Blockchain Testing: Tools, Techniques, and Considerations

Jackson Wonderly, Ayaan M. Kazerouni

1 Introduction

Since the release of the cryptocurrency *Bitcoin* in 2008 [28], the technology behind Bitcoin, *blockchain*, has received a large amount of attention. In this paper we will explain what blockchain and cryptocurrency are, and how they work. We will also give an introduction to *Ethereum*, a flexible blockchain implementation that enables users to create their own *decentralized applications* through the use of *smart contracts*. Finally, we will discuss the current functional and performance testing practices of Ethereum, and blockchain in general.

2 What is Blockchain?

A blockchain is a *shared*, *distributed* ledger that records transactions of assets between two parties. It is made up of *blocks* that typically contain three key items:

- Transaction data
- A timestamp
- A cryptographic hash of the previous block

Each block is therefore securely linked to the previous block, effectively forming a *chain* of blocks, thus the name *blockchain*. Assets on a blockchain can be tangible, intangible, or digital – for example land, insurance claims, or cryptocurrency such Bitcoin[1] or Ether[2] (see 3).

Blockchain is facilitated by a peer-to-peer network made up of its users' machines. With every transaction, the ledger is replicated and stored on each participant's machine, making it *transparent* and easily *verifiable*. Moreover, since the ledger is stored on several machines, there is no central authority or entity that stakeholders depend on. This means there is no central system to attack, making it relatively resilient to attacks or faults.

Since there is no centralized system (like a server), there must be a way for participants themselves to verify and commit transactions to the ledger. Blockchain allows for a "pluggable consensus mechanism" [22] by which the peer-to-peer network agrees on what the blockchain should look like at any given time.

[1] https://bitcoin.org/en/

[2] https://www.ethereum.org/ether

A suitable consensus mechanism needs to provide Byzantine Fault Tolerance (see [25]). The two most popular consensus mechanisms are described below:

- **Proof-of-work** [28]: Every machine on the blockchain is asked to solve a puzzle based on its version of the shared ledger, and machines that store identical versions "team up". The first team to solve the puzzle wins, and all machines update their versions to match the winning team's. The idea is that the most computing power wins, and a majority of the participating machines will have the correct version of the ledger. Ethereum's current implementation (Homestead) implements proof-of-work consensus [6].
- **Proof-of-stake**: *Validators* deposit a stake and take turns proposing and voting on the next block to be included in the blockchain, with the weight of the validator's vote depending on the size of their stake. This method has advantages over proof-of-work such as reduced centralization risks, reduced power consumption, and reduced incentive to attack. Ethereum's upcoming Casper implementation will implement proof-of-stake consensus [32].

2.1 Incentivizing Consensus

The term *mining*, in the context of proof-of-work blockchains, refers to the process of nodes or *miners* in the blockchain network receiving, propagating, verifying, and executing transactions. This includes grouping together transactions into a block and generating a valid proof-of-work for that block. In proof-of-stake blockchains, those who participate in the consensus process are called *validators*. Given that the process of mining and validating requires the miner to dedicate their own resources, miners and validators are generally given incentive by rewarding them with some amount of *cryptocurrency* when they create a new block[10].

3 Cryptocurrency: An Application of Blockchain

Cryptocurrency is type of digital asset that parties may transfer to one another or even create new units of, depending on the currency in question. Blockchain forms the underlying technology upon which cryptocurrency is based. Indeed, the first application of blockchain was the cryptocurrency Bitcoin [28]. Cryptocurrency systems that use proof-of-work consensus rely on a process called *mining* to validate transactions. Cryptocurrency systems that use proof-of-stake consensus rely on a process called *forging*. Since blockchains run on a peer-to-peer to network, users must be given some incentive to participate on the network. Therefore, successful miners are given some amount of the cryptocurrency as a reward, giving them incentive to contribute to the processing power of the network. The consensus mechanisms described in Section 2 are methods of mining cryptocurrency. Mining is the method by which new units of cryptocurrency are created. On the Ethereum blockchain, successful miners are rewarded in *Ether*.

In a typical cryptocurrency system, a user (or users) own a *cryptocurrency wallet*, which contains a private key and a public key. Wallets may be stored in software, on hardware, or even mentally (by remembering private keys or mechanisms to arrive at private keys). Users may use the private key to write transactions to the public ledger ("spend" their cryptocurrency), and their public key allows them to receive currency. The public key acts as a kind of address that other participants can send currency to. All transactions are publicly available on the blockchain, which is a requirement for consensus to be reached.

Cryptocurrencies have skyrocketed in popularity, and as of March 13, 2018 have a total Market Cap of about $350B USD [3].

4 Ethereum: An Implementation of Blockchain

Ethereum[3] is an open-source blockchain platform that includes a Turing-complete programming language. The inclusion of a Turing-complete programming language allows a huge amount of flexibility, allowing developers to create their own decentralized applications through the use of *smart contracts*. Ethereum also has a built-in cryptocurrency, named *Ether*.

4.1 Smart Contracts

The concept of a smart contract was first described by computer scientist Nick Szabo as "a set of promises, specified in digital form, including protocols within which the parties perform on these promises" [31]. Szabo compares a smart contract to a vending machine, where there is a sort of contract between the customer who deposits money, and the autonomous vending machine which receives the money and dispenses change and product. The Ethereum White Paper describes smart contracts as "cryptographic 'boxes' that contain value and only unlock it if certain conditions are met" [7].

In Ethereum, smart contracts are pieces of code that live on the blockchain and have their own address. When a transaction that specifies a smart contract address as the recipient is added to the blockchain, the smart contract is executed. It is also possible for a smart contract to call another smart contract. Smart contracts are also stateful, meaning that they have their own private storage which persists across transactions. Anyone can create their own smart contract and add it the blockchain, but once a smart contract has been accepted into the blockchain it cannot be modified.

4.2 The Ethereum Virtual Machine

Ethereum also comes with the Ethereum Virtual Machine (EVM), which is a sandboxed runtime environment in which all smart contracts are executed. Code running inside the EVM has no

[3] https://www.ethereum.org/

access to the network, filesystem, or other processes. Ethereum smart contracts are generally written in a higher level language which is then compiled to EVM bytecode before being stored on the blockchain. In order to facilitate writing smart contracts, Ethereum provides *Solidity* [4], which is described as a "contract-oriented programming language" and which compiles to EVM bytecode.

4.3 Gas

Given that smart contracts in Ethereum are written in a Turing-complete programming language, one issue that must be addressed is the halting problem. The halting problem says that it is not possible to determine whether an arbitrary program will ever halt, or run forever. This is an issue for Ethereum since a malicious (or incompetent) programmer may write a smart contract that runs forever, wasting the computing resources of nodes in the Ethereum network.

Ethereum addresses the halting problem, and the fact that different transactions may require differing amount of resources to process, by introducing the concept of *gas* [5]. Each transaction must specify a gas allowance, which represents the maximum number of steps that the transaction may take. Each transaction must also specify a gas price, which is the amount of Ether it is willing to pay per unit of gas. Every operation taking during a transaction, including computations and memory operations, consumes gas. If a transaction completes with leftover gas, the sender of the transaction will be refunded for the unused gas. If a transaction runs out of gas before it completes, its execution will be reverted and the sender will not be refunded.

The gas allowance of a transaction applies to *all* operations taken as part of that transaction, including any operations taken by any sub-executions. This means that if some transaction executes smart contract A, and A calls smart contract B, the gas allowance specified by the original transaction will be consumed by the executions of both A and B.

4.4 Consensus Mechanism

Ethereum currently uses a proof-of-work consensus mechanism. One problem that the developers sought to solve when designing Ethereum's consensus mechanism was the issue of centralization. One issue the Bitcoin network is facing is that there are a small number of groups conduct a disproportionate amount of the mining capacity due to the use of application specific integrated circuits, or *ASICs* [30]. These ASICs are designed specifically for mining Bitcoin and able to vastly out-perform commodity hardware. Given that Bitcoin mining is a competitive endeavor where the first node to provide a valid proof-of-work wins, the use of ASICs makes Bitcoin mining largely futile for those using general purpose computers. This centralization is an issue because it can lead to a *51% attack* where a party controlling a large percentage of the

[4] https://github.com/ethereum/solidity

network's computing power can revise transaction history and prevent new transactions from confirming [1].

In order to prevent the centralization problem Bitcoin is facing, the developers of Ethereum designed Ethash, a proof-of-work algorithm that is designed to be ASIC resistant. Ethash achieves this resistance by being memory/IO bound, rather than CPU bound like the Bitcoin proof-of-work algorithm. The argument behind this approach is that "commodity RAM, especially in GPUs, is much closer to the theoretical optimum than commodity computing capacity"[4],

However, there are still a number of issues with proof-of-work, the main one being that users' machines use a large amount of electricity solving proof-of-work puzzles [12]. As a result, Ethereum is planning to move to a proof-of-stake algorithm named *Casper*[32].

4.5 Decentralized Applications

Decentralized applications, referred to within the Ethereum community as "DApps", are services that enable direct interaction between users and providers, e.g. connecting buyers and sellers in a marketplace. DApps typically have a user interface that is built using common web technologies (HTML/JavaScript) and communicate with the Ethereum blockchain using a JavaScript API [6]. DApps also typically have their own set of smart contracts living on the blockchain which are used to encode the application's business logic, as well as provide persistent storage.

Examples of decentralized applications include WeiFund[5] and Eth-Tweet[6]. WeiFund is an application that utilizes smart contracts to conduct crowdfunding campaigns. Eth-Tweet is a decentralized social network similar to Twitter. One feature of Eth-Tweet is that since each post is stored on the blockchain, posts can never be modified or removed, thus preventing censorship.

5 Testing Blockchain Systems and Applications

5.1 Functional testing

With any new software engineering paradigm comes the need for an accompanying shift in testing methodologies. Platforms like Ethereum or Hyperledger Fabric [20] allow developers to develop arbitrary decentralized applications, that will most likely involve some kind of smart contract mechanism (called *chaincode* in Fabric). This creates the need for appropriate *functional testing* tools and techniques: for example, a developer might need to test a smart contract for *correctness* before using it to submit transactions to the blockchain, and they would need to test *the verification of the blockchain*, in order to correctly achieve consensus.

[5] http://weifund.io/

[6] https://github.com/yep/eth-tweet

5.1.1 Client Testing

Ethereum has a repository containing common test cases for all Ethereum implementations [7]. This repository is utilized by developers of Ethereum clients, such as go-ethereum and cpp-ethereum, to help ensure the correctness of their implementations. Each test case in the repository is defined by a JSON object whose structure varies depending on what is being tested. For example, for a test case designed to test the basic verification of the blockchain the object would specify a genesis block (the first block in the blockchain), a set of blocks and transactions to be added to the blockchain, the state of accounts on the network before running the test, the expected state of accounts after running the test, as well as the version(s) of the Ethereum protocol for which the test is applicable. Some of these tests are created using *testeth* [8], which is a tool for generating consensus tests. These tests are each run on their own local, private blockchain. In addition, each client may have its own suite of tests that are more specific to its particular implementation.

5.1.2 Smart Contract and DApp Testing

For developers of decentralized applications, there are a number of tools and frameworks to aid in the development and testing of smart contracts. The Populous framework [9] includes utilities that let developers test their smart contracts against an in-memory Ethereum blockchain that is reset at the beginning of each test run. The most popular development framework for Ethereum, *Truffle* [10], includes *Ganache*, which is a "personal blockchain" that can be used by developers to test their decentralized applications [14]. Ganache includes features that allow the developer to monitor, explore, and configure their personal blockchain. Truffle also includes a testing framework that lets developers write automated tests for their smart contracts. Tests in the Truffle framework can be written in either JavaScript or Solidity. Developers can measure the level of code coverage achieved by the tests they write for their smart contracts using *solidity-coverage* [2].

Developers can also use the official Ethereum test network (named *Ropsten*), or set up their own private local Ethereum network for testing purposes [13]. Developers can acquire Ether for testing their smart contracts on the official test network by using a *faucet* [11]. A faucet is simply a service that sends a small amount of Ether to whoever requests it.

[7] https://github.com/ethereum/tests
[8] https://github.com/ethereum/testeth
[9] https://github.com/ethereum/populus
[10] https://github.com/trufflesuite/truffle
[11] An example of a faucet is http://faucet.ropsten.be:3001/

5.1.3 Formal Verification of Smart Contracts

After several high-profile attacks exploiting vulnerabilities in Ethereum smart contracts [16], which resulted in millions of dollars worth of Ether being stolen or frozen, there has been a significant amount of interest in formal verification methods for smart contracts [17, 23, 15]. Some tools that have been created include:

- *Manticore*, a symbolic execution tool for analyzing binaries and smart contracts [12].
- *Oyente*, a symbolic execution tool for detecting bugs in Ethereum smart contracts [26].
- *Mythril*, a security analysis tool for Ethereum smart contracts that uses symbolic execution and constraint solving [27].

5.2 System Testing

In the past, the Ethereum developers had plans to perform system testing by deploying Ethereum clients inside Docker containers to Amazon EC2 instances, with each client sending logs to a central server. However, the Ethereum system-testing repository [13] has not seen any activity in almost three years. Planned testing scenarios included:

- Testing that clients can connect to each other
- Testing various attacks, such as DDOS and Sybil attacks
- Testing network latency and limited bandwidth conditions

No other documentation could be found on if/how the Ethereum developers conduct system testing.

5.3 Performance Testing

The distributed nature of blockchains – particularly the fact that they run on individual clients without a server – makes performance testing important. Indeed, performance bottlenecks are a major hindrance to widespread adoption of blockchain technology. Bitcoin and Ethereum, the two most popular blockchain implementations, both suffer from performance bottlenecks, preventing them from moving past 3.3-7 transactions per second (TPS) and approximately 15-20 TPS, respectively. As a point of comparison, VISA supports about 45,000 TPS [14]. To effectively scale up and make optimizations where necessary, blockchain developers need to be able to test and measure their systems.

Developers of blockchain networks like Bitcoin, Ethereum, or Hyperledger Fabric scale to achieve widespread adoption. A significant bottleneck in Ethereum is the need for all nodes to process every single transaction. This hamstrings the network's performance, limiting it to the

[12] https://github.com/trailofbits/manticore

[13] https://github.com/ethereum/system-testing

[14] https://www.coindesk.com/information/will-ethereum-scale/

processing capacity of individual nodes on the network. A possible method of overcoming this obstacle would be to increase the block size limit, meaning that block-verification would take place less often (since new blocks would get committed less frequently). However, this would heavily favor supercomputers, and lead to centralization. Therefore, it is not feasible.

There are a few other strategies under investigation. We discuss them below, generally focusing on proposals from Vitalik Buterin's [15] Ethereum Research Group.

5.3.1 Current Limitations and Proposed Solutions

- **Sharding.** An active area of research and development is 'sharding' [19]. The basic idea is that the blockchain is broken into *shards*, where shards are made of chains of *collations*, similar to a traditional blockchain. The *main chain* then contains only the collation headers from all shards, and individual nodes need only process collations in their shards. This effectively parallelizes transaction processing.
- **Layer 2 Protocols.** These refer to techniques to perform operations "off-chain", but that still rely on the main blockchain for security. The main benefit is that the main chain *only holds the information it absolutely needs to*, so individual nodes have to do less work. However, it is difficult to maintain this separation and a trustworthy level of decentralization and security. Some examples are:

  1. *State channels.* [24] If participants need to exchange several state updates over a long period of time, before arriving at a final state, it seems wasteful to have the entire blockchain process each state update as a transaction. State channels allow participants to send state updates *directly to each other*, until the final update is sent to a contract on the blockchain to finalize the transaction. At any time during update exchanges, either participant can verify the state with the main chain, if necessary.
  2. *Plasma.* [18, 29] The key idea behind *Plasma* is the creation of 'child' blockchains attached to the main chain. The child chains can spawn their own children and so on. In this way, interactions (possibly entire DApps) could be run at the child-chain level, without the need for the entire blockchain to process every operation. At any time during execution of the DApp, users could exit the child chain and withdraw their assets to the main chain. An example use case is the Ethereum main network, that is capable of hosting multiple DApps powered by ether. Using Plasma, these DApps could be run on their own child chains whose roots are connected to the main chain. Assets (trading cards, CryptoKitties, etc.) are initially created on the *main chain*, and then moved to the child chain. Then users only interact with the child chain while using the DApp, and DApp-specific logic and smart contacts reside on the child chain. This way, DApps are not wasting processing power to process transactions from other DApps.

[15] A co-creator of Ethereum.

5.3.2 Measuring Blockchain Performance

Currently, there are no widely established benchmarks, metrics, or repeatable procedures for testing the performance of blockchain networks. Frameworks for setting up blockchain networks (like Ethereum or Hyperledger Fabric) do provide tools for functional testing and benchmarking the performance of individual clients, but not the network as a whole.

This lack of standardized methods and metrics has been acknowledged by the blockchain community at large [8, 11]. Hyperledger [16] is an open-source collaborative initiative for the advancement of blockchain technology in industry. The initiative includes multiple projects and frameworks for setting up blockchain networks for different purposes, writing and interpreting smart contracts, and incorporating these systems into industry. A notable project is **Hyperledger Caliper**, a tool for benchmarking multiple blockchain implementations. The Caliper project acknowledges the lack of reproducible or repeatable techniques for benchmarking implementations, and aims to be able to profile blockchain systems in terms of throughput (TPS), latency, resource utilization, etc. Caliper is still in its infancy.

Additionally, Hyperledger has founded a **Performance and Scalability Working Group** (PSWG) whose goal is to standardize metrics and techniques for evaluating blockchain performance. Some performance metrics under proposed by the working group are [9]:

- Blockchain work = transaction throughput * network size
- Transaction throughput = total transactions committed / total time
- Transaction latency = confirmation time - submitting time
- Average transaction latency = $\sum$ (all latencies) / total committed transactions
- Read latency
- Read throughput

Blockbench [21] is a suite of smart contract benchmarks developed to test the performance of Ethereum, Parity, and Hyperledger Fabric. Items measured are:

- Throughput
- Latency
- Fault tolerance
- Scalability

6 Conclusion

Blockchain technology, while being very popular, is also still very new. The benefits of blockchain technology include transparency, verifiability, immutability, as well as the minimization of the amount of trust needed between peers in the network.

[16] www.hyperledger.org/about

Another purported benefit of blockchain technology is increased security due to its distributed nature. However, with blockchain platforms such as Ethereum or Hyperledger Fabric, smart contracts introduce their own vulnerabilities. There are a number of existing tools and frameworks to aid developers in developing and testing their Ethereum smart contracts, and there is active research into the formal verification of smart contracts.

Blockchain performance testing is also still in its infancy. The current generation of blockchain networks have various performance and scalability issues, but performance testing efforts to-date have been fairly ad-hoc. The only current initiative to develop standard benchmarks and performance indicators for blockchain networks is Hyperledger Caliper, which is still in the early stages of development.

References

[1] 51% attack. https://learncryptography.com/cryptocurrency/51-attack. Accessed: 03/13/2018.

[2] Code coverage for solidity. https://blog.colony.io/code-coverage-for-solidity-eecfa88668c2. Accessed: 04/30/2018.

[3] Cryptocurrency market capitalizations. https://coinmarketcap.com/charts/. Accessed: 03/13/2018.

[4] Ethash design rationale. https://github.com/ethereum/wiki/blob/master/Ethash-Design-Rationale.md. Accessed: 03/13/2018.

[5] Ethereum design rationale. https://github.com/ethereum/wiki/blob/master/pages/design-rationale/%5Benglish%5D-design-rationale.md. Accessed 03/13/2018.

[6] Ethereum homestead documentation. http://ethdocs.org. Accessed: 03/13/2018.

[7] Ethereum white paper. https://github.com/ethereum/wiki/wiki/White-Paper. Accessed: 03/13/2018.

[8] Measuring blockchain performance with hyperledger caliper. https://www.hyperledger.org/blog/2018/03/19/measuring-blockchain-performance-with-hyperledger-caliper. Accessed: 04/30/2018.

[9] Metric definition proposal. https://docs.google.com/document/d/1DQ6PqoeIH0pCNJSEYiw7JVbExDvWh_ZRVhWkuioG4k0/edit?usp=sharing. Accessed: 04/30/2018.

[10] Mining. https://github.com/ethereum/wiki/blob/master/Mining.md. Accessed: 03/13/2018.

[11] Performance testing blockchain. https://transactiq.io/blog/performance-testing-blockchain/. Accessed: 04/30/2018.

[12] Proof of stake faq. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ. Accessed: 03/13/2018.

[13] Test networks - ethereum homestead documentation. http://ethdocs.org/en/latest/network/test-networks.html. Accessed: 04/27/2018.

[14] Working with ganache. truffleframework.com/docs/ganache/using. Accessed: 04/27/2018.

[15] Sidney Amani, Myriam Bégel, Maksym Bortin, and Mark Staples. Towards verifying ethereum smart contract bytecode in Isabelle/HOL. *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs - CPP 2018*, 61(i):66–77, 2018.

[16] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on Ethereum smart contracts (SoK). In *Proceedings of the 6th International Conference on Principles of Security and Trust*, volume 10204, pages 164–186, 2017.

[17] Karthikeyan Bhargavan, Nikhil Swamy, Santiago Zanella-Béguelin, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, and Thomas Sibut-Pinote. Formal Verification of Smart Contracts. *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security - PLAS'16*, pages 91–96, 2016.

[18] Vitalik Buterin. Minimal viable plasma. https://ethresear.ch/t/minimal-viable-plasma/426. Accessed: 04/30/2018.

[19] Vitalik Buterin. Ethereum scalability research and development subsidy programs. https://blog.ethereum.org/2018/01/02/ethereum-scalability-research-development-subsidy-programs/, 2018. Accessed 04/27/2018.

[20] Christian Cachin. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[21] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee 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.

[22] M Gupta. Blockchain for dummies, 2017.

[23] Everett Hildenbrandt, Manasvi Saxena, and Xiaoran Zhu. KEVM: A Complete Semantics of the Ethereum Virtual Machine. pages 1–33, 2017.

[24] Liam Horne and Jeff Coleman. Generalized state channels on ethereum, October 2013. Accessed: 04/29/2018.

[25] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[26] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making Smart Contracts Smarter. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, pages 254–269, 2016.

[27] Bernhard Mueller. Smashing Ethereum Smart Contracts for Fun and Real Profit. In *HITBSecConf 2018*, 2018.

[28] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. Accessed: 03/13/2018.

[29] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, 2017.

[30] J Roberts. Does bitcoin have a mining monopoly problem? `http://fortune.com/2017/08/25/bitcoin-mining/`. Accessed: 03/13/2018.

[31] N Szabo. Smart contracts: Building blocks for digital markets, 1996.

[32] V Zamfir. Introducing casper "the friendly ghost". `https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/`, 2015. Accessed: 03/13/2018.