchain technology. Section 2.1 provides an overview of blockchain technology: its definition, characteristics and how it works. Section 2.2 presents and discusses three different layers of blockchain systems: Network, Consensus and Incentives. Section 2.3 gives an introduction about smart contracts and blockchain platforms that support the development of smart contracts. Ethereum blockchain is explained in more detail in Section 2.4. Basic simulation concepts are explained in three Sections 2.5-2.7. Section 2.5 provides an overview of modelling and simulation in general, and a detailed explanation about discrete-event simulations is provided in Section 2.6. Section 2.7 concerns about the steps of a simulation study. Finally, we discuss work related to performance evaluation and simulation tools for blockchains in Section 2.8 before we conclude the chapter in Section 2.9.

2.1 Blockchain Technology

Transactions between parties in current systems are usually conducted in a centralised form, which requires the involvement of a trusted third party (e.g., a bank). However, this could result in security issues (e.g., unauthorised modifications), reliability issues (e.g., single point of failure), performance bottlenecks at the central parties and high transaction fees. Blockchain technology has emerged to tackle these issues by allowing non-trusting entities to interact with each other in a distributed manner without the involvement of a trusted third party.

A blockchain is a distributed ledger that maintains the history of all transactions that have ever occurred in the blockchain network. All network nodes have a copy of this ledger making it replicated and backed up. One advantage of a blockchain is that it allows non-trusting participants to communicate and to securely exchange assets without the need for a trusted third party. Such a ledger has two main purposes, to provide an immutable log of all transactions and to make the transactions visible to anyone inspecting or using the blockchain.

The term blockchain comes from the fact that data about multiple transactions is grouped into blocks. Each block is uniquely identified by its cryptographic hash, and each block is attached and linked to the one that came before it. This results in a chain of blocks. Once a block is generated and attached to the blockchain ledger, the transactions in that block cannot be modified by any node, since it would require the node to rewrite all subsequent blocks. This makes blockchain systems immutable and protected against double-spending attacks [6].

Cryptocurrencies have emerged as the first generation of blockchain technology. Cryptocurrencies are digital currencies that are based on cryptographic techniques and a peer-to-peer network. The first and most popular example of cryptocurrencies is Bitcoin. Bitcoin [77] is an electronic payment system that allows non-trusting parties to transact digital money with each other in a secure manner without going through a middleman (e.g., a bank). However, Bitcoin has limited programming capabilities to support complex transactions. Bitcoin, thus, does not support the creation of complex distributed applications on top of it. Blockchain-based smart contract systems (e.g., Ethereum [25, 49], Hyperledger Fabric [13] and NEO [31]) have then emerged to permit complex distributed applications through smart contracts. A smart contract is basically a computer program that can be attached to the blockchain. It embraces some contractual clauses and it is enforced by the consensus algorithm.

2.1.1 Characteristics and Taxonomy of Blockchain Systems

There are several characteristics or features that can be considered when designing a new blockchain platform or when adopting an existing one. These characteristics include, but not limited to, the following items:

- **Network Permission.** There are two types of blockchains in terms of network permission, which are public and private [99]. In a public (permissionless) blockchain, users or nodes do not need to be trusted or even known to each other. That is, everyone can join the network, read the content of the blockchain, send

a new transaction, write new blocks to the ledger or verify the correctness of the blocks. This, however, comes at the cost of performance (low throughput and high transactions latency), as the network can accept a high number of non-trusted participants. The consensus algorithm (e.g., Proof-of-Work and Proof-of-Stake) in public blockchains should be costly as the network opens for everyone, including malicious writers (see Section 2.2.2). Examples of public blockchains are Bitcoin, Zcash and Ethereum.

In a private (permissioned) blockchain, only users with permissions can join the network, write or send transactions to the blockchain. A company or a group of companies are usually responsible for giving users such grants before joining the network. The consensus algorithm (e.g., Byzantine Fault Tolerance) in private blockchains is often much more efficient as most of the participants in the system are trusted. That is, private blockchains achieve better performance compared to public blockchains. Examples of private blockchains are Hyperledger Fabric and R3 Corda. The focus of this thesis will be on public blockchains.

- **Consensus Protocol.** The consensus protocol of blockchains plays a significant role as it defines the rules that can be followed by the network's nodes to agree on one global blockchain state. It also resolves potential conflicts that may occur due to network latency (see Section 2.2.2). There are many consensus protocols that have been proposed in the literature for both permisionless and permissioned blockchains, including Proof-of-Work (PoW), Proof-of-Stake (PoS), Proof-of-Service, Byzantine Fault Tolerance (BFT) and its variants, Proof of Elapsed Time and other protocols. Among those protocols, PoW is currently the most common protocol used in permissionless blockchains, while BFT is common for permissioned blockchains. We refer to [21, 97] for more detail about consensus protocols in blockchains.

- **Cryptocurrency Support.** A cryptocurrency is a form of digital or virtual currency that uses strong cryptography to securely create new coins and transfer their ownership. It usually works through a blockchain that runs on a distributed peer-to-peer network to maintain transaction records without involving a central authority. It is worth noting that not all blockchains consider cryptocurrencies in their design, and not every cryptocurrency must operate on a blockchain.

- **Smart Contract Support.** Smart contracts are computer programs running on the blockchain network to facilitate and enforce the agreement between participants without the involvement of a trusted third party. Smart contracts can be applied to various applications including Internet of Things (IoT). We refer to Section 2.3 and Chapter 3 for more detail about smart contracts and their applications. Ethereum is the most common permissionless platform for smart contracts, while Hyperledger is a popular permissioned platform for smart contracts.

- **Privacy Support.** Privacy and anonymity of transactions are desirable in some domains that involve sensitive data. In most blockchain systems such as Bitcoin and Ethereum, every node in the network is identified by a unique address (i.e. a public key) instead of the real identity. However, this does not really preserve the privacy of transactions as it is possible to link and trace all transactions submitted by a node through inspecting the address of the node [106]. There are some privacy-preserving blockchain platforms that were particularly designed to make it infeasible to trace the sender and the recipient of transactions as well as the amount of currencies to be transferred. Examples of these platforms are Dash, Zcash and Monero.

Blockchain Platform	**Network Permission**	**Consensus Protocol**	**Cryptocurency Support**	**Smart Contract Support**	**Privacy Support**
Bitcoin	Permissionless	PoW	Yes	No	No
Ethereum	Permissionless	PoW/PoS (Casper)	Yes	Yes	No
Zcash	Permissionless	PoW	Yes	No	Yes
Litecoin	Permissionless	PoW (Scrypt hash algorithm)	Yes	No	No
Dash	Permissionless	PoW/Proof-of-Service	Yes	No	Yes
Peercoin	Permissionless	PoS	Yes	No	No
Ripple	Permissionless (controlled)	RPCA	Yes	No	No
Monero	Permissionless	PoW	Yes	No	Yes
MultiChain	Permissioned	Round-Robin (mining diversity)	No	No	No
Hyperledger	Permissioned	Various protocols (e.g., Kafka, BFT and Proof of Elapsed Time)	No	Yes	No

Table 2.1: Different blockchain platforms and their characteristics.

In the literature, there are a large number of proposed blockchain platforms, each with its own characteristics and design decisions. For instance, some platforms are designed specifically to support rich and complex smart contracts (e.g., Ethereum and Hyperledger), while others are designed to preserve the privacy of users data (e.g., Zcash and Monero). We list the most ten open-source popular blockchain platforms [62] and their characteristics, as depicted in Table 2.1. We note that all

these platforms support cryptocurrencies, apart from Hyperledger and MultiChain as they are platforms for creating and deploying private blockchains.

Bitcoin. The first and most widely known permissionless blockchain platform that is designed as a distributed cryptocurrency system without any central authority. Bitcoin proposes PoW as its consensus protocol to reach a global blockchain state in a non-trusted environment. PoW requires performing computationally expensive computations to maintain the blockchain state, thus it is not an energy-efficient protocol. After the success of Bitcoin, many other alternative cryptocurrencies have been proposed, some with other consensus protocols.

Ethereum. A permissionless blockchain platform that is mainly designed to support the creation and the deployment of complex smart contracts on top of blockchains. Ethereum currently adopts the PoW consensus protocol, aiming to switch to PoS Casper [26] in future. This thesis mainly focuses on the Ethereum blockchain, thus we provide more detail about it in Section 2.4.

Zcash. A permissionless blockchain platform that is designed to preserve the privacy of transactions. Zcash aims to add privacy and anonymity to transactions submitted in the network by using a zero-knowledge proof algorithm called zk-SNARK. Data privacy is highly desirable in some domains with sensitive data such as healthcare. Similar to Bitcoin, Zcash uses the PoW consensus protocols.

Litecoin. A permissionless blockchain platform and a distributed cryptocurrency. Litecoin is based on the Bitcoin software with some modifications. Compared to Bitcoin, Litecoin provides fast transaction confirmation time (four times faster than Bitcoin). In Litecoin a block of transactions is created every 2.5 minutes, instead of every ten minutes as the case in Bitcoin. Litecoin uses the Bitcoin PoW protocol, but with the "Scrypt" hash algorithm that can mitigate the mining centralisation issue [62]. Litecoin can be of interest to domain applications that require fast processing time for transactions and data in a permissionless network.

Dash. A permissionless blockchain platform that provides a privacy-preserving cryptocurrency. Dash, similar to Bitcoin, uses the PoW protocol to secure the network. It, however, improves the Bitcoin network by introducing an additional network tier that is powered by a set of *masternodes*. Those masternodes assure the privacy of transactions through "PrivateSend" and instant transaction validation through "InstantSend" [62]. Dash uses Proof-of-Service protocol to reward those masternodes for their efforts. That is, Dash can be useful for domain applications that require both privacy of transactions and fast transaction validation time.

Peercoin. A permissionless blockchain platform that aims to eventually replace the PoW protocol by a more energy efficient protocol such as PoS. The Bitcoin PoW algorithm requires high energy consumption. Peercoin is the first platform that proposed the use of PoS protocol, by combining it with the PoW protocol [97]. It proposes a metric called "coinage" to determine the stake for nodes in the network [97].

Ripple. A permissionless, but controlled, blockchain platform that acts as a cryptocurrency and a global settlement network for financial transactions. Ripple uses the Ripple Protocol Consensus Algorithm (RPCA), a round-based protocol that is executed by a number of selected validating servers. Since not every node can become a validator, Ripple provides fast and instant payment and is considered as a controlled permissionles network [62].

Monero. A permissionless blockchain platform and privacy-preserve cryptocurrency. It is based on the Cryptonote protocol that comes with a ring signature algorithm (a kind of zero-knowledge proof) that hides the identity of both the sender and the recipient of transactions [76]. Also, it adopts a stealth (one-time) address for transactions, which means that every transaction submitted by a particular node is signed by a different address [62]. This makes it hard to link and trace transactions sent by a particular node. Monero currently uses the Bitcoin PoW algorithm. Similar to Zcash and Dash, it is useful for domain applications where data confidentiality is a requirement.

MultiChain. A blockchain platform that aims to provide certain features to create and deploy permissioned blockchains either within or between organisations [41]. MultiChain does not consider computationally intensive protocols such as PoW since it operates in a private and controlled network. Instead, it implements a round-robin consensus protocol that identifies a set of miners in the network and controls the number of blocks contributed by a miner in a particular time window through a parameter called mining diversity [41].

Hyperledger. An open-source collaborative project that aims to advance permissioned blockchains. It aims to provide an infrastructure of different modules (i.e., smart contract engines) and tools for developing blockchain platforms. Different permissioned blockchain systems have been implemented as variants of the Hyperledger project, including Fabric, Iroha, Sawtooth and Indy. Each of these variants has its own design and consensus protocol. Fabric, which is the first extensible blockchain system for running smart contracts, uses a consensus protocol named Kafka [13].

2.1.2 How Do Blockchains Work?

Blockchain systems rely on a peer-to-peer network, where each node in the network is connected to several different peers. Nodes are responsible for maintaining the blockchain ledger by continuously appending new blocks, and they are often referred to as *miners*. In public blockchains, any node can be a miner, while it is not the case in private blockchains.

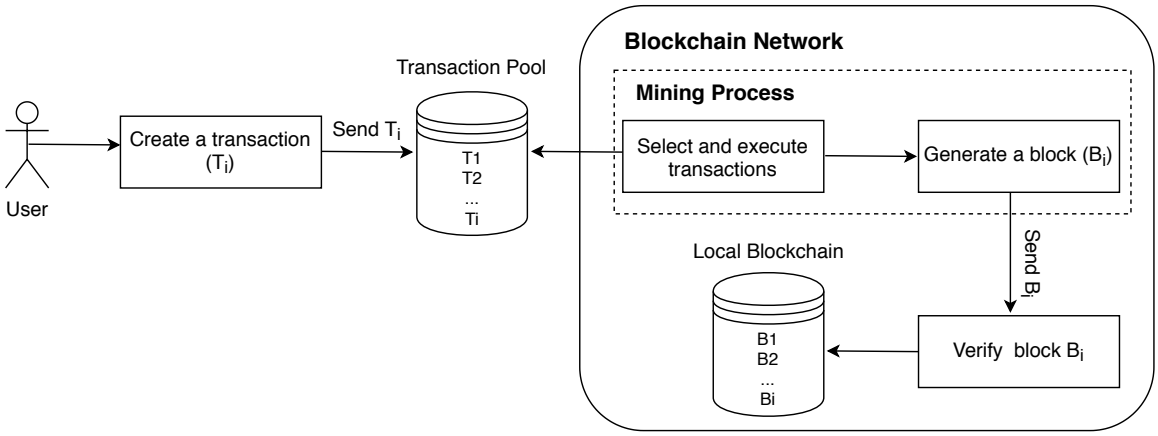

Figure 2.1: The working mechanism of blockchain systems.

Figure 2.1 illustrates the working mechanism of blockchain systems. Users, aka nodes, can create a transaction and then propagate it to the blockchain network.

In the blockchain network, a set of miners are responsible for maintaining the ledger by continuously appending new blocks of transactions. Every miner maintains a transaction pool to keep pending transactions (uncommitted transactions) submitted by nodes in the network. To generate and attach a new block to the ledger, miners first select and execute a number of pending transactions from their pools and then include them in the block by participating in a consensus algorithm such as PoW. The generated block will then be propagated to other nodes in the network. The process of appending new blocks is usually referred to as the *mining process*. Upon receiving the newly generated block, every node has to verify the block before adding it to its local copy of the blockchain. Verifying a block requires checking the correctness of the block construction (e.g., PoW verification) as well as verifying all transactions embedded in the block by executing them. This is to verify the miner's work, and it is often referred to as the *verification process*.

If the majority of the nodes in the network accepted the block, appended it to their local blockchain copies and built upon it, the block will be confirmed and considered

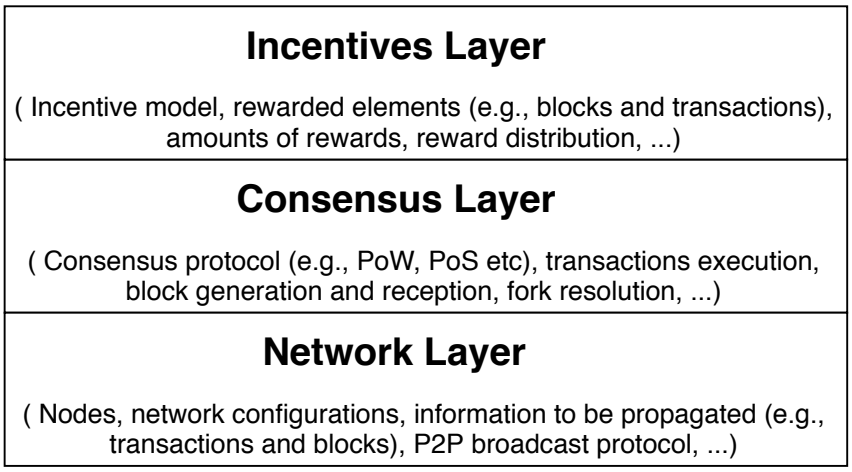

Figure 2.2: Blockchain System Layers.

as part of the global blockchain ledger. The miner that created the block will then receive some reward for its efforts.

2.2 Blockchain Layers

In this section, we present three layers of blockchain systems, namely, network, consensus and incentives, as depicted in Figure 2.2. We will utilise these layers to structure our simulator in Chapter 4. The network layer captures the network's nodes and the underlying broadcast protocol to propagate information between nodes. The consensus layer captures the algorithms and rules adopted to reach an agreement about the current state of the blockchain ledger. The incentives layer captures the economic mechanisms adopted by a blockchain to issue and distribute rewards among the participating nodes.

Before going through these layers in more detail, we list and briefly define the main entities at each layer and the relationship among them. In the network layer, there are two entities, which are *Node* and *Broadcast protocol*. The *Node* entity is the main entity in any blockchain whose role is to participate in the network by creating and submitting new transactions and to maintain the blockchain state by executing transactions and attaching new blocks to the blockchain ledger through engaging in a consensus algorithm. The *Broadcast protocol* entity is responsible for managing the propagation of information entities (*Blocks* and *Transactions*) in the network.

In the consensus layer, there are information entities (*Blocks* and *Transactions*) and blockchain state entities (*Blockchain ledger* and *Transaction pool*). The consensus entities are all maintained by the network nodes. For example, nodes are responsible

for creating transactions and blocks as well as expanding the ledger and the pool every time a new block or transaction arrives in the network. Blocks and transactions play a significant role in updating the blockchain state as follows. The arrival of a new transaction results in updating the system state by inserting the transaction into the transaction pool in order for the nodes to execute it in their forthcoming block. Similarly, the arrival of a new block results in updating the system state by attaching the block to the blockchain ledger.

The incentives layer has the *reward* entity. In blockchain systems that offer incentives, nodes which maintain the system state by attaching new blocks to the blockchain ledger are rewarded for their efforts. The rewarded elements (e.g., blocks and transactions), as well as the calculation and the distribution of such rewards, are the actions that the incentive model determines.

2.2.1 Network Layer

The network layer in blockchain systems contains the nodes in the network, their geographical and relative locations and the connectivity among them. It defines which information is to be propagated as well as the mechanism to propagate such information.

The main constituent in the network layer is a *node*. A node can be an ordinary user who wants to create and submit a transaction to be executed and included in the ledger or a special node, known as *miner*, who maintains and expands the ledger by appending new blocks. A node has a unique identifier and maintains its balance, a local copy of the blockchain ledger and, if the node is a miner, an individual transaction pool. The transaction pool keeps the pending transactions received from other nodes in the network.

Nodes communicate the following information to each other. If a node generates a new transaction, it cryptographically signs it and propagates it to its peers to have it confirmed and recorded in the blockchain ledger. In case the node is a miner, every time it generates a block, it notifies its peers so they can validate it and append it to their copies of the ledger.

As information propagation mechanisms for blockchains, several protocols have been proposed, including relay networks and advertisement-based protocols [50]. In the advertisement-based protocol used in most blockchains [50], the node sends a notification to its peers about the new data (e.g., a transaction). If the recipient node responds by requesting the data, the node will send it. Otherwise, the node will not send it as the recipient node has already had the data.

2.2.2 Consensus Layer

The consensus layer in blockchain systems defines the algorithms and rules for reaching an agreement about the blockchain's state among the network's nodes. Such rules specify which node is eligible for generating and appending the next block to the blockchain ledger, how often blocks are generated as well as how to resolve potential conflicts that may occur when nodes have multiple, differing copies of the ledger.

There are several consensus algorithms such as Proof-of-Work (PoW) and Proof-of-Stake (PoS) that have been proposed for blockchain systems. In PoW, nodes (i.e., miners) invest their computing power to maintain the ledger by attaching new blocks, while in PoS, nodes invest their stake or money. Regardless of what is required to be invested by the nodes, the intuition behind such algorithms is to introduce a cost for maintaining the ledger. The cost introduced should be more than enough to deter nodes from behaving maliciously [97]. At the same time, nodes are only rewarded for their efforts if they follow the rules and maintain the ledger honestly (see Incentives Layer).

To illustrate the consensus layer, we discuss the PoW algorithm here as it is the most common algorithm for permissionless blockchains, used by Bitcoin and Ethereum. In PoW, the computing power invested by a miner determines how frequently that miner will generate and append blocks to the blockchain ledger. To generate a block, the miner has to repeatedly try nonces (random numbers) until the hash of the nonce combined with the block information will be within a certain threshold (referred to as the block difficulty). The only way to find the nonce is by trial-and-error, and thus, the more hash power invested by a miner, the more likely that miner will find the nonce. This process is a competitive task since all miners in the network are competing against each other to find the desirable hash value of the next block. Note that the block difficulty can be dynamically adjusted to control how often blocks are generated.

Due to the delay incurred by propagating blocks between nodes in the network (see Network Layer), other nodes might generate the next block before hearing of another competitive block that has been recently announced. This leads to conflicts, known as forks, which occur when nodes have multiple, differing views of the ledger. The task of the consensus layer in blockchain systems is to resolve such conflicts. Different consensus algorithms use different rules to select which blockchain (fork) should be accepted as the global chain. For example, the PoW algorithm used by Bitcoin and Ethereum resolves the conflicts by adopting the longest chain, as in Figure 2.3. Other

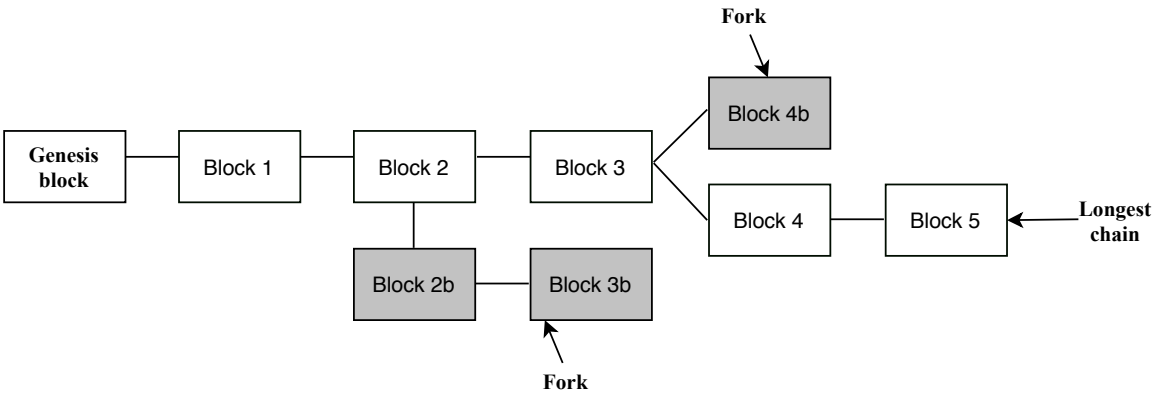

Figure 2.3: The longest-chain rule for fork resolution. Blocks 2b, 3b and 4b are discarded as stale blocks.

proposals such as GHOST [90] select the fork with the heaviest work. Blocks that are not considered in the global chain (represented by a grey background colour in Figure 2.3) will be discarded. These blocks are usually referred to as stale (orphan) blocks, and they do not contribute to the state of the ledger.

2.2.3 Incentives Layer

The incentives layer utilises the blockchain's cryptocurrency to establish an incentive structure, distributing rewards among the participating miners who maintain the blockchain's ledger. The incentive model is essential to maintain any permissionless blockchain system. Incentives should compensate miners fairly for their work and motivate them to behave honestly [3, 5]. The incentives also protect the blockchain system from various attacks (e.g., DDoS attacks in Ethereum [24]) and against malicious behaviours of the nodes (e.g., selfish mining strategies [44]).

In most blockchain systems, rewards are associated with generating blocks and completing transactions, called block reward and transaction fees, respectively. Depending on the chain, there are subtle differences in what is rewarded, e.g., Ethereum also issues a reward for stale (or uncle) blocks, even if they do not make it into the blockchain when conflicts are resolved. When a miner receives a reward (e.g., through appending a new block to the ledger), its balance will increase accordingly. The block reward, in all known blockchains, is set to a fixed amount, while the transaction fee is calculated as a variable amount of cryptocurrencies, depending on effort as well as the prize a transaction submitter is willing to pay.

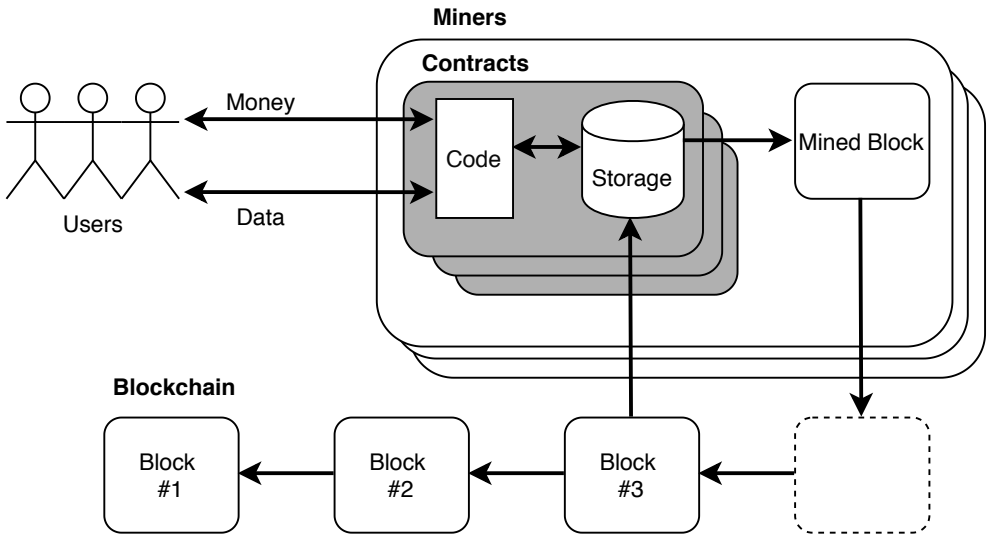

Figure 2.4: Smart contract system [36].

2.3 Smart Contracts

The theoretical concept of smart contracts was first introduced by Szabo in 1994 [93] as "a computerised transaction protocol that executes the terms of a contract". Szabo proposed translating contractual clauses into computer code that can be self-enforced, without the involvement of trusted intermediaries. This concept became a practical reality with the evolution of blockchain technology. Within the blockchain context, a smart contract is a computer program stored on the blockchain. A smart contract acts as a trusted third party between non-trusting participants, and it consists of contract storage, a balance, and program code, as depicted in Figure 2.4. It can be created and made available for use by any node in the network, simply through posting a transaction to the blockchain. Smart contract program code is fixed and cannot be updated once included in the blockchain.

Smart contracts are run by a network of miners who are responsible for maintaining the blockchain. Miners reach consensus on the execution outcome of the smart contract and accordingly update the blockchain. Once deployed, each smart contract is assigned to a unique address and is executed whenever a transaction is created using this address. During the execution of the smart contract, its storage might be updated (i.e., reading from or writing to the storage). Also, a smart contract can exchange cryptocurrency between users. Moreover, a smart contract can invoke and create another smart contract by posting a message, which is not recorded in the blockchain. This message is used by smart contracts either for creating a new smart

contract or for calling functions in other smart contracts.

There are two types of smart contracts, namely, deterministic and non-deterministic smart contracts [30]. A deterministic smart contract is a smart contract that when it is run, it does not require any information from an external party (from outside the blockchain). A non-deterministic smart contract is a contract that depends on information (called oracles or data feeds) from an external party such as the current weather information, which is not available on the blockchain.

Smart Contract Platforms. Smart contracts can be developed and deployed in different blockchain platforms (e.g., Ethereum and Hyperledger Fabric). Different platforms offer different features for developing smart contracts. Here, we will only focus on three platforms, which are Bitcoin, Ethereum and Hyperledger Fabric.

Bitcoin [77] is a public blockchain platform that uses a stack-based bytecode scripting language that is very limited in terms of computing capability [67]. Bitcoin scripting language cannot support the creation of complex smart contracts that contain rich logic. For instance, writing a contract that supports loops or withdrawal limits is not feasible in Bitcoin due to the limitations of its scripting language [25].

Ethereum [25, 49] is also a public blockchain platform that addresses the limitations of the Bitcoin's scripting language by leveraging a Turing-complete language that can support complex applications based on smart contracts on the blockchain (see Section 2.4 for more details).

Bitcoin and Ethereum have scalability challenges in that at most tens to hundreds transaction per second can be processed. Hyperledger Fabric [13] is a private blockchain that seeks to overcome these challenges. Hyperledger Fabric employs a traditional Byzantine fault-tolerant consensus protocol, instead of the Proof-of-Work protocol used in public blockchains. Hyperledger's smart contract technology is called chaincode. It consists of the code that is deployed and executed on the blockchain, the state database (key/value store) and the mining (endorsement) policies.

Smart Contract Applications. There is a multitude of blockchain-based smart contract applications in the literature. According to the survey presented in Chapter 3, research on smart contract applications accounts for 64% of all scientific papers published about smart contracts. Smart contracts have been applied to various domains and topics that rely on a trusted third party, including Internet of Things, cloud computing, healthcare, access control and authentication, voting systems and

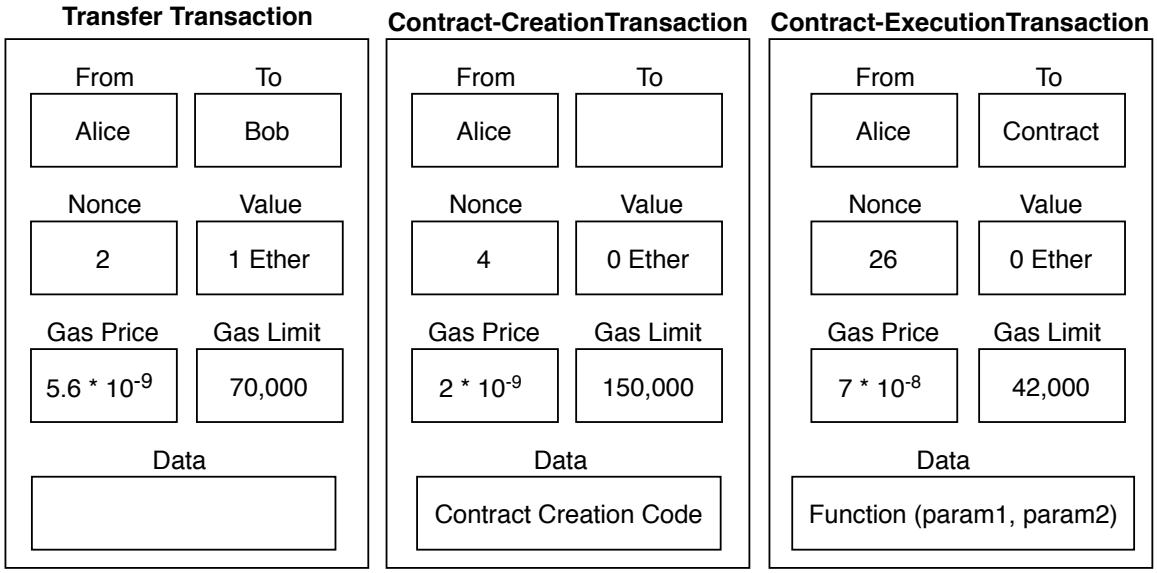

Figure 2.5: Examples of all different types Ethereum transactions.

other applications. In Chapter 3, we present the survey we conducted to explore smart contract topics, with a detailed discussion about smart contract applications.

2.4 Ethereum

Ethereum is the second most popular public blockchain after Bitcoin. The currency of Ethereum is referred to as Ether. Ethereum supports advanced and customised smart contracts with the help of a Turing-complete programming language. Ethereum smart contracts are run within the Ethereum Virtual Machine. Using Ethereum, distributed applications can be developed using different high-level programming languages such as Vyper and Solidity.

In this section, we discuss different aspects of the Ethereum blockchain such as accounts, transactions, blockchains and incentive model. We mainly focus on the aspects that are relevant to the work described in this thesis.

Ethereum account. There are two types of accounts (users), namely, *externally owned accounts (EOAs)* and *contract accounts*. Each account has a unique 160-bits address to be called at as well as a balance. Unlike EOAs, contract accounts have some associated code and a storage space additionally. Different accounts can interact with each other through transactions.

Ethereum transactions. Transactions play an important role in updating the blockchain's state. In Ethereum, transactions are executed in sequence. That means the final blockchain state depends on the order of the transactions. There are two types of transactions, namely, transfer and contract transactions. The former is to move Ether (the Ethereum cryptocurrency) between accounts, while the latter is to either publish a new smart contract to the blockchain or to invoke an existing one.

To publish a new contract, a contract-creation transaction containing the creation bytecode for the contract is submitted to the blockchain. Once the transaction is executed, the contract will be deployed and a unique address will be assigned to it. To invoke that contract, a contract-execution transaction attaching appropriate input data is sent to the contract's address. The input data is the contract's function to be executed and its arguments.

Transactions in Ethereum consist of different fields or attributes, which are nonce, Gas Price, Gas Limit, from, to, value, and data [49], as depicted in Figure 2.5. We briefly explain these attributes as follows:

- **Nonce:** A counter that represents the number of transactions submitted by an account. That is, the nonce is increased by one every time the account sends a transaction.
- **Gas Price:** The amount of money (in Ether) the originator of the transaction is willing to pay for each gas unit consumed by the transaction.
- **Gas Limit:** The maximum amount of gas units the transaction can consume, and it is set by the submitter of the transaction.
- **From:** The submitter address of the transaction.
- **To:** The recipient address of the transaction. This attribute is set to empty for contract-creation transactions.
- **Value:** The amount of currency to be transferred between accounts.
- **Data:** The creation bytecode or the input data to create a new contract or to invoke an existing one, respectively. Thus, the *data* field is only applicable for contract transactions, and it remains empty for financial transactions.

Ethereum Virtual Machine (EVM). The EVM is a stack-based machine that is

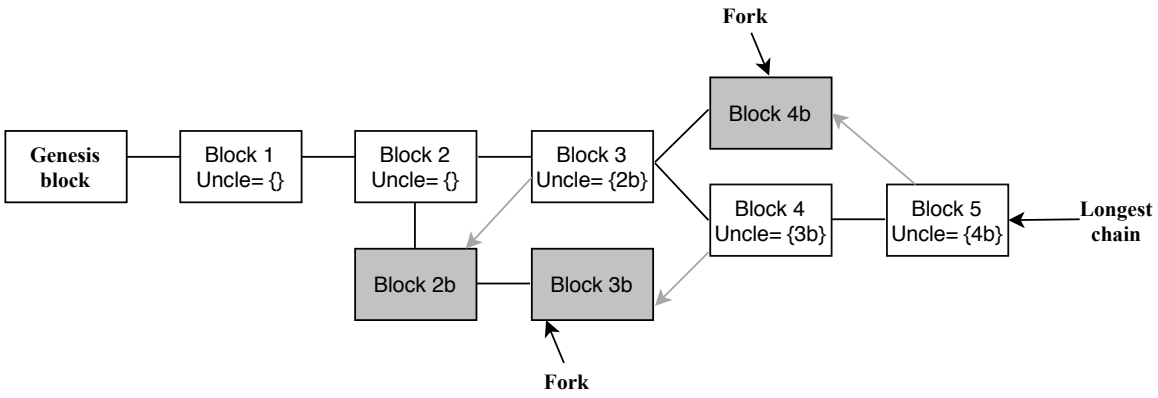

Figure 2.6: Ethereum's uncle inclusion mechanism. Blocks 2b, 3b and 4b are included in future blocks (Block 3-5) as uncle blocks.

responsible for handling the execution of smart contract transactions (both contract-creation and contract-execution) [49]. Every node in the Ethereum blockchain has a copy of the EVM to run contracts independently from other nodes. The EVM consists of a predefined set of instructions, referred to as opcodes. These opcodes run on the EVM in sequence. Every opcode the EVM runs has an associated cost, measured in a unit called Gas. The Gas mechanism will be discussed later on in this section. Note that a smart contract consists of a number of opcodes.

The Ethereum Virtual Machine has been implemented in different programming languages such as Python, Parity and Go. In this thesis, we utilised the Python client to measure the execution time of smart contracts, as we will discuss in Chapter 5.

Ethereum Proof-of-Work (PoW) Algorithm. At the consensus level, Ethereum uses the PoW algorithm with the longest-chain rule to resolve potential forks that may occur due to the network propagation delays (see Section 2.2.2).

Ethereum differs from other blockchains such as Bitcoin in the way that stale blocks can be referenced in future blocks and rewarded for. A stale block that is referenced in a block is referred to as an uncle block. Transactions embraced in the uncle block will be neglected, and they do not contribute to the blockchain's state.

We refer to the process of including uncle blocks in a block as the *uncle inclusion mechanism*. In Ethereum, a maximum of two uncle blocks can be included in a single block. Figure 2.6 shows the uncle inclusion mechanism. Three stale blocks (2b, 3b and 4b) have been referenced in blocks 3-5 as uncle blocks.

Ethereum blockchain. An ordered list of blocks. Each block has a header, a list of

transactions and a list of a maximum of two uncle blocks. The block header contains information relevant to the block such as the hash of the previous block, the block's timestamp, block index, block Gas Limit and other fields.

Ethereum incentive model. Ethereum has a built-in incentive model that rewards miners for maintaining and expanding the blockchain ledger. There are three types of rewards in Ethereum, which are block reward, block's transactions fees and uncle rewards. The block reward is a fixed amount of Ether (currently, 2 Ether) for each new block. The block's transactions fees are the fees associated with all transactions included in the block. The fee for a transaction varies depending on the computational efforts required to execute the transaction. The uncle rewards are a fraction of rewards for generating and including a new block that turns into a stale (orphan) block [49].

Ethereum uses the *Gas mechanism* to calculate the fee for smart contract transactions. Since Ethereum offers a Turing-complete language, users can write a complex contract that might take a long time to be executed. Ethereum, thus, proposes the gas mechanism to limit the possible computation. The intuition behind this mechanism is not only to restrict the computation but also to provide fair incentives for miners.

Every opcode of a smart contract has a predefined gas cost, as specified in [49]. The gas cost for an opcode is determined by the Ethereum foundation, and it depends on the computational resources required to execute the opcode on the EVM. For instance, `ADD` opcode costs three units of gas.

As we discussed earlier, a smart contract can both be published and executed via a transaction. During the execution of a contract transaction, the EVM tallies the amount of Used Gas and charges the submitter of the transaction based on the Used Gas. To avoid non-terminating transactions, the submitter specifies a Gas Limit, and the EMV stops processing if that limit is reached (in which case Used Gas = Gas Limit). The submitter also specifies a Gas Price (expressed in Ether) and the miner then charges the submitter the following transaction fee: Used Gas $\times$ Gas Price. The more opcodes the transaction requires, the more computational effort from the miner, but also the higher the received reward.

It is crucial to ensure that the Ethereum incentive model is fair in order to keep miners well-motivated to participate and to maintain the blockchain ledger honestly [1, 5]. Otherwise, miners may prefer to deviate from the desired behaviour. One issue of Ethereum and some other blockchains is that there are *no miner incentives for verifying* the recipient blocks. As a result, miners might be encouraged to avoid the

verification process, especially if it tends to be computationally intensive, in favour of maximising their revenues, as we will discuss in Chapter 6.

2.5 Modelling and Simulation

Modelling is the method of producing an abstract model that represents a real system of interest, either existing or in design. A system model provides a close approximation of the system it represents by integrating most of its features while at the same time is much simpler than the real system. That is, an excellent model should consider the trade-off between simplicity and realism. A common practice when developing a model is to start with a simple one and then gradually increase the complexity of the model. Generally, a model usually comprises mathematical expressions as well as structural and logical relationships to describe the dynamics of the system [12].

A system model can be classified as deterministic or stochastic; static or dynamic, as depicted in Figure 2.7 [65]. A stochastic model contains one or more random variables or components (e.g., random inter-arrival times for customers in a bank), while a deterministic model has only fixed variables. A dynamic model represents the evolution of a system over time (e.g., a manufacturing model), while a static model represents a system at a specific time. In a static model, therefore, the time is not a significant variable as opposed to a dynamic model. A dynamic model can further be classified as continuous or discrete. In a discrete model, the system state variables change at discrete points in time (not in a continuous manner). An example of a discrete model is the withdrawal service at a bank.

Simulation is a quantitative method, which 'executes' the model to mimic the behaviour of the system [12]. It is quite often that experimentation with a real system is not feasible, impractical or very expensive [72]. Simulation, however, allows experimentation with a model without having to interrupt the real system (if it exists) or implementing a new system for that purpose. With simulation, it is possible to explore different design trade-offs and configuration questions for the system at hand in a timely manner. Simulation can also be used to predict and describe how different conditions and scenarios impact the behaviour of the system. Thus, simulation can be used to answer "What if" questions and to experiment with new designs and policies without interrupting the real system [19].

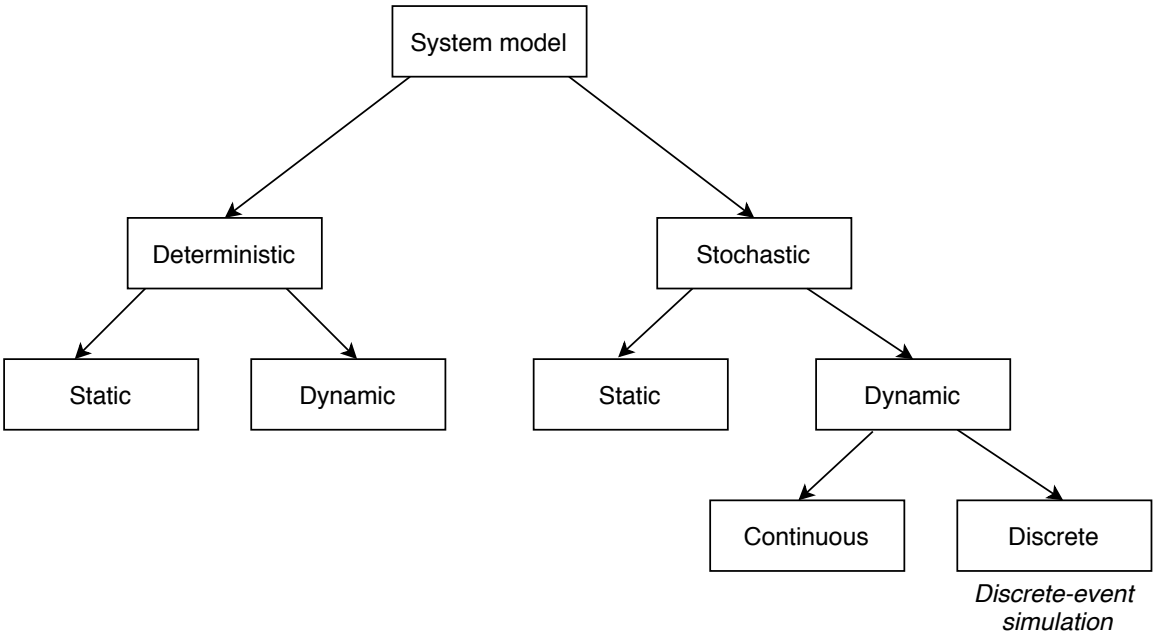

Figure 2.7: The taxonomy of the system model [65].

In general, simulations are a useful tool that can be utilised to study the performance of a system (either existing or proposed one) under a variety of design configurations as well as over a long period of time. Simulation is used before building a new system or changing an existing one. This is to eliminate potential failures, avoid unseen problems and bottlenecks and to improve the performance of the system [72].

Simulation can be classified into two categories, namely, discrete-event simulation and continuous-event simulation [52]. A discrete-event simulation model is both dynamic and stochastic in which the state variables change at discrete moments in time. Human-made systems such as digital computer and information systems are most suitable represented as discrete-event simulation. The focus of this thesis is on discrete-event simulation. In Chapter 4, for instance, we propose a simulator tool named *BlockSim* that is based on a discrete-event simulation approach.

There are two approaches to develop simulation models and tools, namely, general-purpose programming languages (e.g., C++, Java or Python) and special-purpose simulation languages (e.g., Arena and GPSS) [65]. The former is more flexible and familiar, while the latter provides several built-in features (e.g., statistics, event scheduler and animation) that reduce the time required to build models. As stated in [65], there is a debate and conflict about which method is preferable. Also worth noting are simulation frameworks that enable developing simulation models using general-

purpose languages, for instance, OMNeT++ and SimPy for developing models in C++ and Python. We select Python as a general-purpose programming language to develop and implement the BlockSim simulator (see Chapter 4).

2.6 Discrete-event Simulation

A discrete-event simulation (DES) is meant to model the action of a system as a series of events that occur at discrete instants in time. The occurrence of an event triggers an alteration to the system state [20, 88]. The system state remains unchanged between consecutive events.

In a discrete-event simulation, there are two time-advancing mechanisms, which are *next-event* and *fixed-increment* [65]. In a next-event approach, after executing an event, we move the simulation time to the time of the next scheduled event. In a fixed-increment approach, however, we split the simulation time into several small intervals. Events scheduled in a particular interval are then executed together and the system state is changed accordingly. A next-event simulation is typically more efficient as it runs faster than a fixed-increment simulation. This is because not every time interval has to be simulated as some intervals have no events to occur [73].

Here, we further discuss how a next-event discrete-event simulation works, since it is the focal approach used in this thesis. Before explaining and discussing how it works, we briefly identify the major concepts and terminologies, which are: (1) system state, (2) simulation clock, (3) events, (4) event list and (5) event scheduling [65].

- **System state:** A system consists of several entities (e.g., customers and machines) that interact with each other over some time to achieve some goals. The state of a system is a set of variables (e.g., the length of the waiting queue) that maintain the necessary information required to describe the system at a particular point in time. The system state variables are updated as the simulated time evolves, for instance, after the occurrence of an event.

- **Simulation clock:** A variable that represents the current value of the simulated time, and it can be in any suitable measurement unit (e.g., seconds or minutes). The value of the simulated time is usually set to zero before running the simulator. Since discrete-event simulations are dynamic (the system state evolves over time), it is essential to record (keep track of) the current simulation time.

- **Events:** An event is an occurrence whose execution results in a change in some system state variables. An example of an event is the arrival of a new customer. Every event has an associated event time (the time at which the event occurs), an event type as well as any other necessary information required to execute the event. The occurrence of an event is often instantaneous, and it requires performing the actions or activities associated with the event. Such actions depend on the type of event.

- **Event list:** The simulation maintains a list of future (pending) events that are waiting to be executed, known as the future event list (FEL). The list is continuously updated during the simulation by either inserting new events or removing existing ones. The list is usually organised as a priority queue where events are sorted based on their associated event time. In spite of the order in which events are added to the list, the event with the earliest time will always be executed first.

- **Event scheduling:** The process of scheduling and recording new (future) events in the event list. At the start of the simulation model, we usually schedule some initial events to start with as the event list is empty by default. As the simulation progresses, new events can be scheduled and added to the list.

A next-event discrete-event simulation works as follows. At the start of the simulation model, we initiate the simulation clock to zero as well as schedule and add some initial events to the list to start with, as we mentioned earlier. We then scan the event list to determine and select the event with the imminent or earliest time. We advance the simulation clock to the scheduled time of the event. We perform the actions associated with the event, and as a result, we update the system state. The execution of an event may result in inserting new events into the event list or cancelling some existing ones. After executing the event, the event is removed from the list. Then, we select the next event and follow the same procedures until some terminal conditions are satisfied. The terminal conditions can be set as the end of the simulation when the list is empty (no more events), exceeding a specific simulation time or after processing a number of events [65]. Once the terminal conditions have been met, we end the simulation and prepare the statistics of the final output of the simulation.

Because the system state variables only change at the event time, we move the simulation clock to the time of the next event to ignore periods when the system is inactive. This is what makes next-event simulations an appealing approach.

2.7 Steps of a Simulation Study

The methodology for conducting and designing a simulation study consists of different steps [19], as illustrated in Figure 2.8.

- **Problem formulation and setting objectives:** The first step of a simulation study is to define and describe the problem to be studied. It is essential that the problem is well-defined and understood by the participants involved in the study. Also, this step identifies the aim, research questions and objectives of the study.

- **Model conceptualisation:** This step involves designing a simulation model that represents and abstracts the real system. The inputs, outputs and contents of the model should be clearly defined. Also, all assumptions and simplifications about the model need to be stated. The model should also incorporate all mathematical expressions and logical relationships that describe how to calculate the desired output values from the given input values. The complexity of the model usually depends on the scope and aim of the study. That is, the model should not contain details more than what is needed to achieve the objectives of the simulation study.

- **Data collection:** This step is to gather and collect the data needed to feed the model. The data can be directly collected (if available) from an existing running system (e.g., real historical data). In some cases, the data should be obtained by observing the system or by running some controlled experiments. Data is often fitted to theoretical distributions such as an exponential distribution, and then random values are drawn from such distributions during the simulation. This step can be executed in parallel with the model conceptualisation step since they are independent of each other.

- **Model implementation:** This step is to translate the conceptual model into a simulation program. To achieve this task, the modeller or developer should decide on whether to use special-purpose simulation languages (e.g., GPSS) or general-purpose programming languages (e.g., Java), see Section 2.5. The developer is recommended to start with implementing a portion of the model and then steadily increase the level of details until the whole model is successfully implemented.

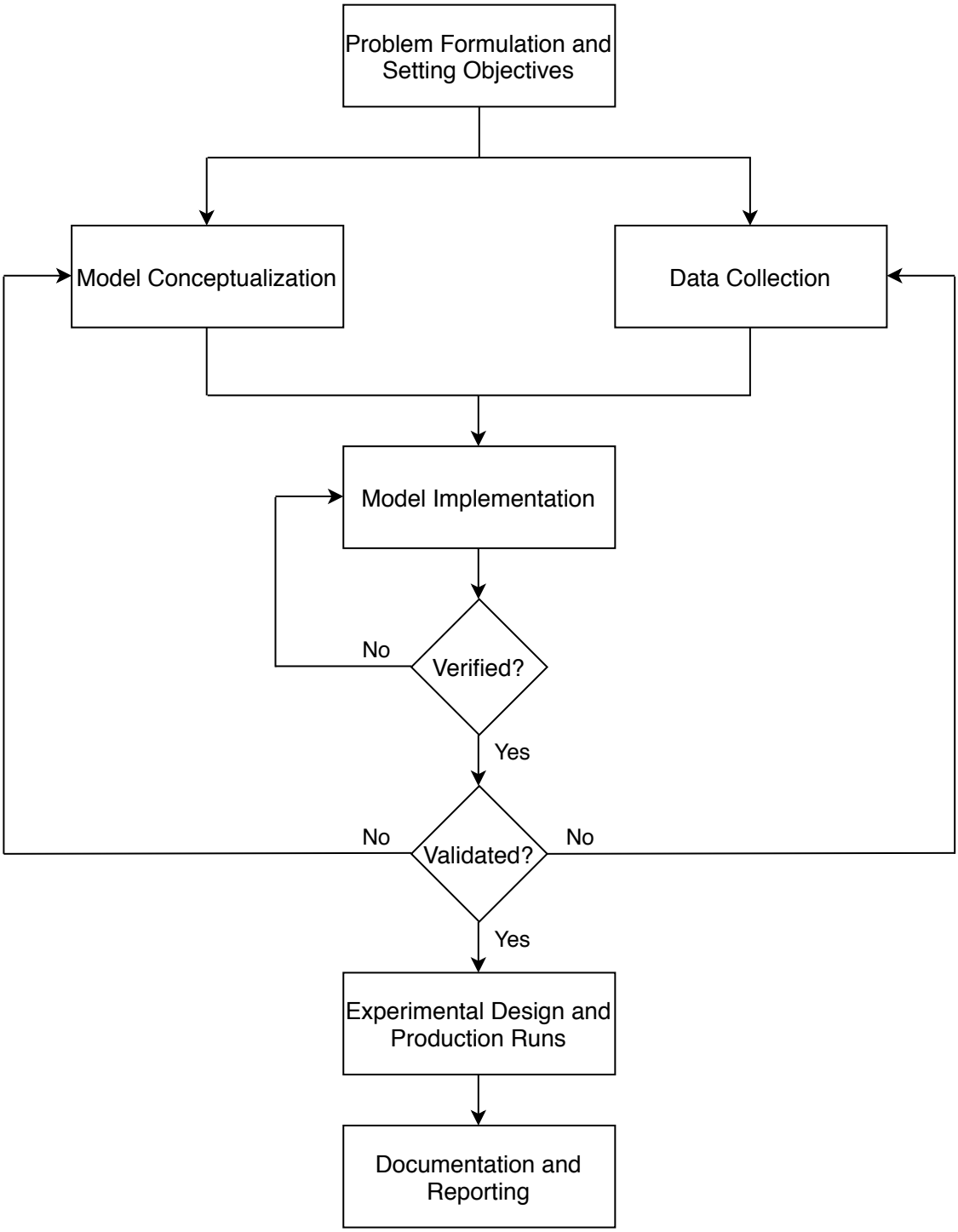

Figure 2.8: Steps of a simulation study (reproduced after [19]).

- **Verification:** This step is to ensure that the conceptual model is correctly implemented. That is, the verification step is a debugging task that mainly ensures the simulation program is error-free and can deliver what the model is intended to do. There are various verification techniques in the literature such as the continuity test and degeneracy test [55]. In a continuity test, the simulation program is run multiple times using different values for input parameters. This is to check if the change in the outcomes is reasonable. A degeneracy test checks for extreme values for input parameters to identify bugs that have not been thought of. Additionally, it could be useful to run simplified cases (e.g., with a short simulation run or a few number of nodes in a network simulator) and then compare the simulation results with that of the analysis. Often, the verification process is carried out after implementing each portion of the model to detect and fix errors in the early stages. Once the whole model is implemented, the simulation program needs also be verified.

- **Validation:** This step is to ensure that the model represents the real system accurately. That is, the simulation program can produce results comparable to that of the real system. A model can be validated against real-system measurements and theoretical results, where possible [55]. Three aspects of the model have to be validated, which are the model's assumptions, the values for input parameters and output values. If the model is invalid, the modeller should revise both the model design, the model assumptions and the data collection exercise. One can start the validation process in conjunction with the model conceptualisation step and continue until the model has been successfully translated into a simulation program.

- **Experimental Design and production runs:** This step is to design and run the simulation study. This involves defining the conditions, procedures and scenarios to be run. Besides, several parameters have to be determined such as the length and the number of replications per simulation run. Furthermore, identifying the input factors to include, how those factors can be varied and whether to use the same or different numbers across different configurations.

- **Documentation and Reporting:** This step is to analyse, document and report the simulations results. The results of the study are analysed using statistical methods (e.g., mean, standard deviation and confidence interval). The results are then documented, and conclusions about these results will be drawn.

The above-mentioned steps are essential for any simulation study. We apply these steps for the simulation studies conducted in this thesis (see Chapters 6 and 7).

2.8 Related Work

In this section, we first discuss some of the literature conducted on performance evaluation of both public and private blockchains. Then, we discuss current research on blockchain simulation models and tools. Finally, we present existing research on performance evaluation of the Ethereum blockchain, with regard to the Verifier's Dilemma and the profit uncertainty problems.

2.8.1 Performance Evaluation of Blockchains

In this section, we survey works on performance measurements, tools and analysis for both public and private blockchains.

Performance Evaluation of Public Blockchains. Aldweesh et al. [2] propose a performance benchmark framework named OpBench for Ethereum Virtual Machine (EVM) that evaluates the CPU usage required to execute the Ethereum smart contract operation codes. Its main purpose is to assess and compare the CPU usage with the gas cost for individual operation codes. It currently supports three different implementations of the Ethereum Virtual Machine, namely, PyEThAPP, Go-Ethereum and Parity. Decker and Wattenhofer [35] conduct a performance measurement study on the Bitcoin network to collect data about block propagation delay and block stale rate by listening to 10000 blocks. Also, they propose a simplified model that estimates the block stale rate by considering the block creation rate and the block propagation delay. In [58], the authors use queuing theory to model the transaction priority mechanism in order to analyse and evaluate transactions latency in the Bitcoin network. Croman et al. [32] analyse the factors that contribute to performance bottlenecks in the Bitcoin network. Their results suggest changing two performance parameters, namely, block size and block creation rate as a first step towards scaling the Bitcoin network. Gervais et al. [50] analyse the performance of PoW blockchains under various network parameters (e.g., block size, block arrival time and propagation mechanism). Papadis et al. [82] propose stochastic models to evaluate the performance of PoW blockchains. In particular, they analyse the impact of block propagation delay and the hash power of the nodes on the block creation rate and block stale rate, using a combination of analytical calculations and simulation experiments. Yasaweerasinghelage et al. [102] demonstrate the usage of architectural

performance modelling to predict the latency of blockchain-based systems. Eyal et al. [43] propose a new protocol named Bitcoin-NG (Next Generation) that aims to improve the performance of the Bitcoin network.

Performance Evaluation of Private Blockchains. Dinh et al. [39] propose a benchmark framework named BLOCKBENCH to evaluate and analyse the performance of private blockchains such as Hyperledger Fabric, Parity and private Ethereum environment. BLOCKBENCH aims to measure and assess the overall performance with regard to latency, throughput, fault-tolerance and scalability through the usage of a set of micro and macro benchmarks. Hyperledger Caliper [27] is another benchmark tool for evaluating the performance of all Hyperledger blockchain variations such as Sawtooth, Burrow and Fabric. Currently, Hyperledger Caliper supports four performance metrics: throughput, latency, success rate for transactions and resource utilisation.

Thakkar et al. [95] conduct an extensive empirical study to evaluate the performance of Hyperledger Fabric blockchain with regard to throughput and latency, by varying five different performance parameters: block size, number of channels, resource allocation, endorsement policy and ledger database. Their analysis results identify several performance bottlenecks and show how to configure these parameters better. Similarly, Kuzlu et al. [63] evaluate the impact of different network workloads on the performance of the Hyperledger Fabric blockchain in terms of throughput, latency and scalability. Ampel et al. [11] evaluate the performance of Hyperledger Sawtooth blockchain using the Hyperledger Caliper benchmarking tool. They test different parameters such as transaction creation rate, batch size, throughput, latency and resource usage in order to identify performance bottlenecks.

2.8.2 Simulation Models and Tools for Blockchains

In this section, we discuss current research on simulation models and tools for blockchain systems. This includes the limitations of existing studies and how the BlockSim simulation tool that we propose in this thesis (see Chapter 4) seeks to address them.

In the literature, there are some attempts to utilise simulation models to evaluate various aspects of blockchain systems. In [102], the authors use architectural modelling and simulation to measure the latency in blockchain systems under different configurations. In [7], the authors propose a simulation model to investigate the impact of profit uncertainty in the Ethereum blockchain. They found that miners in Ethereum are not able to make informed decisions about which transactions to include in their blocks to maximise their revenue. In [78], the authors propose a

simulation model to analyse and evaluate attacks on the Bitcoin network. In [51], the authors use discrete-event simulation to study the behaviour of Bitcoin miners (including selfish-mining strategies) when there is a delay in propagating information among miners.

Besides these proposals, there are some blockchain simulators proposed in the literature. In [50], the authors propose a Bitcoin simulator to analyse the security and performance of different configurations in both the consensus and network layers. Several other Bitcoin-like network simulators are proposed in the literature such as [16, 75, 91]. However, these proposals utilise simulation-based models to study specific aspects of blockchain systems. They neither cross different layers nor cover all common functional building blocks (e.g., blocks and transactions) for blockchain systems. For instance, neither of these proposals model transactions in the blockchain system nor capture the incentives layer in more detail.

In Chapter 4 of this thesis we propose and develop BlockSim as a general-purpose, widely usable, simulation tool for blockchains, to assist in answering a variety of design and deployment questions. Our discrete-event simulator generalises on the ones proposed in the related literature by integrating different layers of the blockchain system to gain a more comprehensive insight into different aspects such as performance, security and incentives. In BlockSim, for instance, we take a step further by considering the functional blocks common across the different implementation of blockchain systems. We design and structure BlockSim to cross different layers of blockchains. Furthermore, we model transactions in two different ways, each of which for specific purposes as well as modelling both Bitcoin and Ethereum blockchains.

2.8.3 Performance Evaluation of the Ethereum Blockchain.

In this section, we discuss existing research on performance evaluation of the Ethereum blockchain, with respect to the Verifier's Dilemma and the profit uncertainty problems. We highlight the gaps in current research and discuss how our work presented in Chapters 6 and 7 of this thesis aims to addresses these gaps with the help of our proposed simulator.

Verifier's Dilemma (Chapter 6). The *Verifier's Dilemma* was first identified by Luu et al. [71], who showed that rational miners would be motivated to skip the verification process to gain an advantage in the race to mine the next blocks. Related to this idea is the mining strategy proposed in [85], whereby a malicious miner purposely designs smart contracts that are computationally expensive to verify, to slow down other miners. In response, [71, 85] showed the profitability of skipping the

verification process in scenarios in which computationally intensive smart contracts were introduced.

This work left several unanswered questions. In particular, it was not known if the above attack was practical, nor was it clear how different miners with different hash powers might benefit from not verifying blocks. In Chapter 6 of this thesis, we address these limitations by evaluating the implications of the Verifer's Dilemma using real Ethereum smart contracts transactions. In addition, we assess both current and future settings of Ethereum, taking into consideration the hash power of miners.

Several solutions in the literature have been proposed to make the verification of complex transactions a more efficient task in order to avoid the Verifier's Dilemma. In [71], the authors proposed a solution in which complex transactions are divided into various smaller transactions that can be incorporated in various blocks. In [15, 38, 104], the authors proposed solutions for executing and verifying smart contracts in parallel. They showed that such solutions could speed up the execution/verification time of contracts compared to that of a sequential solution. In addition, several off-chain solutions (e.g., *YODA* [34] and *Arbitrum* [57], *TrueBit* [94]) for efficient computation of computationally expensive smart contracts have been proposed as an alternative to the protocol used in Ethereum. In these solutions, only a small set of nodes, instead of all nodes, has to perform the verification of complex contracts. Those nodes will be rewarded if they perform the verification correctly, or otherwise, a penalty will be imposed.

In [15, 38, 104], however, the authors did not investigate the parallel verification of smart contracts as a mitigation solution to the implications of the Verifier's Dilemma. In Chapter 6, we propose and evaluate the parallel verification as a solution to reduce the advantage miners would get from not verifying. Besides, we propose and evaluate the intentional production of invalid blocks as a new solution to punish non-verifying miners. We were inspired by the idea of injecting invalid blocks in the network from [94].

Profit Uncertainty (Chapter 7). Ethereum uses the Gas mechanism to set the fee for transactions. Yet, the effectiveness of this mechanism depends on whether the gas cost for each operation code (opcode) is correctly aligned with the computational cost.

Several studies in the literature have assessed the incentive compatibility of the Ethereum Gas mechanism by evaluating the alignment of the received fee with the observed resource usage. At the opcode level, the authors in [1, 2] proposed OpBench system that assesses the alignment of the gas cost for every opcode with its CPU usage

and found that some opcodes were mispriced. Similarly, the authors in [29, 83, 101] found significant inconsistencies for opcodes prices and showed a weak correlation between the gas costs and the CPU and memory usage for individual opcodes. At the transaction level, the authors in [3, 10] compared the fee awarded (in terms of Used Gas) for executing Ethereum smart contract transactions with the invested CPU usage. They found that the amount of Used Gas is not always proportional to the CPU usage, which means that some transactions are more profitable than others.

As the Ethereum incentive model does not provide incentives compatible with the computational costs, the profit a miner would get from executing transactions depends on which transactions have been chosen. With the uncertainty miners perceive in Ethereum regarding the fee and cost for transactions, the profit gained by miners might be impacted. To the best of our knowledge, no work has investigated the impact of the uncertainty miners perceive when selecting transactions on the profit earned under the incompatible Ethereum incentive model. That is, our work presented in Chapter 7 of this thesis is the first investigation attempt on this topic.

2.9 Conclusion

In this chapter, we introduce essential background information related to blockchain technology and simulation techniques. The topics covered in this chapter include an overview of blockchain paradigm, smart contracts, Ethereum, discrete-event simulation and the methodology for conducting a simulation study. In addition, we thoroughly discuss the literature on performance evaluation of blockchain systems, including related work on simulation models and tools.

The core contribution of this thesis is the design and development of a general discrete-event blockchain simulator in addition to the study of two data-driven simulation studies related to Ethereum smart contracts as case studies for the simulator.

Chapter 3

Blockchain-based Smart Contracts: A Systematic Mapping Study of Academic Research

Summary

Smart contracts and their underlying technology, blockchain, have gained wide interest and attention in the last few years. In this chapter, we carry out a longitudinal systematic mapping study of all peer-reviewed technology-oriented research in smart contracts. Our interest is twofold, namely to provide a survey of the scientific literature and to identify academic research trends and uptake. We only focus on peer-reviewed scientific publications to identify how academic researchers have taken up smart contract technologies and established scientific outputs.

We obtain all research papers from the main scientific databases, and using the systematic mapping method classified the papers into six categories, namely, security, privacy, software engineering, application, performance and scalability and other smart contract related topics. Among those categories, we find that the majority of the articles falls into two categories: application (about 64%) and software engineering (21%).

3.1 Introduction

Blockchain-based smart contracts are computer programs that encode an agreement between non-trusting participants. Smart contracts are executed on a blockchain system if specified conditions are met, without the need of a trusted third party.

The aim of this chapter is to identify and to classify all peer-reviewed research that has been conducted on smart contract technology. Importantly, we do not attempt to include all the latest developments in technical, financial or political issues that were communicated through other channels, particularly the Internet. We also are interested in longitudinal (year after year) aspects of academic contributions to smart contracts, to document and analyse the growth in research outputs. To achieve this aim, we follow the systematic mapping study approach proposed in [84] to look for relevant papers in the main scientific databases and to generate a classification map. This is to understand the topics of interest as well as identify gaps for future work.

We conduct the systematic mapping study in longitudinal aspects starting in 2017, then updating it in 2018 before revising it further in 2020. From the mapping study, we find that the number of published articles on smart contracts is increasing significantly every year, reaching over 2500 papers as of March 2020. The results of the study also show six different categories for smart contract topics, which are security, privacy, software engineering, application, performance and scalability and other smart contract related topics. The majority of the papers falls into the applications (about 64%) and software engineering (21%) categories.

The structure of this chapter is as follows. In Section 3.2, we present the methodology used to conduct the systematic mapping study, including the definition of the research questions. Section 3.3 illustrates the results of searching and screening for relevant papers as well as the results for classifying all research papers. In Section 3.4, we discuss and answer the research questions. Section 3.5 concludes the chapter.

3.2 Research Methodology

We select the systematic mapping study proposed by [84] as our research methodology in order to explore the current research related to smart contracts technology. A systematic mapping approach allows us to identify and classify research topics relevant to smart contracts. It also helps us to identify research gaps for future research. As depicted in Figure 3.1, the systematic mapping study is divided into five steps, namely, defining research questions, conducting the search, searching for relevant papers, keywording using abstract in addition to data extraction and mapping process.

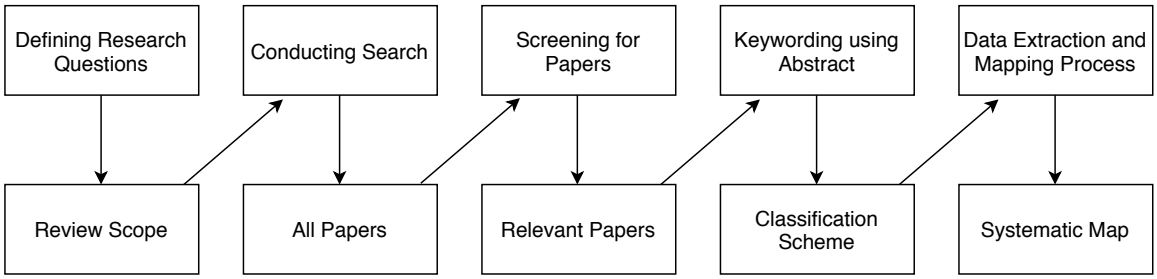

Figure 3.1: Steps of the systematic mapping study [84].

3.2.1 Defining Research Questions

The first step in a systematic mapping study is to define the research questions to be answered by the study. For our study, we define four questions, which are as follows:

RQ1. What are the existing research areas in smart contracts?
RQ2. How does smart contract research evolve year on year in terms of the number and type of publications?
RQ3. What existing applications are there for smart contracts?
RQ4. What are the research gaps?

3.2.2 Conducting the Search

The second step is to search and gather all research papers related to smart contracts based on a specific search term. We choose the term 'smart contract' for this study as the main search keyword. Having identified the keyword for the searching task, we select five different scientific databases to carry out our search. The selected databases are ACM Digital Library, Springer, IEEE Explore, Scopus and ScienceDirect. We only focus on gathering peer-reviewed research papers published in journals, conferences, symposiums, workshops and books.

3.2.3 Screening for Relevant Papers

The third step is to exclude all research papers that are irrelevant to our research questions. To accomplish this step, we follow the screening approach proposed in [103]. In this approach, we first attempt to remove irrelevant research papers based on their titles. If we could not manage to decide on the relevancy of a paper based on its title, we would run through a second step by evaluating the abstract of that paper. In addition to title and abstract based exclusion, we also rely on some exclusion

criteria to remove some papers. We remove papers without English text, without full text available, with no critical contributions such as popular articles, newsletters or grey literature. Moreover, we remove duplicate papers and non-technology based papers.

3.2.4 Keywording using Abstracts

The fourth step is to classify all the relevant research papers based on the keyword approach proposed in [103]. In this approach, we go through the abstract of all papers in order to associate crucial keywords and the key contribution. The purpose of doing so is to classify all research papers under different categories. In some cases where it is difficult to classify a paper using its abstract, we skim the paper quickly to make a proper decision about its category.

3.2.5 Data Extraction and Mapping Process

The last step is to collect all information needed to answer the research questions of our study. We collect various data items embracing the main goal and contributions from each research paper.

3.3 Study Results

This section discusses the results of the systematic mapping study that we conducted on smart contracts. We first discuss the results of searching and screening for relevant papers. Then, we discuss the results of the classification process.

3.3.1 Searching Results

The first step of a systematic study after defining the research questions is to search for relevant papers. We search for all scientific papers in five different scientific databases using a specific keyword 'smart contract'.

Table 3.1 shows the number of research papers on smart contracts that have been retrieved from the five databases over the years (until March 2020). Research on smart contracts started in 2014 with the emergence of the Ethereum blockchain, the most common platform for smart contracts. We note that the number of published papers has been increasing since 2018, and that the greater number of articles is contributed to the Scopus database.

Year	IEEE	ACM	Springer	ScienceDirect	Scopus	All Databases
2014	0	1	0	0	0	1
2015	2	4	3	7	3	19
2016	5	12	21	2	28	68
2017	43	50	75	21	114	303
2018	217	151	224	128	486	1206
2019	421	263	417	292	996	2389
2020	49	33	123	220	184	609
Total	737	514	863	670	1811	4595

Table 3.1: The number of publications on smart contracts per database over the years.

Before 2016, the total number of papers was only 20. This number possibly includes duplicate and irrelevant papers. In 2016, the number of scientific papers increased significantly by 68 papers and since then it further increased dramatically. In the following year (2017), about 300 papers more were published, to reach nearly 400 papers in total. From 2018 and beyond, the number of papers has been growing dramatically, for instance, about 2400 papers were published in 2019. The number of smart contract papers has reached over 4500 by the end of March 2020, with over 600 more papers published in the first three months of 2020.

Interestingly, the number of scientific papers is increasing each year for all the five databases, as clearly depicted in Figure 3.2. This indicates that the popularity of the topic of smart contracts grows every year. For that, we expect even a further increase with respect to the number of papers in the future.

Among the five databases, Scopus dominates the majority of the papers published on smart contracts. It is almost accounting for 40% of total papers published every year. This is expected since Scopus is a comprehensive database that contains papers that are already in other databases. For example, most of the papers in IEEE and ACM libraries are also in Scopus.

The following sections discuss the screening and classifying results. We note that we carried these analysis results in May 2017, and then updated them in June 2018. Due to the significant rise in the number of papers published since the second half of 2018, we have not updated the results. However, the results are valuable and useful to get insight into the direction and trending topics in the general space of smart contracts.

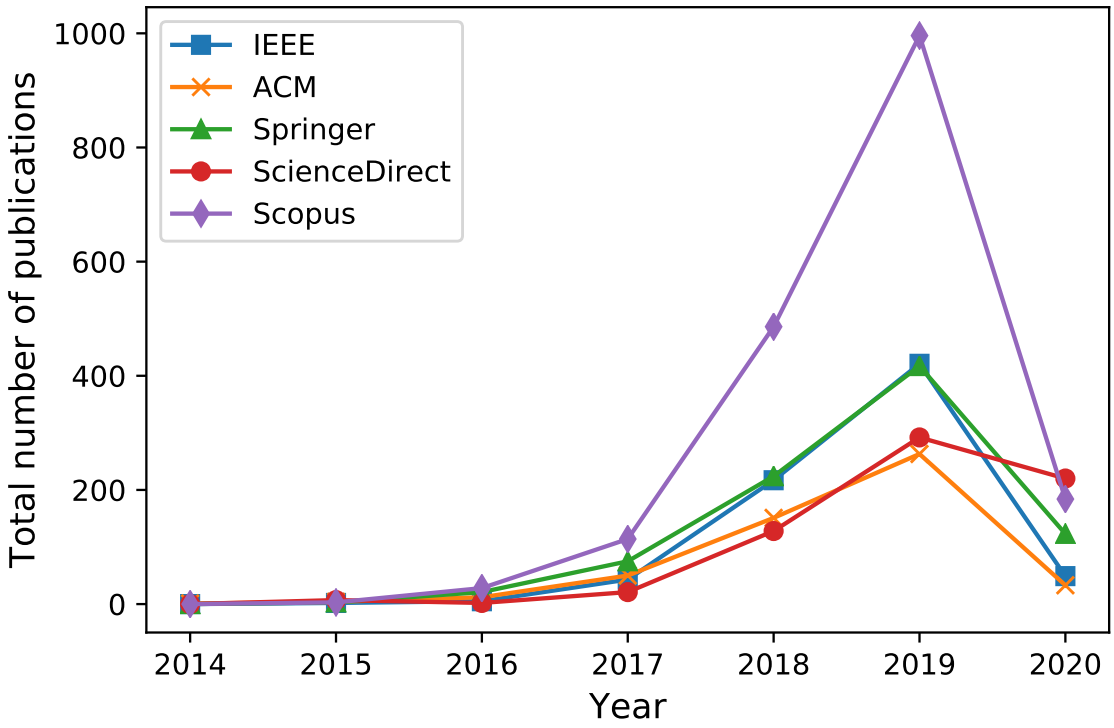

Figure 3.2: The growth in terms of the number of publications per each database over the years.

3.3.2 Screening Results

In this section we discuss the screening results. By applying the searching phase, we managed to gather 617 papers in total, on 12 June 2018. We screened these papers as follow. First, we excluded duplicate papers. We ended up with 407 unique papers. After that, we went through the title and the abstract for all the 407 papers in order to exclude irrelevant papers. We managed to exclude 219 irrelevant papers (about 54% of all papers). There are three reasons why we had a high number of excluded papers. First, many papers were irrelevant to our study, since our focus was to explore smart contracts from a technical perspective. For instance, many papers discussed the topic from an economic or legal point of view. Another reason is that some excluded papers were about cryptocurrencies or blockchain in general (as opposed to smart contracts), which do not contribute to our research questions. The last reason is that some papers were excluded as they only discuss grey literature about smart contracts or discuss the possibility of applying them to different domains such as the Internet of Things, without providing any technical contribution. Therefore, we ultimately included 188 papers in our systematic mapping study.

3.3.3 Classification Results

By applying the Keywording technique, we classified the papers into six categories, namely, security, privacy, software engineering, application, performance and scalability and other smart contract related topics. Security relates to bugs or vulnerabilities that an adversary might utilise to launch an attack in smart contract systems. Privacy includes issues related to disclosing contracts information to unauthorised people. Software engineering refers to any work related to the software development of smart contracts. Application refers to the utilisation of smart contracts to address issues in different domains such as the Internet of Things (IoT). Performance and scalability category refers to the ability of smart contract systems to deliver a reasonable response time as well as to sustain performance when the number of contracts is increasing.

Figure 3.3 shows the percentage of scientific papers in each of the six categories. It is clear that most papers are smart contract applications, accounting for 64% of all the papers. The second most common category is software engineering, with 21% of the papers. Security category dominates 6% of the papers. 3% and 2% of the papers fall into performance and scalability and privacy categories, respectively. It is worth noting that there are some papers (4% of all papers) fall into other smart contract related topics.

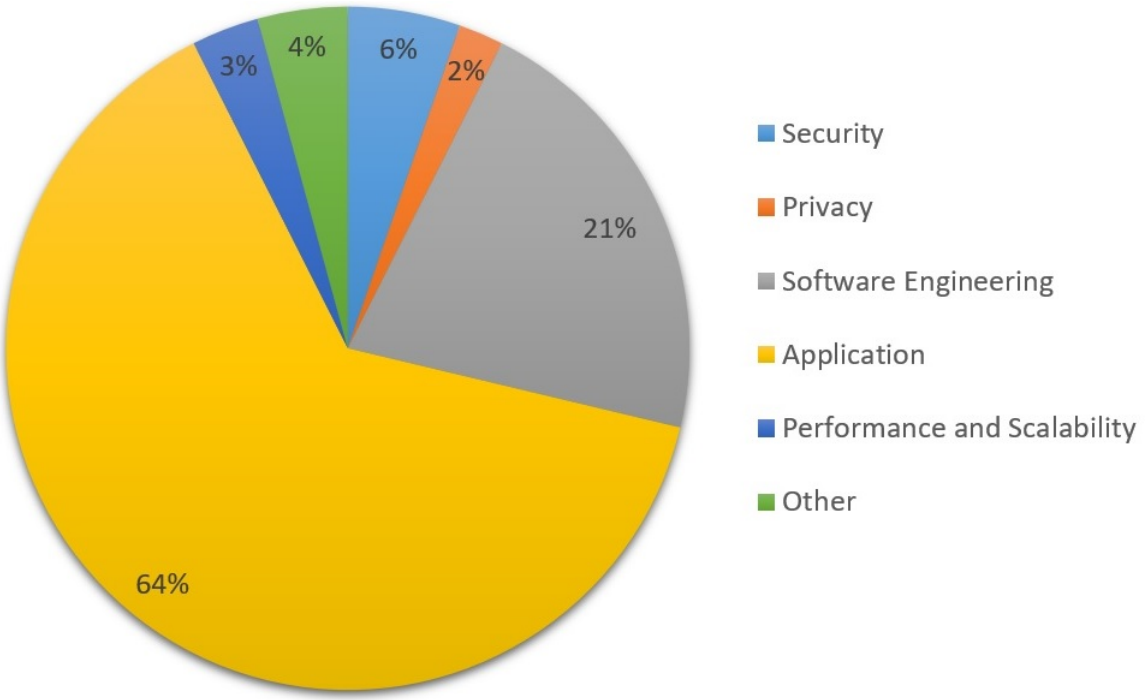

Figure 3.3: The percentage of scientific papers in each category.

Security. We found 10 papers that identify or propose solutions to bugs and vulnerabilities in smart contract systems. These vulnerabilities include transaction-ordering dependency, timestamp dependency, mishandled exception, criminal activities, reentrancy and untrustworthy data feeds. Some papers present detection tools (e.g., Oyente [69] and ReGuard [80]) that can be used by developers to identify common security bugs. Some papers propose assurance methods and machine learning approaches to detect security risks and fraud. Furthermore, one of the papers identifies three types of criminal activities that can be carried out in smart contract systems [56]. Another paper proposes an adaptive incentive mechanism for smart contract systems to defend against denial of service attacks [29]. In addition, Atzei et al. [18] surveyed several vulnerabilities in Ethereum smart contracts and built a taxonomy for such vulnerabilities.

Privacy. We only found four papers that extend blockchain-based smart contract platforms to support privacy and confidentiality. Hawk [61], for instance, is a tool that allows smart contract developers to build privacy-preserving contracts. The rest of the papers focus on extending Hyperledger Fabric to support private data, proposing encrypted data feeds for contracts and proposing a system to ensure the confidentiality of contracts. For example, Zhang et al. [105] propose a Town Crier (TC) solution that acts as a trusted third party between external sources and smart contracts to provide encrypted and authenticated data feeds for smart contracts.

Software Engineering. 40 papers fall into software engineering for smart contracts, covering a wide range of topics. About 20% of the papers focus on verification and validation techniques (e.g., formal modelling techniques [22]) for smart contracts in order to assure that smart contracts are functioning as intended as well as error-free. Six papers are about proposing new platforms (such as Smartdemap [59]) and languages (e.g., Simplicity [81]) for developing smart contracts. Three papers utilise analysis techniques to inspect the code of smart contracts. Moreover, there are three papers that propose solutions to blockchain immutability features by allowing the modification and termination of already deployed smart contracts. Two papers focus on optimising the code of smart contracts by identifying and solving programming patterns with high execution costs.

The rest of the papers focuses on automating the process of developing contracts, building code classifiers and parsers for contracts, human-centered design of contracts,

designing templates for developing contracts, profiling smart contract interactions and identifying common pitfalls in developing safe contracts. Furthermore, there are some papers that propose frameworks for developing secure contracts, proposing a compression method to reuse previously deployed contracts and proposing the integration of a semantic legal layer with the blockchain to support legal contracts.

Applications. We found 120 application-based papers (about 64% of all papers). We classified these applications into several topics, namely, Internet of Things (IoT), cloud computing, financial, data, healthcare, access control and authentication and other applications.

Internet of Things (IoT) Applications: Internet of Things (IoT) refers to physical devices and appliances connected via the Internet. We found 18 papers that apply blockchain-based smart contract technology to IoT. Three applications utilise smart contracts to build an access control system for IoT. Four applications utilise smart contracts to overcome security and privacy issues in the IoT. The rest of IoT-based applications utilise smart contracts for the management of IoT devices, electronic business, data management, data trading, data exchange and data storage.

Cloud Computing Application: We found eight applications that utilise blockchain-based smart contract to overcome various technical issues in cloud computing. These applications address the issues of verifiability of outsourced computation [40], data auditing, resource management of cloud datacentres, negotiation and agreement establishment, data accountability, trust, access control and service level agreement (SLA) monitoring.

Financial Applications: We found several applications that utilise blockchain-based smart contracts for financial purposes such as payment and loan. The identified financial applications embrace fair payment systems, privacy-preserving incentive mechanisms, a smart will, taxation-based payment, car insurance, private and concurrent payment channel network, concert tickets and protocols for data trading and the management of study loan repayment.

Data Applications: These types of applications utilise blockchain-based smart contract to manage and secure general data and information. These applications include data sharing, data management, data indexing, data integrity check, and data provenance and accountability.

Healthcare Applications: We found three blockchain-based smart contract applications in the healthcare domain. These applications are a secure remote patient

monitoring system, an access control framework for electronic health records and trustless medical data sharing among different cloud providers.

Access Control and Authentication Applications: These types of smart contract applications target approaches to user authentication and management of access right policies. Several smart contract based access control systems have been proposed in different domains such as IoT [17], healthcare [33] and cloud computing [92]. With regard to authentication, there are different proposed applications such as a secure mutual authentication for industry 4.0, an enhancement of TLS handshake authentication and a distributed and secure user authentication.

The rest of the applications covers a wide range of different topics including e-voting, supply chain management, intelligent systems (e.g., intelligent transportation systems), smart grid systems, energy-based applications, resource management, reliable decision making, digital rights management, human resource systems and 2 phase commit protocol for distributed consensus protocols [45]. Furthermore, other applications include volunteer time record systems, QoS-aware service composition, logistics management, trustless intermediation in marketplaces, assessment organisation service, Business Process Management (BPM) systems and decentralised applications (DAPPs).

Performance and Scalability. We found six papers that fall into performance and scalability topics in smart contract systems. Some papers propose benchmarking frameworks (e.g., Blockbench [39]) for analysing and monitoring the performance of blockchain-based smart contracts. To overcome scalability issues in smart contract systems, some papers propose solutions to execute smart contracts in parallel instead of sequentially. For more details on the state of the art in performance evaluation of blockchain systems, we refer to this dissertation's background discussion in Chapter 2.

Other smart contract related topics. We found eight papers that fall into other smart contract related topics such as consensus protocols and incentive mechanisms for smart contracts. Some papers propose new secure consensus protocols for smart contracts and identify issues in existing protocols. Other papers focus on incentive mechanisms, system operations and system design for smart contracts.

3.4 Discussion

This section discusses the study results and answers the research questions that we defined in Section 3.2.

RQ1: What are the existing research areas in smart contracts?

The study classifies the research topics on smart contracts into six categories, namely, security, privacy, software engineering, application, performance and scalability and other smart contract related topics. The majority of the research (about 64%) falls into the application category, followed by software engineering (21%). The applications cover a wide range of domains such as IoT, cloud computing, finance, healthcare, access control, authentication and others. Most of these applications are to address technical issues (e.g., security issues) or to get rid of the trusted third parties in existing applications. In the software engineering category, most of the papers utilise analysis techniques for validation and verification purposes or to propose new platforms and languages for developing secure smart contracts. For the performance and scalability category, the papers either propose frameworks for performance analysis or scalable solutions for the execution of contracts. In the security category, most papers focus on identifying and tackling security bugs and vulnerabilities. In the privacy category, most papers focus on the confidentiality of information in smart contract systems.

RQ2. How does smart contract research evolve year on year in terms of the number and type of publications?

Research on smart contracts started in 2014 with the emergence of Ethereum. Since then, the number of publications on smart contracts is increasing significantly every year, as discussed in Section 3.3.1. For instance, the number of publications was increased by over 300% in 2018 compared to that of 2017. Since 2018, the number of papers published every year is at least over 1000 papers, with 609 papers published in the first three months of 2020.

From the analysis results we discussed in Section 3.3.3, application and software engineering categories experience the highest number of publications, accounting for 64% and 21% of the total published papers, respectively. This indicates that smart contract systems widespread very vastly, especially in terms of smart contract applications and development.

RQ3: What existing applications are there for smart contracts?

Smart contract applications cover any solution that utilises smart contract technology to overcome the issues in existing systems or any smart contract tool. We identify 120 application-based papers that make use of smart contract technology in existing systems such as IoT. These applications include cloud-based applications, healthcare-based application, financial applications, data applications, access control and authentication based applications, e-voting, smart grid systems, digital right management, intelligent systems and other decentralised applications. In addition, we identify several smart contract tools that can aid during the process of developing smart contracts, identify security issues, or provide confidentiality for smart contract information.

RQ4: What are the research gaps?

From this study, we are able to identify at least two research gaps in smart contract research. The methodologies used to identify these gaps are by observing issues or limitations from the papers included in this study and by relying on our knowledge in smart contract topic.

The first one is the relative lack of research on scaling blockchain-based smart contract systems. In current systems, smart contracts are executed in sequence, which leads to low throughput, especially if the number of smart contracts becomes relatively large. Although we found a few papers exploring parallel execution of contracts, their proposed solutions are high-level ideas and still not proven to be working properly in smart contract systems. There are some challenges that face parallel execution of contracts such as how to execute contracts that depend on each other at the same time. It is, therefore, essential to conduct research on identifying and tackling performance issues to ensure the ability of blockchain to scale.

A second area is the relative lack of research on performance evaluation of smart contract execution. Performance benchmark approaches, for example, can be beneficial to evaluate the fairness of incentive models within smart contract systems. If the incentive provided by such systems is not compatible with the computational costs, this could result in security attacks as well as poor incentive and cost models [7], which impact the reliability of smart contract systems.

Following from these identified gaps, we conduct a performance benchmark experiment to measure the execution time of real Ethereum smart contracts (Chapter 5). Also, we conduct two performance evaluation studies (Chapters 6 and 7) related

to Ethereum smart contracts. The first one is regarding the impact of sequential verification of contracts on the revenue for honest miners, and how parallel execution of contracts can reduce that impact. The second one is regarding the uncertainty miners face when selecting contract transactions under the unfair Ethereum incentive model, and its impact on the earned profit.

3.5 Conclusion

In this chapter, we conduct a longitudinal systematic mapping study in order to understand current research areas on smart contracts, to identify research gaps for future work and to identify academics trends in uptake and emphasis.

The main insights we gained from this study are as follows. First, we observe that the number of scientific papers on smart contracts is increasing significantly every year, indicating a broad interest in this topic. In addition, we identify and classify smart contract research into six different categories, namely, security, privacy, software engineering, application, performance and scalability and other smart contract related topics. The Application category dominates the majority of research publications, constituting 64% of total articles. Furthermore, we manage to identify two research gaps in smart contract research that we base our simulation studies in this thesis on.

Chapter 4

BlockSim: An Extensible Simulation Tool for Blockchain Systems

Summary

Both in the design and deployment of blockchain solutions many performance-impacting configuration choices need to be made. We introduce BlockSim, a framework and software tool to build and simulate discrete-event dynamic systems models for blockchain systems. BlockSim is designed to support the analysis of a large variety of blockchains and blockchain deployments as well as a wide set of analysis questions. At the core of BlockSim is a Base Model, which contains the main model constructs common across various blockchain systems organised in three abstraction layers (network, consensus and incentives layer). The Base Model is usable for a wide variety of blockchain systems and can be extended easily to include system or deployment particulars. The BlockSim software tool provides a simulator that implements the Base Model in Python. This chapter describes the Base Model, the simulator implementation, and the application of BlockSim to Bitcoin, Ethereum and other consensus algorithms. We validate BlockSim simulation results by comparison with performance results from actual systems and from other studies in the literature. We close the chapter by a BlockSim simulation study of the impact of uncle blocks rewards on mining fairness, for a variety of blockchain configurations.

4.1 Introduction

In the design as well as the deployment of blockchain solutions, many architectural, configuration and parameterisation questions need to be considered. Since it is usually not feasible or practical to answer these questions using experimentation or trial-and-error, model-based simulation is required as an alternative. In this chapter, we propose a discrete-event simulation framework called BlockSim [8] to explore the effects of configuration, parameterisation and design decisions on the behaviour of blockchain systems.

The main goal of BlockSim is to provide simulation constructs that are intuitive, hide unnecessary detail and can be easily manipulated to be applied to a large set of blockchains design and deployment questions. That is, BlockSim has the following objectives: generality, extensibility and simplicity. BlockSim is intended to be used by blockchain designers, analysts and researchers to explore and study performance metrics (throughput and latency), functionality metrics (e.g., stale rates) and system properties (e.g., mining fairness and mining incentives). We note that BlockSim is generally designed for performance evaluation, thus it cannot be used for pure security analysis or formal correctness of consensus protocols and smart contracts.

At the core of BlockSim is a Base Model, which contains model constructs at three abstraction layers: the network layer, the consensus layer and the incentives layer [96]. The network layer captures the blockchain's nodes and the underlying peer-to-peer protocol to exchange data between nodes. The consensus layer captures the algorithms and rules adopted to reach an agreement about the current state of the blockchain ledger. The incentives layer captures the economic incentive mechanisms adopted by a blockchain to issue and distribute rewards among the participating nodes.

The Base Model contains a number of functional blocks common across blockchains, that can be extended and configured as suited for the system and study of interest. The main functional blocks include Node, Transaction, Block, Consensus and Incentives, as we describe in Section 4.2. These are then implemented through a number of Python modules, discussed in Section 4.3, and complemented by modules (event, scheduler, statistics, etc.) that implement the simulation engine.

The public nature of permissionless blockchains provides for particularly powerful opportunities to validate the simulator. We validate the BlockSim simulation results by comparing against theoretical results (invariants such as block rate), against data from the existing public blockchain systems such as Ethereum and Bitcoin and against

results from the literature. The BlockSim simulation results are within a statistically acceptable margin of the real-life or published results, as discussed in Section 4.5. We also demonstrate the use of BlockSim for a simulation study that considers stale rate, throughput and mining fairness, for a range of possible blockchain configurations (not all existing in real-life systems). Using BlockSim we can demonstrate that uncle inclusion (such as in Ethereum) is beneficial for mining fairness.

The structure of the chapter is as follows. Section 4.2 discusses the core Base Model of BlockSim including the design objectives behind it. Section 4.3 presents the implementation of the Base Model. Section 4.4 presents the application of BlockSim to Bitcoin, Ethereum and other consensus protocols as case studies. Section 4.5 discusses the validation of BlockSim against actual systems and studies from the literature. Sections 4.6 and 4.7 show a BlockSim simulation study as well as the evaluation of BlockSim against the design objectives. Finally, we conclude the chapter in Section 4.8.

4.2 BlockSim Base Model

In this section, we introduce the *Base Model* underlying BlockSim, which is designed to model any kind of blockchain system, with application specific extensions as needed. We first define the design principles and goals for BlockSim: generality, extensibility and simplicity. Then, we discuss the design layer by layer: Network Layer, Consensus Layer and Incentives Layer. Within each layer we identify the key functional units (entities) and the actions or activities it executes.

4.2.1 Design Principles

We design a Base Model to fulfill the main design goals for BlockSim, which are:

- **Generality:** we want to be able to use BlockSim for a large set of blockchain systems, configurations and design questions.
- **Extensibility:** BlockSim should be easily manipulated by a designer or analyst to study different aspects of blockchain systems.
- **Simplicity:** the above two objectives should be met while making BlockSim easy to use, both for simulation studies and for extending it.

The art of designing a tool such as BlockSim is to find a useful trade-off between generality and extensibility on the one hand, and simplicity to achieve these two objectives on the other hand. The Base Model is critical in achieving this goal, aiming to find the optimal trade-off among the above three objectives for the domain of blockchain systems.

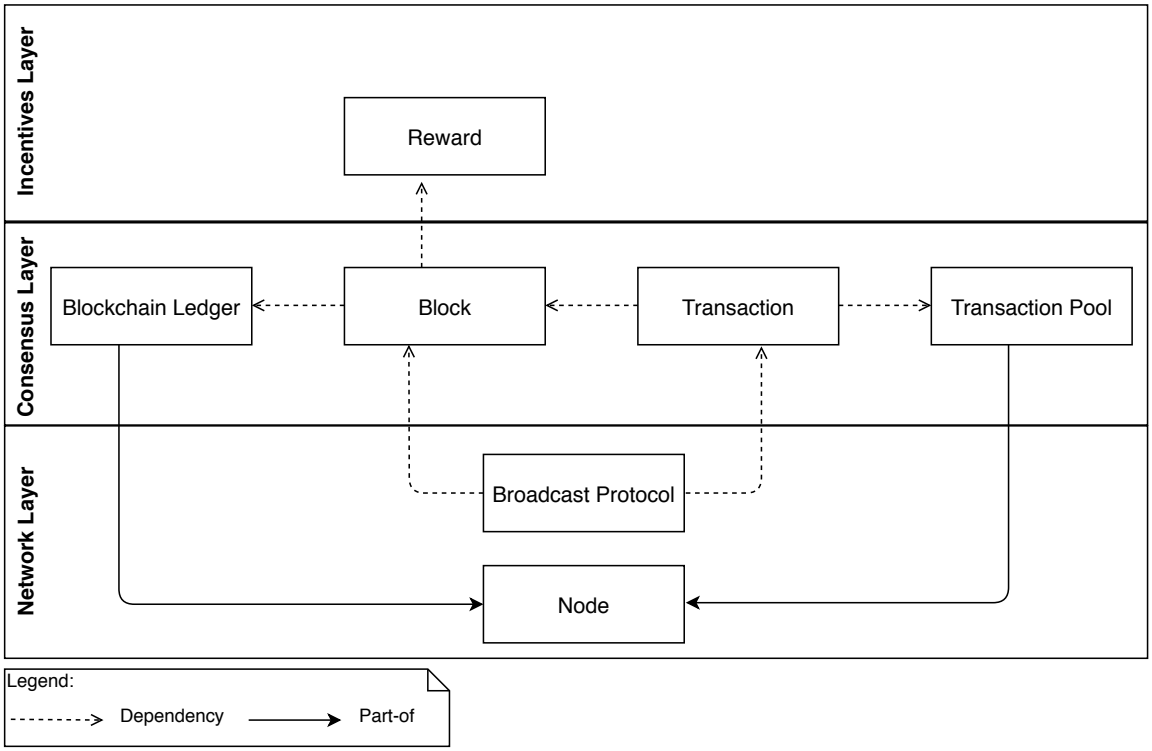

Figure 4.1: BlockSim Model Entities.

The Base Model identifies the key building blocks (e.g., blocks, transactions, nodes and incentives) common across all blockchains BlockSim is meant for, see Figure 4.1. The Base Model dictates how general the model class is that is supported by BlockSim, and particularly how easy it is to build new models. The Base Model will be translated in software modules and therefore also determines if BlockSim can be extended easily, for instance, to provide more detailed models of certain processes that take place in blockchains.

4.2.2 Network Layer

This layer defines two entities *Node* and the underlying *Broadcast protocol*, as depicted in Figure 4.1. The *Node* entity is responsible for updating the system state

variables (e.g., the blockchain ledger and the transaction pool). The *Broadcast protocol* specifies how information entities (e.g., *Blocks* and *Transactions*) are propagated in the network.

Both *Blockchain ledger* and *Transaction pool* entities are part of the *Node* entity (see Figure 4.1). That is, every node maintains and continuously updates these entities. We model nodes as objects that have different attributes such as unique ID, balance, local ledger and transaction pool. The transaction pool and the local ledger are modelled as array lists that can be extended when new transactions and blocks are received. These attributes are common across the different implementation of blockchains. It could, however, be possible to extend this by including more additional attributes, as we will show in Section 4.4.1.

The propagation of information entities depends on the *Broadcast protocol* entity, which can be modelled in detail by accounting for the network configurations, the geographical distribution of the nodes and the connectivity among the nodes, or it can be modelled in an abstraction level by only considering a time delay for propagating information among the nodes. The reason for abstracting the broadcast protocol is to make our simulator as simple as possible by hiding unnecessary details. This will alleviate the user of the simulator from configuring many parameters related to the network configurations such as the broadcast protocol, the geographical distribution of the nodes and the number of connections per node. Having the propagation delay as the only configurable parameter will improve both the efficiency and the usability aspects of the simulator.

4.2.3 Consensus Layer

This layer aims at establishing the rules that nodes can follow to reach an agreement about the blockchain's state. This layer includes four entities, namely, *Transaction*, *Block*, *Transaction pool* and *Blockchain ledger*, as depicted in Figure 4.1.

The *Blockchain ledger* entity depends on the *Block* entity, and the *Block* entity depends on the *Transaction* entity. That is, the blockchain ledger is composed of blocks and blocks are composed of transactions. The *Transaction pool* depends on the *Transaction* entity, as every transaction created is fed into the transaction pool. The *Node* entity maintains these four entities.

Within the consensus layer, there are several activities or actions to be executed by the entities. The creation of blocks and transactions is an example of such activities. The flow of these activities is depicted in Figure 4.2. These activities run continuously, transactions and blocks, for instance, always keep arriving in the network.

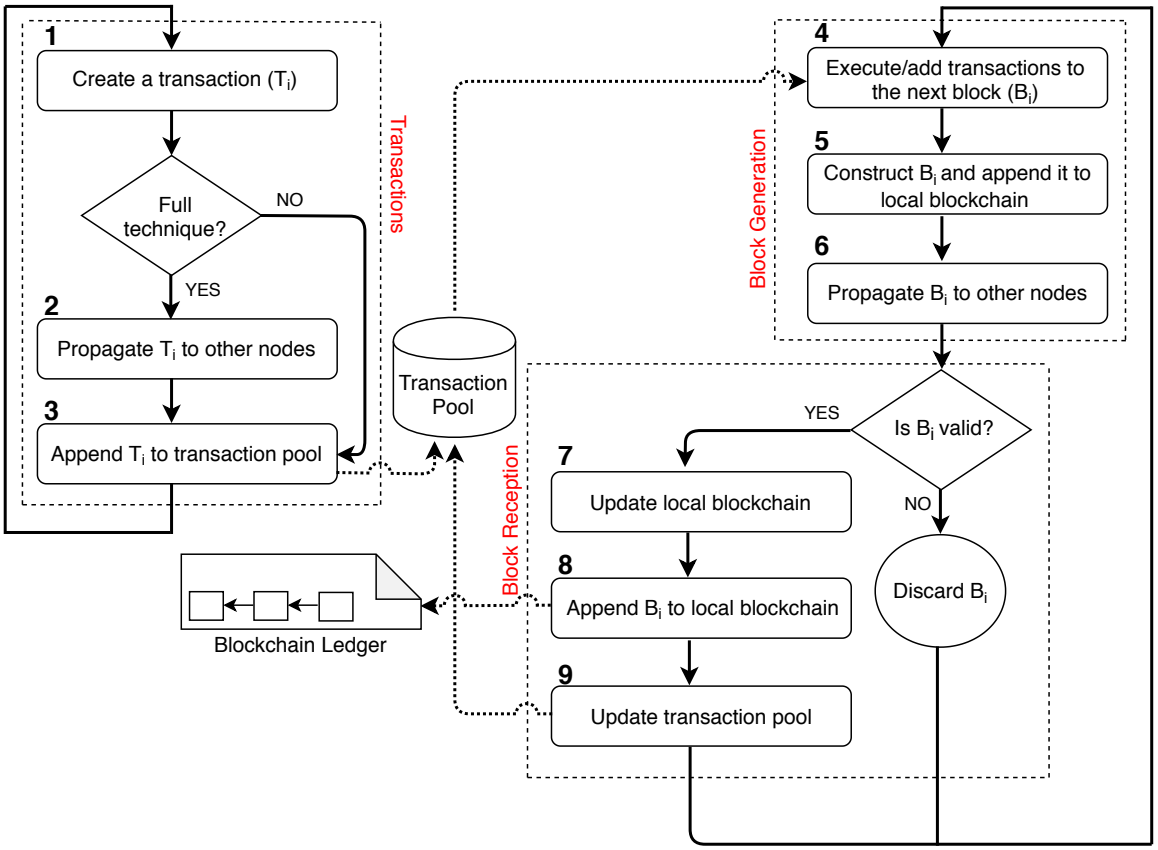

Figure 4.2: Workflow for the consensus activities within the Base Model of BlockSim.

Transactions are one of the building blocks (entities) common across all blockchain systems. It plays a significant role in updating the blockchain's state. The arrival of a new transaction in the network results in updating the transaction pool by inserting that transaction.

We model transactions in two different ways, namely, full and light. The full technique helps to track each transaction in the system (e.g., when a transaction has been created and included in a valid block). This technique models transactions as in any blockchain system, and it is useful if one is interested in, for instance, studying the latency of individual transactions in blockchain systems. However, this type of modelling consumes an enormous amount of computing resources and time during the simulation since each transaction has to be tracked. On the other hand, the light technique does not track each transaction. It is useful when studying the throughput of blockchain systems without caring about the confirmation time of transactions within the system.

In both techniques, we model transactions as objects that have several attributes or fields such as transaction ID, size, fee, timestamp, contents as well as the submitter and the recipient of the transaction. These attributes are almost common across all blockchains, and that some systems have more additional attributes (e.g., Ethereum has also gas-related attributes such as Gas Limit).

Full modelling Technique: In this technique as we discussed in Section 4.2.2, we model an individual transaction pool for each node by assigning an array list for each node as a way to abstract the pool. Each transaction created by a node is propagated to all other nodes in the network. Upon receiving the transaction, the recipient node appends it to their pool. Thus, we model transactions in three different activities labelled from 1 to 3, as depicted in Figure 4.2.

- **Creating transactions:** This involves generating transactions by the participating nodes. The number of transactions to be created per unit of time can be controlled and configured.
- **Propagating transactions:** This requires the creator of the transaction to propagate it to other participating nodes. This is to notify other nodes about the newly created transactions.
- **Appending transactions:** This requires the recipient of the transaction to append it to their transaction pool.

Light modelling Technique: In this technique, we only model a single transaction pool to be shared among all nodes in the network. The intention behind this technique is to provide an alternative and simplified way to model transactions by omitting the propagation process as well as the needs for nodes to update their pools continuously. Thus, the light technique is more efficient and faster during the simulation. However, this technique cannot be used to draw conclusions about the latency of transactions as transactions are not tracked. Nevertheless, it is useful to get indicators about the throughput in blockchain systems.

In this technique, we create a set of transactions (N) and then append it to the shared pool before the mining process, so miners can access the pool to select several transactions to include in their forthcoming block. Note that N should be more than enough for a block, usually enough for two blocks. Once a miner has successfully generated a block, the pool is reset and then filled up with a fresh set of transactions to be included in the next block.

Both techniques could be implemented and then the user would be given a choice to select which method to adopt based on their own needs. For instance, if one is only interested in throughput, there is no need for choosing the full technique since it makes the simulator runs for a very long time.

Blocks are another essential building block (entity) of any blockchain system. Blocks consist of transactions. The arrival of a new block results in an update in the transaction pool and blockchain ledger. The pool is updated by removing all transactions included in the block, while the ledger is updated by appending the newly created block.

We model blocks as objects that have several attributes, namely, depth, block ID, previous block ID, timestamp, size, miner ID and transactions. The block ID is a unique identifier for the block. The block depth indicates the index of the block in the node's blockchain. The miner ID refers to the node that created the block. Each block can accept a list of transactions as its content. These attributes are common across blockchains.

We model blocks in the consensus layer as *Block Generation* and *Block Reception*, see Figure 4.2. Block generation specifies when blocks are generated as well as which node is eligible for appending the next block. It covers all the common actions required by a miner to create and attach a block to the blockchain ledger. The actions embrace executing the block's transactions, constructing and appending the block to the local blockchain and propagating the block to other nodes in the network. Block reception specifies how the network's nodes update their blockchain ledgers upon receiving new blocks. It covers the common activities taken by a node when receiving a newly generated block. Upon receiving a valid block, the recipient node will perform three actions, which are updating the local blockchain if necessary, appending the block to the local blockchain and updating the transaction pool.

The consensus algorithm is responsible for selecting a miner to build the next block. The methodology used to choose a miner varies among blockchains, depending on the adopted consensus protocol. In PoW, for instance, miners are selected based on solving a mathematical task. Once a miner is chosen to construct and append a new block to the ledger, the miner would undertake the following actions. Note that these actions are common across all blockchain systems, and that some specific systems may include other activities (e.g., including uncle blocks in a future block as in Ethereum).

- **Executing and adding transactions to the block:** This requires the miner to select several pending transactions to be executed and included in the next block. Often, miners first sort those pending transactions based on their associated fees. Then, miners select the best transaction according to their ranking criteria, execute it if and only if there is space in the block. The transaction will then be recorded in the block. After that, miners will select the next transaction and continue until the block is full or there is no pending transaction.

- **Constructing and appending the block to the local blockchain:** After preparing the block content (e.g., transactions), the miner would construct the block after which the block will be appended to the miner's local blockchain.

- **Propagating the block to other nodes:** This is to propagate the block to other nodes in the network. This is to notify the network's nodes about the newly generated block.

Once a node has received a new block, it will check its validity. The block is considered valid if it was constructed correctly and all embedded transactions were correctly executed. Beside the block validity, the block must point to the last block in the ledger (the block's depth should be higher than that of the last block). We only model the block depth, and thus, we abstract the validity of the block. If the depth of the received block is not higher than that of the last block, the block will be discarded. Otherwise, the node will perform the following actions.

- **Updating local blockchain:** This requires the recipient node to update its local blockchain, where necessary, before appending the newly received block. This is because sometimes the received block is built on different preceding blocks (a different chain branch) compared to the ones the recipient node has or because it is built on missing blocks. Therefore, the node has to update all the preceding blocks (and fetch all missing blocks if any) according to the ones the received block is following.

- **Appending the block to local blockchain:** This is to append the received block to the local copy of the blockchain.

- **Updating transaction pool:** This requires the recipient node to update its transaction pool, where necessary, upon appending the newly received block. This is to remove all the transactions that have already been executed in the received block from the node's pool.

Transaction pool and blockchain ledger are also important building blocks (entities) since they represent the state of blockchain systems. The transaction pool is updated upon the arrival of a new transaction or block, while the blockchain ledger is only updated once a block has arrived. Nodes are responsible for updating both the pool and the ledger, as every node in the blockchain network maintains a local copy of them (see Section 4.2.2).

The rule of updating the ledger in the case of forks: Nodes at some point in time may have different views of the blockchain ledger due to the network's propagation delay. A significant role of the consensus layer is to define the rules that can be used by the nodes to resolve the forks. For instance, Bitcoin and Ethereum use the longest-chain rule to resolve the forks. That is, nodes update their ledgers every time they receive a block that follows a chain that is longer than their local chains. By doing so, nodes will have the same view of the blockchain ledger. Other systems, however, use different rules (e.g., GHOST [90]).

4.2.4 Incentives Layer

The incentives layer is responsible for designing the underlying incentive model by defining the rewarded elements (e.g., blocks and transactions) as well as distributing the rewards among the participating miners. This layer has the *reward* entity, which depends on the *Block* entity (see Figure 4.1). That is, the rewards are only given to the miners upon appending new blocks to the ledger. The calculation and the distribution of such rewards are considered as actions.

We model the basic incentive model used by most blockchain systems such as Bitcoin. Our model provides a reward for generating a valid block (block reward) and a reward for all transactions included in a block (transaction fee). The block reward is modelled as a fixed amount of cryptocurrency that can be configured and changed by the end-user. The transaction fee is calculated as the multiplication of its size and its prize, where the prize is the amount of money the submitter of the transaction is willing to pay per unit of size. The size and the prize for transactions can also be configured as fixed or variable (random) values. However, it is possible to extend the current model to include different rewards (e.g., rewards for uncle blocks) or change the way how the fee for transactions is calculated. We model the distribution of rewards by increasing the balance of each miner after having a valid block attached to the ledger.

4.3 BlockSim Implementation

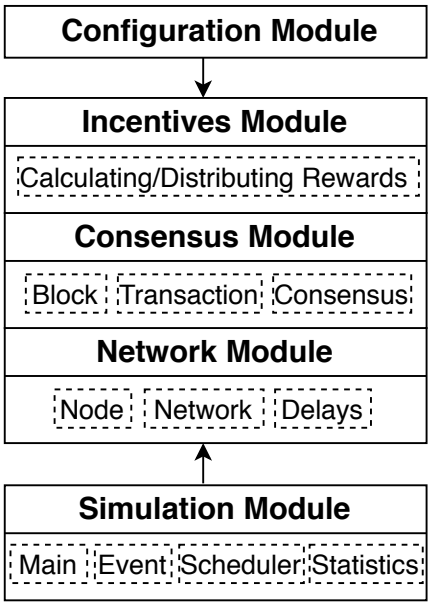

Figure 4.3: BlockSim Implementation Modules.

We present the implementation of the BlockSim simulator using Python 3.6.4[1]. The main modules are given in Figure 4.3. The Simulator Module implements the core engine of the simulator, in particular the event scheduler, which we explain in Section 4.3.1. The main topic of discussion in that section is the granularity at which events are handled, since it heavily impacts the performance of the simulator. This simulation engine module is complemented by the Configuration Module, to be described in Section 4.3.2, which provides the user with ways to configure the simulation model and experiments. Section 4.3.3 explains the implementation of the Base Model, subdivided according to the main layers: Network Module, Consensus Module and Incentives Module.

4.3.1 BlockSim Simulation Engine and Event Scheduler

As depicted in Figure 4.3, the main Simulation Module contains four classes, which are Event, Scheduler, Statistics and Main. We start with explaining our design choices for the event scheduling.

We provide event scheduling at two abstraction levels, the first one considers blocks as the event 'unit', the second considers transactions as the event 'unit'. We

[1]https://github.com/maher243/BlockSim.

explain the block-level events. The class *Event* defines the structure of events in our simulator. In the case of a block-level event it has four attributes: *type*, *nodeID*, *time* and *block*. The attribute *type* indicates how to handle the event, in particular whether the event at hand is to *create a new block* or to *receive an existing block*. The *nodeID* and *time* attributes specify the node that handles the event and the time at which the event takes place. The *block* attribute contains the necessary information for the block to be handled.

```
for i in range (p.Runs):
        clock=0
        if p.hasTrans:
            if p.Ttechnique == "Light": LT.create_transactions()
            elif p.Ttechnique == "Full": FT.create_transactions()

        Node.generate_gensis_block() # generate the gensis block for all miners
        Scheduler.generate_initial_events() # initiate initial events >= 1 to start with

        while  not Queue.isEmpty() and clock <= p.simTime:
            next_event = Queue.get_next_event()
            clock = next_event.time # move clock to the time of the event
            Consensus.handle_event(next_event) # execute the event (create or receive a block)
            Queue.remove_event(next_event) # remove the event from the Queue after executing it

        Consensus.fork_resolution() # resolve the forks
        Incentives.distribute_rewards()# calculate and distribute the rewards between the participating nodes
        Statistics.calculate() # calculate the simulation results (e.g., block statstics and miners' rewards)

        # reset all global variables and the blockchain states for all nodes before the next run
        Statistics.reset()
        Node.resetState()

Statistics.print_to_excel("results.xlsx") # print all the simulation results for all the runs in an excel file
```

Listing 4.1: The implementation of the Main class in BlockSim.

Scheduler class is responsible for scheduling future events and record them in the *Queue*. *Queue* is an array list that maintains all future events, and it is continuously updated during the simulation by either inserting new events or removing existing ones. At the block-level, for instance, once a block is created through a *block creation* event, the *Scheduler* class schedules *block reception* events for other nodes to receive

the block. Also, it schedules a new *block creation* event by selecting a miner to propose and generate a new block on top of the last one.

The function of the *Main* and *Statistics* classes is as one would expect. *Main* runs the simulator. It prepares the setup and then triggers the *Scheduler* class to schedule some initial events. The setup includes the creation of transactions as well as the creation of the first (genesis) block, an empty block that will be attached to the local blockchain for all the nodes in the network. Then, it keeps going through all the events and executes them one by one until the *Queue* is empty or the pre-specified simulation time is reached. The procedures taken to execute the events are similar to any discrete-event simulator [46], which are depicted in Listing 4.1. *Statistics* maintains the results and calculates the statistics of the final output of the simulation, including block statistics (number of blocks included in the ledger and percentage of discarded blocks), throughput and mining profits.

4.3.2 Configuration Module

Type	**Parameter**	**Description**
Blocks	$B_{interval}$	Average time to generate a block in seconds
	B_{size}	Block size in Megabyte (MB)
	B_{delay}	Propagation delay of blocks in seconds
	B_{reward}	Block generation reward
Transactions	hasTrans	Enable/Disabled transactions
	$T_{technique}$	Technique for modelling transactions
	T_{n}	Rate at which transactions can be created
	T_{delay}	Propagation delay of transactions in seconds
	T_{fee}	Transaction fee
	T_{size}	Transaction size in MB
Nodes	N_{n}	Total number of nodes in the network
Simulation	Sim_{time}	Length of the simulation time
	Runs	Number of simulation runs

Table 4.1: Input parameters for the simulator.

This module serves as the main user interface, in which users can select from the available models as well as configuring various parameters related to the participating nodes, blocks, transactions, consensus, incentives and the simulation setups. Table 4.1 summarises the input parameters to be configured before running the simulator. We can, for instance, configure the number of nodes, the block interval time, the volume of transactions to be created per second and other parameters. Besides, our simulator allows disabling transactions if they are not of interest. This can be done by

only setting the parameter *hasTrans* to be "False", without modifying the code of the simulator. Furthermore, it allows selecting a suitable technique (either full or light) for modelling transactions. If we extend the simulator by, for example, including new consensus protocols, this would be reflected in this module to allow the user of the simulator to choose the desired protocol.

4.3.3 Base Model Modules

We discuss the implementation of simulation classes that represent the Base Model of Section 4.2 using the same three layers as before.

Network Module: We implement the network module in two different classes, namely, Node and Network. *Node class* defines the structure of nodes in our simulator. We implement each node as an object in which each node is given a unique ID and a balance. For each node, we assign two array lists to model the local blockchain and the transaction pool. It is worth noting that each node maintains a transactions' pool only if the full transaction technique is applied. Otherwise, a common pool will be shared by all the nodes. *Network class* implements the network latency for propagating both blocks and transactions between the nodes. Currently, we implement the latency as a time delay that can be configured by the user of the simulator in the configuration module. However, it could be possible to extend this class to implement a particular broadcast protocol.

Consensus Module: We implement the consensus module in different classes, namely, Transaction, Block and Consensus. *Transaction class* defines the structure of transactions in our simulator. We implement each transaction as an object that has seven attributes, namely, ID, timestamp, submitter ID, recipient ID, value, size and fee. The end-user can set the size and fee of transactions in the configuration module as fixed values or random values drawn from general distributions, including exponential distribution. This class also implements both full and light techniques for modelling transactions, as we discussed in Section 4.2.3. *Block class* defines the structure of blocks in our simulator. We implement each block as an object that has seven attributes, namely, depth, ID, previous ID, timestamp, size, miner ID and transactions. This class also implements the processes required by the nodes to generate and receive blocks, as discussed in Section 4.2.3. *Consensus class* implements the consensus algorithm as well as the fork resolution rule. It also implements the process of selecting leaders, aka miners, to generate and append new blocks to the ledger. This class is structured to be easy to implement any consensus protocol of

interest. For instance, to implement PoW algorithm with the longest-chain rule to resolve potential forks as the case in Bitcoin and Ethereum.

Incentives Module: This module is responsible for setting the rewarded elements as well as calculating the rewards. Also, it distributes the rewards among the participating nodes by increasing the balance of each node after calculating the rewards. It is, however, possible to extend this module by adding more rewarded elements or changing the way the awards are calculated if required. To make it easier for the end-user, the rewards (e.g., block rewards) can be configured and changed in the configuration module.

4.4 BlockSim Case Studies

BlockSim is designed to be used for any type of blockchain, and to demonstrate this we apply the Base Model of BlockSim to simulate Bitcoin as well as Ethereum. We also discuss how to extend the BlockSim implementation of the Base Model to support any consensus algorithm of interest.

4.4.1 Bitcoin in BlockSim

To simulate Bitcoin we introduce the following modifications and extensions to the core implementation of BlockSim discussed in Section 4.3.

Network Layer. For Bitcoin we abstract the underlying broadcast protocol by modelling the propagation of transactions and blocks as a time delay, as indicated in Section 4.2.2. To parameterise the model one can use DSN Bitcoin Monitoring to obtain the propagation delay of information. The Node module is extended with an attribute for a node's hash power, which we add to the configuration module for the user to set as an input parameter. To distinguish between regular nodes and miners, we can assign zero as the hash power for regular nodes to indicate that the node cannot build blocks (only create and propagate transactions).

Consensus Layer. Bitcoin uses PoW with the longest-chain rule to resolve the forks. As discussed in Section 2.2.2, in PoW miners compete against each other to be allowed to create the next block. They repeatedly draw a random number, combine it with info from the new block and generate a hash. If the hash fulfills some property, the block can be added to the blockchain and forwarded to other nodes. That means miners execute what amounts to a Bernouilli trial and since the number of trials is high, the Bernouilli trials process converges to its continuous-time counterpart, the Poisson Process. That is, the time between successes is exponentially distributed.

In the configuration module, one can set the block difficulty through the $B_{interval}$ parameter, which is the time interval (in seconds) between two consecutive blocks. If multiple chains have the same depth, Bitcoin uses the longest chain to reach a global view of the blockchain ledger by resolving the forks.

Incentives Layer. The incentives in Bitcoin for generating blocks and executing transactions is the same as that of the Base Model. In our main BlockSim implementation, all rewards will be distributed to miners at the end of each simulation run. If needed, the Incentives module can be modified to distribute rewards in run-time. The miner of a block that is finalised and is part of the longest chain receives the block reward and the fees for all transactions included in that block. The rewards can be set in the configuration module.

4.4.2 Ethereum in BlockSim

Ethereum is very similar to Bitcoin but introduces a few additional elements associated with the handling of uncle blocks as well as attributes required for incentives associated with smart contracts.

Network and Consensus Layers. Ethereum allows attaching uncle blocks to a valid block and rewards miners for this. Therefore, we extend the Bitcoin Node module with an unclechain attribute. The unclechain for a node is modelled as an array list storing all chains with uncle blocks that occur during the simulation run. Ethereum allows miners to include a maximum of 2 uncle blocks within the last seven block generations (e.g., an uncle block with a depth 10 can be referenced in a block with a depth less than or equal to 17). We include this logic in the configuration module and allow configuring the maximum number of uncle blocks per block, the number of generations in which an uncle block can be included as well as disabling uncle inclusion mechanism if it is not of interest.

Similarly, we extend the Node module when receiving a block. If the block has a smaller depth or index, the block is appended to the recipient's unclechain as an uncle block to be referenced in a future block. Also, when receiving and appending a valid block to the local blockchain ledger, the miner updates its local unclechain, where necessary, by removing all the uncle blocks that have already been included in the received block.

Incentives Layer. The incentive model of Ethereum, similar to that of Bitcoin, includes block reward and transactions fee. Yet, Ethereum uses the Gas mechanism to calculate the fee for transactions with smart contracts. To determine the fee for transactions and blocks, we therefore require some additional attributes related

to the gas model. For transactions, we add Gas Limit, Used Gas and Gas Price attributes. For blocks, we include the attributes of Gas Limit and Used Gas. We refer to the literature, e.g., [6, 49] for details, but in short, Used Gas multiplied by the Gas Price corresponds to the fee the miner receives, where Used Gas depends on the computational requirements of the smart contract [3], but never exceeds Gas Limit.

Ethereum also introduces rewards for uncle blocks. The uncle reward is distributed between the miner who generated the uncle and the miner who included it in his block, as follows [49]. The miner who generated the uncle gets a variable reward depending on when the uncle has been referenced in a main block. The sooner the uncle is referenced in a block, the higher the uncle reward (R_{uncle}):

$$R_{\text{uncle}} = \left(D_{\text{uncle}} + (G_{\text{uncle}} + 1) - D_{\text{block}}\right) * \frac{R_{\text{block}}}{G_{\text{uncle}} + 1} \tag{4.1}$$

where D_{uncle} is the depth of the uncle, G_{uncle} is the number of generations in which the uncle can be included, D_{block} is the depth of the block and R_{block} is the block reward. The miner who included the uncle in his block will get a fixed reward, which is calculated as $\frac{1}{32}$ * R_{block}. All this is implemented in the incentives module, but the amount of rewards can be set in the configuration module, if required.

4.4.3 Different Consensus Protocols in BlockSim

Thus far we have mainly considered PoW as consensus protocol, but there are many other, including Proof of Stake (PoS), Proof of Authority or message-based consensus algorithms such as Practical Byzantine Fault Tolerance (PBFT) and its many variants [14].

A significant difference between these protocols and PoW is that in PoW miners are not directly selected by the consensus protocol, but instead, miners continuously invest their computing power to create the subsequent blocks. In PoS, for instance, miners would be selected by the protocol based on the amount of stake or cryptocurrencies they hold. The more cryptocurrencies a miner deposited in the system, the more chance they would be selected to generate the next block. Other protocols select miners in a round-robin manner such as Tendermint [64] or based on different metrics [14].

To support approaches such as PoS, we modify the consensus class by changing how miners are being selected to generate the next blocks. Other consensus elements (e.g., transactions, blocks and fork resolution) and modules (simulation, network and incentives) remain unchanged. In general, as long as the output metrics can be

truthfully simulated with events scheduled at the granularity of blocks, BlockSim can be extended in a natural matter. The time consumed by the consensus algorithm would then be represented by a delay. However, if one wants to analyse the impact of specific message sequences on the performance of PBFT style consensus protocols, BlockSim is a less obvious candidate. For efficient (i.e., fast) simulation, one would study such consensus protocols through simulation tools that operate at message-level and not mix different levels of abstractions and time granularity.

4.5 BlockSim Validation

A nice feature of the blockchain design is that it offers invariants (such as the block creation interval) and plenty of publicly available data to validate the results of any simulator. First we compare BlockSim with existing blockchain systems (Section 4.5.1), then we compare with various peer-reviewed studies (Section 4.5.2).

4.5.1 Comparison with Measurements

We compare the results from BlockSim with the most popular public blockchains, Bitcoin and Ethereum. These provide certain 'invariants' that we know to be true, such as the frequency of generating blocks and the proportionality between the miner's hashing share and the probability to win the Proof of Work competition. Bitcoin and Ethereum also provide ample public data to validate our simulator.

Parameters	Bitcoin	Ethereum
$B_{interval}$	596s	12.42s
B_{delay}	0.42s	2.3s
B_{size}	0.83MB	7,997,148Gas
T_{size}	546Byte	Distribution

Table 4.2: Data gathered from Bitcoin and Ethereum, serves as input to the simulation runs used as validation.

Validation of block and transaction metrics. We use the following metrics for validation: number of blocks created, number of uncle or stale blocks (blocks that will not be part of the final chain), and the number of transactions completed per time unit. The results obtained from our simulator and that from the actual systems are reported in Table 4.3. We report both the average and the 95% confidence interval values, for a run of the simulation that corresponds to a full month of real time.

From Table 4.3, we see that our simulator's confidence interval contains the result from the measurements. However, our simulator shows a slightly higher throughput for Ethereum compared to the real data observed. We believe that this is either due to the small sample of transactions retrieved or the fitted frequency distribution.

Bitcoin	Measured	Simulated
B_{included}	146 ± 4	143 ± 5
Stale (uncle) rate	$0.025\,\% \pm 0.051\,\%$	$0.049\,\% \pm 0.069\,\%$
Throughput	2.69 ± 0.09	2.66 ± 0.09
Ethereum	**Measured**	**Simulated**
B_{included}	6083 ± 27	6079 ± 25
Stale (uncle) rate	$12.56\,\% \pm 0.43\,\%$	$12.55\,\% \pm 0.14\,\%$
Throughput	5.99 ± 0.18	6.96 ± 0.03

Table 4.3: Validation of the simulator results by comparison with measurements from Bitcoin and Ethereum. B_{included} is the number of blocks included in the main blockchain per day, the stale (or uncle) rate per day is blocks not in the main chain, and throughput is the number of transactions processed per second.

To obtain the above results, Table 4.2 shows the data gathered from both Bitcoin and Ethereum used as input to the validation runs. That is, we use the values from Table 4.2 for the relevant input parameters given in Table 4.1. We gather the Bitcoin's data from blockchain.info [2], while the Ethereum's data comes from etherscan.io [3]. We collect one month of data for each system as of October 2018. From these sources, we were able to directly collect all the necessary data, apart from the block propagation delay and the transactions' size in Ethereum. However, we obtain the block delay using DSN Bitcoin Monitoring[4] and ETHstats[5]. To obtain the size of transactions in Ethereum, we implement a python script that makes use of etherscan.io APIs to retrieve transactions information. We retrieve the data for the latest 5,000 transactions and then fit a frequency distribution for transactions' size to be used as input in our simulator. For the sake of this experiment, we fit a frequency distribution with the limited collected data.

Validation of PoW. An invariant we can use for validation is the share of blocks each miner generates since it is known that share is equal to the miner's share of the overall hashing power. For instance, if a miner controls 40% of the network's hash

[2]https://www.blockchain.com/explorer
[3]https://etherscan.io/
[4]https://dsn.tm.kit.edu/bitcoin/
[5]https://ethstats.net/

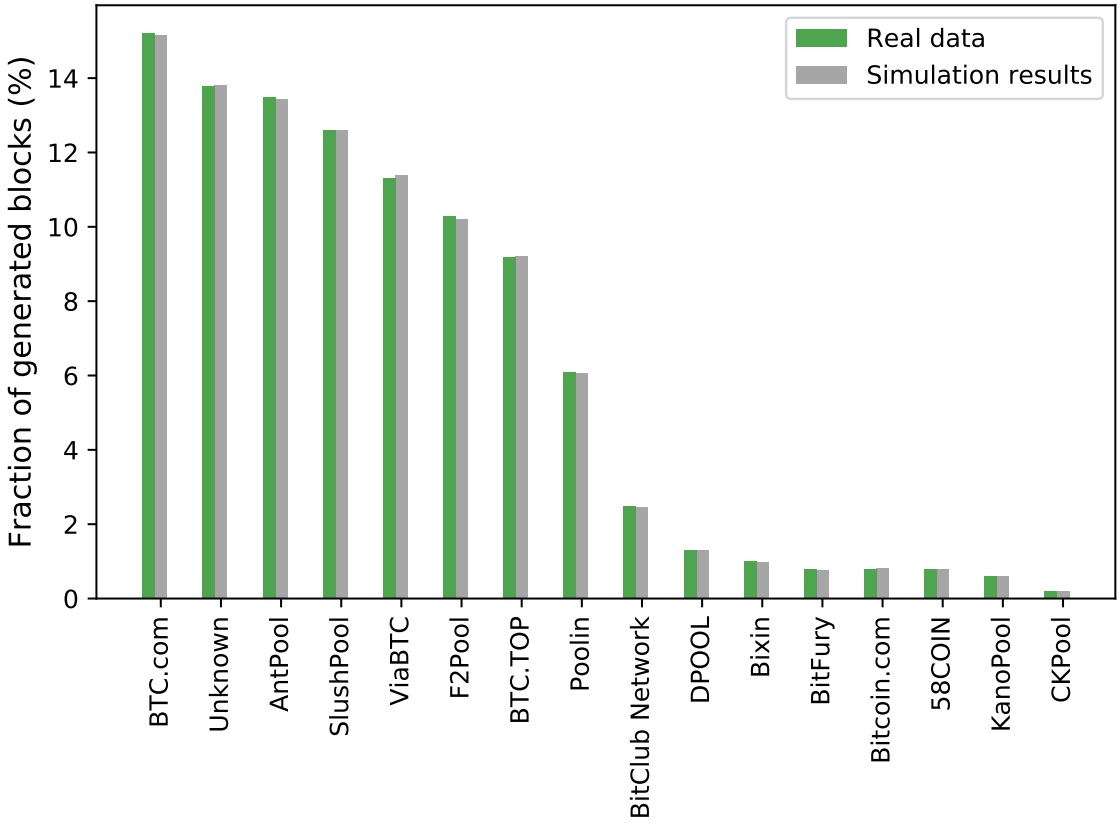

Figure 4.4: Validation of PoW using the fraction of generated blocks given the hashing share of various miners.

power, it should generate 40% of the total blocks. To validate PoW, we collect the estimated hash power as well as the fraction of blocks contributed by Bitcoin miners and miner pools from blockchain.info and input this into our simulator. That is, the simulation is with miners that have the same share of the hashing power as various existing Bitcoin miners.

Figure 4.4 shows the results. We simulate four days of the Bitcoin network, a total of 1000 times and obtain the average fraction of blocks generated by each miner. The x-axis of Figure 4.4 shows the name of the miners and the y-axis shows the fraction of blocks contributed by the miners for both the real Bitcoin network (the green bars) and the simulation results (the grey bars). From Figure 4.4 we see that the simulation results are very close to that of the real Bitcoin network.

	Input Parameters		Stale Rate	
	$B_{interval}$ (s)	B_{delay} (s)	Measured (%)	Simulated (%)
Bitcoin [50]	600	14.7	1.51	1.69 ± 0.08
Bitcoin [35]	600	12.6	1.68	1.73 ± 0.09
Litecoin [50]	150	4.18	1.82	1.88 ± 0.11
Dogecoin [50]	60	2.08	2.15	2.38 ± 0.08

Table 4.4: A comparison between BlockSim and previous studies in terms of the stale rate observed.

4.5.2 Comparison with Peer-reviewed Studies

We also compare the simulator results for the stale rate with that of previous peer-reviewed studies. Decker et al. [35] run an experiment on the Bitcoin blockchain by listening to 10,000 blocks. They found the average block propagation delay is 12.6 seconds and the stale rate is 1.69%. Gervais et al. [50] run some simulation experiments using the configurations of different blockchain systems such as Bitcoin, Litecoin and Dogecoin. They found that their simulation results matched that of the actual systems. To validate our simulator against these studies, we use the same configurations of the block interval ($B_{interval}$) and block propagation delay (B_{delay}) as reported in these studies. We simulate each configuration for a total of 10,000 blocks and report the average results obtained from 10 independent runs, see Table 4.4. From Table 4.4, we see that the stale rates obtained from our simulator are close to the ones reported in previous studies, with a difference of less than 10%.

Input parameters				Mining fairness (% blocks contributed by miners)					
$B_{interval}$ (s)	B_{delay} (s)	Stale rate (%)	Throughput	M1(40%)	M2(30%)	M3(15%)	M4(10%)	M5(5%)	Run time (s)
1	0.5	24.74	1387.74	43.54	30.13	13.51	8.65	4.17	15.9
	2	45.43	997.34	49.42	30.14	10.98	6.501	2.96	16.25
	4	54.81	829.17	54.76	29.33	8.96	4.86	2.09	13.25
	8	62.69	680.77	60.05	28.49	6.71	3.53	1.22	10.51
	16	68.77	569.93	66.36	26.48	4.50	2.0	0.67	8.29
12	0.5	3.49	147.26	40.27	30.15	14.82	9.86	4.90	4.41
	2	11.62	135.11	41.23	30.24	14.40	9.41	4.73	11.19
	4	19.33	123.02	42.60	30.17	13.77	9.09	4.37	14.08
	8	28.8	108.61	44.42	30.10	13.05	8.34	4.08	16.68
	16	39.55	92.09	47.32	30.39	11.83	7.30	3.16	17.64
60	0.5	0.7	30.17	40.06	30.05	14.88	10.15	4.86	2.21
	2	2.79	29.79	40.09	30.07	15.02	9.88	4.95	3.87
	4	5.36	28.6	40.50	29.94	14.94	9.72	4.90	5.93
	8	9.7	27.56	41.23	30.03	14.40	9.60	4.74	10.06
	16	16.45	25.51	42.00	30.24	14.15	9.11	4.51	13.12
150	0.5	0.29	12.23	39.95	29.94	15.07	10.03	5.00	1.86
	2	1.21	12.06	39.81	30.07	15.21	9.89	5.02	2.86
	4	2.23	11.96	40.32	30.06	14.76	9.93	4.94	3.5
	8	4.33	11.62	40.48	29.96	14.86	9.80	4.90	5.11
	16	8.15	11.3	40.89	29.78	14.82	9.77	4.75	7.95
600	0.5	0.08	3.03	39.97	29.98	15.03	9.96	5.06	1.67
	2	0.32	3.05	40.00	29.90	15.11	10.00	4.99	1.84
	4	0.61	3.05	40.10	30.24	14.83	9.85	4.98	2.23
	8	1.14	3.01	39.91	29.99	15.17	9.91	5.02	2.91
	16	2.26	2.99	39.98	29.99	15.07	10.04	4.92	3.67

Table 4.5: The simulation results (stale rate, throughput and the fraction of blocks contributed by each miner) as well as the run time performance (in seconds) for different combinations of block interval and block propagation delay.

4.6 BlockSim Simulation Results

To show the applicability of our simulator, we conduct a simulation experiment to investigate the impact of different consensus and network parameters on the security, performance and mining ecosystem of blockchain systems. We also show the performance of the simulator in terms of run time. We use very similar metrics as in the validation, but for a wider range of parameter values. The main discussion in this section is about how the stale block rate impacts mining fairness and how Ethereum's approach to reward uncle blocks improves mining fairness.

More precisely, we study the impact of different combinations of block interval and block propagation delay on the stale rate, throughput and mining fairness. Stale rate is a security indicator of a blockchain system, and the lower the rate, the better for the security of the system [50]. Throughput represents the number of transactions that can be processed per second, thus directly indicating how well the system performs. Mining fairness indicates that the fraction of blocks a miner includes in the main ledger is proportional to the hash power of that miner. In other words, mining fairness means each miner gets a fair reward compared to its hash power.

Table 4.5 shows the results (stale rate, throughput and mining fairness) for 25 different combinations of different block interval B_{interval} and block delay B_{delay} as well as the run time for every configuration. For ease of presentation, we consider only five miners (M1, M2, M3, M4, M5) with hash powers ranging from 5% to 40%. The hash power for a miner is a configurable parameter (see Section 4.4.1). For all configurations, we set the block size to be 1MB and the average transaction size to be 546 bytes (as in the Bitcoin network). We simulate each configuration for a total of 10,000 blocks and report the average results from 10 independent runs. The confidence intervals are not reported here, but are all within 10% of the average values.

Stale rate. From the stale rate results reported in Table 4.5, we observe the following. First, reducing the block interval, i.e., the time between successive blocks being created, leads to higher stale rates, especially when the block interval is already small. For instance, reducing the block interval from 12 to 1 second in the case of 0.5 second block delay will result in an increase of the stale rate by about sevenfold. When the block interval is small, other nodes could manage to find the next block before hearing of other competitive blocks due to the network latency, leading to conflicts. Also, increasing the block propagation delay leads to higher stale rates. For instance, the

stale rate increases about tenfold when increasing the delay from 0.5 to 16 seconds in the case of 12 seconds block interval. The block delay includes the block's transmission time as well as the verification of the block and its embedded transactions [35]. Thus, the bigger the block size, the more time required to transmit and verify the block. We note that increasing the block size will result in higher stale rates. Furthermore, to ensure the lowest stale rate the block delay should be as small as possible and the block interval as large as possible. For instance, in the case of 600 seconds block interval, the stale rates are minimal since the block delay is only a tiny fraction of the block interval.

Throughput. From the throughput results reported in Table 4.5, we observe the following. First, reducing the block interval leads to higher throughput. This is because more blocks will be generated, and thus, more transactions will be processed. We also observe that the block delay could reduce the throughput significantly, especially when the block interval is small. The number of transactions that can be processed per second is reduced from 147 to 92 when increasing the block delay from 0.5 to 16 seconds in the case of 12 seconds block interval. Furthermore, the block delay does not have a significant impact on the throughput if the delay is too small compared to the block interval. For instance, in the case of 600 seconds block interval, the throughput achieved is almost the same even when the block delay is increased from 0.5 to 16 seconds.

Mining fairness. From the mining fairness results reported in Table 4.5, we observe the following. First and most importantly, we observe a correlation between stale rates and mining fairness. The smaller the stale rates the better the mining fairness and vice versa. In the discussion about stale rates, we observe that reducing the block interval or increasing the block delay can lead to a higher stale rate. That is, reducing the block interval leads to poor mining fairness. In the case of 1 second block interval, for instance, miners with a large hash power (e.g., M1) have a higher fraction of blocks included in the main ledger, and thus gain higher profit, compared to their hash power invested. On the contrary, small miners have a small fraction of blocks included in the ledger, and thus gain less profit, compared to their hash power invested. Similarly, increasing the block delay negatively impacts the fairness of the mining process. For a better mining fairness, the stale rate should be reduced by having the block interval relatively larger than the block delay.

Run time performance. For every combination of configurations, we show the average time (in seconds) it takes the simulator to perform a single run. To obtain the run time results, we use a laptop with a 2.30GHz Intel i5 CPU with 16GB RAM running on Windows 10. From Table 4.5, we observe the following. First, the run time generally takes seconds to simulate 10000 blocks. We note that in this experiment there are five miners and increasing the number of miners would increase the run time since more actions need to be performed in the network. For example, every new miner has to maintain a ledger and update it every time a new block is announced in the network. At the same time, increasing the number of non-miners would not affect the run time that much as they are not participating in maintaining the ledger. Secondly, the run time increases for higher stale rates (setting with small $B_{interval}$ or large B_{delay}). This is because miners need to update their ledgers more frequently than when conflicts are rare. Surprisingly, when the stale rate is high (over 50%) the run time seems to be decreasing. We believe the explanation for this is that although more blocks are in the system, miners neglect most blocks as they arrive when the miner is behind the main chain.

Bitcoin throughput. The current implementation of Bitcoin compromises of 596 seconds block interval and 0.42 second block delay, as reported in Table 4.2. That means the Bitcoin network experiences a low stale rate as well as a good mining fairness. However, it suffers from poor throughput as the number of transactions processed per second is only about 3. We argue that we could securely reduce the block interval of Bitcoin to 60 seconds to improve the throughput by about a factor 10, without any significant impact on the stale rate or mining fairness.

Ethereum mining fairness through uncle inclusion. The current implementation of Ethereum compromises of 12.42 seconds block interval and 2.3 seconds block delay, as reported in Table 4.2. This results in a stale rate of about 12.56% and imperfect mining fairness, but a better throughput than the Bitcoin blockchain. To eliminate the negative impact on the stale rate and mining fairness, Ethereum uses an uncle inclusion mechanism, where stale blocks are included in the main ledger as uncle blocks and the miners of such blocks are rewarded. However, this does not guarantee that miners will receive fair rewards compared to their hash power invested (e.g., a miner with a hash power of 20% should receive 20% of the total rewards distributed in the network). This is especially true as miners get a lower reward for uncle

blocks compared to main blocks as well as they are not rewarded for the transactions included in the uncle blocks.

Miners	(%)	Fraction of rewards without uncle inclusion (%)	Fraction of rewards with uncle inclusion (%)
M1	40	41.32	40.2
M2	30	30.28	30.18
M3	15	14.47	14.91
M4	10	9.34	9.85
M5	5	4.6	4.86

Table 4.6: The fraction of rewards gained by each miner (M1, M2, M3, M4, M5), with and without uncle inclusion mechanism.

We use the same parameters as currently in Ethereum to further explore whether the fraction of rewards a miner would receive with uncle inclusion mechanism is proportional to its hash power. We execute 10 independent simulation runs of 10,000 blocks and report the average results in Table 4.6. From Table 4.6, we see that the fraction of rewards gained by the miners with uncle inclusion mechanism is closer to their hash power than in the case where the uncle mechanism is not applied. Thus, Ethereum indeed achieves a better mining fairness using its uncle inclusion mechanism.

4.7 Discussion: Evaluation of BlockSim against Design Objectives

We evaluate our simulator against the design criteria mentioned in Section 4.2.1, which are generality, extensibility and simplicity.

Generality. Generality refers to the ability to use BlockSim for a variety of analysis questions and for a variety of blockchains. The key technology to achieve generality is the BlockSim Base Model, which has been designed in such a way that many blockchain systems and analysis questions can be answered. The Base Model covers all common building blocks of blockchains such as nodes, transactions, blocks, blockchain ledger, fork resolution and incentive models. We have demonstrated the application of blockchain to analyse Bitcoin and Ethereum, and arguably BlockSim is well-suited for the full class of permissionless blockchain systems. Furthermore, BlockSim achieves generality by supporting different properties and metrics such as performance (both throughput and latency), functionality metrics such as stale rates

and system properties such as mining fairness and mining incentives. To further support this criterion, however, we aim to model and implement different consensus protocols (e.g., Proof-of-Stake) as well as different generic broadcast protocols for the Network layer in a later version of BlockSim.

Extensibility. Extensibility refers to the ability of the BlockSim tool to be extended in a natural manner for various systems and analysis problems. This comes down to the design of the software, which is through modules that can easily be manipulated and extended to investigate different properties or problems of interest. The user of the simulator can use common object oriented programming techniques such as inheritance to extend current modules either by adding new functionalities (classes, methods or attributes) or modifying (overriding) some of the existing ones.

In Sections 4.4.1 and 4.4.2, we show how we extend the base modules of BlockSim to support the implementation of Bitcoin and Ethereum. For instance, we extend the Node module by adding an attribute for a node's hash power.

In addition, we illustrate how to extend the Ethereum model of BlockSim to analyse the implications of the Ethereum Verifier's Dilemma [4] (Chapter 6) and to study the uncertainty problem miners face in Ethereum when selecting transactions (Chapter 7).

As another example, we will briefly explain how to extend BlockSim to support different malicious behaviours of the nodes (e.g., selfish mining strategies). The current implementation of BlockSim assumes that all nodes are honest. To support such behaviours, we can extend the Node module by introducing a new attribute (e.g., selfish) for each behaviour. Note that each behaviour needs to be adequately defined (e.g., by writing a function or a separate class that specifies the procedures involved in this behaviour). To establish selfish mining behaviour for a node, for instance, we configure that node to work on its fork without propagating the blocks it generates to other nodes in the network. Once the behaviours are defined, the user of the simulator has only to access the configuration module and choose which type of behaviours to be studied when defining the nodes, without modifying the underlying code of the simulator.

Simplicity. BlockSim achieves this criterion as it has been implemented in different modules as well as it provides a user interface (a configuration module) that allows the end-user to set up the input parameters for the simulator. This makes BlockSim easy to use and understand. Besides, the current version of BlockSim hides and abstracts

many details. For example, it abstracts all the details of the network layer by only introducing a configurable time delay for information propagation to model this layer. Also, it hides details about the validation process of blocks and transactions. By doing so, BlockSim becomes simple and easy to use and understand. Although hiding and abstracting details can result in an incomplete model, it is possible to extend BlockSim to incorporate these details if required.

4.8 Conclusion

This chapter proposes BlockSim, a discrete-event simulation framework for blockchain systems, capturing network, consensus and incentives layers of blockchain systems. The simulation tool is implemented in Python and is available for general use. We introduce the design and evaluate it against the design objectives of generality, extensibility and simplicity.

BlockSim's results have been validated by comparing it with design properties and measurement studies available from real-life blockchains such as Bitcoin and Ethereum. We also demonstrated the use of BlockSim in a study of stale rate, throughput and mining fairness across a variety of blockchain configurations.

Future work should further demonstrate the extensibility of BlockSim by implementing additional variants of blockchain systems, such as those based on Proof of Stake as well as blockchains augmented with channels. In addition, one can build on the current version of BlockSim and extend it with additional reusable classes that represent other important system aspects and mechanisms, in particular mining pools and channels.

Chapter 5

Data Collection and Distributions of Ethereum Smart Contracts

Summary

To simulate blockchain systems as close to reality as possible, we need accurate estimates of the probability distribution of various variables. In this chapter we obtain distributions for Ethereum smart contract transactions, with respect to Gas Limit, Used Gas, Gas Price and CPU Time. To determine these distributions we use publicly available Ethereum smart contract information, augmented with experimental data for over 300,000 smart contracts obtained on a test bed. We conclude that Gaussian Mixture Models are appropriate for distributions of smart contracts with respect to Used Gas and Gas Price, and use a uniform distribution for the distribution with respect to the Gas Limit. A correlation analysis shows that the CPU Time is strongly correlated with Used Gas and we therefore apply regression techniques to estimate the CPU Time conditioned on Used Gas. We experiment with three ensemble regression methods, namely Random Forest, Gradient Boosting Machine and Adaptive Boosting and conclude that Random Forest is both fast and accurate.

The distributions obtained in this chapter will be used as inputs to the BlockSim simulator in order to conduct the simulation studies in Chapters 6 and 7 of this thesis in a realistic setting. That is, we can draw results that are more representative of the simulated Ethereum blockchain.

5.1 Introduction

There is a wealth of data available for the analysis of Ethereum smart contracts and in its own right it is interesting to understand the distribution of smart contracts

with respect to variables such as Used Gas, Gas Limit and Gas Price (see Section 2.4 for an explanation of these Ethereum specific concepts). Such distributions are also useful, and even necessary, as input to discrete-event simulations of blockchain systems, for instance using simulators such as our BlockSim simulator [8, 9]. Discrete-event simulation is useful to configure, analyse and optimise systems, and realistic distributions make the results more representative of real blockchain systems.

In this chapter we conduct an extensive analysis of Ethereum data in order to estimate distributions of smart contracts with respect to the following parameters: Gas Limit, Used Gas, Gas Price and CPU Time. The basis of our analysis is a set of over 300,000 smart contract transactions available from the live Ethereum system through the Etherscan web site. This provides us with data needed to fit smart contract distributions with respect to Gas Limit and Gas Price. To determine the Used Gas for a smart contract, we execute the smart contract on a local EVM (Ethereum Virtual Machine) to tally up the Used Gas needed for each smart contract. This also provides the CPU Time required for the execution of the smart contract. The chapter describes the data collection and measurement effort in detail in Section 5.2.

We use Gaussian Mixture Models (GMM) to fit distributions with respect to Gas Price and Used Gas since the logarithmic representation of the data resembles a normal distribution (see Section 5.4.1). We estimate the distribution of smart contracts with respect to Gas Limit using a Uniform distribution. A correlation analysis, reported in Section 5.3, demonstrates that CPU Time needed to execute a smart contract is strongly correlated with the Used Gas, as one would expect. Therefore, instead of independently fitting a distribution of smart contracts with respect to CPU Time, we use regression methods to predict the distribution for CPU Time given Used Gas. We compare a number of regression ensemble methods, namely Random Forest, Gradient Boosting Machine and Adaptive Boosting. We discuss the performance of these regression methods in detail in Section 5.5.1 and conclude that Random Forest is both fast and accurate.

The structure of this chapter is as follow. Section 5.2 describes our approach to data collection in addition to our experimental set-up to obtain smart contract attributes (Gas Limit, Used Gas, Gas Price and CPU Time). In Section 5.3, we analyse the correlation between the attributes of smart contracts. Section 5.4 introduces the overall approach to obtaining distributions of smart contracts' attributes. Section 5.5 evaluates the accuracy and performance of the approach proposed in the previous section. Finally, we conclude the chapter in Section 5.6.

5.2 Data Collection

In this section, we describe our approach to data collection. Section 5.2.1 explains the collection of Ethereum data, and Section 5.2.2 explains our experimental set-up required to obtain smart contract information about Used Gas and CPU Time.

5.2.1 Ethereum Transactions Data Collection

Smart contracts are both created and executed through a transaction. In this section, we propose a data collection approach to collect the details (e.g., Gas Limit, Gas Price, and input data) of contract transactions (both contract-creation and contract-execution). For contract-execution transactions, our approach also collects the details of the transaction that created the contract. We make use of the APIs provided by Etherscan[1] to retrieve the details of transactions, and our tool is implemented as a Python script[2].

In our approach, we retrieve the details of transactions by going through blocks. We distinguish between the different types of transactions from their details by checking the recipient address of the transaction. A transaction that is sent to an empty account (no recipient address) indicates a contract-creation one. To distinguish between contract-execution and transfer transactions, we inspect if the recipient address has associated code or not. Unlike contract-execution transactions, transfer transactions are sent to externally owned accounts who have no associated code.

For contract-execution transactions, we also need to retrieve the details of the transaction that created the contract in the first place. To do so, we obtain the details of the first transaction submitted to the recipient address (the contract account). The first transaction submitted is usually the one that created the contract. After retrieving the first transaction, we inspect its details. We decide to neglect any contract-execution transaction that we cannot retrieve or for which we cannot confirm the transaction that created the contract. In other words, for our work to be practicable we only consider contract-execution transactions for which we can retrieve the creation transaction automatically.

In summary, our script determines the type of transactions and distinguishes between contract-creation and contract-execution, and thus, we can use different ways to measure each of them. The focus of this chapter is on contract transactions, so we do not include financial transactions into the final set of transactions prepared for

[1] https://etherscan.io/

[2] https://github.com/maher243/SmartContractsDataCollection

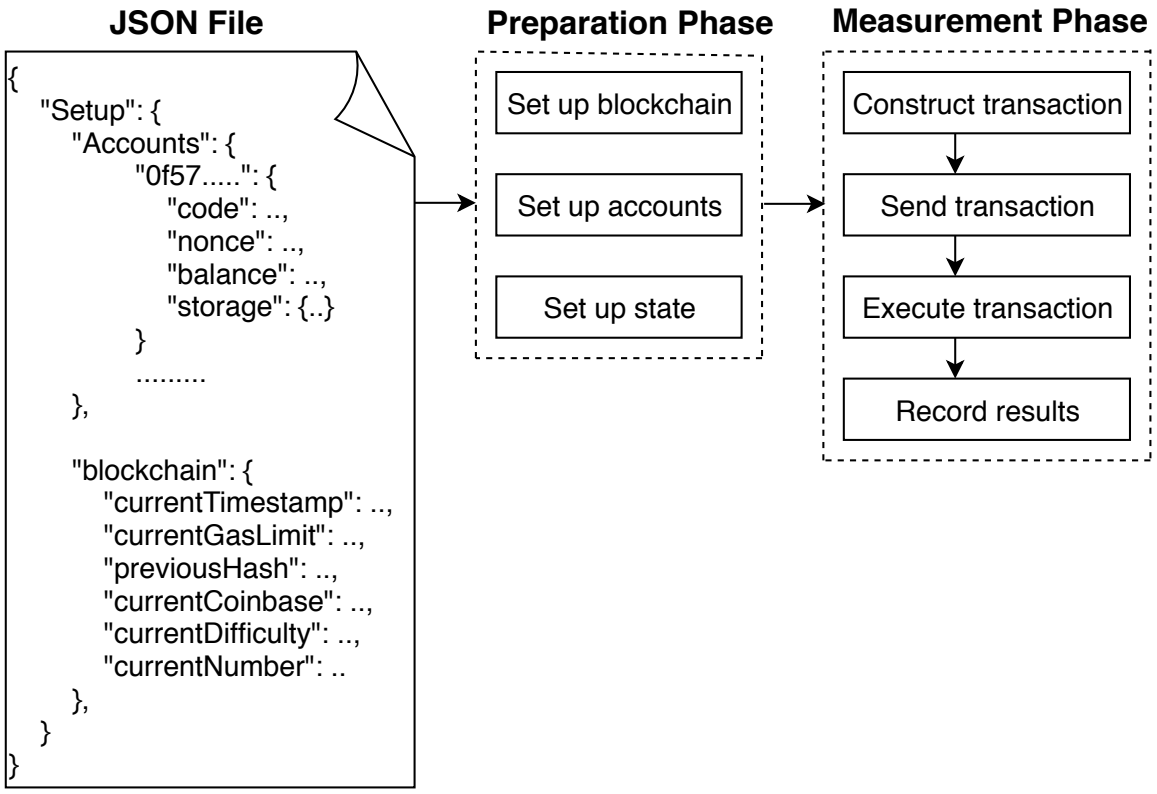

Figure 5.1: Design of the measurement system.

the measurement study. We manage to download the details of about 700 thousand smart contract transactions (for 300 thousand of these we also derived Used Gas and CPU Time, as described in the following section). We select transactions randomly from the set to avoid sample bias.

5.2.2 CPU Time Measurement

To determine the Used Gas and CPU Time for contract transactions, we propose a measurement system that tallies Used Gas and is capable of measuring the CPU usage for smart contracts transactions. Our system isolates the execution of transactions from other computation and overhead (e.g., transaction validation and the Proof of Work overhead)[3].

Our measurement system consists of two phases, namely the Preparation Phase and Measurement Phase, as depicted in Figure 5.1.

In the **Preparation Phase**, we set up the blockchain, necessary accounts and blockchain global state. For the blockchain we set the block difficulty, the block gas

[3]https://github.com/maher243/SmartContractsMeasurment

limit and the block coinbase and create the genesis block as part of configuring the blockchain. We initialise a set of Ethereum accounts, where each account has a unique address and a balance, code and storage. To establish Ethereum's global state we map the account addresses to their associated states.

In the **Measurement Phase**, we construct, send, execute transactions and record results. We construct a transaction by setting its details or fields using the details for transactions we collected from Ethereum. Then, we use the accounts we initialised in the preparation phase to submit and execute the constructed transaction. The execution of a contract-based transaction requires three tasks. First, checking the validity of the transaction (e.g., the signature is valid, the submitter has enough balance and etc). Second, running the input data of the transaction (bytecode) on the EVM. Finally, updating the state upon successful execution. We place a timer before and after the execution of the transaction on the EVM. To measure a contract-execution transaction, we first need to submit and execute a contract-creation transaction. Once the transaction has been successfully executed, we record its Used Gas and the CPU Time it takes to run on the EVM. In our implementation, we use a JSON file to store and read the accounts information as well as the blockchain information. Results are stored in a separate CSV/Excel file.

The experiments we report on in this chapter were obtained from a single machine using the Python PyEthApp[42] client. The machine is a desktop PC with a 3.40GHz Intel i7 CPU with 8GB RAM running on Windows 10. Each transaction is executed 200 times and the average time is then calculated. The 95% confidence interval is always within 2% of the average value.

Among the 700k transactions, we managed to measure 324k transactions (3915 contract-creation and 320109 contract-execution transactions). The reasons we could not measure many transactions are as follows. Some contracts call or rely on other contracts to execute the transaction. In such cases, we cannot measure the transaction unless we first measure the callee contracts. In addition, some transactions depend on other transactions. For instance, to measure a transaction that cancels an order, we need first to initiate a transaction to place the order that we wish to cancel. Finally, some contracts require the sender of the transactions to be the owner of the contract. In that case, the only way to measure these transactions is to modify the source code of the contract.

Before we analyse correlation and obtain distributions, we show some results from our experiments. Figure 5.2 shows the amount of Used Gas versus the CPU Time in seconds for contract-execution (left) and contract-creation (right) transactions.

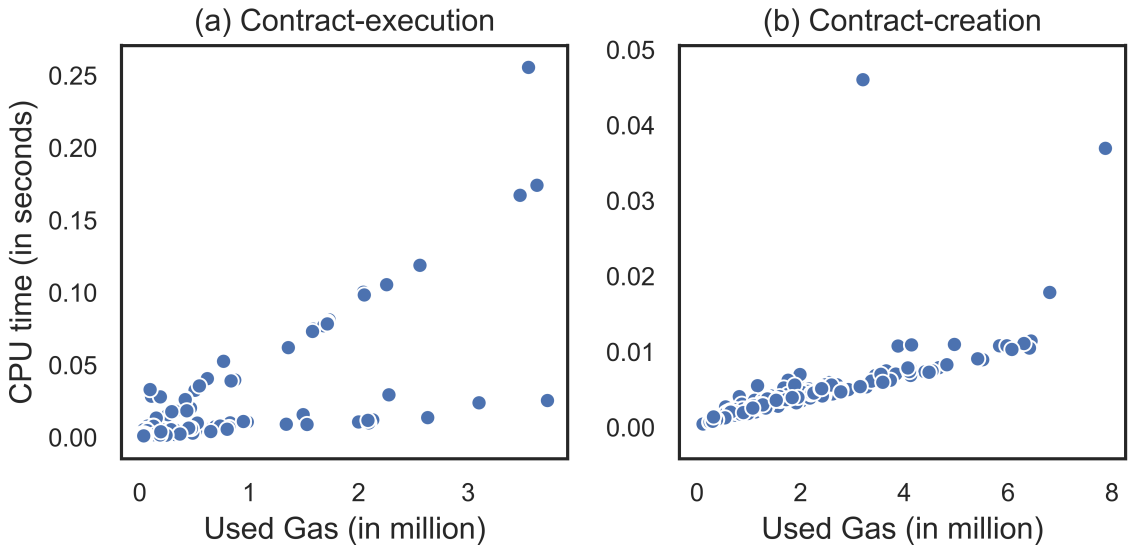

Figure 5.2: CPU Time (in seconds) versus Used Gas (in million) for (a) Contract-execution and (b) Contract-creation.

From Figure 5.2, we clearly see a relationship between Used Gas and CPU Time as the more Used Gas the transaction uses the more CPU Time is required. However, this relationship is not strongly linear as there are some outliers in both sets and the fact that we have two distinct groups for contract-execution transactions. Also, we note that contract-creation transactions are about nine times more profitable than contract-execution transactions in terms of the amount of Used Gas collected per CPU usage, as depicted in Figure 5.3. That is, the profit gained from executing contract transactions varies depending on which transactions have been selected.

In this thesis we only focus on collecting Ethereum smart contracts and obtaining their CPU time in order to fit the appropriate distributions. Thus, we have not analysed why the Used Gas for contract transactions is not linearly proportional to the CPU time required. The main factor that may contribute to this misalignment is because the gas cost for individual opcodes is not properly set, as reported in various studies such as [1, 2, 29, 83, 101]. We intend to analyse the reasons behind this misalignment in future work.

5.3 Correlation Analysis

If parameters are strongly correlated, it would not be appropriate to feed a simulation with independent distributions. Therefore, in this section, we study the correlation between the four different attributes Gas Limit, Used Gas, Gas Price and CPU Time.

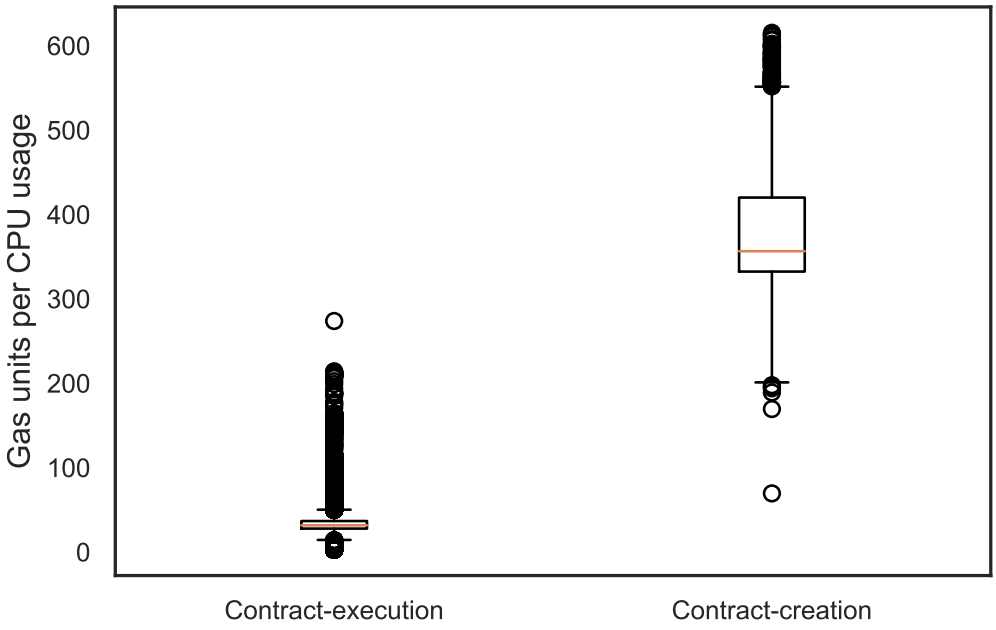

Figure 5.3: Box plot for the amount of gas units per CPU usage for both contract-creation and contract-execution transactions.

We conduct the correlation analysis using both Pearson and Spearman methods. The Pearson method assesses the linear relationship between the variables. In a linear relationship, the variables tend to change together at a constant rate. The Spearman method assesses the monotonic relationship between the variables. The variables in a monotonic relationship tend to change together, but not necessarily at the same rate. We refer to the Pearson correlation value as r_{p} and the Spearman correlation value as r_{s}.

In both Pearson and Spearman, correlation is expressed as a value $r, -1 \leq r \leq 1$, with values of r closer to 1 indicating stronger positive correlation, closer to -1 stronger negative correlation and close to 0 no or almost none correlation between the attributes. Table 5.1 shows the correlation value using the Pearson and the Spearman methods for each pair of attributes, for both creation and execution sets.

For the creation set, we see a strong correlation (r_{p}= 0.89 and r_{s}= 0.99) between Used Gas and the CPU Time. The correlation between the two attributes is stronger if we assume the existence of a non-linear relationship since the Spearman test shows a yet higher correlation value.

The Gas Limit has a medium correlation with the Used Gas (r_{p}= 0.60 and r_{s}= 0.64). A likely cause of this correlation is that the Gas Limit value for a transaction in Ethereum is always greater than or equal to the Used Gas. The Gas Limit can

Attributes	Creation Set r_p	r_s	Execution Set r_p	r_s
Gas Limit and Used Gas	0.60	0.64	0.22	-0.18
Gas Limit and CPU Time	0.49	0.62	0.17	-0.05
Used Gas and CPU Time	0.89	0.99	0.66	0.83
Gas Price and Gas Limit	0.05	0.06	-0.04	-0.10
Gas Price and Used Gas	-0.05	0.01	-0.04	0.03
Gas Price and CPU Time	-0.07	0.00	-0.02	0.01

Table 5.1: The Pearson (r_p) and the Spearman (r_s) correlation between the attributes, for both the creation and the execution sets.

take any value between the Used Gas and the block limit. The Gas Limit shows a weak to medium correlation with the CPU Time (r_p= 0.49 and r_s= 0.62). The Gas Price shows no correlation with any other attribute.

For the execution set, we see a strong correlation (r_p= 0.66 and r_s= 0.83) between the Used gas and the CPU Time. The correlation is stronger if we assume a non-linear relationship between the two attributes, as the Spearman test shows a better correlation value. The Gas Limit has a weak correlation with both the Used Gas (r_p= 0.22 and r_s= -0.18) and the CPU Time (r_p= 0.17 and r_s= -0.05). The Gas Price shows no correlation with any other attribute.

Based on the above correlation analysis, we summarise the correlation between the attributes as follows.

1. The CPU Time attribute has a strong positive non-linear correlation with Used Gas.

2. Gas Limit has a weak to a medium positive correlation with Used Gas.

3. Gas Limit has a weak to a medium positive correlation with the CPU Time. This correlation is slightly stronger for the creation set compared to the execution set.

4. Gas Price is independent of all other attributes, and, indeed, it does not have any relationship with different attributes.

As a consequence, when distributions are used as input to a simulation, particularly CPU Time and Used Gas cannot be considered to be independent. Therefore, we use regression to predict a value for the CPU Time given a Used Gas value.

5.4 Approach to Obtaining Distributions

In this section we introduce our overall approach to obtaining distributions of smart contracts with respect to the parameters Gas Limit, Gas Price, Used Gas and CPU Time:

1. Fit a probabilistic distribution to the Used Gas and the Gas Price values. We use Gaussian Mixture Models, as we explain in Section 5.4.1.

2. Apply non-linear regression models to predict the CPU Time value from the given Used Gas value, see Section 5.4.2.

3. Fit a Uniform distribution to the Gas Limit values, where the minimum value is the Used Gas and the maximum value is the block limit. See Section 5.4.3.

In Section 5.4.4 we provide the overall approach as an algorithmic procedure as well as implementation details pertaining to the parameterisation of the fitting and regression models.

5.4.1 Gaussian Mixture Models for Gas Price and Used Gas

When considering the log of the Used Gas and Gas Price data, its shape resembles a normal distribution or a mixture of normal distributions, as depicted in Figure 5.4. We therefore decided to select Gaussian Mixture Models (GMMs) to fit the log of the data since none of the simple structured distributions fit the data particularly well.

Mixture models are flexible enough to model complex probability distribution functions, where simple probabilistic distributions fail to represent the characteristics of the data accurately [107]. Mixture models can fit and represent any arbitrary and heterogeneous data set with a reasonable accuracy [54, 68, 79]. The most widely used and well-studied class of such mixture models is Gaussian Mixture Models (GMMs) [107]. In GMMs, instead of fitting one Gaussian distribution to the data, several Gaussian distributions are used to represent and capture the data set accurately.

A finite mixture model $p(x|\Theta)$ is defined as the weighted sum of K component densities,

$$p(x|\Theta) = \sum_{i=1}^{K} \phi_i \, p(x|\theta_i), \tag{5.1}$$

where, $\mathrm{x}=[\mathrm{x_i},..., \mathrm{x_n}]^{\mathrm{D}}$ is a sample of n observations of D-dimensional space, $p(x|\theta_i)$ is the i-th component, θ_i is the parameter of the i-th component, ϕ_i is the weight of the i-th component. The weight of a component must be non-negative ($\phi_i \geq 0$) and the

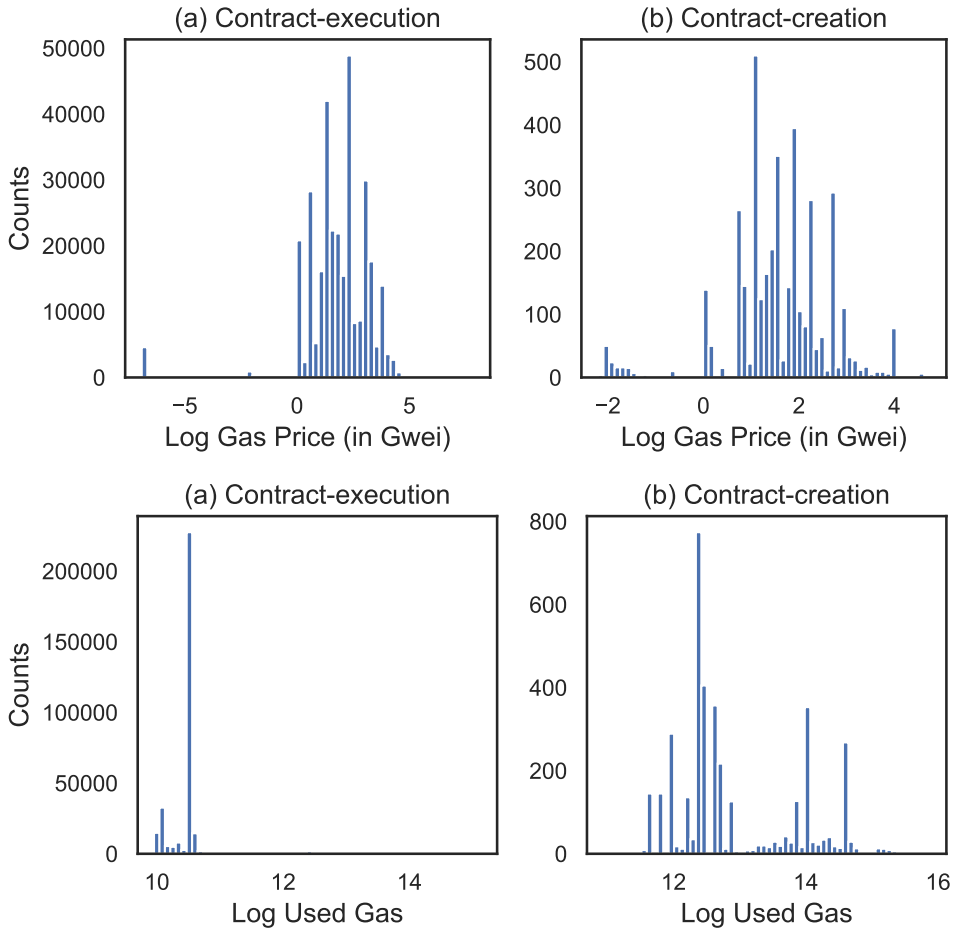

Figure 5.4: Histograms for Log Gas Price (top) and Log Used Gas (bottom) for both: (a) contract-execution and (b) contract-creation.

sum of the weights of all components in a mixture model must be 1 ($\sum_{i=1}^{K} \phi_i = 1$). For GMMs, the i-th component $p(x|\theta_i)$ is represented as a normal distribution, which has the parameters $\theta_i = \{\mu_i, \sigma_i^2\}$. μ_i and σ_i^2 denote the mean and the variance of the component, respectively. It is worth noting that in the case of a multi-dimensional space, the parameters of the i-th component are $\theta_i = \{\mu_i, \Sigma_i\}$, where μ_i is the mean vector and Σ_i is the covariance matrix.

For any application of mixture models, two issues need to be resolved, namely, the number of component densities (K) and the parameters of each component (θ_i) [107]. Selecting a large number of components may lead to a better fit to the data. However, this can result in a very complex model, and there is a risk of being over-fitted [54]. On the contrary, a model with a small number of components is less complex and more robust against over-fitting. However, it may not fit the data accurately and adequately [54].

In the literature, there are different methods proposed to select the right number of components. We apply some of the most common and widely adopted methods Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC), in particular, BIC [89]. Both BIC and AIC introduce a penalty term in an attempt to resolve the over-fitting issue.

There also exist several algorithms to estimate the parameters for mixture models. We use one of the most common algorithms which is the Expectation-Maximisation (EM) algorithm [37]. The EM algorithm is an iterative method that finds and estimates the parameters of a mixture model using maximum likelihood estimation techniques.

In Section 5.4.4 we explain specific implementation choices for our application of GMM to obtain smart contract distributions with respect to Gas Price and Used Gas.

5.4.2 Regression Models for CPU Time

Because of the strong correlation between Used Gas and CPU Time, it might be possible to derive some mathematical equations to directly estimate the CPU Time value for a transaction from the given Used Gas value. As mentioned in Section 5.2.2, however, the relationship between Used Gas and CPU Time is not strongly linear and there are some outliers that exist in both sets. That is, it is not straightforward to come up with some equations to derive the value for the CPU Time given the Used Gas value. Instead, we use regression to obtain a value for the CPU Time, given a Used Gas value. Regression predicts a continuous output variable from given input variables by analysing the relationship between the variables. Ensemble learning methods use a combination of models, instead of a single model, to improve the accuracy of the predictions. The final prediction result is the average result provided by all the models. There are different ensemble methods such as Random Forest (RF) [23], Gradient Boosting Machine (GBM) [48] and Adaptive Boosting (AdaBoost) [47].

In all the three ensemble methods, multiple trees are constructed to determine the final prediction results. RF differs from the two other methods in that it uses a bootstrapping aggregation (bagging) technique to train all the trees at the same time (independently from each other) using a random subset drawn with replacement from the original data set. Both GBM and AdaBoost rely on a boosting technique to train one tree at a time using a random subset drawn without replacement. Each additional tree is trained to correct prediction errors made by the previous tree. This is to improve the accuracy of individual observations.

To predict CPU Time values from the given the Used Gas values, we explore the following ensemble regression models: RF, GBM and AdaBoost. We extensively evaluate these models in terms of accuracy and performance. The evaluation results are discussed in Section 5.5.1. Based on the results, we decided to select RF for predicting the CPU time values.

In Section 5.4.4 we explain specific implementation choices for our application of RF regression to obtain distributions of smart contracts with respect to CPU Time.

5.4.3 Uniform Distribution for Gas Limit

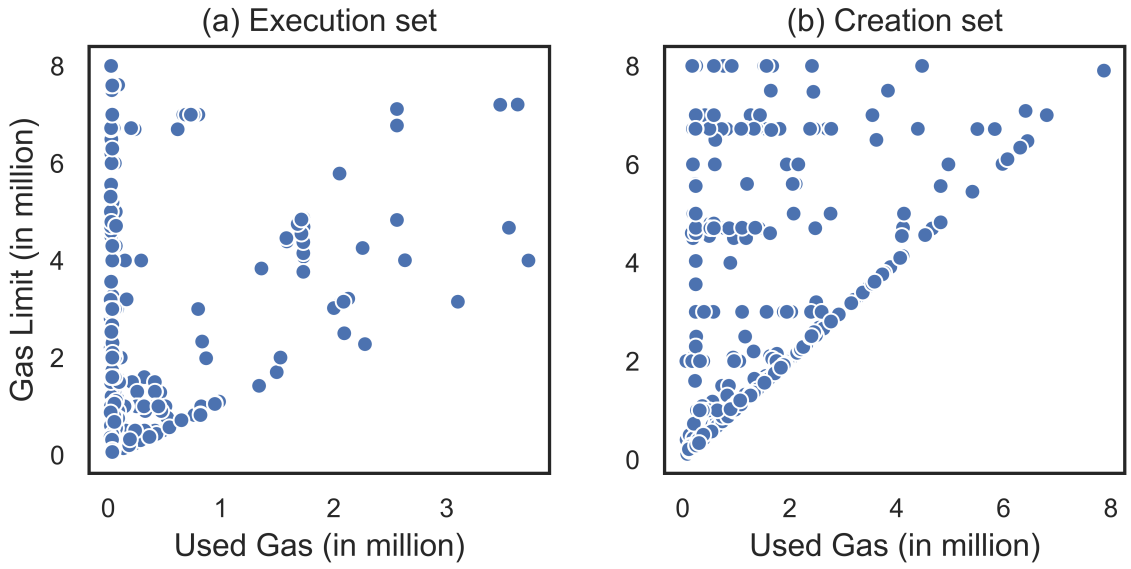

Figure 5.5: Used Gas (in million) versus Gas Limit (in million) for (a) execution set (left) and (b) creation set (right).

For the Gas Limit, it is appropriate to fit a uniform distribution, where the minimum value is the Used Gas and the maximum value is the block limit. This is because the Gas Limit is specified by the submitter of the transaction and it can take any value up to the block limit. Figure 5.5 shows the relationship between Gas Limit and Used Gas for both creation and execution sets, and it is clear that the Gas Limit value can take any value between Used Gas and block limit. Thus, the Gas Limit values will be drawn from a uniform distribution as follows:

$$Gas\ Limit \sim Unif(Used\ Gas, block\ limit) \tag{5.2}$$

The current block limit is about $8 * 10^6$ unit of gas.

5.4.4 Fitting, Regression and Sampling Procedure

Algorithm 1 The fitting and sampling procedure

1: **procedure** FIT A GMM TO LOG($Gas\ Price$)
2: Determine K ▷ Use AIC/BIC
3: Estimate $\sum_{i=1}^{K} \mu_i$, $\sum_{i=1}^{K} \sigma_i^2$, $\sum_{i=1}^{K} \phi_i$ ▷ Use EM algorithm
4: P= GMM(K,$\sum_{i=1}^{K} \mu_i$, $\sum_{i=1}^{K} \sigma_i^2$, $\sum_{i=1}^{K} \phi_i$).fit(log($Gas\ Price$))
5: **procedure** FIT A GMM TO LOG($Used\ Gas$)
6: Determine K ▷ Use AIC/BIC
7: Estimate $\sum_{i=1}^{K} \mu_i$, $\sum_{i=1}^{K} \sigma_i^2$, $\sum_{i=1}^{K} \phi_i$ ▷ Use EM algorithm
8: U= GMM(K,$\sum_{i=1}^{K} \mu_i$, $\sum_{i=1}^{K} \sigma_i^2$, $\sum_{i=1}^{K} \phi_i$).fit(log($Used\ Gas$))
9: **procedure** FIT A RF TO($Used\ Gas, CPU\ Time$)
10: Determine and optimise d, s ▷ Use Grid Search CV
11: T= RF(d, s).fit($Used\ Gas, CPU\ Time$)
12: **procedure** SAMPLE ATTRIBUTES(S_P, S_U, S_L, S_T)
13: S_P= exp($P.sample(n)$) ▷ Sample Gas Price
14: S_U = exp($U.sample(n)$) ▷ Sample Used Gas
15: $S_L = Unif(low = s_u, high = 8 * 10^6, size = n)$ ▷ Sample Gas Limit
16: $S_T = T.predict(S_U)$ ▷ Sample CPU Time

The procedure for fitting distributions to the attributes as well as sampling from such distributions are summarised as Algorithm 1. From top to bottom, it considers the application of GMM to Gas Price and Used Gas, the application of regression for the CPU Time, given values for Used Gas, and finally the sampling procedure.

To fit a GMM to the log Used Gas and to the log Gas Price (line 1-8), we have to determine and estimate parameters such as the number of Gaussian components (K) as well as the mean (μ_i), the variance (σ_i^2) and the weight (ϕ_i) of each component. To determine K, we use the AIC and BIC criteria. We tested a number of K values ranging from 1 to 100 and then selected the best K according to these criteria. To determine the parameters for each component, we use the EM algorithm. After estimating these parameters, we fit the GMM to the data.

To fit an RF model to learn and predict the CPU Time from a given Used Gas value (line 9-11), we have to determine and optimise the model's parameters, which are the number of trees (d) and the number of splits in each tree (s). To fine-tune or optimise the model parameters, we use a grid search technique with K-fold cross validation (CV), where K = 10 as suggested by [60]. We search a number of values ranging from 10 to 500 and a number of values ranging from 1 to 300 to optimise d and s, respectively. Then, we select the best-tuned values for these parameters to fit the RF model.

After fitting distributions to the attributes, we sample (draw) n data points for Gas Price and Used Gas from the fitted GMMs, where $n \geq 1$. Then, we pass the Used Gas data points to the Uniform distribution and RF model to sample the corresponding Gas Limit and the CPU Time values, respectively. The sampling procedure is captured in lines 12-16.

The algorithmic procedure is implemented in Python using a machine learning library called `Scikit-learn` by utilising different packages such as GaussianMixture, RandomForestRegressor, GridSearchCV and KFold.

We implemented a Python class named `DistFit` to fit distributions for the attributes. This class consists of two methods `creationFit` and `executionFit` to fit distributions to the attributes in the creation and the execution sets, respectively. In addition, we implemented a sampling method that takes as input the number of data points (transactions) to be simulated and returns the values of the simulated attributes as a tuple.

5.5 Evaluation of Fitting and Regression Models

In this Section, we evaluate both the accuracy and performance of the fitting and regression models.

5.5.1 Evaluation of Regression Models

As we explained in Section 5.4, we consider three different ensemble regression models (RF, GBM and AdaBoost) to predict the CPU Time value for a transaction from a given Used Gas value. To assess the accuracy of these models, we utilised several score metrics, namely the mean absolute error (MAE), root mean squared error (RMSE) and the coefficient of determination (R^2). Both MAE and RMSE give an indication about the prediction error in the same unit as the data set, and thus, the smaller the error, the better the accuracy [98]. R^2 is a goodness of fit score, and it usually takes a value between 0 and 1, where 1 is the best possible accuracy score [28].

We measure the accuracy of these models for training (seen) and testing (unseen) data to ensure the generality and the robustness of the model against over-fitting. To measure the accuracy of a model with training data, we train the model with the whole data set and then we test it against the same data set. We refer to the accuracy results of the model with seen data as the `training results`.

Of course, a critical issue about relying only on the training results is that although the model may perform well on the training data, the model might not necessarily

	Regression Model	Training Results			Testing Results		
		MAE	RMSE	R^2	MAE	RMSE	R^2
Creation Set	RF	34.29	355.12	0.96	78.47	900.20	0.82
	GBM	34.30	87.17	1	76.87	941.36	0.76
	AdaBoost	171.73	335.89	0.96	202.80	1092.5	0.65
Execution Set	RF	25.73	175.80	0.99	29.64	437.29	0.93
	GBM	88.32	180.14	0.99	90.90	459.53	0.92
	AdaBoost	315.77	1020.72	0.60	316.53	1056.58	0.52

Table 5.2: Accuracy results of the three regression models (RF, GBM and AdaBoost), for both creation and execution sets. MAE and RMSE in microseconds.

perform the same when dealing with testing or new data sets. For this reason, we also measure the accuracy of the model with testing data by applying K-fold cross validation, where K = 10 as suggested by [60]. Specifically, we split the data randomly into 10 folds with equal sizes. Then, we train the model with 9 folds and use the last fold for testing. That means each data point is used 9 times for training and 1 time for testing. We refer to the performance of the model with testing or new data as the `testing results`.

Table 5.2 shows the accuracy training/testing results for the three regression models for both creation and execution sets. To get an insight into the prediction errors reported in Table 5.2, we note that the average CPU Time for transactions is 1548 microseconds in the creation set and 1203 microseconds in the execution set. From Table 5.2, we can see that RF and GBM models outperform the AdaBoost model for both sets (recall that R^2 should be close to 1, while MAE and RMSE should be small). The RF model shows a better prediction accuracy for both training and testing data in the execution set, compared to the GBM model. The average absolute error of the predicted CPU Time value is 30 microseconds for testing data in RF, while it is 91 microseconds in GBM. The GBM model outperforms the RF model in terms of training data in the creation set, but not with the testing data. We note that the accuracy results for the testing data is more important than that of the training data since it shows how the model performs with new data sets that have not been trained on.

To conclude, the RF and GBM models predict the CPU Time value from any given used gas value with a small MAE prediction error, with testing or new data. The average error is always within 8% of the true CPU Time value. In comparison to GBM, the RF model achieves better accuracy for the execution set.

To assess the performance of these regression models, we evaluate both the training

	Regression Model	Training Time	Prediction Time
Creation Set	RF	0.015 ± 0.0002	0.003 ± 4.71e-05
	GBM	0.063 ± 0.0004	0.007 ± 3.96e-05
	AdaBoost	0.092 ± 0.0003	0.769 ± 0.001
Execution Set	RF	2.846 ± 0.005	0.319 ± 0.0002
	GBM	12.277 ± 0.272	0.522 ± 0.0006
	AdaBoost	11.047 ± 0.018	0.790 ± 0.004

Table 5.3: Performance in seconds of the three regression methods, for training and prediction, for both creation and execution sets.

and prediction time for both creation and execution sets. Training time is the time it takes to build and train each model, while prediction time is the time it takes to predict the CPU Time values from the given Used Gas values. We note that the creation set has 3915 data points and the execution set has around 320 thousand data points. Thus, the training and prediction time for the execution set is expected to be higher than that for the creation set.

Table 5.3 shows the performance results for the three regression models for both creation and execution sets. We measured the training and the prediction time for each model 100 times, and then we reported both the average results and the 95% confidence interval. The training and the prediction time for the execution set are higher than that of the creation set, for all three regression models. This is because for the execution set the model needs to take into account a far higher number of data points compared to the creation set.

From Table 5.3, we draw the following conclusions. First, the training time for the RF model is considerably less than that of other models, for both sets. For example, it takes on average about 3 seconds to train the RF model on the execution set, while it takes on average of about 11 and 12 seconds to train AdaBoost and GBM models on the same set. Similarly, the RF model outperforms other models in terms of prediction time. The training time for GBM and AdaBoost is quite similar, while the prediction time for GBM is much faster than AdaBoost. In conclusion, the RF model achieves the best performance among the three models for both training and prediction time.

Based on the accuracy and performance results discussed above, the RF model outperforms the other two models. The GBM model achieves accuracy results comparable to the RF model, but its performance is less strong. The AdaBoost model achieves poor accuracy results compared to other models, and its performance in

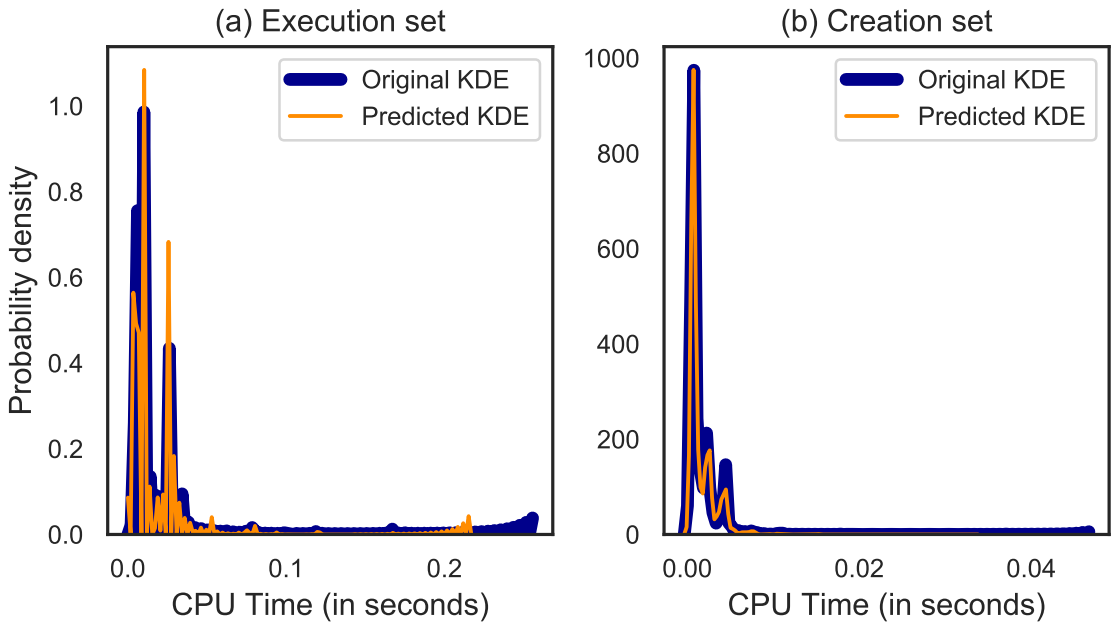

Figure 5.6: KDE for original and predicted CPU Time for both execution set (left) and creation set (right).

terms of the prediction time is also poor. Therefore, we selected the RF model to predict the CPU Time values for both sets.

To further evaluate the accuracy of the RF models, we compare the Kernel Density Estimation (KDE) of the original CPU Time data with that for the predicted data. KDE is a non-parametric smoothing approach for data sets. The predicted CPU Time data is obtained from RF model by passing the Used Gas data. Instead of using the original Used Gas data that we trained RF model on, we sample new data for the Used Gas from GMMs (see Section 5.5.2).

Figure 5.6 shows the KDE for the original and the predicted CPU Time for both the creation and the execution sets. The KDE for the predicted data looks very similar to that of the original one, and that indicates the accuracy of the RF models.

5.5.2 Evaluation of GMMs

To assess the accuracy of the fitted GMMs, we compare the KDE for the original Used Gas and Gas Price data with the data sampled from the fitted distributions. We sample (generate) n data points for both Used gas and Gas Price from the fitted GMMs, where n equals the data points in the original set. Figure 5.7 and 5.8 show the KDE for the original and the sampled Gas Price and Used Gas data for both the creation and the execution sets. The KDE of the sampled data is very similar to that

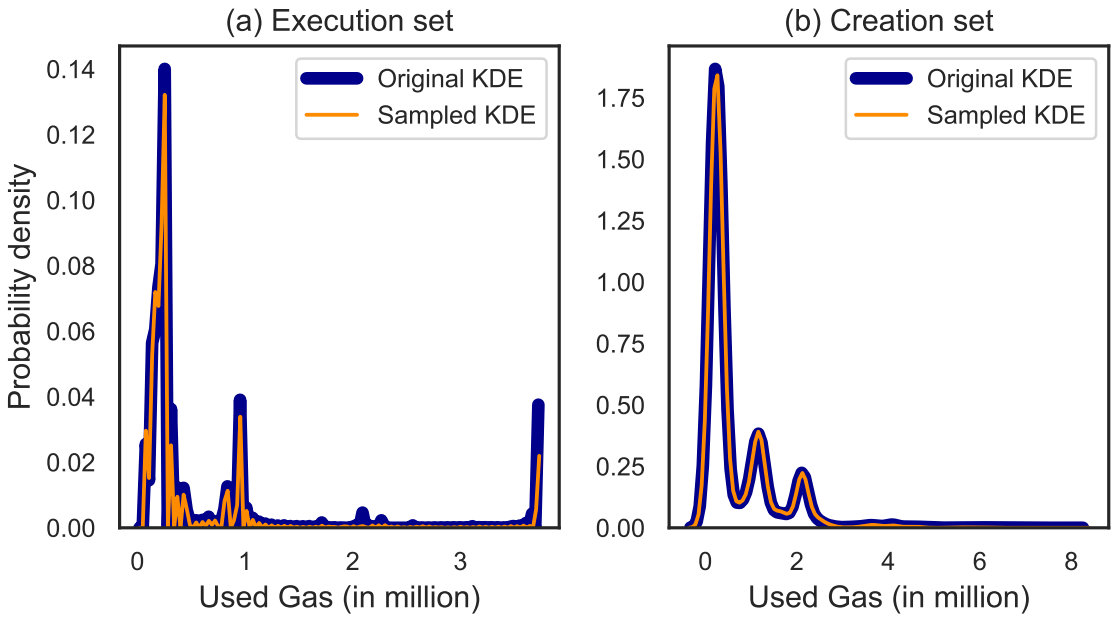

Figure 5.7: KDE for original and fitted Used Gas for both execution set (left) and creation set (right).

	Data	Fitting Time	Sampling Time
Creation Set	Used Gas	0.365 ± 0.012	0.006 ± 0.0002
	Gas Price	0.385 ± 0.015	0.005 ± 7.84e-05
Execution Set	Used Gas	25.236 ± 1.517	0.019 ± 0.0004
	Gas Price	76.821 ± 3.759	0.022 ± 0.0003

Table 5.4: Performance of GMM fitting and sampling, in seconds, for both creation and execution sets.

of the original data for both Gas Price and Used Gas. This gives an indication of the accuracy of the GMMs.

To assess the performance of GMM, we evaluate the fitting and sampling time. Table 5.4 shows the performance results for the GMMs for both creation and execution sets. We measured the fitting and the sampling time for each model 100 times, and then we reported both the average results and the 95% confidence interval.

From Table 5.4, we draw the following conclusions. First, the fitting and the sampling time for Used Gas and Gas Price in the execution set are higher than that for the creation set. As for the regression models in Section 5.5.1, this is due to the higher number of data points in the execution set. Secondly, the fitting time for GMMs is considerably higher than the sampling time for Gas Price and Used Gas, again for both sets. The fitting time depends on the number of different components

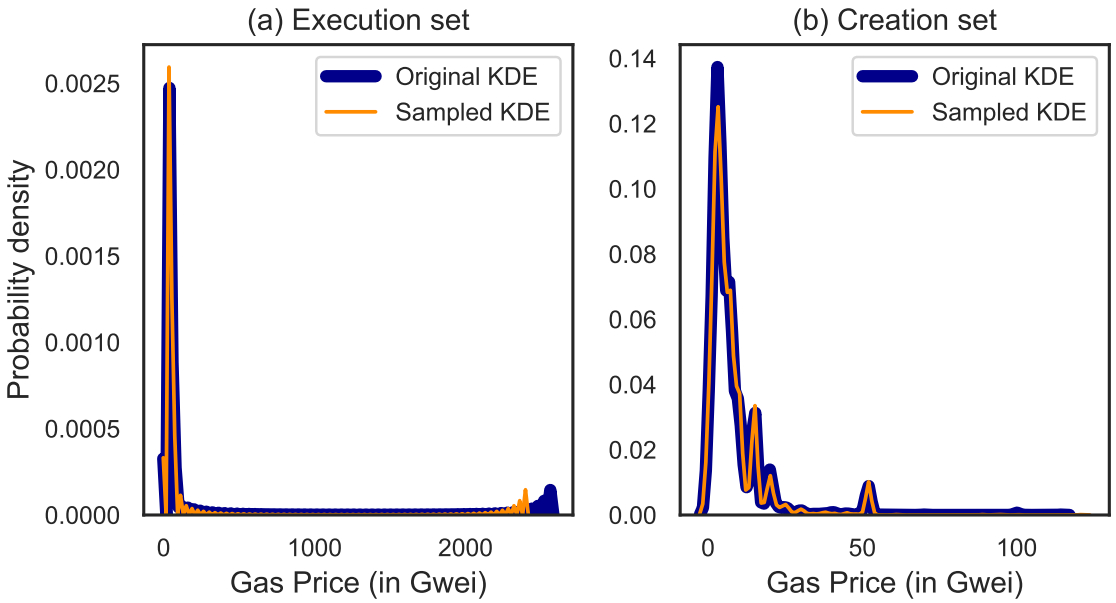

Figure 5.8: KDE for original and fitted Gas Price for both execution set (left) and creation set (right).

(distributions) used to fit the data. Increasing the number of components improves the model accuracy, but at the cost of performance. For example, we use 26 more components to fit the Gas Price compared to the Used Gas and it indeed takes about 77 seconds to fit the model for the Gas Price, while it only takes 25 seconds to fit the model to the Used Gas.

The sampling time is low compared to the fitting time, for both Used Gas and Gas Price. The sampling time is also not impacted by the number of components. For instance, both Gas Price and Used Gas in the execution set have almost the same sampling time, although the number of components is different.

5.6 Conclusion

Based on publicly available data about Ethereum smart contract transactions, we derived smart contract distributions with respect to their Used Gas, Gas Limit, Gas Price and CPU Time. For Used Gas and CPU Time we augmented the data with that obtained from executing smart contracts on our own test bed. In total, we used complete data of about 324 thousand smart contract transactions.

The resulting distributions are of interest in their own right, but predominantly served as input for discrete event simulations of blockchain architectures. We will

use these distributions as inputs to the BlockSim simulator to conduct the simulation studies in the following chapters realistically.

We used Gaussian Mixture Models for the independent distributions with respect to Gas Price and Used Gas (and a Uniform distribution for Gas Limit). Because of strong correlation between Used Gas and CPU Time, we used ensemble regression models for the CPU Time distribution.

Chapter 6

Data-Driven Model-Based Analysis of the Ethereum Verifier's Dilemma

Summary

In proof-of-work based blockchains such as Ethereum, verification of blocks is an integral part of establishing consensus across nodes. However, in Ethereum, miners do not receive a reward for verifying. This implies that miners face the Verifier's Dilemma: use resources for verification, or use them for the more lucrative mining of new blocks?.

In this chapter, we provide an extensive analysis of the Verifier's Dilemma, using a data-driven model-based approach that combines closed-form expressions and discrete-event simulation. To establish a data-driven study, we feed our simulations with distributions for real Ethereum smart contract transactions, see Chapter 5. Our analysis of the Verifier's Dilemma shows that, indeed, it is often economically rational not to verify, in particular for miners with less hashing power. We consider two approaches to mitigate the implications of the Verifier's Dilemma, namely parallelisation and active insertion of invalid blocks, both will be shown to be effective.

6.1 Introduction

Blockchains depend on miners to operate the chain correctly and to jointly guarantee consistency and correctness of the blockchain data and the executed transactions. In public, permissionless, blockchains collaboration of miners is based on incentive mechanisms that provide miners with a certain amount of cryptocurrency for their

efforts. It is clearly important to award fees in such a manner that correct and desired behaviour is encouraged. Well-balanced incentives, together with the miner's interest to keep the system running well, should guarantee the correct behaviour of the blockchain.

Within Ethereum, there is one interesting aspect of the consensus algorithm that is not incentivised directly, namely the verification of transactions and blocks. This leads to an interesting dilemma: should miners verify transactions within blocks if they do not receive a specific fee for it? If all blocks are valid, the verification would not have been necessary and the time spent on verifying could have been used to mine new blocks (which are rewarded by a fee). This Verifier's Dilemma is well recognised, e.g., [71, 94], but has not been systematically analysed. In this chapter we conduct that systematic analysis of the Verifier's Dilemma in Ethereum.

The analysis is involved, and combines a number of analysis techniques to establish the fees miners would collect under different decisions about participation in the verification. We pursue a model-based approach so that we are able to analyse a range of possible scenarios. It is not practical or even possible to obtain insights about the Verifier's Dilemma solely based on observations of the actual Ethereum system. A model-based approach, correctly intertwined with data-driven parameterisation, is the only reasonable approach.

We combine the following techniques. At the core of the analysis is the Ethereum model of BlockSim simulator that we introduced in Chapter 4. We extended it with the functionality necessary for the analysis of the Verifier's Dilemma under various scenarios. Secondly, to run realistic simulation studies, we parameterise the simulator with distributions for Ethereum smart contract transactions that we derived in Chapter 5. As a third and final element in our study, we obtain a number of closed-form results for base scenarios. In these base scenarios no invalid blocks are present, and under that assumption we are able to derive expressions for the rewards miners receive if they do or do not verify blocks.

The conclusion of the above analysis is that under certain conditions it pays off for miners not to verify. Obviously, not verifying blocks puts the correct functioning of the blockchain at risk, since the consensus approach in Ethereum assumes verification to take place. To mitigate this risk, we consider two approaches. First, we consider parallel verification (as proposed in [38]), to decrease the time it takes to verify blocks and therefore decrease the time verifying miners would have to spend before they can mine a new block. Secondly, we consider the idea of injecting invalid blocks on purpose, to penalise miners that do not verify. The reasoning behind that approach

was identified in [94]. By injecting invalid blocks, a non-verifying miner would more often pass on chains with invalid blocks that will be rejected by other miners, which in turn would imply that the non-verifying miner does not receive the block award.

To summarise, the results of our analysis are as follows. It is clear that there are many scenarios in which miners would benefit from not verifying blocks. This is especially true if (1) all or almost all blocks are in fact valid, and (2) if the block limit (the number of transactions in a block) is large. In addition, the impact of verification on the expected reward is larger for miners with less hashing power, who may therefore be tempted more to omit verification. For Ethereum, currently the impact is small but the Verifier's Dilemma will become more important when the block limit increases, as is anticipated [71]. As mitigation approaches, both parallel verification and injecting invalid blocks improve the situation. That is, both make it less lucrative for miners to avoid verifying.

This chapter is structured as follows. Section 6.2 introduces the problem space and providing an explanation of the Verifier's Dilemma. Section 6.2.2 provides the closed-form expressions for the gain in rewards both verifying and non-verifying members could get for the base model, i.e., the case that all transactions are valid. Section 6.3 introduces the two mitigation approaches, namely parallelisation and injecting invalid transactions. Section 6.4 describes how we used the BlockSim simulator and enhanced it to suit our study. The results of the simulation study are provided in Section 6.5, for various scenarios as well as for the two mitigation strategies and the insights gained and conclusions drawn are further discussed in Section 6.6. Finally, we conclude the chapter in Section 6.7.

6.2 Verifier's Dilemma in Ethereum

We first present and discuss in Section 6.2.1 the Verifier's Dilemma in general terms and then we derive in Section 6.2.2 closed-form expression for the rewards received by non-verifying miners. These closed-form expressions hold for scenarios in which all blocks are valid, which we will call the *base model*.

6.2.1 Problem Description

Luu et al. [71] pointed out that verification of blocks consumes computation resources and time, and thus, delays miners in the race of mining the next block. Not only it delays miners, but also it does not provide incentives (a free task) to miners. This

is especially true in smart contract based blockchains since verification of smart contracts involves repeating the execution of the smart contract to check the outputs. [71] also points out that these concerns exacerbate when the block size or limit increases since that increases the number of transactions to verify.

As a result, miners might consider skipping the verification process. Skipping verification allows the miners to turn to the profitable mining of new blocks. The risk of skipping verification is that the miner adds its newly mined blocks to a blockchain that contains invalid blocks. If other miners verify these blocks they will disregard these and the new block and the non-verifying miner will not receive a reward for its new block. The miner needs to decide the following: should I support the blockchain honestly and verify all blocks, possibly at the cost of personal rewards, or shall I skip verifying blocks and instead spend the time on the lucrative mining of new blocks, thus increasing personal rewards? [71] calls this the *Verifier's Dilemma*.

The Verifier's Dilemma has received some attention, we refer to the discussion on related work in Section 2.8. However, there has not been a rigorous analysis of the dilemma using probabilistic modelling techniques such as in this chapter.

6.2.2 Ethereum Base Model

In Ethereum, miners are expected to verify received blocks by executing their transactions in sequence. In this section, we use closed-form solutions to investigate the Verifier's Dilemma in Ethereum and its impact on the fee received by miners. We consider current and likely future configurations of Ethereum in terms of block limit and block interval time.

There are important model assumptions that will hold throughout the chapter. The consequent limitations of these assumptions will be discussed in Section 6.6. We assume that miners can be either solo miners or mining pools, and not miners joining a mining pool. Miners, in particular small miners, within a pool might not be impacted by the Verifier's Dilemma as if they are solo miners. This is because they do not have to do all the verification process by themselves alone. We also assume that miners always follow one static behaviour by either committing or skipping the verification process. That is, our model does not capture dynamic behaviours where miners can change their behaviours, for instance, as a response to the behaviour of other miners. We also assume that miners fill each block by executing as many transactions as they can in order to maximise their revenue. In the real system, miners can generate full, non-full, or even empty blocks, but this is not critical for the analysis of the Verifier's Dilemma. If needed this can be added to the model. All transactions in the network

are assumed to be contract-based transactions, thus ignoring the additional financial transactions that may take place. Such financial transactions take less time to verify and therefore do not impact the Verifier's Dilemma as much, but these can of course easily be added. We also ignore the time it takes to check the hash outcome of the PoW, since that check is almost immediate. Finally, we do not explicitly consider block propagation delay between nodes since this does not affect the issue of the Verifier's Dilemma.

In this section, we derive closed-form results for scenarios in which all miners are honest when executing transactions. That is, all transactions included in a block are valid. The closed-form solutions are to estimate the fraction of fee received by both verifying and non-verifying miners as a function of block limit, block interval time and fraction of hash power. The block limit dictates the number of transactions that can fit in the block. That is, the larger the block limit the more transactions can be included, and thus the more time required to verify the block and its embedded transactions. Of course, the block verification time T_{v} depends on the speed of the machine used. The block interval time T_{b} dictates how often blocks are generated in the network. The fraction of hash power α for a miner dictates the fraction of blocks and rewards a miner can get.

It is worth noting that a miner only verifies blocks that are generated by other miners, not the ones it generates itself. Thus, the average block verification time decreases with the increase of hash power α of the miner, since then there are less blocks generated by others. For instance, a miner with $\alpha = 0.30$ of total network hash power is expected to verify 70% of the total generated blocks, which means that it spends on average $(1 - \alpha)\ T_{\mathrm{v}}$ for verification per block.

Assume that α_v is the fraction of hash power of all verifying miners. Then the extra time δ verifying miners spend per block to perform the sequential verification process is:

$$\delta = (1 - \alpha_v)\ T_{\mathrm{v}} \tag{6.1}$$

The fraction of the expected rewards R_v for verifying miners is then reduced from α_v to

$$R_v = \frac{T_{\mathrm{b}}}{T_{\mathrm{b}} + \delta}\ \alpha_v \tag{6.2}$$

That is, the amount of reward that verifying miners would lose by committing the verification process is $l = \alpha_v - R_v = \delta/(T_{\mathrm{b}} + \delta)\alpha_v$, which non-verifying miners would collect as extra rewards. Assume that $\alpha_s = 1 - \alpha_v$ is the fraction of hash power of all

non-verifying miners. Then, the fraction of the expected rewards R_s for non-verifying miners is increased from α_s to

$$R_s = \alpha_s + l \tag{6.3}$$

Results from this closed-form solution will be validated and discussed in Sections 6.4.2 and 6.5, but to illustrate the implications, assume 10 miners, each controlling $\alpha = 0.1$ of the total network hash power. Among those miners, assume there is only one miner who does not verify. Assume $T_{\mathrm{v}} = 3.18$ and $T_{\mathrm{b}} = 12$ seconds. We calculate the slow down of performing the verification as $\delta = 0.318$. The fraction of fee received by the nine verifying miners is reduced from 0.9 to 0.877. Thus, the non-verifying miner would gain 0.023 more rewards (increase from 0.1 to 0.123, $\approx 23\%$ more than its invested α).

6.3 Mitigation Solutions to the Verifier's Dilemma

We discuss two mitigation solutions for the Verifier's Dilemma, namely parallel verification of transactions and intentional production of invalid blocks.

6.3.1 Mitigation 1: Parallel Verification

Parallel verification was proposed by [38] to speed up the verification process, thereby minimising the lost time to miners. By speeding up the time it takes to verify transactions, a miner would lose less time. Transactions that do not have read/write conflicts with other transactions in the same block can be verified in parallel. The remaining conflicting transactions must still be verified in sequence.

We propose parallel verification as a solution to mitigate the implications of the Verifier's Dilemma. To implement parallel verification in a real system, the Ethereum Virtual Machine needs to support multi-threading. Miners then attach an execution schedule to their proposed blocks. The schedule details which transactions can be processed in parallel (no read/write conflicts) and which must be executed in sequence. We assume miners provide a correct schedule and are well motivated to include the schedule in their blocks.

To obtain a closed-form expression for the received reward, two parameters are added to the parameters of the Ethereum base model (Section 6.2.2), namely the conflict rate and the number of processors. Note that we still assume that all blocks are valid, as in the base model. The conflict rate c is the percentage of conflicting transactions in a block. For example, c= 0.4 means that 40% of the block's transactions

are in conflict with other transactions in the same block. We note that according to [38], the number of conflicting transactions in real blockchains is not very high since there are thousands of different contracts. The number of concurrent processors p is the number of machines the miner has available in parallel. With p processors and conflict rate c, the slow down of performing the parallel verification process (δ_p) is:

$$\delta_p = (c + \frac{1-c}{p})\ \delta \tag{6.4}$$

The fraction of rewards for verifying and non-verifying miners is based on the same equations as in Section 6.2.2.

Apply the parallel verification to the previous example, with $c = 0.4$ and $p = 4$. Then, the slow down of performing the parallel verification is $\delta_p = 0.1749$. The fraction of fee received by the nine verifying miners is reduced from 0.9 to 0.887. Thus, the non-verifying miner would gain 0.013 more rewards ($\approx 13\%$ more than its invested α). In other words, the fraction of rewards obtained by the non-verifying miner increases from 0.1 to 0.113.

6.3.2 Mitigation 2: Intentional Invalid Blocks

In this section, we introduce a solution whereby Ethereum could allocate a special node for intentionally generating invalid blocks as a way to punish non-verifying miners. This special node is assigned a particular hash power (e.g., $\alpha = 0.04$) of the total network hash power. The hash power of the special node simply represents the fraction of the invalid blocks to be purposely generated in the network. We assume this node to verify all blocks generated by other miners, and thus, it always works on the valid branch of the blockchain. The rationale behind this approach is that miners benefit from not verifying because all (or almost all) blocks are valid anyway. However, if incoming blocks could be invalid, the non-verifying miner could end up working on new blocks on top of the invalid ones. Consequently, the non-verifying miner would lose the rewards for those new blocks since other verifying miners will reject the blocks because they were built on top of invalid ones. Since this scenario includes non-valid blocks, we have no closed-form insights for this scenario. However, we will extensively study the result of injecting invalid blocks in the simulation results in Section 6.5.

6.4 Simulator and Validation of Closed-Form Expressions

At the core of our model-based approach to analysing the Verifier's Dilemma is the publicly available BlockSim simulator, which we extended to be able to study the Verifier's Dilemma. In this Section we report in Section 6.4.1 on how we extended the Ethereum model of BlockSim and in Section 6.4.2 on the use of the simulator to validate the closed-form solutions derived in Section 6.2 and 6.3.

6.4.1 BlockSim Simulator Extension

To support the analysis of both the Ethereum base model and the solutions of parallel verification and intentional production of invalid blocks, we introduced the following modifications:

The attributes of transactions: We extended the Transaction module to include several attributes required by the model, which are Gas Limit, Used Gas, Gas Price, and CPU Time. Thus, each transaction created in our simulations has these attributes.

The distribution fitting class (`DistFit`): We defined a new class named DistFit to fit probability distributions to the transaction attributes. This follows the procedure introduced in Chapter 5. We execute the distribution fitting once. During the simulation, when creating new transactions, we sample random values for these attributes from the fitted distributions.

The number of processors (p): This dictates the number of processors a miner could use to verify transactions in parallel. To add this feature to the simulator, we extended the Node module by adding a new attribute named *processors*.

The rate of conflicting transactions (c): This dictates the fraction of transactions that depend on other transactions in the system. To add this feature to the simulator, we introduced a new input parameter called *conflict_rate*. We also extended the Transaction module by adding a new attribute named *dependency* for transactions, to distinguish between conflicting and non-conflicting transactions. Each transaction created will be assigned to a random value (True or False) for the *dependency* attribute based on the *conflict_rate* parameter.

Parallel verification of transactions: To add this feature to the simulator, we modified the execution of the block receiving event as follows. Upon receiving a new block, we distribute non-conflicting transactions between the different processors, after which the conflicting ones will be executed sequentially on a single processor.

Hence, we count the time required to verify transactions in parallel by checking the CPU time attribute for transactions. Prior to starting the verification, the time for all processors is set to 0 (all processors are idle). During the verification process, we keep recording the time when each processor finishes the transaction at hand and pass a new transaction to start afterward.

The intentional production of invalid blocks: To add this feature to the simulator, we first extended the Block module by adding a new attribute named *validity* for blocks, to distinguish between valid and invalid blocks. Each block created will be assigned to a value (True or False) for the *validity* attribute. Then, we set one of the miners to be the network node that always generates invalid blocks. The hash power of this node can be changed to reflect the fraction of invalid blocks to be generated in the network.

6.4.2 Validation of Closed-Form Expressions

To validate the closed-form solutions from Sections 6.2.2 and 6.3.1, we need first to estimate the average time it takes to verify a block and its associated transactions. The verification time depends on which transactions are included in the block. In particular, different transactions take different time as well as blocks may have a very different number of transactions depending on the gas used by these transactions. Hence, we utilised the simulator to simulate different configurations of block limits (the limit is expressed in million (M) units of gas). For each configuration, we simulated 10000 blocks and the statistical results related to the block verification time are given in Table 6.1. The table gives the minimum (min), the maximum (max), the mean, the median, and the standard deviation (SD) for the block verification time, all in seconds.

	Block verification time (T_{v})				
Block limit	min	max	mean	median	SD
8M	0.03	0.35	0.23	0.24	0.04
16M	0.16	0.65	0.46	0.47	0.06
32M	0.51	1.09	0.87	0.87	0.06
64M	1.06	2.08	1.56	1.56	0.19
128M	2.5	3.75	3.18	3.19	0.19

Table 6.1: The statistical results for the block verification time (T_{v}) in seconds for different block limits.

To validate the closed-form expressions (Equations (6.1) to (6.4)) for both the Ethereum base model and the parallel verification solution, we compare the simulation

results with that of the equations. We configured the simulator as follows. We set the block interval time to be 12.42 seconds, which is the minimum observed interval between blocks according to Etherscan[1]. The attributes (Gas Limit, Used Gas, Gas Price, and CPU Time) for transactions are generated from distributions, as discussed in Chapter 5. We set the number of miners to 10, where each miner controls 10% of the total network hash power. Nine miners follow the protocol honestly by executing the verification process upon receiving a newly generated block, apart from one miner who skips the verification process. For the parallel verification, we set the number of processors to 4 and the conflict rate of transactions to 0.4. Then, we record the fraction of fee each miner receives at the end of the simulation.

We run simulation experiments with different configurations of block limits (ranging from 8M to 128M). For each configuration, we simulated the equivalent of 3 days of running time of the Ethereum network and repeated this to have 100 independent runs. Figure 6.1 shows the validation of both the Ethereum base model and the parallel verification by presenting the results from the closed-form solutions as well as from the simulation. The vertical axes show the percentage of the received fee the non-verifying miner receives. One sees from Figure 6.1 that the non-verifying miner always wins, since in this scenario all blocks are valid, so the miner is never penalised for not verifying. The gain can be a full percentage point or more as the block limit increases. Various additional results will be discussed in Section 6.5.

We note that the simulation results slightly differ from that of the closed-form for the larger block limits. The closed-form expressions slightly overestimate the gain miners get from not validating blocks, but the differences are small. Several elements are modelled in more detail in the simulation than in the closed-form expressions, and these may contribute to randomness that causes a difference between closed-form and simulation. We believe that it is fair to conclude from Figure 6.1 that the closed-form expressions are close to the simulation results.

6.5 Results

In this section, we present the main findings from our analysis of the Verifier's Dilemma, under the Ethereum base model as well as under the proposed mitigations of parallel verification and intentional production of invalid blocks. Our main metric of interest is the fee gained or lost by non-verifying miners in various scenarios.

We summarise the main findings that follow from our discussion upfront:

[1] https://etherscan.io/chart/blocktime

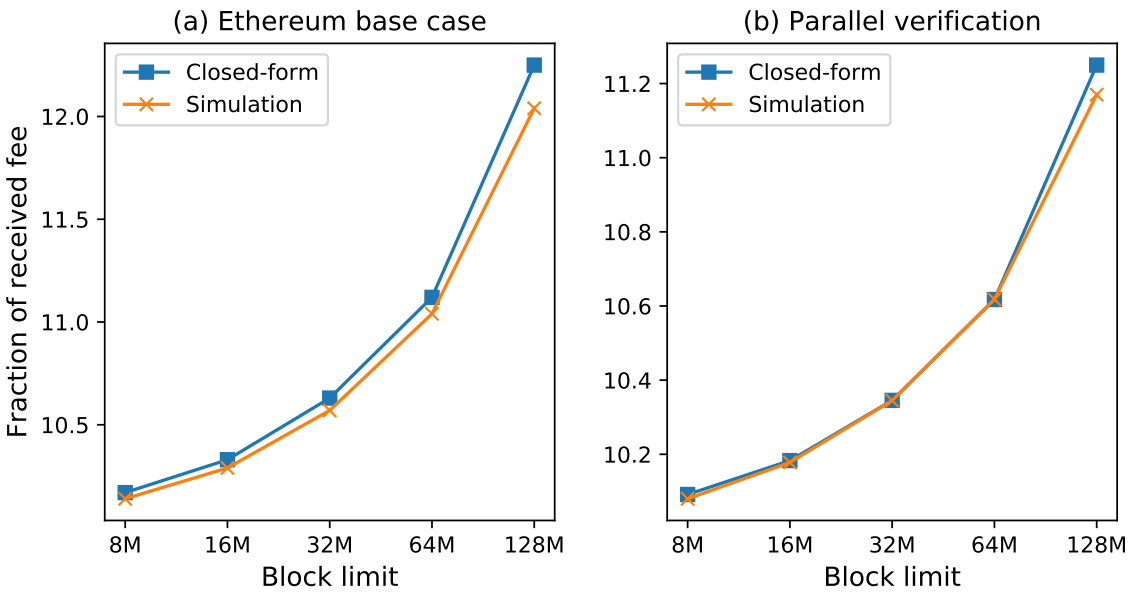

Figure 6.1: Results from the closed-form expressions and the simulation in the fraction of fee received by a non-verifying miner who has 10% of hashing power, for (a) the Ethereum base model and (b) the parallel verification solution.

- The smaller the hash power a miner controls, the more advantage the miner would gain from skipping the verification process.
- In today's Ethereum, miners gain relatively little from skipping the verification (less than 2% of the invested hash power). This is because the block limit in Ethereum is currently small.
- In the future, the Ethereum block limit is expected to increase. In that case, skipping verification becomes considerably more lucrative. This is under the assumption that most miners honestly verify and invalid transactions are rare.
- Parallel verification reduces the benefits miners would get from not verifying blocks. This is especially true if the conflict rate is small and the number of parallel processors is large.
- The mitigation approach to purposely introducing invalid blocks in the network can significantly reduce the benefits of non-verifying miners. This is especially true if the rate of invalid blocks is large or the block limit is small. In this case, miners may be better off to verify.

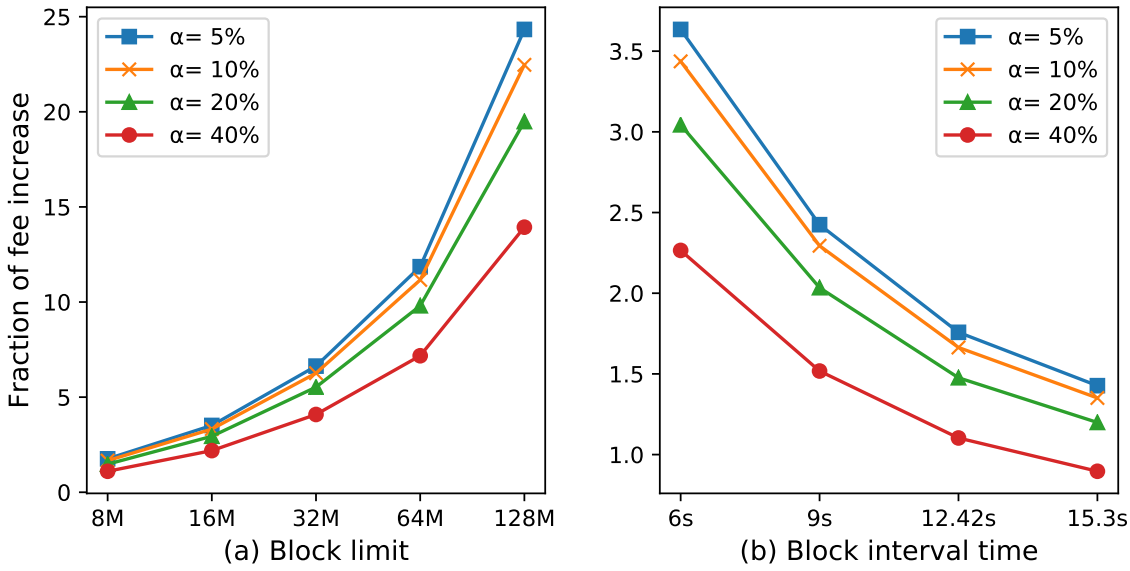

Figure 6.2: The percentage of fee increase for a non-verifying miner with the Ethereum base model: (a) different block limits and (b) different block interval time.

6.5.1 Ethereum Base Model

Figure 6.2 shows the percentage of fee increase a non-verifying miner would gain, for different block limits and different block interval times. The four curves in each of the two plots of Figure 6.2 indicate different fractions of the total hash power owned by the non-verifying miner. In Figure 6.2(a), we consider a block interval time of 12.42 seconds. In Figure 6.2(b), we consider a block limit of 8M, which is the block limit currently used in Ethereum.

From Figure 6.2 we conclude that for the current implementation of Ethereum (block limit = 8M and block interval time is between 12 and 15 seconds), the percentage of fee increase is small (less than 2% of the invested hash power). Yet, this percentage increases significantly with the block limit or the reduction of the block interval time. For instance, a non-verifying miner with $\alpha = 0.05$ would increase its gain from 1.7% for small blocks to a remarkable 22% when the block limit is pushed from 8M to 128M. In addition, we can see that the smaller the hash power a miner controls the larger the increase the miner gets when not verifying blocks. For example, a miner with $\alpha = 0.05$ can increase its fraction of fee to 24% when the block limit is 128M, while it only increases its fraction to about 14% if $\alpha = 0.40$. This is because a miner has to verify all the blocks that were mined by others, which amounts to $(1 - \alpha)$ of the network blocks, as we discussed in Section 6.2.2. In other words, in Ethereum, small miners spend more time on verification than large miners because

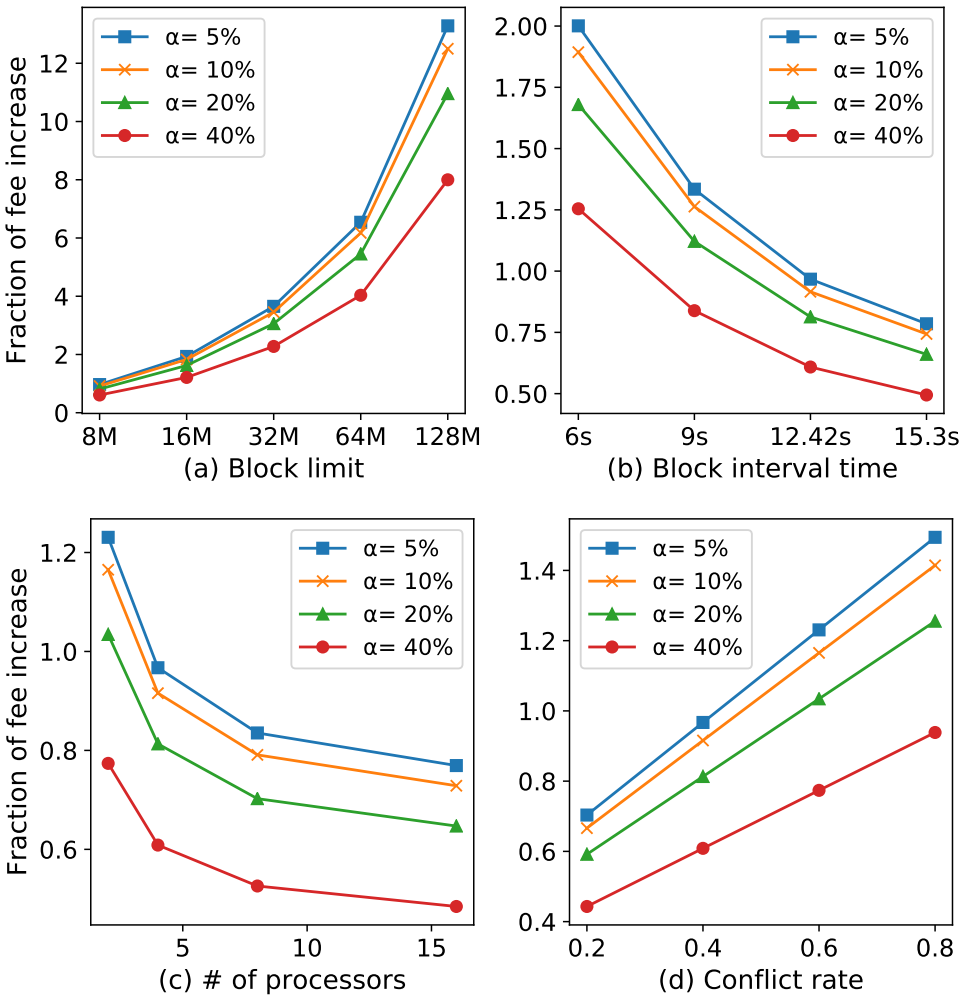

Figure 6.3: The percentage of fee increase for a non-verifying miner with parallel verification: (a) different block limits, (b) different block interval time, (c) different number of processors and (d) different conflict rates.

they receive more new blocks from other miners. Therefore, small miners have more to gain from stopping with verifying.

6.5.2 Parallel Verification

Parallel verification of transactions is a solution that we proposed in Section 6.3.1 to minimise the advantage non-verifying miners would gain by reducing the overall time required for the verification process. Figure 6.3 shows the percentage of fee increase that a non-verifying miner would gain, for different block limits, different block interval times, different number of processors and different conflict rates for transactions. As in Section 6.5.1, the different curves represent different hash powers

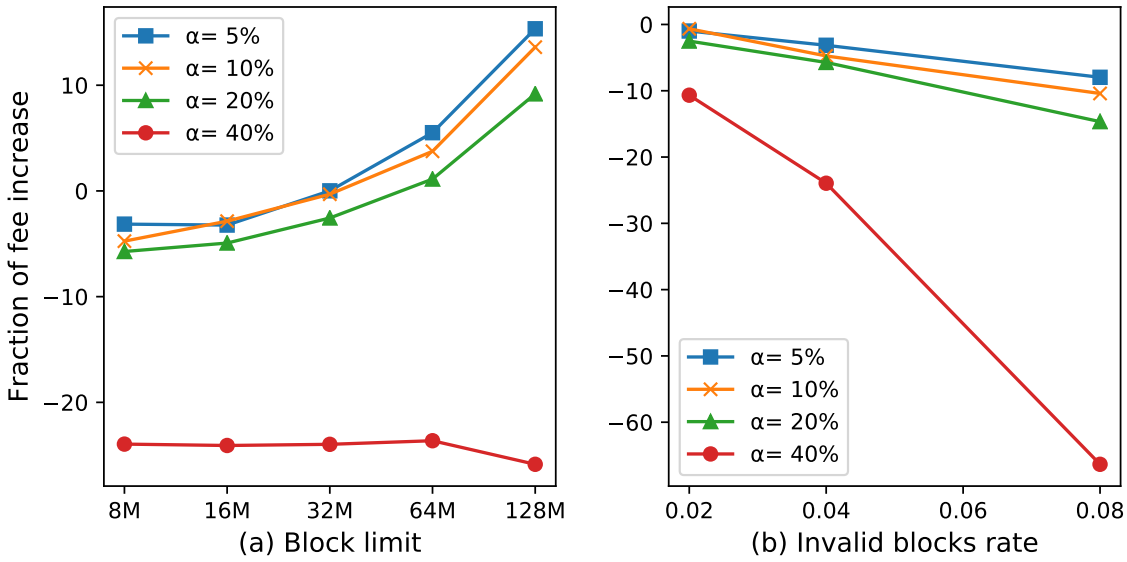

Figure 6.4: The percentage of fee increase for a non-verifying miner with the intentional production of invalid blocks: (a) different block limits and (b) different rates of invalid blocks.

for the non-verifying miner.

From Figure 6.3 we see that although the percentage of fee increase rises with the block limit or the reduction of the block interval time, the advantage is reduced almost to half that of the Ethereum base model (see Figure 6.2). This is for modest parallelisation, with only 4 processors and a conflict rate of 0.4. In addition, from (c) and (d) we see that the advantage decreases further with the increase of the number of processors or with a small rate of conflicting transactions. For instance, assume an 8M block limit and 0.4 conflict rate. Then the increase a non-verifying miner with α= 0.10 would get goes down from about 1.2% to 0.7%, when increasing the number of processors from 2 to 16. To summarise, the advantage a miner would gain by skipping the verification is minimised when shifting from the Ethereum base model to the parallel verification solution. The degree of reduction depends on the conflict rate and the number of concurrent processors.

6.5.3 Production of Invalid Blocks

The idea behind intentionally introducing invalid blocks (Section 6.3.2) is to punish non-verifying miners. To assess if this approach can be useful, we modify the BlockSim simulation classes to account for the possibility of having invalid blocks. We run simulation experiments with different configurations of block limits and different rates

of invalid blocks. The rate of invalid blocks refers to the hash power of the special node that only generates invalid blocks. For each configuration, we simulated 1 day of the Ethereum network and reported the average results obtained from 100 independent runs. For these experiments, we considered a block interval time of 12.42 seconds.

Figure 6.4 shows the percentage of fee increase that a non-verifying miner would gain given some fraction of invalid blocks in the network, for different block limits and different rates of invalid blocks. The different curves represent different hash powers for the non-verifying miner. In Figure 6.4(a), we consider a block interval time of 12.42 seconds and an invalid blocks rate of 0.04. In Figure 6.4(b), we consider a block interval time of 12.42 seconds and a block limit of 8M.

From Figure 6.4 we see that the fee increase for non-verifying miners is significantly reduced when inserting invalid blocks in the network. For instance, the fee increase a non-verifying miner with α= 0.10 would get decreased from about 22% to 13.6%, when the rate of invalid blocks is 0.04 and the block limit is 128M.

Even more interesting, non-verifying miners might get *less* reward than one would expect based on their hash power. That is, we establish a situation in which verifying is preferred over not verifying. This is especially pronounced when the block limit is small or when the rate of invalid blocks is large. For example, a non-verifying miner with α= 0.10 would lose about 5% fee when the block limit is 8M and the rate of invalid blocks is 0.04. That means that conducting the verification process in that case is more profitable than skipping it.

We also note that miners with large hash powers (e.g., $\alpha \geq 0.20$) are affected more when not verifying blocks, compared to miners with small hash powers. For example, a non-verifying miner with $\alpha = 0.05$ can lose about 3% of its expected fraction of fee when the block limit is 8M and the rate of invalid blocks is 0.04, while it would lose about 24% of its expected fraction of fee if $\alpha = 0.40$.

To summarise, purposely introducing invalid blocks into the blockchain could discourage miners from not verifying received blocks. The degree of deterrence depends on the rate of invalid blocks in addition to other blockchain configurations such as block limit.

6.6 Discussion

In this section, we discuss the impact of the assumptions made and the internal validity threats to our evaluation of the Verifier's Dilemma. Also, we present the limitations and challenges of applying the proposed mitigation solutions.

6.6.1 Assumptions

We made several assumptions in Sections 6.2 and 6.3 to evaluate the Verifier's Dilemma. Here, we discuss the assumptions that may impact the analysis results, which are as follows.

Miners and mining pools. We assume that miners can be either solo miners or mining pools, but we do not distinguish miners within a pool. Miners, especially those with less hashing powers, often join a mining pool to get more stable rewards than if they act as solo miners [66]. Miners within a pool collaborate with each other to complete the verification process. That means miners within a pool tend to spend less time on verification individually. However, our analysis of the Verifier's Dilemma is for a pool as a whole and not for the individual miners that constitute a pool.

Static mining behaviours. We studied the Verifier's Dilemma assuming that miners always follow one behaviour (commit or skip the verification process). That is, we have not considered the case where miners switch between the two behaviours and the case where miners dynamically change their behaviours according to the behaviour of other miners. It is possible to find miners who verify some blocks and skip others in order to reduce the risk of accepting invalid blocks (e.g., verifying a block in every ten blocks). It is also possible for miners to adjust their behaviours based on the behaviour of other miners. Instead of only considering a static analysis, it would be nice to study the Verifier's Dilemma as a game theoretic analysis that considers dynamic mining strategies as future work to obtain rich analysis results.

Different types of transactions. We studied the Verifier's Dilemma if transactions are contract-related. However, there are many financial transactions in Ethereum and since these can be verified very quickly the advantage of not verifying blocks may not be as large as in Section 6.5. In that sense, our analysis should be considered a worst case analysis. We believe the main insights derived from our analysis remain valid, even if exact values and results may be different.

Full blocks of transactions. We assume that blocks are filled with transactions, but in the real system it is possible to have non-full or even empty blocks. In that case verifying blocks takes less effort. However, by design the block reward is decreasing over time and is expected to be removed eventually [53], only leaving transaction fees as an incentive. Without block rewards, miners will be encouraged to fill up their blocks with transactions to maximise their rewards.

6.6.2 Threats to Validity

We discuss two threats to the internal validity of our evaluation, namely, the selection of contract transactions and the instrumentation used.

Selection of contract transactions. This can impact the analysis results, and thus, it acts as a potential threat to correctness. To mitigate this threat, we base our evaluation on a large data set that constitutes over 300 thousand contract transactions. Also, we select transactions from the Ethereum network randomly to avoid sample bias. We believe in doing so we establish the most reliable representation of the real system.

Instrumentation. The CPU time to verify each transaction is based on the measurements obtained from a particular machine. Since we base parameter values in this chapter on experiments on a selected machine, the outcomes may not generalise for other architectures. In reality, miners will use different, typically more powerful machines and the specifications of machines are expected to further improve in the future. Also with more powerful machines and/or increased PoW difficulty, the Verifier's Dilemma will be a problem when the block limit reaches a particular threshold or if more complex contracts are permitted. Of course, the analysis approach in this chapter can be applied using data from different machine, and run simulations with different parameter values.

6.6.3 Limitations of the Proposed Solutions

Although we show the effectiveness of the two proposed solutions, there are some limitations and challenges to the adoption of these solutions.

Parallel verification of transactions. Parallel verification discourages skipping verification. However, the implementation of parallel verification on a real blockchain system is challenging [6, 101] and requires multi-threading support in the EVM. Complications arise because a miner needs to attach a table to its block in order for the verifier to know which transactions can be run in parallel. Producing the table is challenging since it requires knowing conflicting and non-conflicting transactions [38]. Moreover, one trusts the miners to produce the correct table.

Intentional insertion of invalid blocks. Inserting invalid block not only make skipping the verification a less beneficial strategy, it often makes verifying the preferred strategy. Although this solution can be easily adopted in Ethereum, its introduction would likely face some challenges. Producing invalid blocks in the network decreases the overall performance of the system and honestly verifying miners as they

are expected to verify those invalid blocks and then reject them. In practice, one would expect Ethereum to be very hesitant adding such overhead to the system.

6.7 Conclusion

This chapter provides an extensive analysis of the Verifier's Dilemma, following a data-driven, model-based approach that combines closed-form expressions with discrete-event simulation and utilises machine learning techniques to parameterise and configure probability distributions used by the simulator. This is the first extensive analysis of the Verifier's Dilemma we are aware of. The insights we gained in this chapter can be of assistance in anticipating the implications of the Verifier's Dilemma under future developments, e.g., when the block limit increases in Ethereum, or when Proof of Work is replaced. Of particular importance for the fairness of blockchain systems is that our analysis shows that small miners are more impacted by the verification demands, and will be more tempted not to verify. Our results also indicate that, counter-intuitively, problems associated with the Verifier's Dilemma exacerbate if there are less invalid transactions. This leads to the insight that future blockchain systems may operate better if designers or operators assure that some transactions are invalid. We suggest that similar analysis as reported in this chapter should be carried out for future system designs and operational developments, to anticipate the consequences of the Verifier's Dilemma.

Chapter 7

Data-Driven Analysis of the Impact of Profit Uncertainty in Ethereum

Summary

In Ethereum, miners face uncertainty about the fee and the cost of individual transactions. That is, they are not able to make a proper decision of which transactions to select to maximise their revenue. In addition to the uncertainty miners face, the Ethereum incentive model is not incentive-compatible as the award miners would get from executing transactions is not proportional to the computational costs. That means some transactions are more profitable than others. With the lack of incentive compatibility the implications of the uncertainty problem can even exacerbate.

We provide an extensive analysis of the impact of the uncertainty problem on the received profit, using data-driven and simulation approaches. We design a model to simulate different transaction selection strategies for scenarios with and without uncertainty. We show that the uncertainty miners perceive when selecting transactions has a significant impact on the per block profit. Also, we show such uncertainty could negatively impact the PoW profit (the fraction of blocks a miner would generate) in future implementations when the block limit becomes relatively large.

7.1 Introduction

In Ethereum, users send transactions to the network to transfer cryptocurrencies, deploy a new smart contract or to invoke an existing one. Miners execute these transactions in blocks, and in turn, they collect their associated fees. Each block has

a limit on how many transactions it can have, and thus, miners usually prioritise transactions by selecting the most profitable ones [74].

However, several studies have shown that the Ethereum incentive model is not proper as the award miners would get from executing transactions is not proportional to their computational costs. That means some transactions are more profitable than others. In addition to the imperfect incentive model, miners do not know the exact income and cost of transactions beforehand. The only information available to the miners before selecting a transaction is the maximum income they can get from it. Since miners face uncertainty about the income and the cost of transactions, they are not able to make informed decisions of which transactions to select and execute to maximise their profits.

In this chapter, we conduct an extensive analysis of the uncertainty problem miners perceive in Ethereum when selecting transactions and its impact on the received profits. To accomplish this analysis, we combine the following techniques. First, we design a simulation model to simulate the decisions of miners when selecting transactions and design different transaction selection strategies for scenarios where miners are both certain and uncertain about the income and the cost of transactions. Secondly, we extend the Ethereum model of the BlockSim simulator with the functionality necessary to support the implementation of the model and the selection strategies. Thirdly, to obtain realistic simulation results, we feed the simulator with the distributions for real transactions (see Chapter 5).

We run simulation experiments to compare the profits (block profit and PoW profit) earned by uncertain miners with that of certain miners to draw conclusions about the implications of the uncertainty problem. With 'certain' miners we mean hypothetical miners that would have available all information about costs and rewards of all transactions. The main conclusions of our evaluation are as follows. First, the uncertainty miners face has a significant impact on the received block profit, especially if the pool size (the number of pending transactions in the network to select from) is large enough. With the lack of such uncertainty, miners can get four times as much block profit compared to the case where uncertainty is present. Furthermore, although there is no significant impact on the PoW profit in today's Ethereum, the effect will become serious in future settings when the block limit goes beyond 32 million units of gas.

The structure of this chapter is as follows. Section 7.2 introduces the uncertainty problem in Ethereum and proposes a model to investigate the problem. In Section 7.3, we establish different selection strategies for transactions to study the implications of

the uncertainty problem. Section 7.4 describes how we used the BlockSim simulator and enhanced it to suit our simulation study. Sections 7.5 and 7.6 present the main findings and discuss the insights gained and conclusions drawn. We conclude the chapter in Section 7.7.

7.2 Uncertainty Issue in Ethereum

We first present and discuss in Section 7.2.1 the uncertainty issue in general terms and then we propose in Section 7.2.2 a model for studying this issue by simulating the selection decisions for miners.

7.2.1 Problem Description

As stated in Section 2.4, Ethereum uses the Gas mechanism to calculate the fee for smart contract transactions. Each opcode of a smart contract uses a predefined amount of gas, as specified in [49]. The EVM tallies the amount of Used Gas and charges the submitter of the transaction based on the Used Gas. To avoid non-terminating transactions the submitter specifies a Gas Limit, and the EVM stops processing if that limit is reached (in which case Used Gas = Gas Limit). The submitter also specifies a Gas Price (expressed in Ether) and the miner then charges the submitter the following transaction fee: Used Gas $\times$ Gas Price. The more opcodes the transaction requires, the more CPU effort from the miner, but also the higher the received reward.

The profit a miner can get from executing a transaction takes into account the income (transaction fee) and the cost (CPU and storage costs) of the transaction [74]. The fee offered by the originator of the transaction is considered as an income from a miner's perspective. The computational work required by the miner to execute a transaction is the cost of the transaction. In blockchain systems, each miner can select any subset of transactions from their pool to execute and include in their block. Since each block has a limit of how many transactions it can have, miners usually priorities transactions by selecting the most profitable ones [74].

In [3], the authors showed that the Ethereum incentive model is not incentive-compatible as the rewards a miner would receive from executing transactions related to smart contracts are not proportional to the computational costs. For instance, some transactions consume extensive CPU Time, while they offer a small fee and vice versa. That means the profit miners would gain varies depending on which transactions miners would execute and include in their blocks. In addition to the

lack of incentive compatibility, the only information provided to the miner about the profit of executing a transaction is the maximum income (Gas Limit and Gas Price). Miners do not know the exact income (Used Gas) they can obtain from a transaction before executing it. Not only this but also miners do not know in advance the cost of executing the transaction. This makes miners uncertain about the profit they can gain from executing a transaction. Thus, they are not able to make informed decisions about which transactions to include in their blocks to maximise their block profits.

Furthermore, in PoW-based systems like Ethereum, miners compete against each other to maximise their PoW mining profits by generating more valid blocks. With such uncertainty, miners might select complex transactions that take a long time to run, delaying them from starting the PoW task earlier. As a result, miners generate fewer blocks (gain less profit) than expected.

7.2.2 The Model

In Ethereum, miners face uncertainty during the selection of transactions. In this section, we propose a model that simulates the decisions that miners take to select and include transactions in their block in order to analyse the impact of the uncertainty issue.

There are important model assumptions that will hold throughout the chapter. The consequent limitations of these assumptions will be discussed in Section 7.6. We assume that miners can be either solo miners or mining pools, and not miners within a mining pool. Individual miners within a pool might be less impacted by the uncertainty problem as they collaborate with each other. We consider an optimisation analysis where we evaluate the profit gained by miners given the transaction selection strategy. That is, we do not consider the behaviour of all miners in the network and the potential impact of it on the system. For instance, what impact can be on the system if all or some miners follow one or multiple strategies. We also assume that miners fill each block by executing as many transactions as they could to maximise their block profit. However, in a real blockchain system, miners can generate full, non-full or even empty blocks. We assume full blocks of transactions to study the uncertainty issue under worst case scenarios, but of course, one can introduce non-full blocks to the model if required. We also assume that all transactions in the network are contract-based transactions, thus ignoring financial transactions that may take place. Financial transactions take less time to execute (less complex) compared to contract transactions, but they can be added to the model if needed. The intention here is to study the uncertainty issue under the worst case scenarios. We neglect the

block propagation delay by assuming that blocks are received immediately by other nodes, as it does not affect the uncertainty issue. Finally, as a first approximation, we assume that the CPU Time required to execute a transaction is representative of its cost.

The proposed model is divided into three main parts, namely, model inputs, model contents and model outputs.

Model inputs: The model takes the miner's pool as an input. The pool has several pending transactions that are waiting to be executed. Each transaction has the following attributes: Gas Limit, Used Gas, Gas Price and CPU Time (see Section 2.4 for details about gas-related attributes). In Chapter 5, we explained the data collection exercise for gathering these attributes. We collected real Ethereum transactions data and then fitted the appropriate distributions. To study the uncertainty problem more realistically, we will feed the model with the fitted distributions.

Model Contents: The model contents describe all steps and formulas needed to calculate the model outputs given the model inputs. To model the decisions that miners take to select and execute transactions in a block, we first sort pending transactions and then select a subset of these transactions to execute in the block.

- Sorting transactions: The first step is to sort all pending transactions in the miner's pool based on their profits. In Section 7.3, we will design five different sorting strategies.

- Selecting and executing transactions: After sorting all pending transactions, the miner selects the first transaction and then checks if it can fit in the block or not. If it does not fit, the miner can select the next one. Otherwise, the miner executes it and then check if the block still has space for other transactions or not. If the block does have space, the miner can select the next transaction. The miner repeats this process until the block is full (no more transactions can fit in the block).

After executing every transaction, the income and the cost of the transaction are calculated and recorded. The income is the transaction fee (Gas Price X Used Gas), while the cost is assumed to be the CPU Time required to execute the transaction.

The block income is calculated as the sum of the income gained from all transactions in the block

$$Block\ Income(in\ Ether) = \sum_{i=1}^{n} F_{\mathrm{i}} \tag{7.1}$$

where F_{i} is the income (fee) of the ith transaction and n is the total number of transactions executed in the block.

The block cost is calculated as the sum of the CPU Time required to execute all transactions in the block

$$Block\ Cost(in\ second) = \sum_{i=1}^{n} T_{\mathrm{i}} \tag{7.2}$$

where T_{i} is the CPU Time consumed by the ith transaction and n is the total number of transactions executed in the block.

The block profit that miners can get from executing all transactions in the block is then calculated as the ratio between the block income and the block cost:

$$Block\ Profit = \frac{Block\ Income}{Block\ Cost} \tag{7.3}$$

To model PoW profit, the model should support the essential elements of a PoW-based blockchain such as block generation and block reception. After filling a block with transactions, the miner will engage in solving the PoW task to form the block. Upon a successful task, the block will be propagated to other nodes to have it confirmed and included in the blockchain ledger. The details about these elements can be found in Chapter 4.

The PoW profit that miner i can get is simplified as the fraction of blocks that have been successfully included in the blockchain ledger, which is as follows:

$$PoW\ Profit = \frac{B_{\mathrm{i}}}{B_{\mathrm{total}}} \tag{7.4}$$

Where $\mathrm{B_i}$ is the fraction of blocks contributed by miner i and $\mathrm{B_{total}}$ is the sum of the blocks contributed by all miners.

Model outputs: The model has two main outputs, namely, the block profit and the PoW profit. These outputs have been explained in the model contents.

Group	Strategy	Information	Selection Criteria	Income	Cost
Baseline	S1: Gas Price	Gas Limit	Gas Price	NO	NO
	S2: Maximum income	Gas Price	Gas Limit * Gas Price	NO	NO
Optimised	S3: Exact income	Used Gas Gas Price	Used Gas * Gas Price	YES	NO
	S4: Exact cost	Gas Limit Gas Price CPU Time	$\frac{\text{Gas Price}}{\text{CPU Time}}$	NO	YES
	S5: Exact profit	Used Gas Gas Price CPU Time	$\frac{\text{Gas Price}}{\text{CPU Time}}$ Used Gas	YES	YES

Table 7.1: The strategies of the baseline and the optimised groups.

7.3 Transactions Selection Strategies

To investigate the impact of the uncertainty issue, we will compare the profit gained for scenarios where miners are uncertain about the income and the cost of transactions with the case when certainty is present. This can be accomplished by evaluating different strategies for selecting and including transactions in the block to be created.

In this section, we establish five different strategies to simulate the decisions that miners can take to select a subset of transactions to execute and include in their forthcoming block. We classify these strategies into two groups, namely, *baseline* and *optimised*. The baseline group is to simulate the decisions of miners under the Ethereum condition, where miners are uncertain about the income and the cost of transactions. The only information available to the miners in the baseline group is the maximum income (Gas Limit and Gas Price) they can get from executing transactions. The optimised group is to simulate the decisions of miners with the absence of such uncertainty. Table 7.1 illustrates the strategies in both the baseline and the optimised groups. The "Information" column specifies which transactions' attributes are available to the miners before selecting transactions. The "Selection Criteria" column states the decisions that miners take to sort and select transactions. The last two columns show the differences between the five strategies in terms of income and cost certainty. For example, miners are certain about both the income and the cost of transactions in the exact profit strategy.

7.3.1 Baseline Group

In the baseline scenarios, there are two possible strategies that miners can use to select transactions from their pools, which are as follows.

- Gas Price strategy: In this strategy miners sort transactions based on the Gas Price attribute. That is, miners select transactions that offer the highest Gas Price values.
- Maximum income strategy: In this strategy miners sort transactions based on the maximum income they might get from executing transactions. That is, miners select transactions that offer the maximum income, which can be calculated as follows:

$$Maximum\ income = Gas\ Limit * Gas\ Price \tag{7.5}$$

7.3.2 Optimised Group

In the optimised scenarios, we assume miners know the income and/or the cost of transactions, thus we evaluate three optimised strategies for selecting transactions, which are as follows.

- Exact income strategy: This strategy is to see whether the certainty about the income of executing transactions would help miners increase their profit. In this strategy, we assume miners know the exact income (fee) they can get from transactions in advance, but not the cost. That is, miners select transactions that offer the highest exact income, which can be calculated as follows:

$$Exact\ income = transaction\ fee = Used\ Gas * Gas\ Price \tag{7.6}$$

- Exact cost strategy: This strategy is to see whether the certainty about the cost of executing transactions would help miners increase their profit. In this strategy, we assume miners know the exact cost (CPU Time) of transactions in advance, while they are still uncertain about the exact income of transactions. That is, miners select transactions that offer the highest expected income per cost unit, which can be calculated as follows:

$$Exact\ cost = \frac{Gas\ Price}{CPU\ Time} \tag{7.7}$$

- Exact profit strategy: This strategy is to see whether the certainty about both the income and the cost of executing transactions would help miners increase their profit. In this strategy, we assume miners know both the exact income and the exact cost of transactions in advance. That is, miners select transactions that offer the highest exact profit, which can be calculated as follows:

$$Exact\ profit = \frac{Exact\ income}{CPU\ Time} \tag{7.8}$$

7.4 BlockSim Simulator Extension

This section reports on how we extend the BlockSim simulator to support the analysis of the uncertainty issue miners face in Ethereum, described in Section 7.2.1. In particular, we extend and modify the Ethereum model of the BlockSim simulator to incorporate the model we propose in Section 7.2.2 as well as the selection strategies we design in Section 7.3. The modifications we made to the BlockSim simulator are as follows:

- We extend the `Transaction` module to include the CPU Time as an attribute in addition to other attributes such as Gas Limit, Used Gas and Gas Price.
- We define a new class named `DistFit` to fit probability distributions to the transactions' attributes (see Chapter 5 for more detail). We fit these distributions before starting the simulator. Then, when we want to create new transactions, we can sample random values for these attributes from the fitted distributions.
- We extend the `Node` module to include different sorting strategies that miners can use to sort pending transactions in their pool. We explain the details about these strategies in Section 7.3. The current implementation of the BlockSim simulator assigns the default sorting strategy to all miners. In the default strategy, transactions are sorted based on the value of the Gas Price attribute.
- We slightly modify the `Consensus` module of BlockSim to support the time it takes miners to execute transactions in a block. The current implementation omits this detail by only accounting for the PoW time, as explained in Chapter 4. Therefore, we modify the time it takes a miner to create a block to account for the PoW time plus the time required to execute all the transactions in the block.

- We extend the `Statistics` module to calculate and print the model outputs, which are the block income, the block cost, the block profit and the PoW profit. For convenience, we print the results in an Excel file.

7.5 Results

In this section, we present the main findings from our analysis of the uncertainty miners face in Ethereum when selecting transactions. Our main metrics of interest are the block profit and the PoW profit gained by miners in various scenarios, with and without the presence of such uncertainty. The block profit dictates the ratio between the income and the cost of transactions included in a block. The PoW profit dictates the fraction of blocks accepted in the blockchain ledger, which reflects the mining rewards.

We consider current and likely future configurations of Ethereum in terms of block limit and pool size. The block limit dictates the number of transactions in a block, and it is expected to increase in the future to scale the system. The pool size refers to the number of pending transactions in the network that miners can select from. The pool size fluctuates over time but might increase significantly in the future when Ethereum becomes more popular than now. Etherscan shows the pool size of the Ethereum network at every minute for the last four days. By looking at the data in Figure 7.1 (5-9 July 2019), we can see the pool size varies between about 4,000 and just over 25,000 transactions. That is, it would be valuable to consider different configurations for both block limit and pool size to study the uncertainty issue.

For all experiments in this chapter, we set the number of miners as 5, where each miner controls 20% of the total network hash power. We assign each miner to one of the five transaction selection strategies that we described in Section 7.3. In this section, for the ease of reference, we will refer to the five strategies as S1, S2, S3, S4 and S5. S1 and S2 represent miners in the baselines strategies, while S3-S5 represent miners in the optimised strategies.

We summarise the main observations that follow from our discussion upfront:

- **Impact of uncertainty on the block profit.** The uncertainty miners perceive in Ethereum about the income and the cost of transactions has a significant impact on the block profit miners can get (up to a factor 4). This is especially true when the pool size increases as it means that more transactions are available for miners to select from.

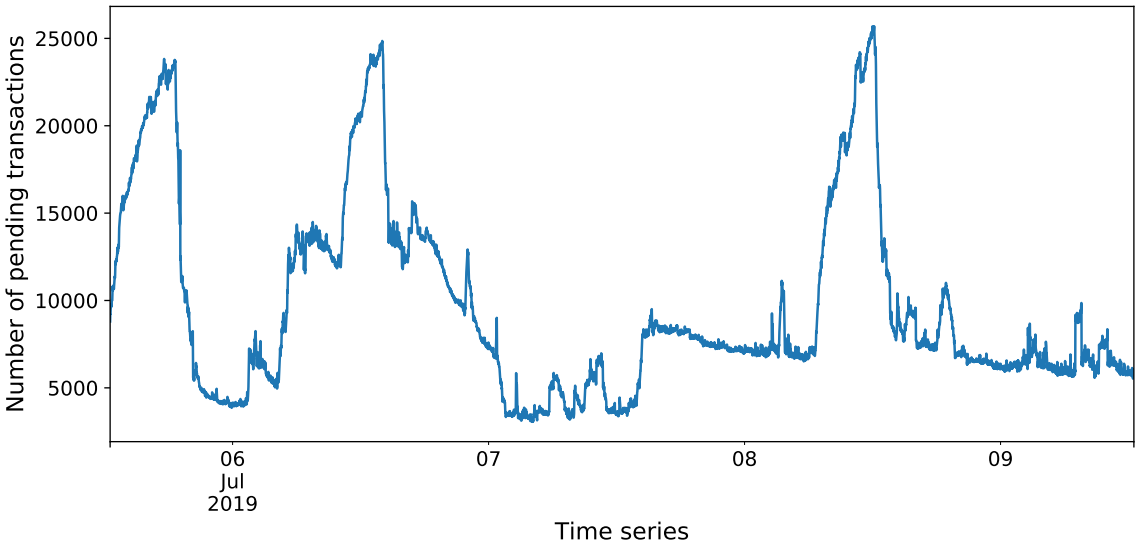

Figure 7.1: Pool size of Ethereum's transactions.

- **Impact of uncertainty on the PoW profit.** The uncertainty miners face has only an impact on the PoW profit when the block limit increases beyond 32 million units of gas. That is, there is no impact on the current implementation of Ethereum as the block limit is small (8 million units of gas).

7.5.1 Impact of Uncertainty on the Block Profit

In this section, we present and discuss the impact of the uncertainty on the block profit gained by miners. Table 7.2 summaries the results for 25 different configurations of different block limits and pool sizes for all the five strategies. The results are the block income (BI) in Ether, the block cost (BC) in seconds and the block profit (BP). For each configuration, we report the average result from 1000 independent simulation runs. The confidence interval is not reported here, but it is within 2% of the average result.

From Table 7.2, we can see a significant difference between the five strategies in terms of the block income, the block cost and the block profit. It is clear there is a trade-off between the block income and the block cost. For instance, S1, S2 and S4 achieve the highest block income, while incurring the highest block cost. However, maximising the block income does not mean maximising the block profit as the profit should consider both the income and the cost imposed by executing transactions. Thus, we will compare, in more detail, the block profit gained by miners in the baseline strategies with that of the optimised strategies to get insights about the

Block limit	Pool size	S1			S2			S3			S4			S5		
		BI	BC	BP	BI	BC	BP	BI	BC	BP	BI	BC	BP	BI	BC	BP
8M	4k	0.39	0.24	1.65	0.43	0.24	1.85	0.15	0.06	2.78	0.35	0.22	1.55	0.2	0.06	3.85
	8k	0.47	0.24	2.01	0.5	0.22	2.34	0.17	0.05	3.85	0.41	0.21	1.98	0.23	0.04	5.8
	16k	0.55	0.23	2.43	0.59	0.22	2.75	0.19	0.05	4.96	0.48	0.19	2.49	0.27	0.03	8.09
	32k	0.65	0.22	3.03	0.69	0.22	3.36	0.22	0.04	6.58	0.54	0.18	3	0.33	0.03	11.45
	64k	0.76	0.22	3.64	0.81	0.22	3.95	0.25	0.03	8.53	0.59	0.17	3.51	0.41	0.03	15.54
16M	4k	0.69	0.49	1.44	0.71	0.49	1.49	0.25	0.12	2.32	0.62	0.47	1.33	0.35	0.1	3.57
	8K	0.85	0.48	1.82	0.86	0.48	1.84	0.28	0.1	2.91	0.75	0.43	1.74	0.37	0.07	5.56
	16k	1.01	0.46	2.23	1.01	0.45	2.28	0.3	0.09	3.79	0.9	0.4	2.28	0.44	0.06	8.04
	32k	1.19	0.45	2.7	1.18	0.44	2.72	0.36	0.08	5.66	1.04	0.38	2.76	0.53	0.05	11.36
	64k	1.38	0.43	3.25	1.38	0.43	3.27	0.42	0.06	7.81	1.11	0.35	3.17	0.67	0.05	14.74
32M	4k	1.06	0.95	1.12	1.04	0.91	1.16	0.42	0.24	1.88	1	0.95	1.05	0.62	0.26	2.45
	8k	1.4	1	1.41	1.34	0.97	1.39	0.44	0.2	2.3	1.3	0.95	1.37	0.64	0.16	4.13
	16k	1.71	0.99	1.76	1.63	0.98	1.68	0.47	0.2	2.5	1.5	0.84	1.78	0.67	0.1	6.58
	32k	2.05	0.9	2.29	1.92	0.96	2.02	0.53	0.18	3.35	1.85	0.78	2.38	0.8	0.09	9.35
	64k	2.41	0.88	2.75	2.28	0.9	2.56	0.65	0.14	5.64	2.11	0.74	2.86	1	0.08	12.51
64M	4k	1.48	1.7	0.88	1.42	1.53	0.93	0.96	0.73	1.37	1.39	1.81	0.77	1.17	0.78	1.55
	8k	2.13	1.92	1.11	1.98	1.78	1.11	0.76	0.44	1.84	2.02	1.92	1.05	1.16	0.46	2.57
	16k	2.81	2.02	1.39	2.56	1.92	1.34	0.83	0.38	2.27	2.64	1.91	1.38	1.24	0.3	4.33
	32k	3.42	2.02	1.71	3.14	1.98	1.59	0.86	0.39	2.23	3.01	1.67	1.81	1.24	0.18	7.17
	64k	4.13	1.81	2.29	3.72	1.97	1.89	0.98	0.35	3.02	3.73	1.54	2.43	1.52	0.15	9.88
128M	4k	1.96	3.12	0.63	1.89	3.05	0.62	1.9	2.63	0.73	1.85	3.46	0.53	1.91	2.56	0.75
	8k	2.94	3.39	0.87	2.74	3.05	0.9	1.89	1.42	1.35	2.78	3.64	0.77	2.34	1.56	1.53
	16k	4.23	3.85	1.1	3.8	3.52	1.08	1.43	0.86	1.72	4.05	3.84	1.05	2.24	0.87	2.58
	32k	5.6	4.09	1.37	4.94	3.84	1.29	1.58	0.73	2.18	5.3	3.82	1.39	2.34	0.55	4.37
	64k	6.86	4.07	1.69	6.14	4.01	1.53	1.64	0.78	2.12	6.02	3.31	1.82	2.37	0.32	7.43

Table 7.2: A summary of the experiment's results for different configurations of block limits and pool sizes for all the five strategies (S1, S2, S3, S4, S5).

impact of the uncertainty miners face when selecting transactions.

Since the block income and the block cost for the two baseline strategies (S1 and S2) were almost the same, the block profit gained by miners in these strategies is the same. That is, there is no difference between the two baseline strategies in terms of the block profit.

In the first optimised strategy (S3), miners can get up to 1.4 times more block profit compared to miners in the baseline strategies. That is, the certainty about the income of transactions can help miners to increase their block profit significantly. This is a bit surprising since miners in this strategy still uncertain about the cost of transactions. However, this might be because miners in this strategy select transactions that offer the highest income (fee), which are more likely to be contract-creation transactions.

In the second optimised strategy (S4), however, miners are not able to achieve any significant improvement in terms of the block profit compared to miners in the baseline strategies. That is, the certainty about the cost of transactions does not help

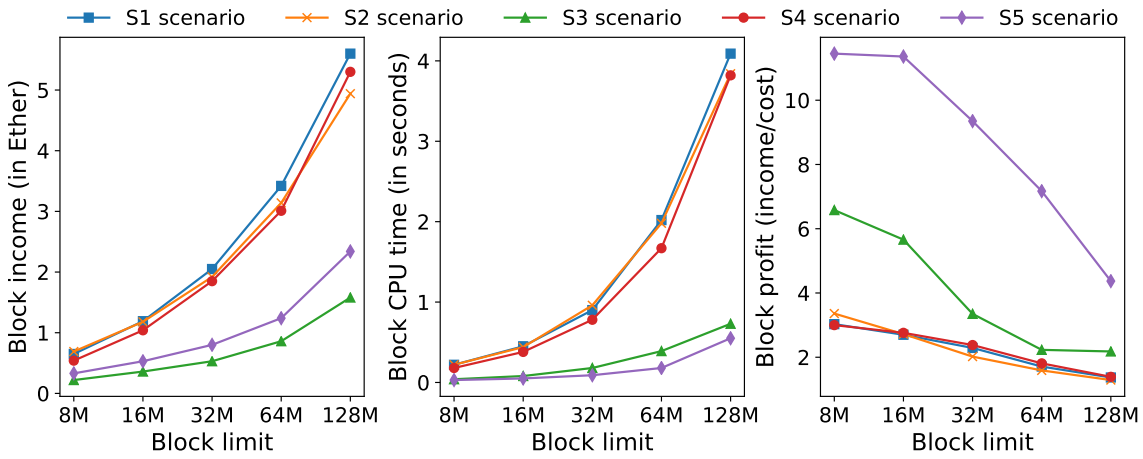

Figure 7.2: Block income (left), block cost (middle) and block profit (right) for different block limits.

miners increase their block profit. This might be due to the impact of the uncertainty miners perceives in this strategy about the block income.

In the third optimised strategy (S5), miners can get up to four times as much block profit compared to miners in the baseline strategies. That is, the certainty about both the income and the cost of transactions can help miners to maximise their block profit significantly. Also, miners in this strategy can gain more block profit compared to other optimised strategies since they are certain about both the income and cost of transactions.

Impact of the block limit growth. Figure 7.2 shows the block income (left), block cost (middle) and block profit (right) for different block limits for all the five strategies. The pool size here is 32,000 transactions. The x-axis shows the block limit in millions of gas units, while the y-axis shows the result of interest. The five curves represent the five selection strategies.

From Figure 7.2, we can see that when the block limit increases, both the block income and the block cost also increase for all the five strategies, at roughly the same pace. This is expected since increasing the block limit means more transactions can be included. Regarding the block profit, it is unclear whether the increase of the block limit can lead to a rise in the block profit. This is because when the block limit increases both the income and the cost also increase, leading to roughly the same block profit. Therefore, the impact of the uncertainty miners face does not grow with the rise of the block limit. For instance, miners with certainty (e.g., S5 strategy) can

get up to four times as much block profit compared to uncertain miners regardless of the block limit.

It is worth noting that the block profit in Figure 7.2 seems to be decreasing when the block limit increases for all the five strategies. This is because we are using the same pool size over different configurations of block limits. When the pool size is small compared to the block limit, the profit is expected to be less since there will not be many choices of transactions for miners to select from.

Impact of the pool size growth. Figure 7.3 shows the block income (left), block cost (middle) and block profit (right) for different pool sizes for all the five strategies. The block limit here is 32 million units of gas. The x-axis shows the pool size, while the y-axis shows the result of interest.

From Figure 7.3, we can see that when the pool size increases, the block income also increases for all the five strategies. This is expected since increasing the pool size means more choices of transactions are available to miners to select from. However, the rate of increase is different among the strategies. Uncertain miners (S1 and S2) can get 1.2 times more block income when the pool size increases from 4000 to 64000. Miners with S4 strategy can almost achieve the same increase rate as uncertain miners. On the contrary, certain miners with strategies S3 and S5 can only increase their block income by up to 60%. This is because they aim at maximising their block profit by considering both the block income and the block cost.

Regarding the block cost, we can see different behaviours between the five strategies when the pool size increases. The block cost for uncertain miners (S1 and S2) remains almost the same, even when the pool size increases. On the contrary, the block cost for certain miners (S3-S5) decreases with different degrees. For example, the block cost decreases by 22% for S3, 41% for S4 and 69% for S5, when the pool size increases from 4,000 to 64,000. This is because when the pool size increases, miners with certainty can select the best (cheap) transactions, compared to uncertain miners who do not know the cost of transactions beforehand. That is, miners with certainty can decrease the block cost significantly with the rise of the pool size, while uncertain miners maintain roughly the same block cost.

Regarding the block profit, we can see that when the pool size increases, the block profit also increases for all the five strategies. However, we observe differences between the five strategies in terms of the degree of increase. Miners with certainty (e.g., S5) can get four times more block profit when the pool size increases from 4,000 to 64,000, while miners with uncertainty (e.g., S1) can only get 1.2 times more block

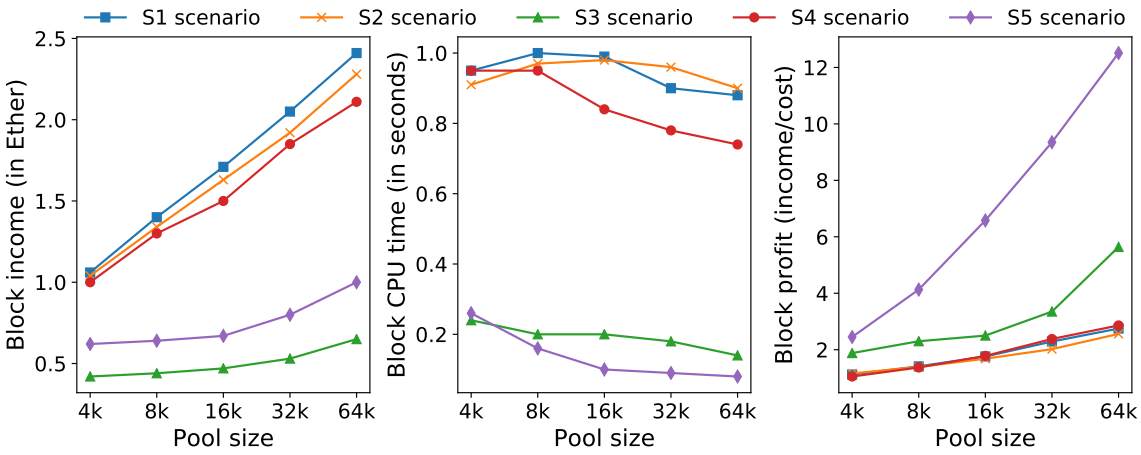

Figure 7.3: Block income (left), block cost (middle) and block profit (right) for different pool sizes.

profit. This indicates that although with uncertainty, miners can increase their block profit when the pool size increases, the increase can even be larger with the absence of such uncertainty.

To summarise, the impact of the uncertainty miners face becomes even more obvious with the growth of the pool size. With the lack of uncertainty, miners can better optimise their selection of transactions when the pool size increases, leading to a significant increase in their block profit as apposed to uncertain miners.

7.5.2 Impact of Uncertainty on the PoW Profit

In this section, we present and discuss the impact of the uncertainty on the PoW profit gained by miners. The PoW profit here dictates the fraction of generated blocks (the fraction of fee received). Table 7.3 summaries the PoW profit results for 25 different configurations of different block limits and pool sizes, for all the five strategies. For each configuration, we simulate one day of the Ethereum network and then report the average results from 10 independent runs. The confidence interval is not reported here, but it is within 5% of the average result.

From Table 7.3, we can see some differences between the five strategies in terms of the PoW profit, especially when the block limit goes beyond 32 million units of gas. For the baseline strategies, our simulation results show that the average PoW profit for both strategies was almost the same. That is, that there is no difference between the two baseline strategies in terms of the PoW profit.

Block limit	Pool size	S1 (%)	S2 (%)	S3 (%)	S4 (%)	S5 (%)
8M	4k	20.027	20.009	19.91	19.656	20.397
	8k	20.027	20.218	19.701	19.978	20.075
	16k	20.062	20.129	19.99	19.643	20.173
	32k	20.08	19.912	19.812	19.913	20.282
	64k	19.788	19.674	20.199	20.172	20.166
16M	4k	19.917	19.778	20.245	19.598	20.465
	8k	19.736	19.523	20.035	20.006	20.704
	16k	19.885	19.7	20.074	19.868	20.474
	32k	19.55	19.899	20.092	19.998	20.462
	64k	19.47	19.79	19.943	20.298	20.499
32M	4k	19.603	19.395	20.724	19.629	20.65
	8k	19.316	19.843	20.563	19.483	20.796
	16k	19.754	19.427	20.432	19.462	20.925
	32k	19.616	19.489	20.386	19.803	20.705
	64k	19.484	19.718	20.12	20.024	20.655
64M	4k	19.329	19.401	21.286	19.398	20.586
	8k	19.233	19.469	21.062	18.942	21.298
	16k	19.208	19.182	21.102	19.368	21.145
	32k	19.166	19.019	21.1	19.221	21.489
	64k	19.017	19.144	20.55	19.503	21.788
128M	4k	19.641	20.158	20.356	19.117	20.725
	8k	19.169	18.856	22.082	18.807	21.088
	16k	18.293	18.578	22.363	18.196	22.57
	32k	17.996	18.2	22.445	18.535	22.824
	64k	18.956	18.128	21.962	18.261	22.694

Table 7.3: A summary of the PoW profit (the fraction of generated blocks) for different configurations of block limits and pool sizes for all the five strategies (S1, S2, S3, S4, S5).

Similar to the block profit discussed in Section 7.5.1, certain miners in the optimised strategies (S3 and S5) are able to obtain higher PoW profit as opposed to uncertain miners. In the first optimised strategy (S3), miners can get more PoW profit compared to miners in the baseline strategies when the block limit increases beyond 32 million units of gas. That is, the certainty about the income of transactions can help miners to increase their PoW profit by up to 25%. In the second optimised strategy (S4), however, miners are not able to achieve any significant improvement in terms of the PoW profit compared to miners in the baseline strategies. That is, the certainty about the cost of transactions does not help miners increase their PoW profit. In the third optimised strategy (S5), miners can get more PoW profit compared to miners in the baseline strategies when the block limit increases beyond 32

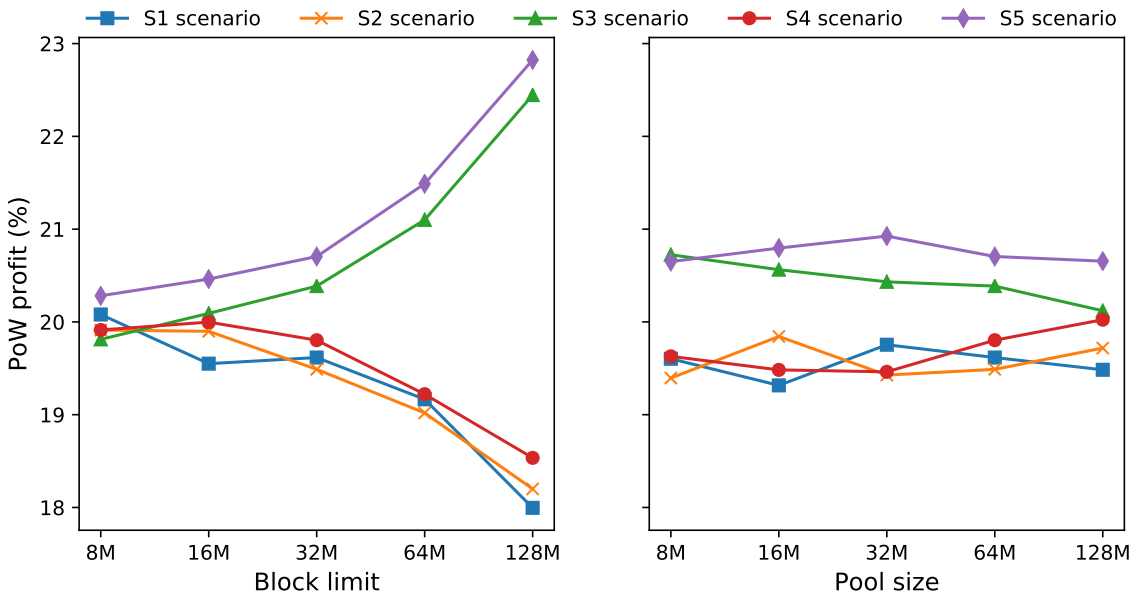

Figure 7.4: PoW profit for different block limits (left) and different pool sizes (right).

million units of gas. That is, the certainty about both the income and the cost of transactions can help miners to maximise their block profit by up to 30%.

Impact of the block limit growth. Figure 7.4 (left) shows the PoW profit for different block limits for all the five strategies. The pool size here is 32,000 transactions. The x-axis shows the block limit in millions, while the y-axis shows the PoW profit. From Figure 7.4, we can see that when the block limit increases, the impact of the uncertainty on the PoW profit also increases. For the current implementation of Ethereum, all uncertain and certain miners receive roughly the same fraction of fee. Thus, there is no impact of such uncertainty on the PoW profit. However, when the block limit goes beyond 32 million, the effect will become visible. For instance, miners with certainty (e.g., S5) can generate up to 27% more fee than uncertain miners (e.g., S1) when the block limit reaches 128 million.

Impact of the pool size growth. Figure 7.4 (right) shows the PoW profit for different pool sizes for all the five strategies. The block limit here is 32 million units of gas. The x-axis shows the pool size, while the y-axis shows the PoW profit. From Figure 7.4, it is unclear whether the increase in the pool size can have an impact on the PoW profit. We showed that in Section 7.5.1, the rise in the pool size could help miners to increase their block profit since miners will have more options for

transactions to select from. However, this is not the case for the PoW profit as the pool size can only help miners to increase their per block profit by offering more choices of transactions to select from.

7.6 Discussion

In this section, we discuss the impact of the assumptions made and the internal validity threats to our evaluation of the uncertainty problem. Also, we discuss the limitations of our analysis.

7.6.1 Assumptions

We made several model assumptions in Section 7.2.2 to evaluate the uncertainty problem in Ethereum. Here, we discuss four assumptions that can impact the analysis results, which are as follows.

Miners and Mining pools. Similar to the study of the Verifer's Dilemma in Chapter 6, our analysis is applicable for miners and mining pools as a whole. That is, our analysis of the uncertainty problem has not considered individual miners that constitute a pool. Miners within a pool might be less impacted by the uncertainty problem depending on how the pool works and how tasks are distributed among the participating miners.

Type of transactions. Similar to the study of the Verifer's Dilemma in Chapter 6, we studied the uncertainty problem considering only contract-related transactions with the aim of providing a worst case analysis. That is, we neglected financial transactions as they are trivial to execute. We believe the results are still valid even that it might be less intensive if we introduced financial transactions,

Full blocks of transactions. Similar to the study of the Verifer's Dilemma in Chapter 6, we studied the uncertainty problem considering only full blocks of transactions with the aim of providing a worst case analysis. The impact of uncertainty should be different if we introduce non-full or even empty blocks. However, we believe miners are more encouraged to fill their blocks to maximise their transactions fees, especially when the block reward is eventually removed [53].

Real cost of transactions. To study the uncertainty problem, we need to feed the model with the cost of transactions. As a first attempt to explore the uncertainty space, we assume that the CPU Time required to execute a transaction is representative of its cost. However, the real cost of a transaction should be related to the actual energy cost, or at least consider other overhead (e.g., memory usage)

beyond the CPU usage. Although the main insights derived from our analysis remain useful, more accurate and reasonable results can be obtained by feeding the model with the real energy cost.

7.6.2 Threats to Validity

We discuss three threats to the internal validity of our evaluation, which are as follows:

Selection of contract transactions. Similar to the study of the Verifer's Dilemma in Chapter 6, the selection of transactions can be a threat to the correctness of our analysis. However, we base our evaluation on a large number of transactions that have been selected in a random way, see Chapter 5. That is, we establish the most reliable and less bias representation of the real system.

Instrumentation. Similar to the study of the Verifer's Dilemma in Chapter 6, the CPU time to execute each transaction is based on the measurements obtained from a particular machine. That is, the outcomes may not generalise for other architectures. However, it is possible to reproduce the analysis results using CPU Time data from different machines.

Ethereum incentive model. The effect of uncertainty problem only holds on the current implementation of the Ethereum Gas mechanism as it does not provide the right incentives for miners. We expect Ethereum to improve its Gas mechanism to properly and adequately set the fee for transactions. In that case the uncertainty miners face would have no impact on the received profit as miners would always gain rewards that are compatible with the efforts spent, regardless of whether they face uncertainty or not.

7.6.3 Limitations

We discuss two limitations of our analysis, which are as follows.

Analysis of collective behaviour. Our analysis of the uncertainty problem is a classic optimisation analysis, where we compare the profit a miner would get by following each of the five transaction selection strategies. That is, our analysis does not consider the selection of such strategies given the behaviour of other miners. One way to address this limitation is by considering a game theoretic analysis that accounts for the behaviour of other miners.

As our analysis has not considered the behaviour of other miners in the network, we cannot draw conclusions about the impact on the system if, for instance, all or some miners follow one strategy of selecting transactions. Assume that miners follow the

best strategy (e.g., strategy S5) by selecting transactions based on the ratio between rewards and cost. In this case, transactions that offer a small reward or incur a high cost will wait for long time to be selected or even worse they might never be selected. We intend to extend this optimisation analysis to a game theory by considering the behaviour of all miners as future work.

Number of miners. In our analysis presented, we only consider five miners, each following a different strategy. The number of miners in our current optimisation analysis does not really matter. This is because our aim is to compare the profit a miner may get when there is certainty about the rewards and/or the cost of transactions (optimised strategies) with the current Ethereum case where miners are uncertain about both rewards and costs of transactions. However, the number of miners might matter if we consider a game theoretic analysis that accounts for the behaviour of others miners. For example, an uncertain miner who follow the baseline strategy S1 might be impacted more if there is a large number of miners following the best strategy S5. We note that one can easily change the number of miners in the network in the simulator and then run the desirable analysis.

7.7 Conclusion

This chapter provides an extensive analysis of the uncertainty problem miners perceive in Ethereum when selecting transactions, following a data-driven approach and discrete-event simulation techniques. To the best of our knowledge, this is the first analysis attempt of the uncertainty problem we are aware of.

The main insights we gained from this analysis are as follows. First, the uncertainty miners face can significantly reduce the earned block profit (up to a factor four), especially when there are a large number of pending transactions to select from. Besides, such uncertainty can have a considerable impact on the PoW profit in future implementations when the block limit becomes relatively large.

We suggest that similar analysis as reported in this chapter should be carried out for future developments, e.g., when Proof of Work is replaced or when Ethereum has adjusted its incentive model. Also, we suggest reproducing the analysis results after feeding the model with the real energy cost for transactions as opposed to the CPU time metric we used in this chapter.

Chapter 8

Conclusion and Future Work

Summary

In this chapter, we first introduce and list the main contributions of this research. Then, we provide a summary of the work presented in this thesis and point to some directions for future work.

8.1 Thesis Contribution

In this thesis, we contribute to the field of blockchain analysis as follows. First and most important, we propose BlockSim as a framework and tool for blockchain systems to address the limitations of existing simulation tools proposed in the literature. Unlike existing tools, BlockSim simulator is designed to be generic, extensible and easy to use. BlockSim can assist blockchain designers, analysts and researchers to explore various performance and system properties.

Another contribution of this thesis is two extensive data-driven simulation studies related to Ethereum smart contracts, regarding the Verifier's Dilemma and the profit uncertainty problems. We provide (to the best of our knowledge) the first data-driven rigorous analysis of these problems using probabilistic modelling techniques with the help of our BlockSim simulator. Thirdly, to run these studies realistically, we collect real Ethereum smart contracts data and then transform it into distributions to parameterise the simulator. Finally, we conduct a systematic mapping review to understand research trends on the general domain of smart contracts, and to identify areas for future research. In this section, we provide a summary of these contributions as follows:

1. Conducting a systematic mapping study to explore the current research on blockchain-based smart contracts. From the mapping study, we provide a survey

of the scientific literature, identify academic research trends and uptake and identify gaps for further research **(Chapter 3)**.

2. Designing and developing a generic blockchain simulator named `BlockSim` that is flexible enough to support the analysis of a large variety of blockchains and a wide set of analysis problems. BlockSim covers different blockchain layers, and provides simulation constructs that are intuitive, hide unnecessary detail and extensible. At the core of BlockSim is a Base Model that we extend to support the implementation of Bitcoin and Ethereum blockchains. BlockSim is implemented in Python, and it is validated against real-life systems and measurement studies from the literature **(Chapter 4)**.

3. Conducting an extensive analysis to estimate the distributions for Ethereum smart contract transactions, with respect to different attributes. To determine these distributions, we use publicly available Ethereum smart contract information, augmented with experimental data for over 300,000 smart contract transactions obtained on a test bed. The estimated distributions are then fed as inputs to the BlockSim simulator to conduct data-driven simulation studies **(Chapter 5)**.

4. Conducting an extensive data-driven analysis of the Ethereum Verifier's Dilemma, using a data-driven model-based approach that combines closed-form expressions and discrete-event simulation. We show that, indeed, it is often economically rational not to verify, in particular for miners with less hashing power. We consider two approaches to mitigate the implications of the Verifier's Dilemma, namely parallelisation and active insertion of invalid blocks, both shown to be effective **(Chapter 6)**.

5. Conducting an extensive analysis of the impact of the uncertainty miners perceive in Ethereum when selecting transactions, using data-driven and simulation approaches. We show that such uncertainty has a significant impact on the profits miners can gain **(Chapter 7)**.

8.2 Thesis Summary

In this section, we summarise the work that has been carried out in this thesis, highlighting the main contributions and results obtained.

8.2.1 Academic Research on Smart Contracts (Chapter 3)

Blockchains and smart contracts have received increasing attention in recent years, also in academic circles. One contribution of this thesis is to identify and to classify all peer-reviewed research that has been conducted on smart contract technology, with the aim to document the growth of research outputs and to identify gaps for future work.

We conduct a systematic mapping study in longitudinal aspects to document and analyse the growth in research outputs related to smart contracts. From the mapping study, we find that the number of published papers on smart contracts increases significantly every year, reaching over 2500 papers as of March 2020. The results of the study also show six different categories for smart contract topics, which are security, privacy, software engineering, application, performance and scalability and other smart contract related topics. The majority of the papers falls into application (about 64%) and software engineering (21%) categories.

From the systematic study, we identify at least two research areas that can be further explored, and we base our work (Chapters 5-7) on these identified areas.

8.2.2 BlockSim Simulation Framework (Chapter 4)

A major contribution of this thesis is the proposal of a discrete-event simulation framework called BlockSim to explore the effects of configuration, parameterisation and design decisions on the behaviour of blockchain systems. BlockSim aims to provide simulation constructs that are intuitive, hide unnecessary detail and can be easily manipulated to be applied to a large set of blockchains design and deployment questions (related to performance, reliability, security or other properties of interest). That is, BlockSim has three design objectives, namely, generality, extensibility and simplicity.

At the core of BlockSim is a Base Model, which includes model constructs at three abstraction layers: the network layer, the consensus layer and the incentives layer [96]. The Base Model includes a number of functional blocks (e.g., Block, Transaction and Node) common across blockchains, that can be extended and configured as suited for the system and study of interest. The Base Model of BlockSim is implemented through a number of Python modules and complemented by modules (event, scheduler, statistics, etc.) that implement the simulation engine. To illustrate the extensibility of BlockSim, we extend and modify the Base Model of BlockSim to support

the implementation of both Bitcoin and Ethereum blockchains. We validate BlockSim against existing public blockchain systems such as Ethereum and Bitcoin and results from the literature. We show BlockSim can produce statistically acceptable simulation results.

To demonstrate the usefulness of BlockSim, we conduct a simulation study that considers stale rate, throughput and mining fairness, for a range of possible blockchain configurations (not all existing in real-life systems). Using BlockSim we can demonstrate that the Ethereum uncle inclusion mechanism is beneficial for mining fairness. Also, we show that the block interval (i.e. the time between two consecutive blocks) for Bitcoin can be securely reduced to improve the throughput by a factor 10.

8.2.3 Data Collection and Distributions (Chapter 5)

Another contribution of this thesis is the collection of Ethereum smart contract data and the application of appropriate distributions to transform the data into inputs suitable for the BlockSim simulator. This contribution can be divided into several parts, as follows:

Data collection approach. We propose a data collection approach that is capable of collecting the details (e.g., Gas Limit, Gas Price and input data) of Ethereum smart contract transactions. Our approach makes use of the APIs provided by Etherscan explorer, and it is implemented as a Python script and made available to the public.

Measurement system. We also propose a measurement system that tallies the Used Gas and is capable of measuring the execution time of Ethereum smart contract transactions. Currently, our system is implemented on top of the Ethereum Python client but can be applied to other clients if required.

Approach to obtaining distributions. Using our data collection approach and the measurement system, we manage to collect the details of over 300,000 smart contract transactions with respect to the following attributes: Gas Limit, Used Gas, Gas Price and CPU Time. Then, we conduct an extensive analysis to estimate the distributions for these attributes. We use Gaussian Mixture Models to fit distributions to Used Gas and Gas Price since the logarithmic representation of the data resembles a normal distribution. We use Uniform distribution to estimate the distribution of Gas Limit. Due to the correlation between Used Gas and CPU Time, we apply regression methods to predict the CPU Time needed to execute a contract transaction, given the

Used Gas. We compare a number of regression ensemble methods, and then consider Random Forest as it is both fast and accurate.

The resulting distributions have been evaluated in terms of accuracy and performance, and are implemented as a Python class that can be integrated with the Ethereum model of the BlockSim simulator in order to generate representative and realistic smart contract transactions.

8.2.4 The Ethereum Verifier's Dilemma (Chapter 6)

Within Ethereum, miners do not receive incentives for verifying the received block and its associated transactions. This leads to an interesting dilemma: should miners verify transactions within blocks if they do not receive a specific fee for it? If all blocks are valid, the verification would not have been necessary and the time spent on verifying could have been used to mine new blocks (which are rewarded by a fee). This Verifier's Dilemma is well recognised, e.g., [71, 94], but has not been systematically analysed. Our fourth contribution is an extensive model-based analysis of the Verifier's Dilemma in Ethereum.

We combine the following techniques to conduct our analysis of the Verifer's Dilemma. At the core of the analysis is the Ethereum model of BlockSim simulator that we extend with the functionality necessary for the analysis under various scenarios. Secondly, to run realistic simulation studies, we feed the simulator with distributions that we obtain for smart contract transactions, see the previous section. Finally, we derive a number of closed-form expressions to estimate the rewards miners receive if they do or do not verify blocks for base scenarios where no invalid blocks are present.

The conclusion of the analysis results is as follows. There are many scenarios in which miners would benefit from not verifying blocks. This is especially true if (1) all or almost all blocks are in fact valid, and (2) if the block limit is large enough. Within the current implementation of Ethereum, the implications of the Verifier's Dilemma are small but the impact will become more important when the block limit increases.

To mitigate the implications of this dilemma, we consider two approaches. First, we consider parallel verification to decrease the time verifying miners would have to spend before they can mine a new block. Secondly, we consider the idea of injecting invalid blocks on purpose, to penalise miners that do not verify. By injecting invalid blocks, a non-verifying miner would more often pass on chains with invalid blocks that will be rejected by other miners, which in turn would imply that the non-verifying

miner does not receive the block award. Both approaches improve the situation by making it less lucrative for miners to avoid verifying.

We recommend the Ethereum foundation to seriously consider some solutions to mitigate the implications of the Verifier's Dilemma before scaling the system. In this thesis, we suggest two effective mitigation solutions for Ethereum to consider, which are parallel verification and active injection of invalid blocks.

8.2.5 The Impact of Profit Uncertainty (Chapter 7)

In Ethereum, miners face uncertainty about the fee and the cost of individual transactions. That is, they are not able to make an informed decision of which transactions to select to maximise their revenue. In addition to the uncertainty miners face, the Ethereum incentive model is not incentive-compatible (i.e. the award miners get from executing transactions is not aligned with the computational cost), which exacerbates the implications of the uncertainty problem. Our fifth contribution is an extensive analysis of the uncertainty miners face during the selection of transactions, and its impact on the received revenue in terms of block profit and PoW profit.

We combine the following techniques to conduct our analysis. First, we design different transaction selection strategies for scenarios where miners are both certain and uncertain about the income and the cost of transactions. Secondly, we extend the Ethereum model of the BlockSim simulator with the functionality necessary to support this analysis. Thirdly, to obtain realistic simulation results, we feed the simulator with the distributions for real transactions.

The conclusion of the simulation results is as follows. The uncertainty miners face has a significant impact on the received block profit, especially when the pool size is relatively large. Within the current implementation of Ethereum, the impact on the PoW profit is negligible, but it will become severe in future settings when the block limit is relatively large.

To eliminate the impact of such uncertainty, we suggest the Ethereum foundation to adjust its Gas incentive model to provide fair transaction rewards before scaling the system. With a proper incentive model, the uncertainty miners face would have no impact on the received profit as miners would always gain rewards that are compatible with the efforts spent, regardless of whether they face uncertainty or not.

8.3 Future Work

In this section, we introduce a number of opportunities for future research that can be explored to improve and extend the work presented in this thesis. These opportunities are as follows.

1. **Extending BlockSim Simulator**: In Chapter 4, we propose BlockSim as a general and extensible simulation framework and tool for blockchains. The current version of BlockSim is designed to model public blockchains and contains models for the most popular public blockchains (Bitcoin and Ethereum). One opportunity is to extend BlockSim to support the analysis and study of private blockchains such as Hyperledger. Secondly, as BlockSim currently only models the PoW algorithm used in Bitcoin and Ethereum, another way to extend BlockSim is by considering different consensus algorithms such as PoS and variants of PBFT. In addition, BlockSim can be extended by modelling the underlying broadcast protocol for the peer-to-peer network, instead of modelling the propagation of information as a time delay. That would help in the decision of selecting the right and optimal broadcast protocol.

2. **Data Collection and Fitting:** In Chapter 5, we collect data about Ethereum smart contracts and transform it into distributions in order to provide inputs for the simulator. There are several ways to improve the work presented in this chapter, as follows. First, it would be nice to try more additional distributions and regression methods to fit fast and more accurate distributions to the data. For instance, one can consider different kinds of mixture models such as Phase-type using *HyperStar tool* [86, 87], as an alternative to GMM, to fit distributions to the Used Gas and Gas Price attributes. Secondly, as our measurement data shows that the received fee (in terms of Used Gas) for contract transactions is not well-aligned with the CPU usage, another potential future work is to analyse and understand the factors (e.g., individual opcodes) that contribute to this. This thesis only focuses on collecting and fitting distributions to contract data, and thus we have not considered the analysis of the reasons behind this misalignment. Furthermore, the experiment we perform in this chapter to obtain the Used Gas and the CPU Time is based only on a single machine running the Python Ethereum Virtual Machine (EVM). However, there are several implementations of the EVM that miners can use to

run smart contracts, as mentioned in Section 2.4. Conducting the same experiment using different machines running on different EVM clients would provide more insightful (and possibly different) results about how well the received fee is aligned with the computational efforts. Finally, future work may extend the proposed measurement system to consider computational effort beyond CPU usage such as memory usage, and relating the effort directly to the real energy cost.

3. **The Verifier's Dilemma Problem:** In Chapter 6, we conduct an extensive data-driven analysis of the Ethereum Verifier's Dilemma using closed-form expressions and discrete-event simulations. There are some opportunities to improve this work in the future. First, we only study the Verifier's Dilemma under the PoW algorithm as currently implemented in Ethereum. However, since Ethereum and other blockchains are planning to move to more efficient protocols such as PoS [26, 97], studying the impact of this dilemma under different consensus protocols is of interest. Secondly, we have not derived closed-form expressions for scenarios where invalid blocks are present. We leave this for future work as with expressions one can get insightful results very quickly compared to simulations. Thirdly, our analysis has not considered dynamic mining behaviours and collective behaviour of miners. Future work may extend our analysis by proposing a game theoretic analysis that covers both dynamic and collective mining behaviours. Furthermore, the two proposed mitigation solutions (parallelisation and injection of invalid blocks) face some limitations, as discussed in Section 6.6. That is, addressing these limitations to enable the application of the proposed solutions in the real system would be valuable to the community. Finally, future work may consider different mitigation solutions such as *Sharding* [70] and study their effectiveness in comparison with the solutions we proposed.

4. **The Profit Uncertainty Issue:** In Chapter 7, we conduct an extensive data-driven analysis of the profit uncertainty miners face when selecting transactions. There are various aspects that can be improved as future work. First, we assumed the CPU usage is representative of the cost of a transaction. However, the cost should be related to the actual energy cost. That is, further research is required to feed our model with the actual energy cost for transactions to study the impact of such uncertainty. Similar to the Verifier's Dilemma, we study the uncertainty issue under the PoW algorithm, and that future work

may consider different consensus algorithms. Instead of a typical optimisation analysis, considering the collective behaviour of miners in the network is of interest as future work. Furthermore, proposing a run-time system that allows miners to choose the best (most profitable) transactions could be of interest. Finally, our results show the impact of the profit uncertainty under the current Ethereum incentive model, where the fee of transactions is not proportional to the computational cost. Thus, enhancing the Ethereum incentive model would be an excellent contribution to eliminate the implications of the uncertainty issue.

Bibliography

[1] A Aldweesh, M Alharby, and A van Moorsel. Performance benchmarking for Ethereum opcodes. In *Proceedings of the 15th International Conference on Computer Systems and Applications*, pages 1–2. IEEE, 2018.

[2] A Aldweesh, M Alharby, M Mehrnezhad, and A van Moorsel. OpBench: A CPU performance benchmark for Ethereum smart contract operation code. In *Proceedings of the 2019 IEEE International Conference on Blockchain*, pages 274–281. IEEE, 2019.

[3] A Aldweesh, M Alharby, E Solaiman, and A van Moorsel. Performance benchmarking of smart contracts to assess miner incentives in Ethereum. In *Proceedings of the 14th European Dependable Computing Conference*, pages 144–149. IEEE, 2018.

[4] M Alharby, R Castagna Lunardi, A Aldweesh, and A van Moorsel. Data-driven model-based analysis of the ethereum verifier's dilemma. In *Proceedings of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 209–220. IEEE, 2020.

[5] M Alharby, A Aldweesh, and A van Moorsel. Blockchain-based smart contracts: A systematic mapping study of academic research (2018). In *Proceedings of the 2018 International Conference on Cloud Computing, Big Data and Blockchain*, pages 1–6. IEEE, 2018.

[6] M Alharby and A van Moorsel. Blockchain-based smart contracts: A systematic mapping study. In *Proceedings of the 4th International Conference on Computer Science and Information Technology*, pages 125–140. AIRCC Publishing Corporation, 2017.

[7] M Alharby and A van Moorsel. The impact of profit uncertainty on miner decisions in blockchain systems. *Electronic Notes in Theoretical Computer Science*, 340:151–167, 2018.

[8] M Alharby and A van Moorsel. BlockSim: A simulation framework for blockchain systems. *ACM SIGMETRICS Performance Evaluation Review*, 46(3):135–138, 2019.

[9] M Alharby and A van Moorsel. Blocksim: An extensible simulation tool for blockchain systems. *Frontiers in Blockchain*, 3:28, 2020.

[10] M Alharby and A van Moorsel. Fitting and regression for distributions of Ethereum smart contracts. In *Proceedings of the 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services*, pages 248–255. IEEE, 2020.

[11] B Ampel, M Patton, and H Chen. Performance modeling of Hyperledger Sawtooth blockchain. In *Proceedings of the 2019 IEEE International Conference on Intelligence and Security Informatics*, pages 59–61. IEEE, 2019.

[12] D. R Anderson, D. J Sweeney, T. A Williams, J. D Camm, and J. J Cochran. *An introduction to management science: quantitative approaches to decision making*. Cengage Learning, 2015.

[13] E Androulaki, A Barger, V Bortnikov, C Cachin, K Christidis, A De Caro, D Enyeart, C Ferris, G Laventman, Y Manevich, S Muralidharan, C Murthy, B Nguyen, M Sethi, G Singh, K Smith, A Sorniotti, C Stathakopoulou, M Vukolić, S. W Cocco, and J Yellick. Hyperledger Fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15. ACM, 2018.

[14] S. D Angelis, L Aniello, R Baldoni, F Lombardi, A Margheri, and V Sassone. PBFT vs Proof-of-Authority: Applying the cap theorem to permissioned blockchain. In *Italian Conference on Cyber Security (06/02/18)*, January 2018.

[15] P. S Anjana, S Kumari, S Peri, S Rathor, and A Somani. An efficient framework for optimistic concurrent execution of smart contracts. In *Proceedings of the 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pages 83–92. IEEE, 2019.

[16] Y Aoki, K Otsuki, T Kaneko, R Banno, and K Shudo. SimBlock: A blockchain network simulator. In *Proceedings of the 2nd Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, pages 325–329. IEEE, 2019.

[17] H. F Atlam, A Alenezi, R. J Walters, G. B Wills, and J Daniel. Developing an adaptive risk-based access control model for the Internet of Things. In *Proceedings of the 2017 IEEE International Conference on Internet of Things and IEEE Green Computing and Communications and IEEE Cyber, Physical and Social Computing and IEEE Smart Data*, pages 655–661. IEEE, 2017.

[18] N Atzei, M Bartoletti, and T Cimoli. A survey of attacks on Ethereum smart contracts (SoK). In *Proceedings of the International Conference on Principles of Security and Trust*, pages 164–186. Springer, 2017.

[19] J Banks. *Discrete-event system simulation*. Pearson Education India, 1984.

[20] J Banks. Introduction to simulation. In *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*, pages 7–13, 1999.

[21] S Bano, A Sonnino, M Al-Bassam, S Azouvi, P McCorry, S Meiklejohn, and G Danezis. SoK: Consensus in the age of blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 183–198. ACM, 2019.

[22] K Bhargavan, A Delignat-Lavaud, C Fournet, A Gollamudi, G Gonthier, N Kobeissi, N Kulatova, A Rastogi, T Sibut-Pinote, N Swamy, and S Zanella-Béguelin. Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, page 91–96. ACM, 2016.

[23] L Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[24] V Buterin. Transaction spam attack: next steps, `https://goo.gl/uKi9Ug`, 2016.

[25] V Buterin. A next-generation smart contract and decentralized application platform. *White Paper*, 3(37), 2014.

[26] V Buterin and V Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[27] H Caliper. Blockchain performance benchmarking for Hyperledger, `https://hyperledger.github.io/caliper/`.

[28] A. C Cameron and F. A Windmeijer. An R-squared measure of goodness of fit for some common nonlinear regression models. *Journal of Econometrics*, 77(2):329–342, 1997.

[29] T Chen, X Li, Y Wang, J Chen, Z Li, X Luo, M. H Au, and X Zhang. An adaptive gas cost mechanism for ethereum to defend against under-priced dos attacks. In *Proceedings of the International Conference on Information Security Practice and Experience*, pages 3–24. Springer, 2017.

[30] K Christidis and M Devetsikiotis. Blockchains and smart contracts for the Internet of Things. *Ieee Access*, 4:2292–2303, 2016.

[31] I Coelho, V Coelho, P Lin, and E Zhang. Community yellow paper: A technical specification for NEO blockchain, `https://neoresearch.io/assets/yellowpaper/yellow_paper.pdf`, 2020.

[32] K Croman, C Decker, I Eyal, A. E Gencer, A Juels, A Kosba, A Miller, P Saxena, E Shi, E Gün Sirer, D Song, and R Wattenhofer. On scaling decentralized blockchains. In *Proceedings of the International Conference on Financial Cryptography and Data Security*. Springer, 2016.

[33] G. G Dagher, J Mohler, M Milojkovic, and P. B Marella. Ancile: Privacy-preserving framework for access control and interoperability of electronic health records using blockchain technology. *Sustainable Cities and Society*, 39:283–297, 2018.

[34] S Das, V. J Ribeiro, and A Anand. Yoda: Enabling computationally intensive contracts on blockchains with byzantine and selfish nodes. *arXiv preprint arXiv:1811.03265*, 2018.

[35] C Decker and R Wattenhofer. Information propagation in the Bitcoin network. In *Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing*, pages 1–10. IEEE, 2013.

[36] K Delmolino, M Arnett, A Kosba, A Miller, and E Shi. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *Proceedings of the International Conference on Financial Cryptography and Data Security*, pages 79–94. Springer, 2016.

[37] A. P Dempster, N. M Laird, and D. B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[38] T Dickerson, P Gazzillo, M Herlihy, and E Koskinen. Adding concurrency to smart contracts. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 303–312. ACM, 2017.

[39] T. T. A Dinh, J Wang, G Chen, R Liu, B. C Ooi, and K.-L Tan. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1085–1100. ACM, 2017.

[40] C Dong, Y Wang, A Aldweesh, P McCorry, and A van Moorsel. Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 211–227. ACM, 2017.

[41] B Ekbote, V Hire, P Mahajan, and J Sisodia. Blockchain based remittances and mining using CUDA. In *Proceedings of the 2017 International Conference On Smart Technologies For Smart Nation*, pages 908–911. IEEE, 2017.

[42] Ethereum Foundation. PyEth client, `https://github.com/ethereum/pyethapp?files=1`.

[43] I Eyal, A. E Gencer, E. G Sirer, and R Van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *Proceedings of the 13th USENIX Conference on Networked Systems Design and Implementation*, page 45–59. USENIX Association, 2016.

[44] I Eyal and E. G Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Proceedings of the International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[45] P Ezhilchelvan, A Aldweesh, and A van Moorsel. Non-blocking two phase commit using blockchain. In *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, pages 36–41. ACM, 2018.

[46] G. S Fishman. *Discrete Event Simulation: Modeling, Programming, and Analysis*. Springer Series in Operations Research. Springer-Verlag, New York, NY, 2001.

[47] Y Freund and R. E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[48] J. H Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

[49] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, `https://ethereum.github.io/yellowpaper/paper.pdf`. *Yellow Paper*, 2019.

[50] A Gervais, G. O Karame, K Wüst, V Glykantzis, H Ritzdorf, and S Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.

[51] J Göbel, H. P Keeler, A. E Krzesinski, and P. G Taylor. Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay. *Performance Evaluation*, 104:23–41, 2016.

[52] B. R Haverkort. *Performance of computer communication systems: A model-based approach*. Wiley Online Library, 1998.

[53] Y Hu, A Manzoor, P Ekparinya, M Liyanage, K Thilakarathna, G Jourjon, and A Seneviratne. A delay-tolerant payment scheme based on the Ethereum blockchain. *IEEE Access*, 7:33159–33172, 2019.

[54] T Huang, H Peng, and K Zhang. Model selection for Gaussian mixture models. *Statistica Sinica*, 27(1):147–169, 2017.

[55] R Jain. *The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, 1990.

[56] A Juels, A Kosba, and E Shi. The ring of gyges: Investigating the future of criminal smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 283–295. ACM, 2016.

[57] H Kalodner, S Goldfeder, X Chen, S. M Weinberg, and E. W Felten. Arbitrum: Scalable, private smart contracts. In *Proceedings of the 27th USENIX Conference on Security Symposium*, page 1353–1370. USENIX Association, 2018.

[58] Y Kawase and S Kasahara. Transaction-confirmation time for bitcoin: A queueing analytical approach to blockchain mechanism. In *Proceedings of the International Conference on Queueing Theory and Network Applications*, pages 75–88. Springer, 2017.

[59] M Knecht and B Stiller. Smartdemap: A smart contract deployment and management platform. In *Proceedings of the IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 159–164. Springer, 2017.

[60] R Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, page 1137–1143. Morgan Kaufmann Publishers Inc., 1995.

[61] A Kosba, A Miller, E Shi, Z Wen, and C Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Proceedings of the 2016 IEEE Symposium on Security and Privacy*, pages 839–858. IEEE, 2016.

[62] T.-T Kuo, H Zavaleta Rojas, and L Ohno-Machado. Comparison of blockchain platforms: A systematic review and healthcare examples. *Journal of the American Medical Informatics Association*, 26(5):462–478, 2019.

[63] M Kuzlu, M Pipattanasomporn, L Gurses, and S Rahman. Performance analysis of a Hyperledger Fabric blockchain framework: Throughput, latency and scalability. In *Proceedings of the 2019 IEEE International Conference on Blockchain*, pages 536–540. IEEE, 2019.

[64] J Kwon. Tendermint: Consensus without mining, `https://tendermint.com/static/docs/tendermint.pdf`. *Draft v. 0.6, fall*, 1:11, 2014.

[65] L. M Leemis and S. K Park. *Discrete-event simulation: A first course*. Pearson Prentice Hall Upper Saddle River, NJ, 2006.

[66] Y Lewenberg, Y Bachrach, Y Sompolinsky, A Zohar, and J. S Rosenschein. Bitcoin mining pools: A cooperative game theoretic analysis. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 919–927. Citeseer, 2015.

[67] A Lewis. A gentle introduction to smart contracts, https://bitsonblocks.net/2016/02/01/a-gentle-introduction-to-smart-contracts/. 2016.

[68] B. G Lindsay. Mixture models: Theory, geometry and applications. In *Proceedings of the NSF-CBMS Regional Conference Series in Probability and Statistics*, pages i–163. JSTOR, 1995.

[69] L Luu, D.-H Chu, H Olickel, P Saxena, and A Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, page 254–269. ACM, 2016.

[70] L Luu, V Narayanan, C Zheng, K Baweja, S Gilbert, and P Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.

[71] L Luu, J Teutsch, R Kulkarni, and P Saxena. Demystifying incentives in the consensus computer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 706–719. ACM, 2015.

[72] A Maria. Introduction to modeling and simulation. In *Proceedings of the 29th conference on Winter simulation*, pages 7–13, 1997.

[73] N Matloff. Introduction to discrete-event simulation and the simpy language. *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, 2(2009):1–33, 2008.

[74] A Miller. Gas economics, https://github.com/LeastAuthority/ethereum-analyses/blob/master/GasEcon.md.

[75] A Miller and R Jansen. Shadow-Bitcoin: Scalable simulation via direct execution of multi-threaded applications. In *Proceedings of the 8th USENIX Conference on Cyber Security Experimentation and Test*. USENIX Association, 2015.

[76] M Möser, K Soska, E Heilman, K Lee, H Heffan, S Srivastava, K Hogan, J Hennessey, A Miller, A Narayanan, and N Christin. An empirical analysis of traceability in the monero blockchain. *Proceedings on Privacy Enhancing Technologies*, 2018(3):143–163, 2018.

[77] S Nakamoto. Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf. *White Paper*, 2008.

[78] T Neudecker, P Andelfinger, and H Hartenstein. A simulation model for analysis of attacks on the Bitcoin peer-to-peer network. In *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management*, pages 1327–1332. IEEE, 2015.

[79] H. D Nguyen and G McLachlan. On approximations via convolution-defined mixture models. *Communications in Statistics-Theory and Methods*, 48(16):3945–3955, 2019.

[80] I Nikolić, A Kolluri, I Sergey, P Saxena, and A Hobor. Finding the greedy, prodigal, and suicidal contracts at scale. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 653–663. ACM, 2018.

[81] R O'Connor. Simplicity: A new language for blockchains. In *Proceedings of the 2017 Workshop on Programming Languages and Analysis for Security*, pages 107–120. ACM, 2017.

[82] N Papadis, S Borst, A Walid, M Grissa, and L Tassiulas. Stochastic models and wide-area network measurements for blockchain design and analysis. In *Proceedings of the 2018 IEEE Conference on Computer Communications*, pages 2546–2554. IEEE, 2018.

[83] D Perez and B Livshits. Broken metre: Attacking resource metering in EVM. *arXiv preprint arXiv:1909.07220*, 2019.

[84] K Petersen, R Feldt, S Mujtaba, and M Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, pages 1–10. BCS Learning Development Ltd., 2008.

[85] B. B. F Pontiveros, C. F Torres, and R State. Sluggish mining: Profiting from the verifier's dilemma. In *Proceedings of the International Conference on Financial Cryptography and Data Security*, pages 67–81. Springer, 2019.

[86] P Reinecke, T Krauß, and K Wolter. Cluster-based fitting of phase-type distributions to empirical data. *Computers & Mathematics with Applications*, 64(12):3840–3851, 2012.

[87] P Reinecke, T Krauß, and K Wolter. Hyperstar: Phase-type fitting made easy. In *Proceedings of the 2012 Ninth International Conference on Quantitative Evaluation of Systems*, pages 201–202. IEEE, 2012.

[88] S Robinson. *Simulation: The Practice of Model Development and Use*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2004.

[89] L Scrucca, M Fop, T. B Murphy, and A. E Raftery. Mclust 5: Clustering, classification and density estimation using gaussian finite mixture models. *The R Journal*, 8(1):289, 2016.

[90] Y Sompolinsky and A Zohar. Secure high-rate transaction processing in Bitcoin. In *Proceedings of the International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[91] L Stoykov, K Zhang, and H.-A Jacobsen. Vibes: Fast blockchain simulations for large-scale peer-to-peer networks. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos*, pages 19–20. ACM, 2017.

[92] I Sukhodolskiy and S Zapechnikov. A blockchain-based access control system for cloud storage. In *Proceedings of the 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering*, pages 1575–1578. IEEE, 2018.

[93] N Szabo. Smart contracts, `https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html`. 1994.

[94] J Teutsch and C Reitwießner. A scalable verification solution for blockchains, `https://people.cs.uchicago.edu/teutsch/papers/truebit.pdf`. *White Paper*, 2017.

[95] P Thakkar, S Nathan, and B Viswanathan. Performance benchmarking and optimizing Hyperledger Fabric blockchain platform. In *Proceedings of the 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 264–276. IEEE, 2018.

[96] A van Moorsel, A Aldweesh, M Alharby, and P Ezhilchelvan. Benchmarks and models for blockchain, `https://icpe2018.spec.org/fileadmin/user_upload/documents/icpe_2018/ICPE2018_Blockchain_Keynote_Aad_van_Moorsel.pdf`. Keynote at International Conference on Performance Engineering, 2018.

[97] W Wang, D. T Hoang, P Hu, Z Xiong, D Niyato, P Wang, Y Wen, and D. I Kim. A survey on consensus mechanisms and mining strategy management in blockchain networks. *IEEE Access*, 7:22328–22370, 2019.

[98] C. J Willmott and K Matsuura. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, 30(1):79–82, 2005.

[99] K Wüst and A Gervais. Do you need a blockchain? In *Proceedings of the 2018 Crypto Valley Conference on Blockchain Technology*, pages 45–54. IEEE, 2018.

[100] X Xu, C Pautasso, L Zhu, V Gramoli, A Ponomarev, A. B Tran, and S Chen. The blockchain as a software connector. In *Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture*, pages 182–191. IEEE, 2016.

[101] R Yang, T Murray, P Rimba, and U Parampalli. Empirically analyzing Ethereum's gas mechanism. In *Proceedings of the 2019 IEEE European Symposium on Security and Privacy Workshops*, pages 310–319. IEEE, 2019.

[102] R Yasaweerasinghelage, M Staples, and I Weber. Predicting latency of blockchain-based systems using architectural modelling and simulation. In *Proceedings of the 2017 IEEE International Conference on Software Architecture*, pages 253–256. IEEE, 2017.

[103] J Yli-Huumo, D Ko, S Choi, S Park, and K Smolander. Where is current research on blockchain technology?—a systematic review. *PLOS ONE*, 11:1–27, 2016.

[104] L Yu, W.-T Tsai, G Li, Y Yao, C Hu, and E Deng. Smart-contract execution with concurrent block building. In *Proceedings of the 2017 IEEE Symposium on Service-Oriented System Engineering*, pages 160–167. IEEE, 2017.

[105] F Zhang, E Cecchetti, K Croman, A Juels, and E Shi. Town Crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 270–282. ACM, 2016.

[106] Z Zheng, S Xie, H Dai, X Chen, and H Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *Proceedings of the 2017 IEEE International Congress on Big Data*, pages 557–564. IEEE, 2017.

[107] X Zhuang, Y Huang, K Palaniappan, and Y Zhao. Gaussian mixture density modeling, decomposition, and applications. *IEEE Transactions on Image Processing*, 5(9):1293–1302, 1996.