Department of Informatics
Technical University of Munich

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

MASTER'S THESIS IN INFORMATICS

A Cross-Protocol Blockchain Benchmark System

Andrei Lebedev

Technical University of Munich

Department of Informatics

Master's Thesis in Informatics

A Cross-Protocol Blockchain Benchmark System

Ein protokollübergreifendes Blockchain-Benchmark-System

Author:	Andrei Lebedev
Supervisor:	Prof. Dr.-Ing. Georg Carle
Advisor:	Dr. Vincent Gramoli, Dr. Holger Kinkelin, Filip Rezabek, M. Sc.
Date:	July 15, 2022

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, July 15, 2022

Location, Date

Signature

Acknowledgements

I would like to thank my advisors, Dr. Holger Kinkelin and Filip Rezabek from the Technical University of Munich, and Prof. Vincent Gramoli and Dr. Gauthier Voron from the Swiss Federal Institute of Technology Lausanne. They were always available if I had any questions, provided continuous feedback, took part in the project, and taught me how to structure the work efficiently.

I would also like to thank Prof. Dr.-Ing. Georg Carle, for making this collaboration work possible. It would not have been possible to carry out this project without the team and the resources at the Chair of Network Architectures and Services.

Finally, I am grateful to my parents and friends, who were supportive throughout the whole duration of my studies.

Abstract

With the recent advent of blockchains, we have witnessed a plethora of blockchain proposals. These proposals range from using work to using time, storage, or stake to achieve consensus about which block is appended to the chain. As a drawback, it makes it difficult for the application developer to choose the suitable blockchain to support their applications. Even though many publicly available data about blockchain protocols, scalability, and performance are available, detailed baseline measurements that compare them in a fixed setup are missing. The publicly available results cannot be easily compared, as the setups, the number of nodes, locations, and load are not unified. As the setup conditions vary and, in some cases, lack details, it makes these results irreproducible and hard to compare.

In this thesis, we extend the prototype blockchain benchmark tool Diablo, which so far only supported Ethererum and uses a master-worker architecture. The improved Diablo-v2 blockchain benchmark tool generalizes Diablo's functionality to offer compatibility with other blockchain protocols. To achieve this goal, we analyze six state-of-the-art blockchain protocols and design an extensible interface for our tool based on this analysis.

Second, we create Minion, a deployment and orchestration tool for Diablo-v2. Minion provides the facilities to deploy the required software, setup and configure a blockchain network, run the benchmark, and collect the results. Minion can use machines from Amazon Web Services and was extended to support the local testbed at the Chair of Network Architectures and Services.

Finally, we evaluate the blockchain protocols on Amazon Web Services and the local testbed. We use AWS to make extensive measurements with up to 200 geographically distributed machines. Even though the local testbed cannot reach such scale, properly designed experiments can assess the scalability aspects, and in addition controlled environment helps with assessing bottlenecks in a higher level of detail.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

With the growing adoption of blockchain technology, the number of readily-available solutions has multiplied dramatically. As of March 2021, approximately five thousand distinct cryptocurrencies have been reported on a single website [1]. Each of these implementations aims at offering improvements through distinctive features, focused on the performance and application to various use-cases. Although a number of these variants could, in theory, be running on multiple instances of the same blockchain, they are often packaged as their own standalone implementation with distinctive features. A recent survey [2] highlights the breadth of the blockchain landscape through a classification of blockchains, listing eight different protocols to select nodes that are tasked with proposing blocks, 13 different consensus protocols, and 9 data structures to store transaction information. This diversity illustrates a probably small subset of all blockchain implementations that exist today.

This plethora of blockchain proposals raises the question of which proposal is the ideal blockchain for a particular application. Unfortunately, most of these proposals are not reported in scientific publications. They are at best presented in the form of white papers that present a 10-000-foot-view of their implementation details. As an example, the Ethereum yellow paper [3] presents the technicalities of the Ethereum Virtual Machine but does not explain how Ethereum participants can reach consensus on a unique block at a given index of the chain. In order to analyze the underlying protocols of such blockchains, researchers typically had to look at the available source code before being able to reason about the correctness of the protocols [4].

Another approach is for researchers to evaluate blockchains as black boxes by generating workloads and measuring their performance. Following this approach, many announce-

ments were made online about the performance of a specific blockchain. As an example, Avalanche was recently claimed to achieve 4500 TPS with a 2 second latency on its official website [5]. However, the environmental settings are not communicated. This could be confusing, especially given that a technical report or white paper presented a peak throughput at 1300 TPS [6].

There have been some thorough scientific publications about blockchain performance [7]–[12]. These usually provide enough details to make the results reproducible. The detailed environmental settings give the reader the ability to get similar performance by re-running the experiments. Most of these evaluations [7]–[12], however, evaluate few blockchains on synthetic workloads that are not representative of realistic use-cases. With the growing amount of use-cases and the ability of modern blockchains to run Decentralized Applications (DApps), one can evaluate blockchain when running real applications under an existing workload trace.

1.1 Previous Work

We improve and generalize Diablo benchmark framework [13] prototype, which supports Ethereum [3], and uses a master-worker architecture. We describe Diablo in detail in Section 2.1. For protocol evaluation, in addition to synthetic token transfer workloads, we use five realistic smart contracts developed by Aymeric Bacuet [14] to recreate real-work workload scenarios. We explain how each smart contract was derived in Section 2.2.

1.2 Problem Statement

The authors of different blockchain protocols claim impressive performance. These results are usually obtained in isolation and are often non-reproducible, which makes them hard to compare and verify. Existing blockchain benchmark frameworks are either specifically focused on testing a single protocol or do not support or provide workloads that reflect the real-world usage of the blockchain system. Furthermore, we are not aware of the framework which came with a generic solution to automatically execute the experiments in the cloud and local environments.

In this thesis, we aim to create a blockchain protocol benchmarking framework, which would be easy to extend by developers to support different protocols, covering the API differences. As a result of extensibility, we need to come up with a design of a generic workload specification, which would allow us to run the same experiment regardless of the underlying protocol. Then, we need the results of the experiments to be comparable, and therefore there should be a list of metrics that are relevant across different blockchain

implementations. As cloud compute environments and local testbeds have their own advantages and drawbacks, we want to create a tool that will allow us to easily deploy the network in the cloud or on-premise infrastructures and automate the experiment execution and the collection of the results.

1.3 Goals and Research Questions

Based on the problem statement, we define two research goals and the questions that help us to achieve the goals

G1 Create an extensible blockchain benchmark framework which can run in the cloud and local environments

Q1 Which architecture would cover the protocol differences while being easy to use and maintainable?
In Chapter 4, we describe the limitations of existing architecture and client interface, and derive improvements and generalizations in Chapter 5.

Q2 How to distribute and scale the transaction load in the framework?
We explain the limitations of Diablo workload definition syntax and explain the cases it does not cover in Chapter 4, and propose a new syntax in Chapter 6.

Q3 What metrics are relevant across different blockchain protocols?
Chapter 7 provides an evaluation based on different metrics, such as throughput, latency, and the ratio of committed transactions to submitted transactions.

Q4 What are the differences between the local and the globally distributed test environments?
We first analyze the differences between Amazon Web Services and iLab testbed in Chapter 4. Later, we look at the empirical findings in Section 7.

G2 Use the new framework to create initial measurements

1.4 Methodology and Structure

In Chapter 2, we describe the architecture of Diablo prototype, which serves as the basis of the benchmarking framework that we propose in this thesis. We describe the smart contracts that we used for the experiments that simulate the real world workload. We provide an overview of the 6 blockchain protocols we used to derive a generic and extensible architecure.

Chapter 3 gives an overview of different blockchain benchmark framework proposed and developed previously, and lists their limitations.

We conduct an analysis of Diablo prototype and its architectural limitations in Chapter 4. We also look at the differences of AWS and iLab testbed, their pros and cons as an experimental environment. We note which aspects of the blockchain protocols can be better tested with the suitable environment.

In Chapter 5, we derive the design of the new Diablo-v2 benchmarking framework and the orchestration and deployment tool Minion. We use the study of different blockchain protocols and the analysis of the original Diablo prototype as the foundation for the new generic framework. For Minion, we consider the analysis and characteristics of AWS and iLab environments.

We explain the implementation details of Diablo-v2 and Minion in Chapter 6. We give the details on the specification file formats used in Diablo-v2.

Chapter 7 describes the experimental evaluation of 6 blockchain protocols in AWS and iLab testbed environments. We assess the scalability and robustness in geographically distributed realistic and fully controlled local environments. We explain the challenges in comparing the results from different testbeds and the deployment specifics which might affect the observations.

Finally, we conclude our work in Chapter 8, where we provide an overview of the work done, its outcomes and our observations, and propose future work.

1.5 Contributions

The main contributions of the thesis are:

1. We propose Diablo-v2 (DIstributed Analytical BLOckchain benchmark framework), written in 10,083 lines of Go code, that allows developers to evaluate their blockchain with real applications. We derive a generic and extensible architecture and client interface by analyzing 6 state-of-the-art blockchains, including Algorand [15], Avalanche [6], Ethereum [3], Diem [16], Quorum [17] and Solana [18].
2. We provide an orchestration and deployment tool called Minion, which allows to setup blockchain networks and conduct the experiments in an automated fashion. We make sure that the developers can use public environments, as well as local testbeds by implementing support for Amazon Web Services and Plain Orchestrating Service [19].

3. We evaluate the different aspects of the 6 mentioned blockchain protocols, such as scalability and robustness in geographically distributed setting with up to 200 AWS machines, as well as local iLab testbed with full control over the network environment.

1.6 Collaboration acknowledgements

Chris Natoli is the author of the original Diablo prototype. Diablo-v2 and Minion were developed together with Gauthier Voron. Aymeric Bacuet developed the smart contracts used for the experiments. Vincent Gramoli took part in the experiment design, analysis of the results, and writing the paper [20].

Chapter 2

Background

This chapter gives an overview of the original Diablo framework prototype, its architecture, smart contracts used for experimental evaluation, and blockchain protocols which serve as the foundation of the new Diablo client interface and architecture.

2.1 Diablo Benchmark Framework

Diablo (DIstributed Analytical BLOckchain benchmark framework) [13] is a prototype benchmark framework for blockchain protocols initially developed by Chris Natoli in 2021. It serves as the basis of this thesis and is extended and generalized as explained in Section 1.3. It is important to note that the original prototype did not come with any generic deployment scripts, which allowed us to simply conduct the experiments in different environments. The experiments in the technical report were semi-manually executed on an OpenStack cluster. It allows comparing the protocols in the same environment, using configurable workloads with different transaction types, such as native transfers or smart contract invocations.

The architecture and components of Diablo are displayed in Figure 2.1. Diablo has master-worker architecture, where *Primary* acts as an orchestrator and result aggregator, and *Secondaries* produce the workload and collect results for individual transactions.

The main task of the *Primary* is to generate the workload, distribute it over the secondaries, and send commands to secondaries to start the benchmark and collect the results when they become available. The Workload Generator is an interface to be specialized for each blockchain protocol, which creates a transaction in a protocol-specific format. Listing 2.1 shows some of the interface methods. All the interface methods operate on

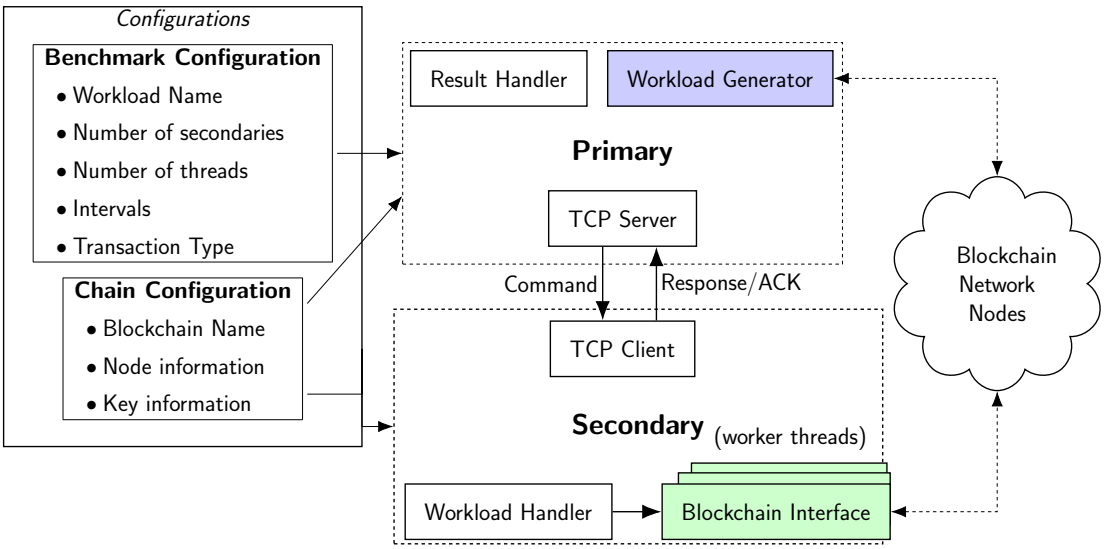


Figure 2.1: The architecture of Diablo

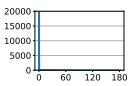
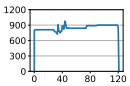
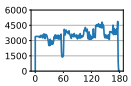
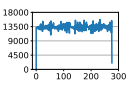
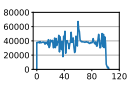
DApp	Exchange	Mobility service	Web service	Gaming	Video sharing
Workload					
Source trace	Nasdaq	Uber	Fifa	Dota 2	YouTube
Characteristics	Burst	Compute intensive	Contended	High sending rate	Very high sending rate

Table 2.1: Decentralized applications and their associated workload

byte slices in order to be generic, and the methods perform protocol-specific encoding and decoding.

2.2 The Decentralized Applications Suite

The five default decentralized applications (DApps) contributed by Aymeric Bacuet [14] are used to measure the performance of blockchains in a realistic setting. In this thesis, we use them together with Diablo-v2 that we develop by extending the original Diablo prototype. As summarized in Table 2.1, each of these DApps illustrates a distinct behavior and runs a workload trace taken from a real centralized application. Each workload figure shows the number of submitted transactions per second (TPS on the y-axis) for the duration of the run (seconds on the x-axis). For the sake of compatibility with all blockchains, DApps are implemented in both the Solidity v0.7.5 (language supported originally by Ethereum but also by Avalanche, Quorum, and Solana), the PyTeal v5 (Python language binding for Algorand smart contracts), and the Move v3 (language for Diem smart contracts) programming languages.

```
1  // Workload definitions for ease of use: [secondary][worker
       ][time][txlist][txbytes]
2  type Workload [][][][][]byte
3
4  type WorkloadGenerator interface {
5          // CreateContractDeployTX creates the raw signed
               transaction that will deploy a contract
6          CreateContractDeployTX(fromPrivKey []byte,
               contractPath string) ([]byte, error)
7
8          // CreateInteractionTX create a signed transaction
               that performs actions on a smart contract at the
               given address
9          CreateInteractionTX(fromPrivKey []byte,
               contractAddress string, functionName string,
               contractParams []configs.ContractParam, value
               string) ([]byte, error)
10
11         // CreateSignedTransaction creates a transaction
               that is signed and ready to send from the given
               private key.
12         CreateSignedTransaction(fromPrivKey []byte,
               toAddress string, value *big.Int, data []byte)
               ([]byte, error)
13
14         // GenerateWorkload generates the workload specified
                in the chain configurations.
15         GenerateWorkload() (Workload, error)
16 }
```

Listing 2.1: WorkloadGenerator interface

Exchange DApp / Nasdaq

Exchange DApp is designed as a decentralized exchange (DEX) with a workload trace taken from the National Association of Securities Dealers Automated Quotations Stock Market (Nasdaq). The Nasdaq experiences a boom of trades at its opening at 9 AM Eastern Time Zone. The number of trades for Google (GOOGL), Apple (AAPL), Facebook (FB), Amazon (AMZN), and Microsoft (MSFT) were extracted from the official website [21]. These workloads proceed in a burst by experiencing an initial demand of about 800 TPS for Google, 1300 TPS for Amazon, 3000 TPS for Facebook, 4000 TPS for Microsoft and 10,000 TPS for Apple before dropping to 10–60 TPS. The accumulated workload, denoted GAFAM, runs for 3 minutes and experiences a peak of 19,800 TPS before dropping between 25–140 TPS.

The exchange DApp is implemented as an `ExchangeContractGafam` smart contract with functions `checkStock`, `buyGoogle`, `buyApple`, `buyFacebook`, `buyAmazon`, `buyMicrosoft`. Each order consists of invoking the corresponding `buy*` function that, in turn, checks the availability of the stocks before updating the number of available stocks and emitting a corresponding event. More specifically, the process consists of a fungible token available in limited supply implemented by a single integer counter. Each transaction buys one token by decrementing the counter after checking that this counter is greater than 0.

Gaming DApp / Dota 2

Gaming DApp executes the trace of the most popular game on Steam, which is a Multiplayer Online Battle Arena video game called Dota 2 [22]. The number of Steam users peaked at 26.85 million in March 2021 [23]. The DApp comprises players who interact with each other and with the environment.

The gaming DApp is implemented as a smart contract `DecentralizedDota` whose `update` function moves the positions of 10 players along the x-axis and y-axis of a 250-by-250 map so that they turn back whenever they reach the limit of the map. The trace lasts for 276 seconds invoking at an almost constant update rate of about 13,000 TPS, which is particularly demanding.

Web service DApp / Fifa

The web service DApp is modeled as the number of requests to the Fifa website during the 1998 soccer world cup. More than 1.35 billion requests to the Fifa website were recorded over the course of the 84 days of the world cup with an average requests length of 3689 bytes. In particular, during the final match on June 30^{th}, 1998, between 11:30 PM and 11:45 PM, the total number of requests reached 3,135,993 for an average

request per minute of 209,066, a total of 8.5 GB transferred and an average of 580 bytes transferred per minute. During the most demanded minute of this period, 215,241 requests were sent, translating into an average of 3587 TPS.

The web service DApp is implemented as a simple `Counter` smart contract, with an `add` function that gets incremented at each request Hence its workload is highly contended. The duration of the workload is 176 seconds, sending an overall $3,500$ transactions at a rate varying from 1416 to 5305 requests per second.

Mobility service DApp / Uber

The mobility service DApp is based on a study of Uber requests in New York City (NYC) from 2018 [24]. The study reports a peak of 16,496 requests per hour between January 2015 and March 2015. As the demand grew since 2015, this peak throughput does no longer reflect the Uber demand. The average Uber demand between 2015 and 2019 grew from 70,348 to 556,387 requests per day with an increase of monthly users of 7.91.[1] The current Uber demand is approximated in NYC to $16,496 \times 7,91 = 130,483$ requests per hour, which translates into 36 TPS. In the first quarter of 2019, nearly 1,55 billion Uber trips were booked around the world, while 63,48 million Uber trips were booked in NYC alone [25]. As the NYC demand represents 1/24 of the world demand, to extrapolate this demand to Uber worldwide, the demand is derived to $24 \times 36 = 864$ TPS.

The mobility service DApp consists of a `ContractUber` smart contract whose function `checkDistance` computes the Euclidean distance between the customer (the requester) and 10,000 drivers in an area (a 2-dimension grid) of $10,000 \times 10,000$ in order to match the closest driver to the customer. As neither the PyTeal nor the Move languages support floating points or define the square root function $\sqrt{}$ to compute Euclidean distances, Newton's integer square root function is implemented in Solidity, PyTeal, and Move languages and used to compute the Euclidean distance. As Algorand DApps state is limited to key-value pairs, the PyTeal implementation of `ContractUber` only stores the position of one driver and computes the Euclidean distance to this unique driver 10,000 times. As the function executes a loop with 10,000 iterations computing the distance, the mobility service DApp is computation intensive.

Video sharing DApp / YouTube

The video sharing DApp is based on the number of videos uploaded to YouTube [26]. More precisely, from the YouTube traffic observed during the three months of 2007, the

[1] `https://toddwschneider.com/dashboards/nyc-taxi-ridehailing-uber-lyft-data/`.

Blockchain	Prop.	Consensus	VM	DApp lang.
Algorand [15]	probabilistic	BA⋆ [15]	AVM	PyTeal
Avalanche [6]	probabilistic	Avalanche [6]	`geth`	Solidity
Diem [16]	deterministic	HotStuff [28]	MoveVM	Move
Quorum [17]	deterministic	IBFT [29]	`geth`	Solidity
Ethereum [3]	eventual	Clique [30]	`geth`	Solidity
Solana [18]	eventual	TowerBFT [31]	eBPF	Solidity

Table 2.2: Blockchains evaluated in the thesis.

day with the peak request rate and the hour within this day with the peak request rate of 1,680,274 transactions per hour was extracted. This result was normalized to obtain a request rate of 467 transactions per second. Between 2007 and 2021, the number of videos uploaded to YouTube has been multiplied by 83 [27]. Hence the average throughput is approximated to $467 \times 83 = 38,761$ TPS, which makes this DApp very demanding. The video sharing DApp corresponds to a smart contract called `DecentralizedYoutube` with an `upload` function that gets some video data as a parameter and assigns the requester's address to the data before emitting a corresponding event.

2.3 Blockchain Protocols

In this section, we describe the six blockchains with smart contract support that we compare using Diablo-v2. They are listed in Table 2.2.

Algorand

Algorand [15] is a proof-of-stake blockchain that elects a subset of nodes, through sortition, that can append the next block. It does not fork with a high probability, so the transaction is final as soon as it is included in a block.[1] Algorand features a blocking API that waits for the transaction to be committed before returning to the client. Although it makes it natural to use this blocking call to detect the commit of each transaction, Diablo-v2 was too demanding. Hence we made Diablo-v2 polls every appended block to detect transaction commits, which significantly improved Algorand's performance.

Avalanche

Avalanche [6] is a blockchain also offering instant finality with high probability. There are now three blockchain protocols in Avalanche: one featuring the Ethereum Virtual

[1] `https://algorand.foundation/algorand-protocol/core-blockchain-innovation`.

Machine (C-Chain), one supporting only native transfers (X-Chain), and another one for metadata management. To evaluate the DApps, we used C-Chain, which exposes a web socket streaming API (shared by Ethereum and Quorum) to access the current blockchain head or the latest block. Avalanche supports the London release improvement of Ethereum (improvement `#1559` of August 5th, 2021) with the new gas fee structure with tips, which means the gas fee is computed dynamically (differently from Ethereum's original method). Avalanche limits the gas per block to 8M gas and the lower bound the period between blocks to 1.9 seconds[1].

Diem

Diem, formerly known as the Libra blockchain [16], was initiated by Facebook. It features a variant of the HotStuff protocol that solves the consensus problem deterministically (hence avoiding forks) while reducing the communication of traditional consensus protocols in good executions.

Ethereum

Ethereum [3] is the second largest blockchain in market capitalization. As the default version of Ethereum uses the proof-of-work cryptopuzzle resolution, which inherently limits its throughput, we exclusively used the Ethereum proof-of-authority consensus protocol, called Clique, as available in `geth`. This version still limits the block period to 15 seconds by default. Just like Avalanche, the Ethereum API exposes a web socket streaming API to access the current blockchain head or the latest block.

Quorum

Quorum [17] is a blockchain initiated by J.P. Morgan and currently maintained by Consensys. It features different consensus algorithms: Raft, which only tolerates crash failures, and IBFT and QBFT, which both tolerate Byzantine failures and partial synchrony. As Quorum features the `geth` Ethereum Virtual Machine, with the latest changes from the Berlin upgrade (April 15th, 2021), it also features the Clique proof-of-authority consensus algorithm. However, it does not feature the more recent London gas fee computation used by Ethereum and Avalanche. Similar to Ethereum and Avalanche, Quorum exposes a web socket streaming API to access the current blockchain head or the latest block.

[1] `https://snowtrace.io/chart/blocktime`.

Solana

Solana is a recent blockchain that is highly optimized for special hardware features (GPU and Intel instructions). Similar to Ethereum, Solana may fork and needs to wait for 30 confirmations (additional appended blocks) before a stored transaction can be considered final [32]. Its algorithm builds upon proof-of-history and "depends on messages eventually arriving to all participating nodes within a certain timeout" [18]. To append a block every 400 milliseconds, Solana replaces the Merkle Patricia Trie of Ethereum with a simplified data structure and replaces the ECDSA signature scheme with EdDSA (ED25519). Solana uses its own API, also based on a web socket, that allows the client to specify a commitment level. The clients listen for new blocks with the desired commitment level by subscribing to a web socket interface. Interestingly, Solana fetches the block hash before issuing transactions because the last block hash needs to be signed as part of the issued transaction.

Collaboration acknowledgements

The author of the thesis implemented Avalanche and Solana support in Diablo. Gauthier Voron contributed Algorand and Diem client interfaces. Chris Natoli added support for Ethereum and Quorum.

2.4 Amazon Web Services and i8 testbed

To evaluate different blockchain protocols using our framework, we need a set of machines where we can run the blockchain network and the benchmarking framework. In this thesis, we have AWS (Amazon Web Services) and the chair testbed to availability.

Amazon Web Services

If we begin looking at the interaction with the service from the bottom-up, we start with Machine Images. An image contains a snapshot of storage containing the operating systems and required software. An instance is a virtual server, which is created by specifying the instance type and the machine image. The instance type describes the hardware configuration of an instance, and the same machine image can be reused with different instance types. A fleet is a group of instances that can be created with a single request. Fleets can be created in different regions, which represent different data centers around the globe.

Instances get a public IPv4 address when they are created, which can be used to access the instance.

AWS provides an API to perform actions with the service, a command-line tool, and SDKs in different programming languages to use the mentioned API.

i8 testbed

With the chair testbed, we get to use the bare-metal servers, as well as deploy virtual machines on them. The testbeds are controlled with the Plain Orchestrating Service (pos). Before moving to the notions related to the servers, we start with the calendar. Because there are bare-metal servers, the user has to create a calendar entry for the server they want to use. After the calendar entry is created, the user can make an allocation that allows them to execute different commands with pos, such as rebooting the server. Here, we also have a concept of an image, which contains the operating system and applications. Nodes represent servers and virtual machines.

The testbed network topology is heterogeneous, meaning that servers have a different number of NICs and accessibility from the management node. For example, the iLab testbed is only accessible through a jump host and does not have connectivity to other machines from the coinbase testbed.

pos also provides an API to manage the servers, a command-line tool, and Python SDK for the API.

Chapter 3

Related Work

In the following chapter, we outline the related work comprised of blockchain benchmark frameworks and explain their limitations.

Hyperledger Caliper [8] is a blockchain benchmark framework enabling users to evaluate the performance of blockchains developed within the Hyperledger project, such as Fabric, Sawtooth, Iroha, Burrow, and Besu. It also supports Ethereum and has plans to extend to other blockchains in the future. Caliper provides pre-defined workloads, specifying the calling contract, functions, and the rate of transaction sending over time. Unfortunately, all these workloads are synthetic, and we are not aware of any pre-defined DApps with realistic workloads that can be used with Caliper.

Blockbench [7] is one of the most notable benchmarking frameworks for blockchains, as it supports a number of micro and macro benchmarks. The most significant aspect of Blockbench is that it ports over the notable YCSB workload from the database systems community. Aimed at private blockchains, Blockbench benchmarks the different layers of the blockchain, such as consensus or data storage, with tailored workloads, allowing fine-grained testing and measurement of the effectiveness of each of these layers. The evaluation metrics available show throughput and latency, but also the tolerance of faults through injected delays, crashes, and message corruption. Blockbench's complexity introduces difficulty when extending to other blockchains or introducing new workloads.

The variety of Ethereum adapted blockchains motivated the development of Chainhammer [9], a benchmark tool focused on the performance of Ethereum-based blockchains under extremely high loads. Chainhammer, unlike others, does not follow a workload curve but provides continuous high load generation, aiming to measure the throughput in extreme situations. The design is specialized, meaning that there is little flexibility in

modifications to support other workloads. Conversely, as Chainhammer is exclusively for Ethereum, it can perform post-benchmark analysis and obtain metrics on information critical to the Ethereum infrastructure, such as transaction cost analysis and the structure of blocks.

Most evaluations that were made on blockchains are ad hoc and do not aim at comparing very different blockchain designs on the same ground. Previous works have, for example, focused on permissioned blockchains as one can find in the context of the Internet of Things [10]. Others have evaluated exclusively Byzantine fault tolerant blockchains [11].

Chapter 4

Analysis

In this chapter, we perform a review of the Diablo prototype. We focus on the parts which are critical for the extensibility and generalizability of the framework and point out the limitations of the prototype. Furthermore, we discuss the differences between AWS and iLab testbed and which characteristics of the protocols can be better evaluated in the right environment.

4.1 Benchmark framework

In this section, we focus on identifying the design decisions which create issues when we try to extend the framework to support other blockchains and negatively affect the performance. We started our work based on commit `823ad60` of Diablo.

4.1.1 Transaction signing

We notice that the workload generator interface and the architecture assume that all the transactions can be pre-signed on the primary and then distributed over the secondaries in order to be sent to the blockchain network. This creates problems with several protocols that we want to use in our experiments.

Solana requires to include a recent block hash to every transaction submitted to the network in general. Solana also has an implementation of transaction nonces which involves deploying a special smart contract to the user account. This smart contract then allows storing the current block hash when any transaction involving the user account is executed. However, this still does not allow us to prepare the whole transaction workload in advance since we don't know the block hashes which will appear during the experiment execution.

Ethereum "London" update introduced a dynamic fee policy for transactions, which resulted in transactions failing to commit due to being underpriced. This change also affects Avalanche since it uses Ethereum Virtual Machine for its C-Chain as well.

To address this issue, the blockchain interface should not expect the transactions to be pre-signed. The contents of the transactions should be opaque to the framework, and the required logic should be contained within the protocol-specific implementation.

4.1.2 Blockchain interface

The workload generator and client interfaces consist of 11 and 15 methods, respectively, which were designed with Ethererum JSON-RPC API.

First, the actual implementation does not use all of the methods when the experiment is executed. For example, the client interface contains methods for contract deployment and querying the blocks, but they are not called in the generic part of Diablo. This might be misleading for the developers who want to integrate new protocol support in Diablo.

Second, this creates an issue that it is too restrictive for other blockchain protocols, which might have a different interaction pipeline. For example, the implementation assumes that a single transaction is required to deploy a smart contract to the blockchain network. However, Solana API uses UDP for communication and therefore restricts the message size sent over the network, resulting in multiple transactions being required to deploy a smart contract.

Lastly, the interface assumes that the methods will be called inside a concrete implementation of the interface. The problem that arises from such an approach is that the data passed between the methods have to be encoded and decoded, creating an additional burden for the developer.

As in the previous aspect, the contents of transactions and protocol-specific notions should be opaque to the framework. The client implementations for the protocols should hide all the details of operations, such as smart contract deployment or block listening. The benchmarking framework should only operate on transaction submissions and later get the commit timestamp based on their interface.

4.1.3 Workload specification

As described in Section 2.1, the workload specification consists of the number of threads, number of secondaries, and time series representation of workload, defined in per-second granularity.

The limitation of such a simple description is that the workload is uniformly distributed over all the secondaries. While this works well if all the secondaries have the same hardware specifications, there is a possibility that if the machines have different compute power, the secondaries with slower hardware might fail to produce the expected workload.

Furthermore, such a specification format does not allow to specify access patterns between different secondaries and blockchain nodes. In some cases, it might be required that some secondaries only access a subset of blockchain nodes, for example, when traffic between some pairs of secondaries and blockchain nodes is more expensive.

To solve this problem, it is possible to assign deployment-specific tags to blockchain endpoints and later specify which tag a particular secondary should use for its connections. As for the workload distribution, the configuration can specify transaction per second workload on a per-secondary basis. This will add complexity to the configuration, and the configuration will not be portable across different deployments. However, it will be possible to convert a simple uniform workload distribution description to the new format in an automated fashion.

4.1.4 Workload scheduling

As mentioned, the workload is specified in per-second granularity. During the experiment execution, the framework schedules the transactions to be sent every second and sends all the transactions for a particular time point immediately. On a small scale, such behavior results in local load bursts, followed by a period of inactivity until the next time point.

In order to prevent the idle time between the bursts, the framework can schedule the transactions uniformly throughout a single second, as the timestamps can be represented as the number of milliseconds or microseconds.

4.1.5 Bandwidth usage

All the transactions are expected to be created and signed on the primary and then distributed over the secondaries to be sent to the blockchain network. Such an approach results in high bandwidth usage between primary and secondaries, as the signed transactions contain all the required protocol-specific data.

To address this aspect, the transactions can be generated on the secondaries, and the primary can only distribute the descriptions of the transactions, which contain data, such as source and destination account identifiers, and the token amount to be transferred.

4.1.6 Command-line parameters

Diablo uses benchmark and chain configuration files to define an experiment. Both files are passed to primary and secondaries as command-line parameters. This adds overhead to the developers as they have to distribute the files to all the machines used by Diablo, and make sure that the same files are used in the experiment.

Instead of specifying the configuration file for every binary, the network connections between the primary and the secondaries can be reused to share the settings, such as blockchain endpoints or other parameters.

4.1.7 Output format

The output file contains per-thread results for each client in a secondary, per-secondary aggregated results, and total results. Per-thread results include a list of transaction latencies and metrics such as average and median latency, average throughput, per-second window throughputs, number of successfully committed transactions, and transactions that failed to be committed. Per-secondary aggregated results contain the same data and metrics, the difference being that all the lists of transaction latencies from threads are used in the calculations. Total results include maximum, minimum, and average throughput and latency, plus aggregated latency lists and per-second window throughputs.

The problem with such output format is that the data about timepoints when individual transactions were sent and committed are lost and replaced with inferred data – latencies and throughputs. For example, if we want to include ramp-up and tear-down periods, we need a possibility to consider only a subset of the experimental results, removing the transaction information for the mentioned periods. With the current output format, we cannot filter out the unneeded transactions based on their submit and commit times.

The solution, in this case, would be to output the whole list of transaction submissions metadata, containing submission time and commit time. Having individual submission and commit times will allow selecting a subset of the results to account for ramp-up and ramp-down, for example. It will also allow deriving additional metrics later if required.

4.2 AWS and the i8 chair testbed comparison

In this section, we look into the differences of cloud provided virtual machines and local testbeds on the example of Amazon Web Services and the i8 chair testbed.

First, we discuss the benefits of cloud environments for the blockchain protocol evaluation. One of the important points is scalability in terms of computing power. AWS

provides a vast range of machine types, for example, from 2 vCPUs and 4 GB RAM to 96 vCPUs and 192 GB RAM. Protocols can be optimized for different hardware with multiprocessing capabilities, such as GPU and CPU with vector extensions or CPU with specific architecture. Such factors are taken into account by the providers, and machines with different hardware are also available.

While multiple machines of different types can be spawned in the same datacenter, cloud providers also typically have multiple data centers across the globe. This brings us to the second benefit of AWS, which is geographical distribution. Currently, Amazon has AWS datacenters in more than ten regions. With this feature, we can create networks of hundreds of machines, which allows us to easily test the scalability aspect of the protocol in terms of the number of blockchain nodes.

The fact that the datacenters are distributed across the globe provides us with a network with realistic latency and bandwidth. Even though virtualization is present in the setup, the machines share the actual hardware and network links. As the datacenters are located on different continents, we are provided with the latencies limited by the physical properties of the connection and the actual distance and underlying network topology between the locations. As different services on the machines communicate with each other and are accessed by the users, the bandwidth of the links is being used. This allows taking another important aspect of real networks into account during the evaluation, which is background traffic.

The example of measured latency and bandwidth is displayed in Figure 4.1. We see the pairwise RTT in milliseconds and throughput between 10 geographically distributed regions. The measurements were done with `iperf3` and its default parameters. The traffic was generated over 10 seconds. In the table, we can see that the distance between the locations affects the latency. We observe minimal RTTs between the closely located regions – Milan and Stockholm, Bahrain and Mumbai, Oregon and Ohio. The maximal RTTs are observed in distant regions, such as Sydney and Cape Town.

On the other hand, such an environment makes it hard to perform reproducible tests. The utilization of the network links between the datacenters changes throughout the day, as the services may be accessed more during the day and less at night. The usage is reflected in latency and bandwidth, with lower latency and higher bandwidth available at hours with reduced usage and higher RTT and lower throughput being observed at peak usage hours.

In order to account for the variance in the network parameters, local testbeds can be used to perform the measurements. In this environment, the whole network can be exclusively used by the system under test. For example, with the iLab testbed, we have

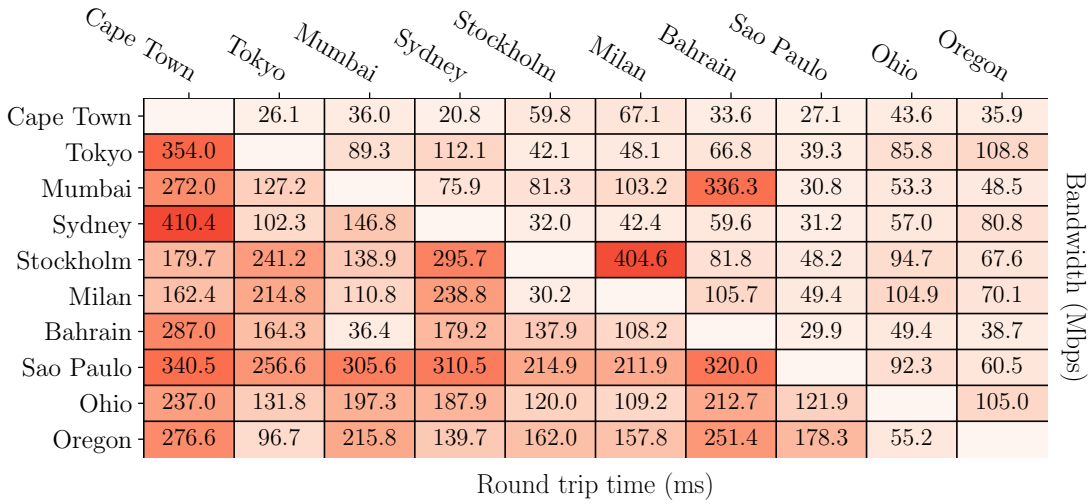


Figure 4.1: Round trip time and bandwidth between the 10 selected AWS regions

measured an average of 1.1 millisecond RTT using the same approach as with AWS. Given that the latency between the nodes does not change, we can introduce arbitrary delays to evaluate the tolerance of the blockchain protocol against the network delays.

Such property of the iLab testbed network as fixed latency between the nodes allows us to replicate the latencies of geographically distributed cloud networks at a particular point in time. With `tc-netem` tool, we can specify the added delay on a network interface of the machine used for running the experiments. Ideally, such a setup can be used to reduce the usage of cloud environments, produce similar results, and reduce the cost of the experiments.

In order for the deployment solution to be backend-independent, it should operate on a unified protocol, such as SSH. In this case, it will be possible to operate on any set of servers that are accessible with SSH on the host.

4.3 Summary

In this chapter, we analyzed the Diablo prototype and discovered the limitations that prevent it from being extensible to support different blockchain protocols. We also looked at the differences between cloud environments and local testbeds and their advantages and disadvantages. We came to the conclusion that both environments should be considered as they can be used for evaluating different aspects of the protocols.

Therefore, we can provide a list of requirements for an extensible and generic blockchain benchmarking framework.

R1 Flexibility
The framework should support different modes of workload generation, such as pre-signed or unsigned, to support different protocols.

R2 Simplicity
The client interface that has to be implemented in order to support a new protocol should be simple and not contain unused methods.

R3 Configurability
The framework configuration should account for different deployment topologies and hardware.

R4 Granular scheduling
The framework should schedule the transaction submissions with subsecond granularity to prevent unnecessary idle time.

R5 Minimal overhead
Resources used inside the framework, such as the bandwidth required to distribute the information about transactions, should be minimal.

R6 Usability
The framework configuration should prevent an incorrect combination of parameters.

R7 Extensibility
It should be possible to derive different metrics from the framework's experimental results.

R8 Interoperability
It should be possible to deploy the framework in different environments, such as cloud VMs or local testbeds.

Chapter 5

Design

The goal of this chapter is to create the generic and extensible architecture for the Diablo-v2 benchmarking framework and design the orchestration and deployment tool Minion. The design is based on the analysis from the previous chapter. We derive an extensible approach for Minion based on interfaces of AWS and pos.

5.1 Diablo-v2

This section describes the improved architecture for Diablo-v2 that allows comparing different blockchain protocols on the same ground in different environments.

In Figure 5.1, we see the components of Diablo-v2. To facilitate distributed workload generation, Diablo-v2 comprises two main components, a single *Primary* and multiple *Secondaries*. For the sake of extensibility, Diablo-v2 offers a blockchain abstraction with four functions that a developer can implement to compare their new blockchain protocol to existing blockchains, and Diablo-v2 also offers a workload specification language for a developer to add new DApps.

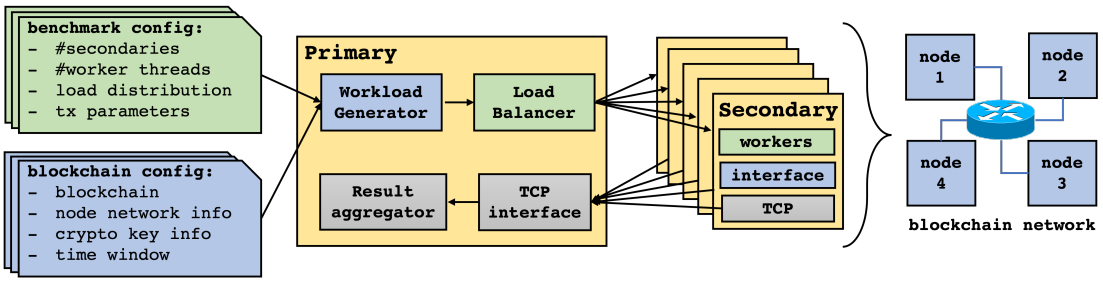


Figure 5.1: The architecture of Diablo-v2

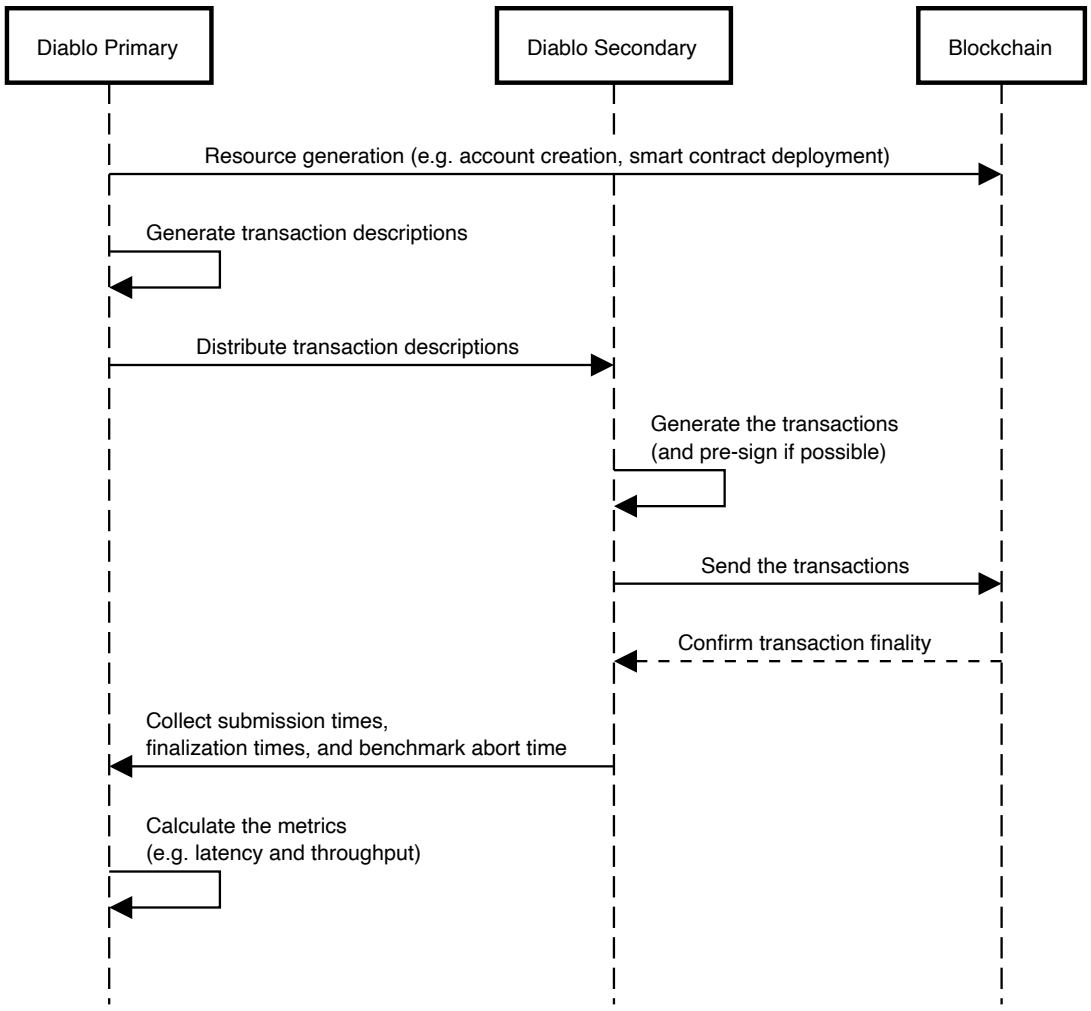


Figure 5.2: Diablo-v2 interaction sequence

Figure 5.2 gives an overview of Diablo-v2's workflow. Below, we describe the responsibilities of the primary and the secondaries in detail.

5.1.1 Primary

The purpose of the Primary machine is to coordinate the experiment: it generates the workload and dispatches it between Secondaries, launches the benchmark, aggregates the results, and reports them back.

Prior to starting the benchmark, its workload generator parses the benchmark and blockchain configuration files. A benchmark configuration file indicates the requests type, whether requests are native transfers or DApp invocations, and their distribution between Secondaries and blockchain nodes over time. For each workload invoking

DApps, the Primary also deploys the smart contracts listed in the benchmark configuration file. The blockchain configuration file is necessary to generate the workload appropriately because the transaction distribution depends on the number and locations of the deployed blockchain nodes. Then, the Primary transmits a description of the transactions to the Secondaries, waits for all Secondaries to be ready, and informs all Secondaries when to start the benchmark. Such an approach satisfies **R5**, as we minimize the used bandwidth by transferring the descriptions of the transactions instead of the fully signed transactions.

Once the benchmark is complete, each Secondary sends its results to the Primary, and a results aggregator collects them to output a file, indicating the start time and end time of each transaction (as recorded by the Secondaries). These timestamps can then be used post-mortem to generate time series and analyze the distribution of latencies (e.g., Fig. 7.6) or more simply, to output aggregated values like the average (e.g., Fig. 7.3).

5.1.2 Secondary

Secondaries are responsible for the presigning of the transactions and the execution of the workload, interacting directly with blockchain nodes for the system under test. The number and specification of the Secondaries are typically chosen to match the resources allocated to the blockchain (cf. Table 7.1) to be able to stress test the blockchain. Note that each Secondary can send requests to multiple blockchain nodes.

The Secondary interacts with the blockchain through a client interface specific to each blockchain. The current clock is recorded as the submission time right before a transaction is sent. The Secondaries constantly check if the submission time is not too late compared to the time demanded by the Primary and emit a warning otherwise. Each worker thread constantly polls the blockchain nodes to obtain the last block and check whether it contains sent transactions. When a sent transaction is detected within a block, the current clock time is recorded as the decision time for this transaction.

5.1.3 Blockchain abstraction

To make Diablo-v2 compatible with various blockchain implementations, we abstract away the main components of a blockchain. The Diablo-v2 benchmark specification interacts with the resulting blockchain abstraction. Let C be the set of clients. A blockchain is modelled as a tuple $\langle E, R, I \rangle$ where E is the finite set of *endpoints* that act as blockchain nodes, R is a finite set of resources (e.g., account balance, smart contract state) maintained in the blockchain state, and I is a potentially infinite set of interactions types (e.g. asset transfer, smart contract function invocation) between a

client and the blockchain. An interaction event is denoted as a tuple $\{(c, i, r, t)\}$ with $c \in C$, $i \in I$, $r \in R$ and $t \in \mathbb{R}$.

The benchmark specification contains a function M mapping the Secondaries to the blockchain endpoints, a set φ^R of resources needed for the test and the interactions $\{(\varphi^c, \varphi^i, \varphi^r, t)\}$ where $\varphi^c \in \varphi^C$, $\varphi^r \in \varphi^R$, $t \in \mathbb{R}$. More precisely, $M : S \times E \Rightarrow \varphi^C$ where S is the set of Secondaries, E is the set of endpoints, both available only at runtime, and φ^C is the set of specified clients, each specified client is implemented by an explicit worker thread. The two types of interactions are `transfer_X` to transfer `X` coins from one account to another one and `invoke_D_Xs` to invoke a Dapp `D` with the parameters `Xs`.

To add a new blockchain, one has to implement at least one of these interaction types as well as 4 functions that convert the benchmark specification to an executable test program: (i) s`.create_client(`E`)` where $s \in S$, (ii) `create_resource(`φ^r`)` and $\varphi^r \in \varphi^R$, (iii) `encode(`φ^i, r, t`)` where $t \in \mathbb{R}$ and $r \in R$, and (iv) `trigger(`i, r, t`)`.

Such an abstraction satisfies **R2** mentioned in Section 4.3, as it allows the framework to be extended to support multiple different protocols. It also satisfies **R1**, as the transactions are opaque to the framework behind the interaction interface and can be either pre-signed or signed just as they are sent to the blockchain network.

5.2 Minion

In order to automate the deployment of Diablo-v2, we designed an orchestration and deployment tool called *Minion*. It allows allocating machines from a provider, preparing the hosts, and installing the required software to run the blockchain network and Diablo, run the experiments and collect the results. Minion features an extensible design that allows to easily add support for different machine providers.

Figure 5.3 gives an overview of Minion's workflow. We define three actors in the interaction sequence. The Minion host is the host that is responsible for orchestration and deployment. It interacts with the machine provider, where it can request the machines and perform other provider-specific operations, and with the system under test, which consists of machines used to run the blockchain network and Diablo-v2. Minion host stores all the required parameters for the experiment and the results of the measurements. The machine provider allocates the machines and returns a list of handles to be used by Minion for communication with the machines. The allocated machines are then used to run the blockchain network and Diablo-v2.

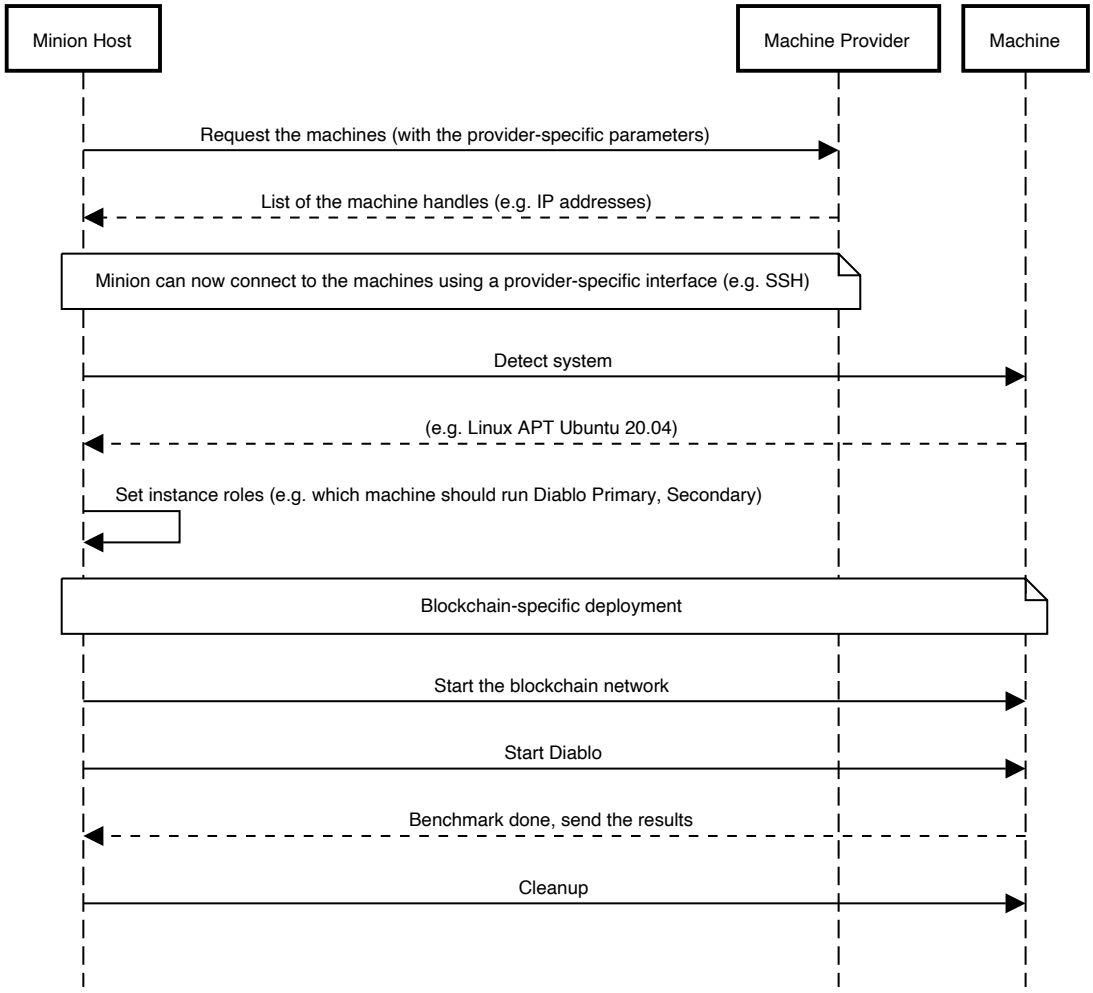


Figure 5.3: Minion interaction sequence

First, the Minion host starts the operation by requesting the machines from the provider, specifying the required parameters if needed. The result of this request is the list of handles that can be used by Minion to interact with the machines, execute the required commands, and transfer the files. Next, we detect the system on the machines in order to install the software. Such an approach allows us to have a set of installation scripts for each operating system and easily add a set of scripts for a new distribution if required.

After the software is installed, the roles are assigned to the allocated machines. The roles define which software will be run on the node and the addresses of the related machines if required. In this phase, we can also specify the number of the blockchain nodes or Diablo-v2 secondaries we want to have on a single host.

Then, Minion executes blockchain-specific deployment scripts. The scripts prepare the hosts to run the blockchain network and Diablo-v2. The scripts allow delegating the work, which requires the installed blockchain binaries to the allocated machines and performing the generation with the gather-scatter operation. For Diablo-v2, information about the blockchain network and the workload specification is distributed to all the nodes which will run the experiment.

The experiment execution phase begins with the start of the blockchain network. Since Minion waits until a blockchain-specific start script exits, it is possible to implement the health check, which waits until the network is ready to accept the requests from the clients. When the blockchain network initialization is completed, Minion starts Diablo-v2 primary and the secondaries. After Diablo-v2 has finished collecting and aggregating the results of the experiment, they are transferred to the Minion host, along with the logs of the blockchain nodes.

As described, such architecture decouples the interaction between the machine provider and the machines themselves, making it possible to use the tool in different environments. Therefore, **R8** is satisfied.

Chapter 6

Implementation

In this chapter, we outline the implementation details of Diablo-v2 and Minion. We present the workload specification format used in Diablo-v2 and the extra steps implemented in Minion to prepare the iLab testbed hosts for the experiments.

6.1 Diablo-v2

Diablo-v2 is implemented in Golang, as it provides all the required facilities for concurrency and networking. Also, Golang is a popular language choice, as it is used by Algorand, Avalanche, Ethereum, and Quorum.

6.1.1 Workload specification

Using the notation described in Section 5.1.3, the benchmark configuration file specifies the function M, the set φ^R and the interactions $\{(\varphi^c, \varphi^i, \varphi^r, t)\}$ using YAML format. For example, the gaming DApp configuration file in Listing 6.1 defines 4 variables: `acc` (line 4) is a set of 2,000 user accounts, `dapp` (line 5) is a set containing one instance of the `dota` DApp. Those two variables form the φ^R set.

The variable `loc` (line 2) is the set of Secondaries tagged with the string `us-east-2` (an AWS availability zone) and `end` (line 3) is the set of all endpoints. These two variables are used in the definition of the M function (lines 7-10) which defines 3 clients invoking the DApp `dapp` from accounts in `acc` with the parameters parsed from `update(1, 1)` at the rate specified in the `load` section (lines 16-19): each client sends 4,432 TPS for the first 50 seconds then 4,438 TPS for the next 70 seconds which is the end of the test.

```
let:
  - &loc { sample: !location [ "us-east-2" ] }
  - &end { sample: !endpoint [ ".*" ] }
  - &acc { sample: !account { number: 2000 } }
  - &dapp { sample: !contract { name: "dota" } }
workloads:
  - number: 3
    client:
      location: *loc
      view: *end
      behavior:
        - interaction: !invoke
            from: *acc
            contract: *dapp
            function: "update(1, 1)"
          load:
            0: 4432
            50: 4438
            120: 0
```

Listing 6.1: Workload file example

The schema satisfies **R3**, as the workload can be specified for each blockchain client individually and distributed according to the hardware capacity of the secondaries. Using the `load` description, the primary schedules the transactions uniformly across the time periods, satisfying **R4**.

6.1.2 Diablo-v2 configuration

Diablo-v2 has the following command-line arguments, which can be used to specify the parameters and the configuration of the experiment.

```
diablo primary -vvv --port=5000 \
   --env="accounts=accounts.yaml" \
   --env="contracts=dapps-directory" \
   --output=results.json --compress --stat \
   10 setup.yaml workload.yaml
```

To run the Primary, we specify the verbosity level, port number for the secondaries to connect to, the path to the accounts file, DApps source codes, output file path, compress output (with gzip), printing statistics to standard output, number of Secondaries (10 in the example), blockchain setup file, and workload specification file.

```
diablo secondary -v --tag="us-east-2" \
    --port=5000 127.0.0.1
```

To run the Secondary, we again specify the verbosity level, port and address of the Primary to connect to and a tag to indicate the Secondary location for collocation with blockchain nodes.

Such design satisfies **R6**, as the configuration files are only passed to the primary, and the secondary gets all the required information for the experiment from the primary.

6.1.3 Results file format

The results file uses JSON format and contains all the concrete data regarding the clients and interaction events defined in the workload file. In Listing 6.2, we show a part of an example results file to describe its structure. On the top level, there is the seed used for operations requiring a random generator and the list of the locations corresponding to the secondaries. For each location, we store its IP address and port, associated tags, and the information about the clients that were running in the specified location. For every client, we report its index, reference to the client in the workload file, and the list of performed interactions. Finally, for every interaction, we also store the reference to the interaction type in the workload file. Then, depending on whether the interaction was submitted, committed, aborted, or had an error, we store the corresponding timepoints or a boolean flag for the error.

The format satisfies **R7**, as using the timepoints of individual interactions, we can calculate the metrics for the subsets of data and derive throughput and latency.

6.2 Minion

Minion is implemented in Perl as a set of modules and the main script, which contains the main deployment scenario.

6.2.1 pos support

To support pos, we first create a Cli module that wraps command-line interface tool commands, such as managing an allocation or a node.

We extend the Ssh module to create a Node module, similarly to the AWS Instance module. In Node, we wrap the calls to the Cli module with the correct arguments, such as node name. To initialize the Ssh parent, we pass the correct hostname, which differs from the node identifier in the case of iLab machines, by appending `.ilab` to the node identifier. We also create a method to get the correct IPv4 address for communication

```
{
   "Seed":883378932,
   "Locations":[
      {
         "Address":"3.139.56.33:57098",
         "Tags":[
            "3.139.56.33:57098"
         ],
         "Clients":[
            {
               "Index":0,
               "Kind":"/home/ubuntu/deploy/diablo/workload.
                  yaml:18:7",
               "Interactions":[
                  {
                     "Kind":"/home/ubuntu/deploy/diablo/
                        workload.yaml:21:24",
                     "SubmitTime":0.100119845,
                     "CommitTime":0.614523155,
                     "AbortTime":-1,
                     "HasError":false
                  }
               ]
            }
         ]
      }
   ]
}
```

Listing 6.2: Results file example

between the nodes. The method fetches the IPv4 address of `eno0` in the case of coinbase machines and the address of `eth-static` for the iLab machines.

We also create an Allocation module, similar to the AWS Fleet module. Here, we store the list of related nodes and wrap the calls to `allocate` and `free` CLI methods.

Therefore, such a structure allows us to have the following sequence of actions to run the experiment. First, we create a calendar entry and a corresponding allocation. Then we set the image and reset the nodes. At the end of the experiment, we stop the nodes and free the allocation along with the calendar entry.

6.2.2 iLab support

For the iLab testbed, several features have been taken into account to easily run the experiments with the blockchain networks we used.

The switches assign link-local IPv6 addresses to the machines. Since there are multiple interfaces on the machines, and several of them have an IPv6 address assigned, we should use scoped addresses to connect to other machines in the testbed. Such addresses did not work with Diem, which is implemented in Rust and uses `std::net::TcpStream` to create the connections. To eliminate the issue, we manually assign IPv4 addresses for the experiments. We use the addresses of form `10.20.ord(isle).index`, where isle is in `a-s,r`, and index is in `1-6`.

6.2.3 Network template preparation

We noticed that the network creation process might take a significant amount of time, which is too expensive to accept on each experiment run. For example, Algorand requires running a subprocess during the generation of cryptographic material for each node, which becomes an issue when the network size is on the order of hundreds of nodes. For Diem, the account creation function of the provided tooling runs sequentially in the blocking mode, and each individual call may take a couple of seconds. To solve this problem, we prepare all the files required to run a network of nodes, and we store the archive with the files for different sizes of the network. When we need to run an experiment, we unpack the archive and use the current IP addresses of nodes.

6.3 Blockchain protocols

Algorand

We experimented the Algorand version with commit `116c06e` dated from Nov. 23^{rd} 2021 and available at `https://github.com/algorand`. In addition to Solidity, a version of

each DApp was implemented in PyTeal because Algorand only supports the Transaction Execution Approval Language (Teal), which is bytecode and requires a conversion from the PyTeal higher level language.

Avalanche

For the experiments, we used the `master` branch with commit number `7840200`. We initially tried to setup the Avalanche experiments using the RSA4096 cryptographic signature scheme as recommended by Avalanche. However, this signing process was taking too long due to the scale of our experiments. As we could not make Avalanche work after replacing RSA4096 by ED25519, we opted for using ECDSA instead.

Diem

Like Ethereum, Diem requires that each transaction contains a sequence number, i.e., a monotonically incremented integer. The difference with Ethereum is that Diem nodes only accept a maximum of 100 transactions from the same signer in their memory pool, limiting the rate at which a unique signer can submit transactions. To bypass this limitation, we made workloads submit from 2,000 different accounts in most deployment configurations, however, we noticed that the provided setup tools would fail systematically after creating 130 accounts. This is why we restricted the number of accounts to 130 in the `community` and `consortium` configurations.

We experimented the testnet branch from Aug. 21^{st} 2021 with commit number `4b3bd1e` of the Diem repository `https://github.com/diem/diem`. Diem testnet branch is dated Aug. 20 of 2021, while the main branch was updated at the time of writing (Feb. 27, 2022). Even though the testnet branch seems outdated, the official Diem tutorial still recommends using the testnet branch for development purpose: `https://developers.diem.com/docs/tutorials/tutorial-my-first-transaction/`.

Ethereum

We evaluated the `geth` version from the `master` branch with commit hash `72c2c0a` from Dec. 12 of 2021 available at `https://github.com/ethereum/go-ethereum`. In August 2021, the "London" update to the gas calculation introduced the notion of tips. With this new version, the gas fee changes at every block, which can impact the execution of transactions: when the fee increases then the transaction risks to be underpriced. This is why, we tried to adjust the fee dynamically during the execution of the benchmark—this implied signing transactions online.

Quorum

We experimented the `master` branch with commit hash `919800f` of Quorum from 2 Nov. of 2021 available at `https://github.com/ConsenSys/quorum`. Given that Clique is vulnerable to message delays [4] and Raft is vulnerable to arbitrary failures, we exclusively run Quorum with IBFT in our experiments.

Solana

We experimented the commit number `0d36961` of the `master` branch of Solana from March 12 of 2022, as available at `https://github.com/solana-labs/solana`.

Previous tests ran by the Solana team all consisted of requesting the last block hash before issuing concurrently transactions withdrawing from different accounts. We could not use this technique while evaluating realistic DApps because Solana requires the hash to be created less than 120 seconds before the transaction request is received while DApps can run for longer. To cope with this limitation, the Solana-Diablo-v2 interface periodically fetches the last block hash.

Chapter 7

Evaluation

This chapter focuses on evaluating six state-of-the-art blockchain protocols with Diablo-v2 on Amazon Web Services and the local testbed environments. First, we describe the design of the experiments for both environments. Second, we provide the results of our experiments on AWS and iLab testbed. Lastly, we explain the challenges of comparing the results in different environments and the deployment specifics of different protocols.

7.1 Design

In this section, we describe the deployment configurations we used for the experiments on AWS and iLab testbed, including specifications of the machines and network topologies.

7.1.1 AWS

We deployed Diablo-v2 and the blockchains in different configurations with up to 200 virtual machines ranging from c5.xlarge (2 vCPUs and 4 GiB memory) to c5.9xlarge (36 vCPUs and 72 GiB memory) and spread equally among different geo-distributed regions in five continents: Cape Town, Tokyo, Mumbai, Sydney, Stockholm, Milan, Bahrain, São Paulo, Ohio, Oregon. Table 7.1 lists these different configurations. They range from a `datacenter` scenario with extensive resources to a `testnet` of collocated machines, to a geo-distributed `devnet`, to a large scale `community` of machines, to a large-scale `consortium` of modern machines. The table indicates the number of blockchain nodes deployed, the hardware they use, and how many regions they are spread across.

Configuration	Blockchain nodes			Regions
	number	#vCPUs	memory	
`datacenter`	10	36	72 GiB	Ohio
`testnet`	10	4	8 GiB	Ohio
`devnet`	10	4	8 GiB	all
`community`	200	4	8 GiB	all
`consortium`	200	8	16 GiB	all

Table 7.1: The experimental settings

We deployed both the blockchains and Secondaries in the different deployment configurations to measure the impact of geo-distribution on the blockchain performance. In all cases, we applied the same geo-distribution strategy to the blockchain nodes and to the Secondaries: each Secondary submits its requests to its collocated blockchain node so as to mimic requests being routed from a client towards its closest blockchain node. In all these configurations, a single Primary was used for setting up the experiment and gathering the performance results. As the Primary is not involved during the performance monitoring phase, its location does not impact the experimental results.

Datacenter

The `datacenter` configuration aims at showcasing the blockchain's peak performance in an idealized setting. Such a configuration features powerful c5.9xlarge machines located in the closed network of a single datacenter, the Ohio AWS availability zone. These machines are not commodity hardware as each machine features 36 vCPUs, and 72 GiB memory, the bandwidth and latency between machines are 10 Gbps and 1 ms[1], respectively, which is not representative of an open network. Instead, this configuration allows evaluating blockchains when a lot of resources are available.

Testnet

The `testnet` configuration features small c5.xlarge machines located in a single datacenter, the Ohio AWS availability zone. This typically corresponds to a testnet setting where blockchain developers typically run their blockchains in order to assess performance and stability during development phases. As the machines are cheaper to rent than c5.9xlarge, they allow the testnet to run for a long period of time, allowing for continuous deployment.

[1] `https://aws.amazon.com/ec2/instance-types/c5/`.

Devnet

The `devnet` configuration geo-distributes the machines in an open network to assess the performance in a setting involving the network latencies over long distance communications. This intends to mimic the performance one could expect from a blockchain devnet, where external beta testers or preliminary validators from different regions could participate in the evaluation of the blockchain before a release of a mainnet available to internet users.

Community

The `community` configuration increases the number of machines to about the number of countries around the world. This configuration aims at mimicking the performance of the blockchain as if it was used in a geo-distributed environment involving as many blockchain participants as there are jurisdictions (there are currently 195 universally recognized self-sovereign states in the world). Such a highly distributed setting is often considered to be particularly censorship resistant by not being strongly affected by political decisions in only one of the jurisdictions where it operates.

Consortium

The `consortium` configuration geo-distributes 200 blockchain nodes similar to the `community` configuration. However, it features more powerful c5.2xlarge machines that better represent modern computers featuring 8 vCPUs and 16 GiB of memory. This aims at mimicking a consortium of individuals or institutions, like the R3 consortium [33], who have the resources to devote modern machines without specialized hardware to participate in the blockchain service.

7.1.2 iLab

To run the experiments on the iLab [34] testbed, we use `eth-static` interface, which is a dedicated 10 gigabit network between all the testbed machines. As shown in Figure 7.1, the network consists of 7 isles (named a, b, c, d, e, f, s) of 6 machines each, plus an isle of 3 machines (named r), giving 8 isles and 45 machines in total. Every two isles (a and s, b and r, c and d, e and f) are connected to a switch, and there are overall 4 switches, and all of them are connected to each other.

The machines have the following hardware:

- Intel Core i7-8700 CPU @ 3.20GHz (6/12 cores/threads)
- 64 GB RAM
- 500 GB SSD

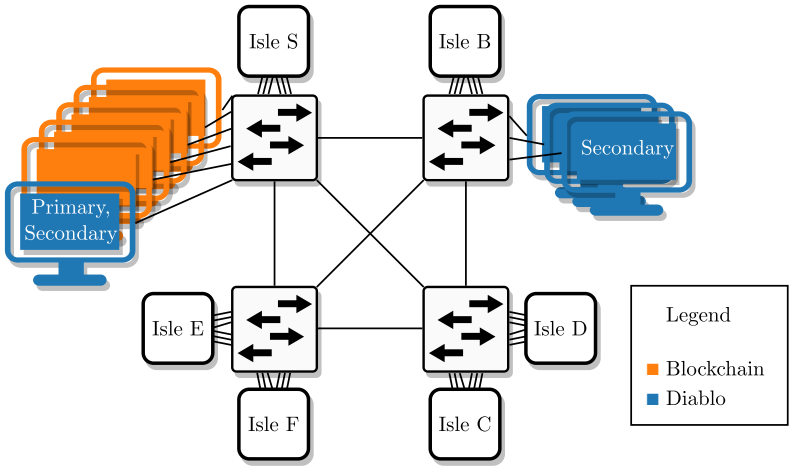


Figure 7.1: iLab topology

- Intel X550T 10 GbE NIC
- Debian 11 Bullseye

To distribute the workload generation over the testbed, we spread the Secondaries across all the available isles. We use the first machines of isles a-s and the machines of isle r for workload generation, giving us 10 machines for Secondaries in total. We use the remaining machines for blockchain nodes in different configurations. For simplicity, we vary the number of blockchain nodes as a multiple of 5, since we have 5 machines left from isles a-s. Same as in AWS setup, for the Primary, we use one of the machines which run the Secondary, as the Primary does not use any resources when the workload is applied to the blockchain network.

7.2 Results

In this section, we stress test the blockchains described in Section 2.3 under the realistic DApps of Section 2.2, synthetic workload, and real but less demanding workload traces to compare the scalability, robustness, universality, and availability of these blockchains.

7.2.1 AWS

To provide an overview of blockchains performance executing realistic DApps, we deploy each DApp of Section 2.2 in the `consortium` deployment configuration (200 8 vCPUs–16 GiB machines spread over 10 countries in 5 continents) and generate the workload associated with each of these DApps. For each run, we make sure that Diablo-v2 uses enough Secondaries to not be the bottleneck.

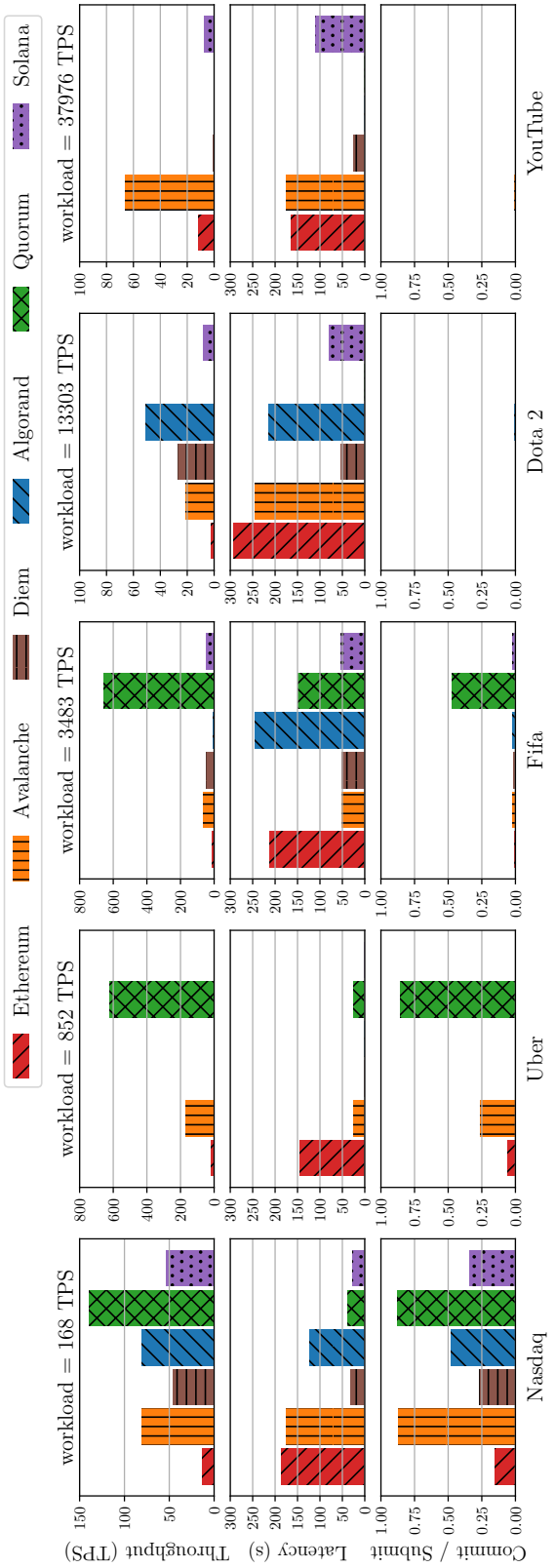


Figure 7.2: Evaluation of blockchain performance when executing realistic DApps

Figure 7.2 shows the average throughput, average latency, and the proportion of committed transactions for each blockchain-DApp pair. For each DApp (column), shows the average workload effectively submitted by Diablo-v2 (top of each column), average throughput (first row), average latency (second row), and proportion of committed transaction (third row) for each blockchain. Each blockchain is deployed on 200 c5.2xlarge AWS instances spread among 10 datacenters. We observe that for the Exchange DApp, which has the lowest average workload, Nasdaq, of 168 TPS only, Avalanche and Quorum commit more than 86% of the transactions, all the other blockchains commit 47% or less of the transactions. For the most demanding workload, the YouTube workload, the proportion of commits is lower than 1% for all evaluated blockchains. In addition, when the average workload is 852 TPS (like the Uber workload), or 3,483 TPS (like the Fifa workload), only Quorum maintains a throughput higher than 622 TPS while the other blockchains have a throughput lower than 170 TPS. For higher workloads (like Dota 2), no blockchain maintains a throughput higher than 66 TPS. Finally, among all DApps, no blockchains commit with a latency lower than 27 seconds. We indicate below what are the causes of this performance gap and how a blockchain developer can use Diablo-v2 to find these causes on their own blockchain.

Scalability and deployment

Using Diablo-v2, we quantify *scalability* as the ability to allow a large number of unprivileged users to participate in the blockchain execution. To this end, we deploy the blockchains on networks of different sizes composed of machines ranging from enterprise grade hardware with high computational power (`datacenter`) to commodity hardware with modest computational power (`community`). We then measure their performance when stressed with a synthetic workload.

More precisely, we deploy each blockchain on four deployment configurations: `datacenter`, `testnet`, `devnet` and `community`. For each configuration, we use Diablo-v2 to emulate clients sending native transactions to the blockchain for 120 seconds at a constant rate of 1,000 TPS, which is the same order of magnitude as the average load of the Visa system[1]. We measure the average throughput and average latency for each blockchain. If the measured throughput is close to the workload of 1,000 TPS, then we conclude that the blockchain handles the simple payment use case for the configuration.

[1] Visa claims 150 million transactions per day = 1,736 TPS on average (`https://usa.visa.com/run-your-business/small-business-tools/retail.html`)

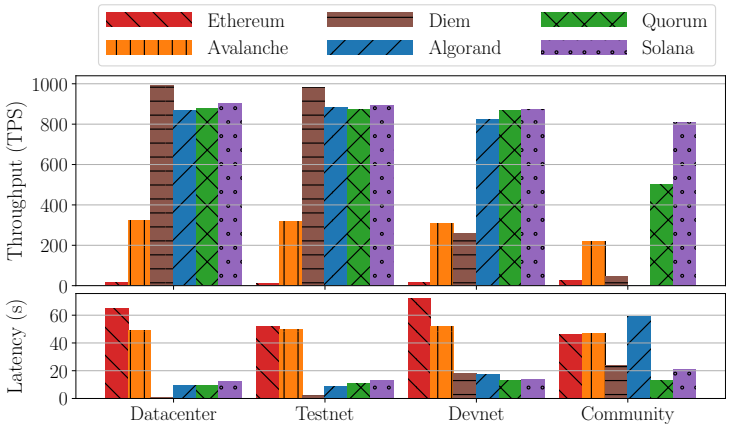


Figure 7.3: Throughput and latency with a constant workload of 1,000 TPS

Figure 7.3 shows the average throughput and average latency for each blockchain on the four configurations. We observe that only Solana handles a 1,000 TPS constant workload for all configurations while maintaining a throughput higher than 800 TPS with a latency below 21 seconds. Solana uses an eventually consistent consensus based on a verifiable delay function which puts away all communication steps but a broadcast. By using a verifiable delay function, Solana makes the block generation delay independent of the number of cores the participant uses. By removing most of the communication steps, Solana also performs better in more challenging network settings. Quorum also stands out in the community setup with a throughput of 499 TPS for a latency of 13 seconds. Quorum uses a well known deterministic consensus algorithm that does not introduce artificial delays and provides immediate finality. In addition, Quorum benefits from many blockchain specific optimizations by using `geth` as a base code. For all blockchains, there is no significant difference between the datacenter and the testnet configurations. In all the configurations, Diem achieves the best throughput (more than 982 TPS) and the best latency (2 seconds or less) but only on configurations with a local setup. We conjecture that Diem is designed to provide very low latency and is optimized to run on network setups with a low round-trip time (RTT). Over the remaining blockchains, only Algorand achieves a throughput higher than 820 TPS when deployed on the `devnet` configuration, which is a geodistributed network. In particular, the best average throughput that Algorand reaches in 885 TPS on the `testnet`. We conjecture that the other blockchains, namely Avalanche and Ethereum, are designed to run at a relatively low throughput regardless of the available computational power or network bandwidth.

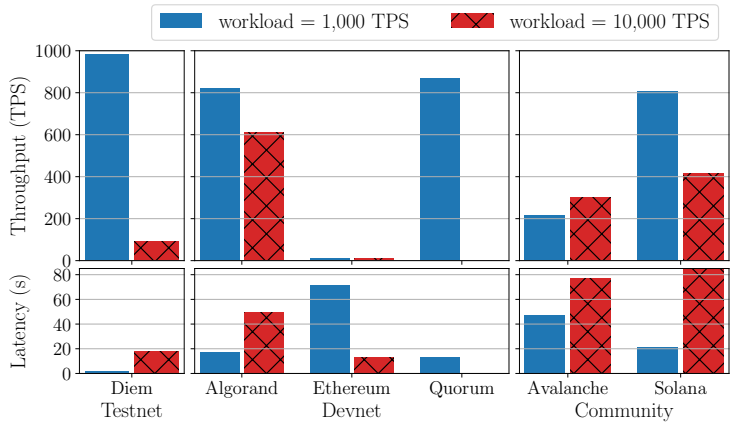


Figure 7.4: Throughput and latency of each blockchain with a constant workload

Robustness and denial-of-service attacks

To better understand whether blockchains are *robust* to high demand, we used Diablo-v2 to inject high workloads and test whether the blockchain collapses or continues treating requests further. Intuitively, a blockchain is more robust if a higher workload is needed to penalize its latency and throughput. This property is desirable to measure how hard it is for an adversarial user to perform a denial-of-service attack on the blockchain by submitting transactions at a high rate. The test consists of deploying the blockchain in a deployment configuration where it performs well under the moderate workload and observing whether a higher workload leads to performance degradations.

To compare the blockchain robustness, we deploy each blockchain on its most suited deployment configuration and observe its performance when stressed with a high workload. To this end, we configured Diablo-v2 to send native transactions to the blockchain for 120 seconds at a constant rate of 10,000 TPS, which is 10× higher than the sending rate in the deployment challenge. Although Diablo-v2 can send transactions at higher rates, we found this workload to be sufficient to show some interesting behaviors of the tested blockchains.

Figure 7.4 compares the throughput and latency of each blockchain when stressed with workloads of 1,000 TPS and 10,000 TPS. Diem and Quorum are the most negatively affected by the higher workload: Diem throughput is divided by 10 while Quorum throughput drops to 0. Interestingly, Diem and Quorum are the only blockchains we evaluated that use a deterministic Byzantine fault tolerant (BFT) consensus. These algorithms were originally designed to commit as many client requests as possible, a behavior that easily leads to saturated memory pools or network queues when exposed

to high workloads. When applied to blockchains this effectively results in a vulnerability to DoS attacks[1].

Algorand and Solana are more robust as their throughputs are divided by 1.45 and 1.94, respectively, while the latencies of Algorand and Solana are multiplied by 2.43 and 4, respectively. These results show that despite being affected by a high workload, these two blockchains do not completely collapse, and the performance decrease likely results from the inability of the underlying hardware to handle too many requests. Interestingly, Avalanche throughput is not negatively affected by the higher workload, as its throughput is multiplied by 1.38, which makes it comparable to Solana throughput for the same workload. This confirms the conjecture about scalability that Avalanche throttles its throughput. It is hard to say something about Ethereum results since this blockchain only commits 0.09% of the transactions when the workload is 10,000 TPS.

Universality and DApp executions

To understand whether a blockchain is *universal* in that it can handle requests that are made arbitrarily complex, we test whether the blockchains can handle a large variety of DApps with a potentially complex execution logic. To this end, we first deploy the smart contracts of the DApps of Section 2.2 on the blockchains and then execute the real workload that invokes the functions of these contracts.

To test if a blockchain can execute arbitrary programs, we use the Mobility service DApp, which is CPU intensive and generates a 810–900 TPS workload during 120 seconds. We test whether the blockchains can provide the service delivered by Uber by measuring the throughput and latency and verifying that it matches the demand. As one can expect this workload to be more demanding than with the native transfers generated above, we deploy the blockchains in the `consortium` configuration (see Table 7.1), which has the same number of machines and the same network as the `community` configuration but with more powerful machines.

Figure 7.5 shows the throughput and latency for each blockchain running the computationally intensive Mobility service DApp on the `consortium` configuration. When the blockchain is unable to execute the smart contract, Figure 7.5 shows an **X** letter instead. Algorand, Diem, and Solana are unable to execute the DApp because the client reports an error of type `"budget exceeded"` indicating that the execution ran out of gas or timed out. This execution limit is hard-coded and cannot be lifted by paying a higher gas fee in the transaction. We conjecture this limit is hard-coded to prevent a

[1] Generating 10,000 TPS with Diablo-v2 costs less than 8 USD/hour on AWS.

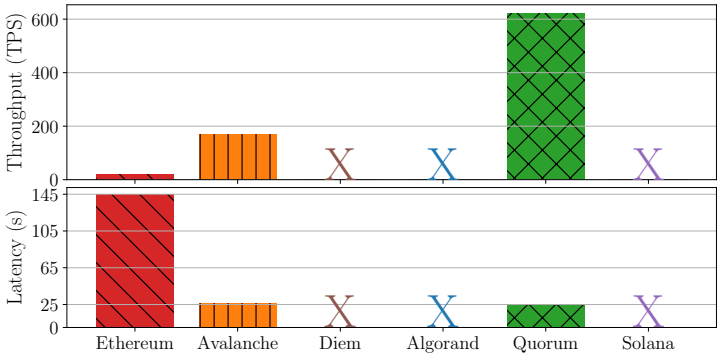


Figure 7.5: Throughput and latency with a workload between 810 TPS and 900 TPS

rich adversary from slowing down or completely stopping the blockchain by executing compute intensive tasks in smart contracts. Interestingly, the three blockchains able to execute the DApp use the `geth` implementation of the Ethereum Virtual Machine (EVM), which comes with no hard limit on the gas budget of a transaction. Over these three blockchains, Quorum has the highest throughput of 622 TPS which is close to the average workload, while the two other blockchains, Avalanche and Ethereum, have a throughput lower than 169 TPS.

Availability despite load peaks

We measure the *availability* of a blockchain as its ability to commit submitted transactions in a timely manner even when stressed with load peaks. A blockchain is more available when it handles more intense bursts of transactions with low latency and without dropping any transaction. This property is desirable for a blockchain to handle realistic workloads where users are likely to send many transactions to the blockchain at the same time and expect to receive a confirmation from the blockchain within a reasonable delay. To measure the availability of the blockchains, we first deployed each blockchain in the `consortium` configuration (see Table 7.1) and then generates short bursts of transactions of varying intensities, extracted from the Exchange DApp / Nasdaq workload. In particular, we use Diablo-v2 to send `buy` transactions at the same rate as the trade rate during the Nasdaq opening for 3 companies: Google, Microsoft, and Apple. Finally, we measure the proportion of dropped transactions and the latencies of committed transactions.

Figure 7.6 shows the cumulative distribution function (CDF) of the transaction latencies for all blockchains under three workloads, with a peak load of 800 tx (Google), 4,000 tx (Microsoft), or 10,000 tx (Apple) followed by a low workload. Over the tree workloads, only Quorum commits all the transactions. Specifically, when stressed with the Apple

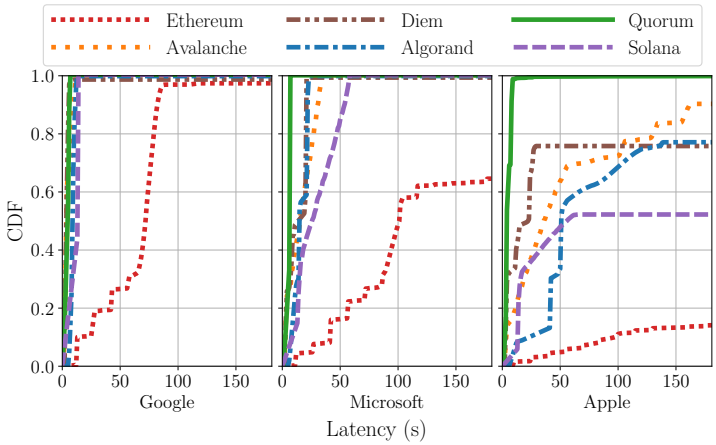


Figure 7.6: CDF of the transactions latency

workload, which consists of an initial load peak of 10,000 transactions during the first second, Quorum commits all transactions, among which 91% of the transactions are committed with a latency of 8 seconds or less. Interestingly, Quorum commits its transactions with similar latencies of 7 seconds or less when stressed with lower load peaks. Quorum uses IBFT, a deterministic BFT consensus that was historically designed to never drop a client request. We conjecture that this design choice is still present in the Quorum blockchain, as we already mentioned before.

The other blockchain based on a deterministic BFT consensus, Diem, only commits 75% of the transactions, all of them in less than 30 seconds. Diem drops transactions during the load peak because of the limited size of the mempool on each blockchain node. While this dropping mechanism prevents Diem from committing all transactions during high load peaks, it also makes it less prone to completely collapse during constant loads, as opposed to Quorum. Algorand and Solana also drop transactions, as shown by their CDF plateauing at 77% and 52% of committed transactions, respectively, whereas Avalanche and Ethereum keep committing transactions until the end of the experiment. While it takes up to 162 seconds for Avalanche to commit some of the transactions, this blockchain manages to commit 90% of the submitted transactions. Despite its low throughput, Avalanche is the second blockchain to commit the most transactions.

As opposed to the Apple workload, the Google workload presents an initial load peak of 800 transactions during the first second. As a result, all the blockchains commit more than 97% of the Google workload transactions. In addition, all the blockchains but Ethereum commit all the Google workload transactions in less than 14 seconds, while Ethereum does it in 118 seconds. The Microsoft workload has a moderate load peak of 4,000 transactions during the first second. On this workload, while all blockchains but

Ethereum commit all transactions, they take more time to do so, with the exception of Quorum, which commits all of its transactions with a latency of 7 seconds. Specifically, Solana has its maximum latency rising from 1 second for the Google workload to 59 seconds, while Algorand, Avalanche, and Diem have their maximum latency going from 10-14 seconds to 22-37 seconds. On the Microsoft workload, Ethereum commits only 64% of the transactions.

Discussion

Next, we summarize the key results of the evaluation. While the realistic DApp evaluation reveals that current blockchains are not yet ready to deliver the same performance as centralized infrastructures to provide common services, the in depth analysis of blockchains' performance shows that some blockchains fulfill some of their promises and identifies key factors of performance.

First, it appears that a blockchain using eventual consistency, like Solana, scales more easily to networks with many nodes. A decent throughput is also achieved by Quorum, a blockchain based on long studied consensus protocols but which also benefits from modern engineering techniques. More importantly, two blockchains, Diem and Avalanche, fail at using more challenging configurations most likely because they simply do not consider these configurations as a use case: high RTT networks for Diem and large hardware resources for Avalanche.

Second, the two blockchains using BFT consensus protocols, namely Quorum and Diem, are the most impacted by constantly high workloads. The Algorand, Avalanche, and Solana blockchains, which use either probabilistic or deterministic consensus protocols, maintain a non negligible throughput when stressed with high constant workloads.

Third, Quorum and Diem, the least robust blockchains in the face of peak loads, are also the blockchains committing the largest portion of transactions under reasonable delay. This seems to indicate that there is a tradeoff between robustness and availability.

Lastly, only the three blockchains using the `geth` implementation of the EVM, Avalanche, Ethereum, and Quorum, execute smart contracts with complex and computationally demanding logic. The other blockchains having a virtual machine with a hard limit on the computational cost of a transaction are unable to provide complex services.

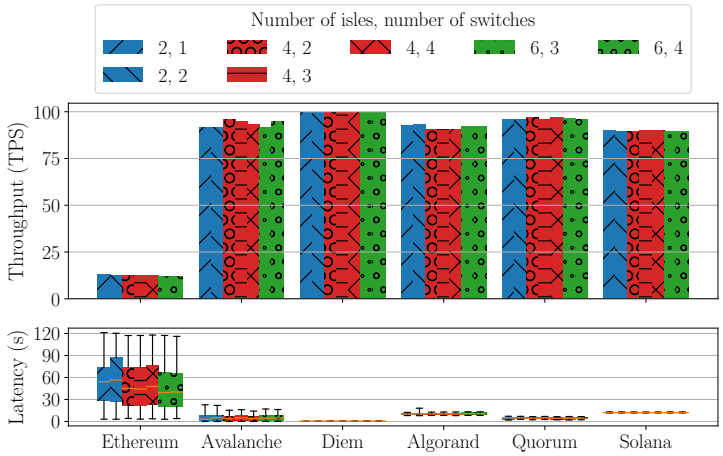


Figure 7.7: Throughput and latency, 100 TPS workload, varied number of isles and switches

7.2.2 iLab

With the local testbed, we focus on finer-grained small-scale experiments which look into how the network scales when the number of blockchain nodes is increased and how the network delay affects the performance of the protocols.

Inter-switch Communication

As we have two isles connected to a single switch, we have multiple possible configurations with the experiments involving two, four, or six isles. With two isles, they can either be connected to a single switch or be connected to two different switches. With four isles, two switches can be fully utilized, or there can be a partial utilization of three or four switches. With six isles, either three or four switches can be used. All of these configurations may affect the performance of the blockchain network.

In the next experiments, we send a constant workload of native transfer transactions over 2 minutes to the blockchain network.

Figure 7.7 shows the throughput and latency for each blockchain when stressed with a workload of 100 TPS. We can see that for all of the protocols, the measured throughput stays consistent and is not affected by possible delays added by the switches. For Ethereum, we see a slight decrease in the median latency as we scale up the number of blockchain nodes.

Next, we experiment with the same setups and a workload of 1,000 TPS in Figure 7.8. Here we start to notice a significant variance in results compared to the previous experiment. First, Avalanche fails to handle the workload, and the latency for the transactions the network manages to commit jumps from 7 seconds to 53 seconds, which is a 7.6 times

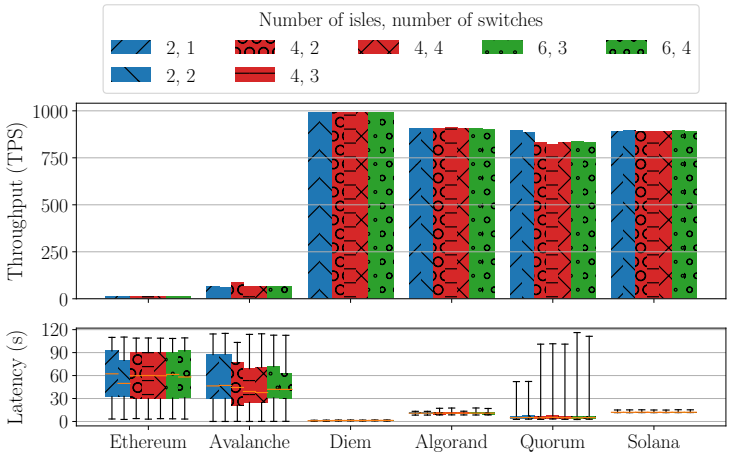


Figure 7.8: Throughput and latency, 1,000 TPS workload, varied number of isles and switches

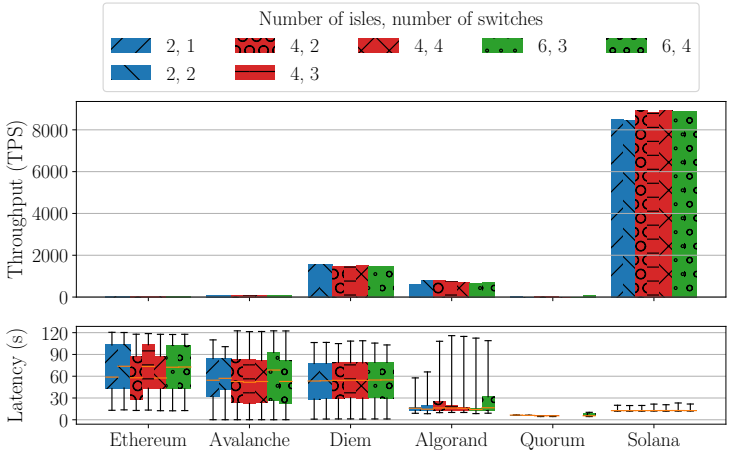


Figure 7.9: Throughput and latency, 10,000 TPS workload, varied number of isles and switches

increase in the average latency. We also notice that the maximum latency of committed transactions in Quorum starts to depend on the number of nodes. We previously noticed such behavior on a larger scale with Hyperledger Iroha [35].

Lastly, in Figure 7.9, we present the results of the experiments with the same setups and a workload of 10,000 TPS. Here, Solana shows the best results with regard to handling a very high workload.

Overall, we make a conclusion that the number of switches does not affect our measurements in a noticeable way and proceed to use the configuration with the minimal number of switches in the next experiments.

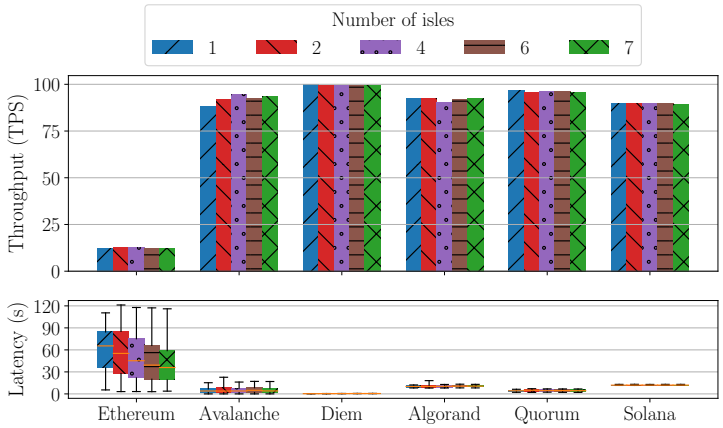


Figure 7.10: Throughput and latency, 100 TPS workload, varied number of isles

Isle scalability

To evaluate the scalability of the blockchain protocols in terms of the size of the network, we create networks of sizes 5 (1 isle), 10 (2 isles), 20 (4 isles), 30 (6 isles), and 35 (7 isles). We fully utilize the testbed, as it consists of 45 machines in total. We stress the network with the constant workload of native transfer transactions over 2 minutes with a varied rate.

In Figure 7.10, we compare the latency and the throughput of each protocol under the constant workload of 100 TPS. We again note that the measured throughput stays consistent between the different sizes of the network for all the blockchains. With Ethereum, we notice a pattern that the median latency tends to become smaller as the network size increases. We cannot increase the size of the network to observe the behavior further. However, the decrease becomes smaller with each network size increase.

Figure 7.11 shows the latency and the throughput for the blockchain protocols under test with the 1,000 TPS workload. The maximum observed latency in Quorum tends to increase with the size of the network. However, at the same time, the throughput does not have a noticeable impact.

We display the latency and the throughput of all the tested protocols in 7 configurations under a workload of 10,000 TPS in Figure 7.12. We observe an increase in the throughput in Solana as we use 20 nodes. This behavior can be explained by the fact that Solana uses the available processing power of the machines, and its performance scales with the available hardware. For Algorand, we reached its announced peak throughput in the experiment. For Diem, the relatively poor performance compared to Solana

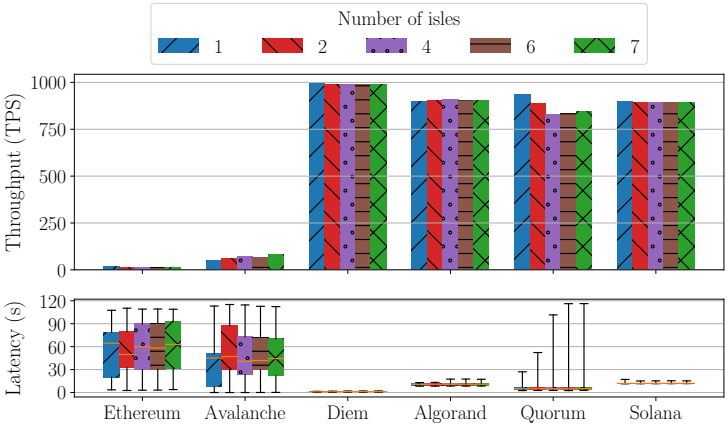


Figure 7.11: Throughput and latency, 1,000 TPS workload, varied number of isles

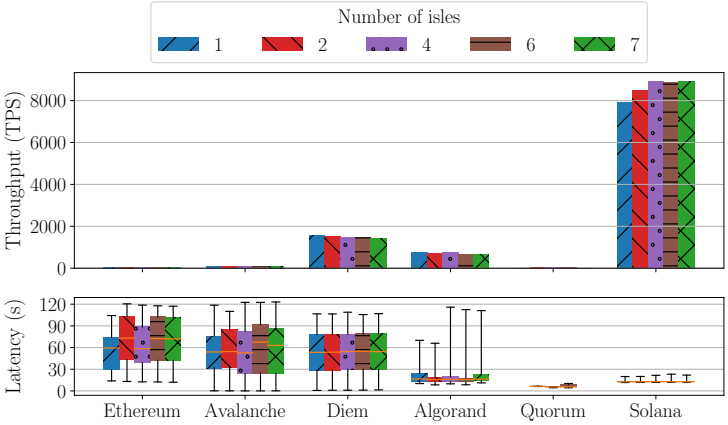


Figure 7.12: Throughput and latency, 10,000 TPS workload, varied number of isles

Added	0	50	100	150	200	250	300
Measured	1.13814	49.758	99.663	149.689	199.73	249.782	299.727

Table 7.2: Added and average measured RTT (ms) between the isles

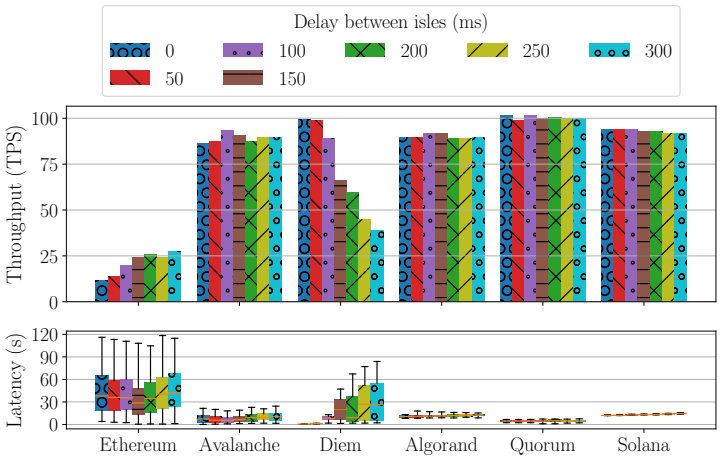


Figure 7.13: Throughput and latency, 100 TPS workload, varied delay between the isles

can also be explained by the experiment limitation regarding the number of available accounts.

Emulated latency

To evaluate the tolerance against network delays and simulate a real-world geo-distributed environment, we use a network of 35 machines, 7 isles in total. Each of the isles represents a separate location with a fixed delay to other locations. For simplicity, we use equal delay values for all the isles. We experiment with delays of 50, 100, 150, 200, 250, and 300 milliseconds.

In Table 7.2, we compare the added and measured RTT between the isles in the testbed to verify that the changes we did with `tc` are correctly applied in the whole network. We see the measured values slightly below the target value because we subtracted the baseline RTT from the added value as we were making the changes.

In Figure 7.13, we compare the throughput and the latency of the protocols under test with 100 TPS workload, and varied added delay between the isles. We can notice that the performance of Algorand and Quorum stays consistent regardless of the added delay. For Solana, the median latency increases from 12.01 to 14.53 milliseconds. The important observation is that the Diem performance drops significantly with the added delay, and the throughput decreases by more than 50%. We can infer from the observation

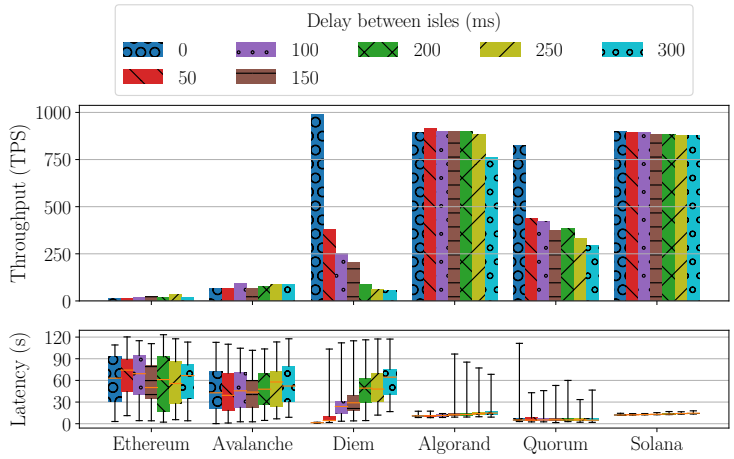


Figure 7.14: Throughput and latency, 1,000 TPS workload, varied delay between the isles

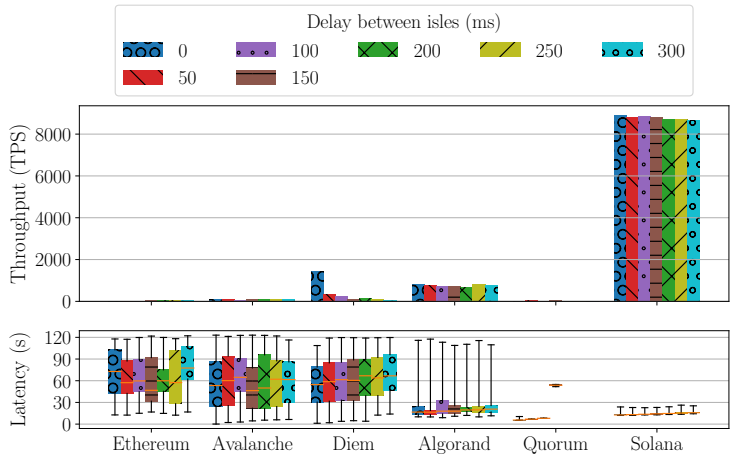


Figure 7.15: Throughput and latency, 10,000 TPS workload, varied delay between the isles

that the protocol was optimized for low-latency setups and is not suitable for real-world networks in the current state.

We display the latency and the throughput for all the tested protocols with 1,000 TPS workload in Figure 7.14. Compared to the previous workload, we see that the throughput of Quorum is halved as we add even 50 millisecond delay between the isles. At the same time, the latency stays the same for the different delay settings. For Diem, we see the same behavior of decreased throughput and increased latency. For the other protocols, the performance stays consistent with the increase of the delay in the network.

Figure 7.15 shows the throughput and the latency measures for the protocols under the 10,000 TPS workload. As before, Ethereum and Avalanche show minimal throughput,

and Quorum fails to handle the provided workload. We notice the same performance drop for Diem. Solana and Algorand show consistent performance regardless of the added delay.

7.3 Result Comparison

For the comparison of the results of the evaluation in AWS and iLab environments, we use the results of Sections 7.2.1, Scalability and deployment, and 7.2.2, Emulated latency with 1,000 TPS workload of native transfer transactions. The corresponding results are displayed in Figures 7.3 and 7.14. As we see in both figures, for the low-latency setups, as `datacenter` or `testnet`, Diem reaches its maximum throughput given the provided workload. On the other hand, when we increase the latency between the nodes and use geographically distributed regions as in `devnet` or `community`, we observe a significant drop in the throughput of Diem. The results correspond to the experiments with the increased latency in the iLab environment. As for the other blockchains, such as Algorand or Solana, they are optimized for the public networks and increased latencies, and therefore we don't see the drop in throughput, as shown in the figures for AWS and iLab environments.

7.4 Limitations

In this section, we look at the different aspects of the evaluation which can be taken into account in order to increase the depth of understanding of the blockchain protocols. While we performed an extensive set of tests in various environments, there are still more factors and variables that can be changed and which can affect the performance of the protocols.

First, we performed the experiments using the default system configurations supplied by the image provider. However, blockchain protocols like Solana recommend[1] different operating system tuning, such as increasing the size of UDP buffers, as Solana uses UDP for communication, or increasing the limit of memory mapped files. Such tweaks can significantly improve the performance of the protocol but should be examined separately for each protocol.

Second, protocols such as Algorand have different node types which have different modes of operation. Algorand separates relay nodes and participation nodes, where relay nodes

[1] `https://docs.solana.com/running-validator/validator-start#linux`.

are responsible for communication in the network, and participation nodes participate in consensus. In our deployment scenario, we ran both a relay and a participation node on each machine. Such topology can be suboptimal and not exactly represent a typical deployment. Instead, the Algorand main network can be analyzed, and such topology can be replicated in a private deployment for the performance evaluation.

Third, in our measurements, we calculate the throughput based on the transactions sent by Diablo. We store the hashes of the transactions and compare them to the hashes received in the block subscription or the query for the individual transaction. If the hashes match, we store the commit time of the specific transactions. Such metric only accounts for the transactions generated by Diablo. However, the Solana protocol includes voting transactions into the blocks, meaning that the calculated throughput can be higher if those transactions are included.

Another important point is that while we used the dynamic fee interface for Avalanche, we still observed that some transactions were dropped due to the insufficient fees specified in the transactions. There are multiple possible approaches to solve this issue. On the one hand, we tried to calculate the transaction fees online during the experiment run using the data provided by the blockchain network. It is possible that the approach was not perfect, and therefore the calculation logic can be reviewed and improved. On the other hand, it might be possible to specify static fees in the Avalanche configuration so that they do not provide overhead for the experiment, allowing only to benchmark the raw transaction processing performance. Also, while we experimented with C-Chain, it is important to measure the performance of X-Chain as well.

Lastly, the differences in the cloud environment and the lab testbed do not allow strict comparison of the metrics. Due to the hardware differences, we can only look at the tendencies and the order of change, but not the exact numbers.

Chapter 8

Conclusion

In this section, we summarize the work done in the thesis. We go into detail about how we addressed the research questions defined in Chapter 1. Additionally, we propose the directions for future work and how Diablo-v2 and Minion can be extended.

8.1 Summary

We were faced with the problem that the existing blockchain benchmark frameworks do not provide the ability to compare different protocols with real-world workload traces, making it hard for the application developers to choose the right solution for their use case and to evaluate the claims of the white papers. Furthermore, the currently present benchmark tools do not provide a universal deployment solution for local and cloud environments.

In order to solve the problem, we designed and implemented Diablo-v2, a blockchain benchmark framework, which extends the original Diablo prototype. In order to derive a generic architecture, we analyzed 6 state-of-the-art blockchain protocols, covering different aspects of their APIs. Diablo-v2 features a flexible workload definition syntax that accounts for different network topologies and the hardware of the machines running the benchmark.

We implemented Minion, the tool which allows deployment and orchestration of the blockchain networks and Diablo-v2. While Minion allows allocating the machines on Amazon Web Services, we extended it to support Plain Orchestrating Service, which is used at the local testbed at the Chair of Network Architectures and Services.

Lastly, we carried on an extensive evaluation of the blockchain protocols on Amazon Web Services and the local testbed using Diablo-v2. Using it both in the cloud setting with up to 200 geographically distributed machines and on a local testbed with the 6 blockchain protocols has shown that we addressed the requirements of the framework being generic, extensible, and easy to use.

8.2 Research Questions

In this section, we reiterate the answers to the research questions that we initially defined.

Which architecture would cover the protocol differences while being easy to use and maintainable?
Analyzing the Diablo prototype, we have seen that there are different factors that should be taken into account when designing a generic blockchain benchmarking framework, such as generating the workload and signing the transactions in particular. We reduced the prototype interface for Ethereum, which had 15 methods, to a concise interface with 4 methods which allowed us to integrate 4 different APIs. The strict specification of the API allows the developers not to overcomplicate their client implementation for a target protocol. The final interface design is described in Section 5.1.3.

How to distribute and scale the transaction load in the framework?
Diablo prototype has shown that simple workload definitions only provide basic functionality and cover a limited number of use cases. The workload definition derived in the thesis explicitly specifies blockchain endpoints and workload generator locations, which allows to easily control the communication patterns between Diablo-v2 and the blockchain network. Furthermore, the specification enables defining the generated workload individually for every workload generator, which creates a possibility to use machines with different hardware specifications. We show the workload file schema in Section 6.1.1.

What metrics are relevant across different blockchain protocols?
In Section 7.2, we showed that metrics such as transaction latency and throughput and the ratio of committed to submitted transactions could already provide significant insight into the performance of different blockchain protocols. For example, by analyzing the results of synthetic and realistic workload traces, we were able to make conclusions about the causes of specific protocol behaviors. We can conclude that BFT protocols are highly impacted by high workloads and that probabilistic protocols show better scalability when the number of clients is increased.

What are the differences between the local and the globally distributed test environments?
On the one hand, geo-distributed cloud providers allow experimenting with different hardware, having flexibility regarding the number of virtual CPUs and amount of RAM. Services such as AWS have data centers across the globe, which makes it possible to carry out experiments with realistic network conditions and background traffic spread across multiple continents. Local test environments, on the other hand, provide full control over the network conditions. Both environments play an important role in evaluating blockchain protocols, as they show different aspects of systems under test. We discuss the benefits and drawbacks of both environments in Section 4.2, and later show that we can compare the results from the environments in Section 7.3.

8.3 Future Work

While we focused on transaction-related metrics in the thesis, such as throughput, latency, and the ratio of committed transactions to submitted transactions, there are other important metrics that can help to compare different blockchain protocol implementations. Metrics such as network bandwidth, CPU, and disk usage can provide insight into how efficient is a particular implementation. These metrics can help to estimate the costs of maintaining a blockchain network over a long period of time. Moreover, blockchain users also perform reads on the network, in addition to transactional workload. Query performance might differ depending on when a particular value was written into the ledger. The generic architecture of Diablo-v2 and Minion allows the tools to be extended to support these metrics. The code to create queries has to be implemented Diablo-v2, similarly to how the transaction creation is currently implemented. Minion can be used to install the required software to collect CPU and disk metrics, run the tools along the blockchain binaries, and collect the metrics after the experiment execution.

Another aspect that can be studied is network partitioning. Partitioning is defined as the loss of connectivity between the different nodes of the blockchain network. Partitioning can affect aspects such as safety and liveness. From the safety perspective, the network can fork, meaning that the nodes start maintaining different versions of the ledger. Or the network can stop accepting transactions completely and fail to continue the normal operation after the network connectivity is restored, meaning that liveness property is violated. Here, `netem` also can be used to block communication between the different machines in the network.

As mentioned in Section 7.4, we used Algorand network topology where every physical machine had both relay and non-relay nodes running. Such a setup may not exactly replicate a real-world network and may create additional load on the network and the machines. Experiments can be done where only a subset of machines run the relay nodes responsible for communication routing. The other mentioned limitations can be resolved as well by conducting additional research.

Bibliography

[1] *Coinmarketcap*, Accessed:2021-05-06, 2021. [Online]. Available: `https://coinmarketcap.com/`.

[2] C. Natoli, J. Yu, V. Gramoli, and P. J. E. Veríssimo, "Deconstructing blockchains: A comprehensive survey on consensus, membership and structure", arXiv, Tech. Rep. 1908.08316, 2019. [Online]. Available: `http://arxiv.org/abs/1908.08316`.

[3] G. Wood, *Ethereum: A secure decentralised generalised transaction ledger*, Yellow paper, 2015.

[4] P. Ekparinya, V. Gramoli, and G. Jourjon, "The Attack of the Clones against Proof-of-Authority", in *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'20)*, 2020.

[5] *Avalanche: Blazingly fast, low cost, & eco-friendly*, Accessed:2021-12-06, 2021. [Online]. Available: `https://www.avax.network/`.

[6] T. Rocket, "Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies", Tech. Rep., 2018, Accessed on 11/26/2021. [Online]. Available: `https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV`.

[7] 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*, 2017, 1085–1100.

[8] *Hyperledger caliper*, Accessed:2021-05-06, 2021. [Online]. Available: `https://hyperledger.github.io/caliper/`.

[9] A. Krüuger, *Chainhammer: Ethereum benchmarking*, Accessed:2021-05-06, 2017. [Online]. Available: `https://github.com/drandreaskrueger/chainhammer`.

[10] R. Han, G. Shapiro, V. Gramoli, and X. Xu, "On the performance of distributed ledgers for internet of things", *Internet of Things*, vol. 10, 2019.

[11] G. Shapiro, C. Natoli, and V. Gramoli, "The performance of byzantine fault tolerant blockchains", in *Proceedings of the 19th IEEE International Symposium on Network Computing and Applications (NCA'20)*, 2020, pp. 1–8.

[12] T. Crain, C. Natoli, and V. Gramoli, "Red belly: A secure, fair and scalable open blockchain", in *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P'21)*, 2021.

[13] H. Benoit, V. Gramoli, R. Guerraoui, and C. Natoli, "Diablo: A distributed analytical blockchain benchmark framework focusing on real-world workloads", EPFL, Tech. Rep. 285731, 2021.

[14] A. Bacuet, "Comparative analysis of the scalability of the ethereum and algorand blockchains with the benchmarking of decentralised applications using diablo", EPFL, Tech. Rep., 2022.

[15] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies", in *Proc. 26th Symp. Operating Syst. Principles*, 2017, pp. 51–68, ISBN: 978-1-4503-5085-3.

[16] S. Bano, M. Baudet, A. Ching, *et al.*, *State machine replication in the libra blockchain*, Accessed: 2019-10-01, 2019. [Online]. Available: `https://developers.libra.org/docs/assets/papers/libra-consensus-state-machine-replication-\in-the-libra-blockchain.pdf`.

[17] J. Chase, *Quorum whitepaper*, en, Accessed: 2020-12-04, 2019. [Online]. Available: `https://github.com/ConsenSys/quorum/blob/master/docs/Quorum\%20Whitepaper\%20v0.2.pdf` (visited on 09/14/2019).

[18] A. Yakovenko, *Solana: A new architecture for a high performance blockchain v0.8.13*, Accessed: 2021-12-06, 2021. [Online]. Available: `https://solana.com/solana-whitepaper.pdf`.

[19] S. Gallenmüller*, D. Scholz*, H. Stubbe, and G. Carle, "The pos Framework: A Methodology and Toolchain for Reproducible Network Experiments", in *The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21)*, Munich, Germany (Virtual Event), Dec. 2021. DOI: `10.1145/3485983.3494841`.

[20] V. Gramoli, R. Guerraoui, A. Lebedev, C. Natoli, and G. Voron, "Diablo-v2: A benchmark for blockchain systems", EPFL, Tech. Rep. 294268, 2022, p. 14. [Online]. Available: `http://infoscience.epfl.ch/record/294268`.

[21] *Nasdaq*, Accessed: 2022-20-04, 2022. [Online]. Available: `https://www.nasdaq.com/`.

[22] Steam, *Dota 2*, Accessed: 2022-20-04, 2013. [Online]. Available: `https://store.steampowered.com/app/570/Dota_2/`.

[23] Statista, *Number of peak concurrent steam users from january 2013 to september 2021*, Accessed: 2022-20-04, 2021. [Online]. Available: `https://www.statista.com/statistics/308330/number-stream-users/`.

[24] A. Brodeur and K. Nield, "An empirical analysis of taxi, lyft and uber rides: Evidence from weather shocks in nyc", *Journal of Economic Behavior & Organization*, vol. 152, pp. 1–16, 2018.

[25] Statista, *Number of rides uber gave worldwide from q2 2017 to q4 2020*, Accessed: 2022-20-04, 2022. [Online]. Available: https://www.statista.com/statistics/946298/uber-ridership-worldwide/\#:~:text=In\%20the\%20fourth\%20quarter\%20of,percent\%20year\%2Don\%2Dyear.

[26] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: A view from the edge", in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2007, 15–28.

[27] Statista, *Hours of video uploaded to youtube every minute as of february 2020*, Accessed: 2022-20-04, 2022. [Online]. Available: https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/.

[28] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness", in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019.

[29] R. Saltini and D. Hyland-Wood, "IBFT 2.0: A safe and live variation of the IBFT blockchain consensus protocol for eventually synchronous networks", arXiv, Tech. Rep. 1909.10194, 2019.

[30] T. Hay, *Hyperledger besu: Understanding proof of authority via clique and ibft 2.0 private networks (part 1)*, Accessed: 2022-09-05, 2021. [Online]. Available: https://consensys.net/blog/quorum/hyperledger-besu-understanding-proof-of-authority-via-clique-and-ibft-2-0-private-networks-part-1/.

[31] A. Yakovenko, *Tower bft: Solana's high performance implementation of pbft*, Accessed: 2022-09-05, 2019. [Online]. Available: https://medium.com/solana-labs/tower-bft-solanas-high-performance-implementation-of-pbft-464725911e79.

[32] S. labs, *Solana confirmations*, Accessed: 2022-20-04, 2022. [Online]. Available: https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_state/mod.rs#L34.

[33] C. Natoli and V. Gramoli, "The balance attack or why forkable blockchains are ill-suited for consortium", in *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017, Denver, CO, USA, June 26-29, 2017*, 2017, pp. 579–590.

[34] M.-O. Pahl, "The iLab Concept: Making Teaching Better, at Scale", *IEEE Communications Magazine*, vol. 55, no. 11, pp. 178–185, 2017. DOI: 10.1109/MCOM.2017.1700394. [Online]. Available: http://ieeexplore.ieee.org/xpl/

articleDetails.jsp?tp=&arnumber=8114571&contentType=Journals+%26+Magazines.

[35] A. Lebedev, "Feasibility of blockchain-based logging and data processing in iot networks", Technical University of Munich, Interdisciplinary Project, 2021.