sustainability

Article

Cost Efficiency Evaluation of an On-Chain, Decentralized Ride-Sharing Platform

Georg Aschauer, Erik Sonnleitner * and Marc Kurz

Department for Smart and Interconnected Living (SAIL), University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg, Austria
* Correspondence: erik.sonnleitner@fh-hagenberg.at

Abstract: Traditional taxi providers are slowly losing their clients to online taxi providers, which are becoming increasingly popular. These companies typically provide a commercial, centralized platform in order to provide these services. The concept of centralization gives those providers significant power over their users, such as demanding and changing fees and collecting data over provided services. We aim to discuss how a decentralized approach using blockchains can solve those problems. A ride-sharing platform with the most essential functionality is implemented using a decentralized application based on smart contracts, which are, in turn, deployed on different blockchains. Various effects with respect to implementing and providing a decentralized ride-sharing service are taken in to account, especially regarding the effective cost for users. The fees of the developed platform heavily depend on which blockchain it is deployed on. Blockchains may compete in term of cost with traditional ride-sharing platforms, but choosing a suitable blockchain for the application is essential. Furthermore, the prototype shows that the time it takes for a transaction to be manifested into a block and subsequently confirmed is multiple times longer than a traditional web service response. These new problems require a new approach to implementing those requests that do not negatively affect the user experience.

Keywords: blockchain; smart contracts; decentralized applications; ride-sharing

check for updates

Citation: Aschauer, G.; Sonnleitner, E.; Kurz, M. Cost Efficiency Evaluation of an On-Chain, Decentralized Ride-Sharing Platform. *Sustainability* **2023**, *15*, 6230. https://doi.org/10.3390/su15076230

Academic Editor: Fausto Cavallaro

Received: 8 February 2023
Revised: 22 March 2023
Accepted: 1 April 2023
Published: 4 April 2023

1. Introduction

Online ride-sharing platforms became very popular in the past decade and are slowly replacing traditional taxi services [1]. These ride-sharing platforms usually use centralized servers to handle the communication, payment, and other aspects of a ride. The most prominent representatives of these online ride-sharing platforms include Uber, Lyft, and many more. However, significant problems and concerns persist:

- *Privacy*: Since the providers of centralized platforms have complete control over the servers that host their services, this allows them to collect all the user data which is exchanged during the rides of their users [2]. This data includes GPS information of all drivers and riders, the exact time of each ride, phone numbers, and much more.
- *Centralization*: Since the providers of centralized ride-sharing platforms have complete control over the servers on which the services run, this gives them much power. This power includes changing user conditions, such as changing the fee per ride at any time. It also allows the provider to ban any user from using their services at any time without reason. None of this is in the interest of the actual users of the platform.

1.1. Hypotheses

We form the hypothesis, that a fully decentralized and trust-less (i.e., without central authority) system based on public blockchains may be able to fulfill all major requirements typically associated with commonly known centralized alternatives, e.g., the ones given above. We strive to achieve a privacy-preserving implementation by only storing the minimum amount of data required, most of which may also be stored in an encrypted state. The decentralized approach has no effect on cost or usability for drivers and riders. There are also no compromises in the primary feature set offered by a decentralized ride-sharing platform compared to a centralized one.

1.2. Goals and Research Questions

We aim to implement a prototype of a decentralized ride-sharing platform on a public blockchain. This prototype includes at least one frontend application through which users can interact with the platform, similar to a traditional one. This prototype should enable clients—carpooling service providers referred to as drivers as well as users wanting to use such a service referred to as riders—to operate without depending on a central system or company. This decentralized approach also means that conflicts between both parties must be handled without a third party in the implementation. Specifically, the following research questions should be answered:

1. **Research Question 1**: How do the transaction costs of a decentralized ride-sharing platform implemented on Ethereum and other compatible blockchains compare to fees charged on a centralized platform such as Uber on a per-ride basis?
2. **Research Question 2**: Are there any noticeable effects on user experience when using a frontend application that interacts with smart contracts compared to centralized services?

1.3. Outline

This work is organized into seven chapters. Section 2 explains the basic concepts of blockchain technology and smart contracts. Section 3 discusses related work on blockchains, smart contracts, and decentralized ride sharing. It discusses some ideas to implement privacy and handling conflicts on a decentralized ride-sharing platform. Section 4 describes how a decentralized ride-sharing platform is implemented. This implementation includes smart contracts, a unit test suit, and two frontend applications. Section 5 describes how data is collected from the implemented platform to compare it to a decentralized platform. Section 6 describes how the collected data is used to compare the centralized platform and what the results mean, before drawing a conclusion in Section 7.

2. Technical Background

This chapter explains the technical concepts of blockchain technologies and smart contracts, and covers some tools and standards used to develop smart contracts.

2.1. Dapp-Enabled Blockchains

The idea of a blockchain was first described in the Bitcoin white paper [3]. It describes a peer-to-peer network open for everyone to join that can handle payments without relying on a central entity. Ref. [4] describes blockchain as the technology which solves the double-spending problem in decentralized digital cash systems. It archives this by clever use of cryptography. Information such as transactions is stored in a block that links to a single previous block by their hash, thus creating a chain. Users can have accounts referred to as wallets, which consist of public and private key pairs. Transactions are signed by the private key of a user. Those blocks are immutable and transparent. Different blockchains use different mechanisms to archive those properties [3,5].

However, not all blockchains allow for the building and running of decentralized applications (or *Dapps*). Three of the more prominent ones are listed below:

- **Ethereum**: The corresponding white paper [6] describes the idea of an alternative blockchain with a different focus than Bitcoin. Its goal is to run Turing Complete smart contracts on that blockchain. It uses a different ledger model compared to Bitcoin, the account model. The state of each account is stored in a Merkle tree and includes a counter to ensure each transaction is only processed once, the actual balance of the account, in the case of a smart contract its byte code, and the state information of the smart contract [7]. Transactions create a new state for the involved accounts in the next block. Unlike Bitcoin, Ethereum stores the transaction as well as the current state as mentioned previously in each block. Since only small portions of the state stored in the Patricia tree change with each new block, this allows the new state to reference unchanged branches of the tree via their hashes. This mechanism lets miners delete old blocks from their disk since all the state information can be found in the last block. In [8], M. Bez et al. raise some concerns about Ethereum's scalability. Every transaction has to be processed by every node on the network. This redundant processing means that as the blockchains' state grows, fewer nodes have the required hardware to do so, which means the network's decentralization suffers.

- **Polygon** is a blockchain project which attempts to solve the scaling problems of Ethereum. It claims full compatibility with Ethereum applications by supporting smart contracts written for the EVM (Ethereum Virtual Machine) and providing a way to exchange messages with the Ethereum network. The core component of the project is the Matic sidechain described in the corresponding whitepaper [9]. This side chain uses proof of stake as a consensus mechanism and a much lower block time than Ethereum. Every few blocks, a checkpoint is created. A checkpoint is created by fitting all block hashes from the previous checkpoints into the Merkle tree. The root hash of that Merkle tree is then validated by other members of the network and published to a particular smart contract on the Ethereum network. After the root hash is published to Ethereum, other members of the network can challenge the published hash for a certain amount of time before it becomes a final checkpoint.
- **Harmony One** [10] is another blockchain project that attempts to provide better scalability than Ethereum while preserving security and decentralization. Their solution focuses on sharding. Harmony One uses proof of stake as its consensus mechanism with some differences from other proof of stake blockchains, such as using multi-signature to collect and aggregate the validators' votes [11].
  The network consists of a beacon chain as well as multiple shards, each with a subset of validators. The beacon chain provides the randomness function for selecting a block proposer in the shard as well as the identity information. The shard chains store their state and handle transactions indecently from each other. Blocks are grouped in epochs. Every epoch the validators are randomly assigned to a new shard.

2.2. Smart Contracts

2.2.1. Concept and Context of Contracts

The idea of smart contracts was first mentioned by Nick Szabo [12,13]. He describes how contracts written on paper, typically enforced by governments, are the very foundation of the free market economy. He proposes a new digital form of contracts called smart contracts. He describes the four principles when designing smart contracts as observability (everyone should be able to see and prove the execution of a smart contract); verifiability (it should be possible to prove that a smart contract has been executed or failed); privity (knowledge and control over the smart contract should only be described to as many parties as necessary, whereas each additional party with control is a new attack vector); and enforceability (reducing the need for enforcement to a minimum.).

Nick Szabo [12] also describes how cryptographic protocols are the foundation of archiving these principles and explains how they can solve them in detail. He also mentions digital currencies and smart properties. Smart contracts can enable a transfer of ownership of physical property via digital representations; for example, a car could only get started with a digital key which can be transferred via a smart contract in exchange for digital currency.

A smart contract is a script stored on a blockchain [14], and is created by sending a transaction containing the code in some form to an address. Each contract has its unique address on the network. A user can send a transaction to that given address to invoke the smart contract, which is then subsequently executed by every network node receiving the transaction, with the data included in the transaction serving as arguments. Ethereum [5] was one of the first blockchains to support implementing Turing Complete smart contracts and stores the byte-code of deployed smart contracts and their state in Merkle Trees.

Transactions in Ethereum require gas, which is paid in the native asset of Ethereum, Ether [5]. The higher the complexity of the code executing in the smart contract, the higher the required amount of gas, as described in great detail by Mokaluse et al. [15]. Any excess gas after the code execution is sent back to the user. If the user does not send enough gas with the transaction for the code execution, the smart contract's state is rolled back, but all gas is consumed. This mechanism attempts to prevent denial of service attacks on Ethereum since the execution of smart contracts stops once there is no gas left, and gas is an expensive resource. Smart contracts can also invoke other smart contracts, a process which is implemented and activated through messages. Messages are similar to transactions, but can never come from an external source. Such a message only contains the unspent gas of the original transaction sent to the smart contract [7]. The code of smart contracts is typically written in the domain-specific language Solidity and compiled to a bytecode language called the Ethereum Virtual Machine code (EVM). The bytecode of a contract is then executed as infinite loop, until either the execution runs out of gas or it reaches a return statement.

2.2.2. Standards and Tools

Solidity is a primary high-level, object-oriented, statically typed programming language used to develop smart contracts for Ethereum and other EVM compatible blockchains, as described in [16,17]. Although alternatives exist, Solidity is the most prominent language for smart contract development.
Truffle is a JavaScript framework for building, deploying, and testing decentralized applications on Ethereum and other EVM-compatible blockchains [18]. It allows for creating applications containing multiple smart contracts, compiling, linking, and generating the corresponding API in JSON format.
Metamask [19] is a browser extension that handles account management. It allows user to store their private keys in different ways, such as through hardware wallets, while allowing to connect web applications to an Ethereum-compatible blockchain.

3. Related Work

There are multiple blockchain concepts, but most blockchains either follow the concept of being implemented permissionless (i.e., open for everyone to participate) or permissioned (i.e., restricted access). Wüst and Gervais [20] discuss these different categories of blockchains, how their properties compare to centralized databases and how to determine if blockchains fit a use case. They further compare the properties of public verifiability, transparency, privacy, integrity, redundancy, and trust anchor between blockchains and centralized databases. Their work also proposes a guideline to determine whether a blockchain is a suitable solution for a problem. They say that to choose a blockchain as a solution, and where the problem requires multiple parties that do not trust each other, it needs to store some form of state information, and cannot agree on a trusted third party, as shown in their flow chart shown in Figure 1. If all these requirements are met, their work also shows which type of blockchain is a fitting solution. Following the flow chart for the use case of a decentralized ride-sharing platform, we would come to the conclusion that a permissionless blockchain is a fitting solution.

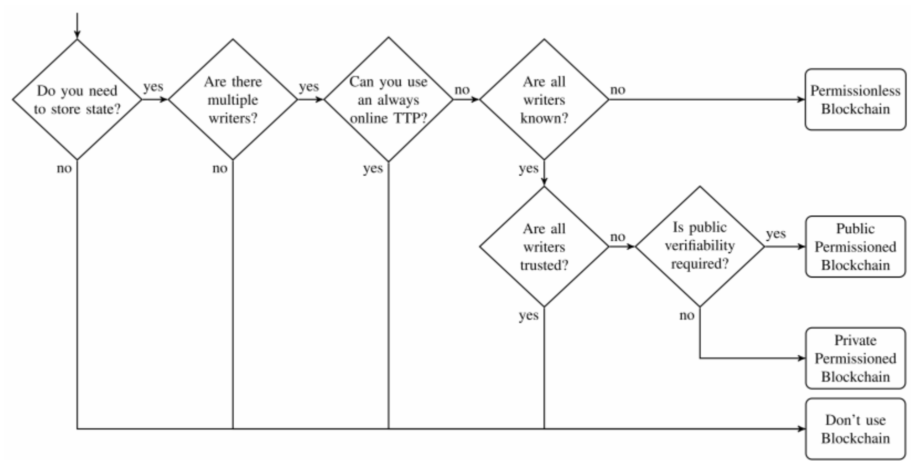
Do you need to store state?
yes
no
Are there multiple writers?
yes
no
Can you use an always online TTP?
no
yes
Are all writers known?
no
yes
Permissionless Blockchain
Are all writers trusted?
no
yes
Is public verifiability required?
yes
no
Public Permissioned Blockchain
Private Permissioned Blockchain
Don't use Blockchain

Figure 1. Flow chart to find out whether a blockchain is a suitable solution for a given problem. Reprinted from Ref. [20]. Copyright 2018, Wüst et al.

Sánchez [21] first proposed a fully decentralized ride-sharing system. Their approach uses a peer-to-peer network, where the drivers and riders are the members. In order to preserve privacy, their idea is only to save generalized location and time data. This is done by a technique called spatial cloaking. A region is split into cells which can be based only on any suitable criteria. For example, a city could be split into districts, as shown in Figure 2. The same approach is used for the time data by creating time cells with a certain interval. One problem in their approach is that the driver learns the real identity of the rider and can then track his future rides [22].

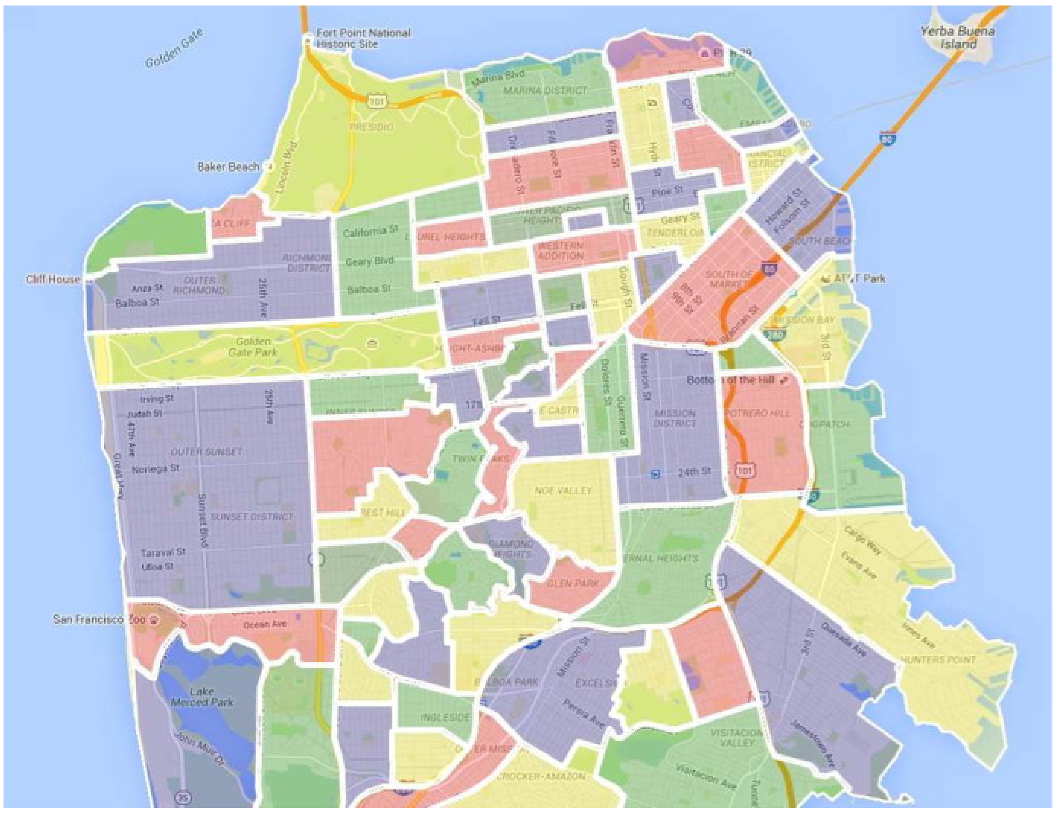

Figure 2. Spatial cloaking example for San Francisco, where partitioning is done by neighbourhoods. Reprinted from from Ref. [21]. Copyright 2016, D. Sanchez et al.

Semenko and Saucez [23] propose an alternative decentralized ride-sharing platform using a blockchain. Their idea is to build an overlay network that includes all business actors. Those business actors are referred to as service nodes and are responsible for matching riders and drivers. In their approach, nodes of the network do not have a real incentive to act in the interest of the driver or riders, as payment for the rides is still supposed to be done in cash.

Baza et al. [24] discuss how a simple decentralized ride-sharing platform could be implemented on a blockchain, although their definition of what a ride-sharing platform should be differs from the one used in this work. In their definition of a ride-sharing platform, drivers allow riders to join on routes or parts that they would drive anyway. Their work focuses on preserving the privacy of the users of the platform, with their main concept being based on the idea mentioned in [21]. They also use spatial cloaking to generalize the data in cells. They use static squares to group location data into cells, as seen in Figure 3. The rider, therefore, only provides a region cell and a time cell when looking for a ride.

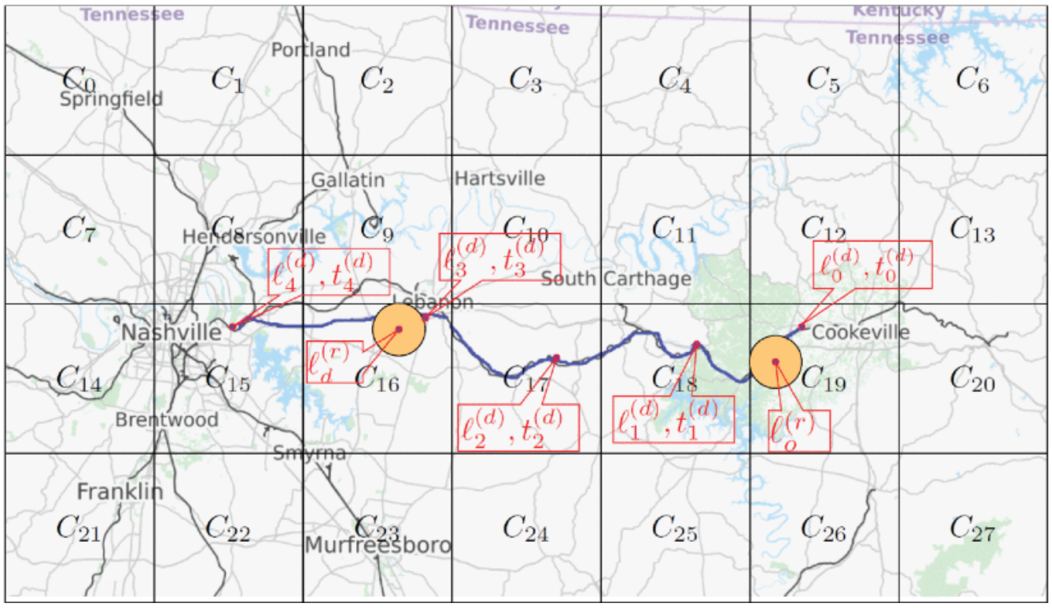

Figure 3. Visualization of a route using spatial cloaking example for Tennessee, where partitioning is done by static squares. Reprinted from Ref. [24]. Copyright 2020, Baza et al.

Drivers can look for requests in cells contained in their route or are near their planned route and then send an offer. The offer includes the exact location and exact time, encrypted by the public key of the rider as well as the price. The price is not encrypted in order to increase competition and provide the lowest price for the rider. The rider can then decrypt the received offers with his private keys and select an offer that fits his destination time and location requirements. When joining the ride, the rider has to authenticate to the driver. This is done by zero-knowledge proof, where the rider proves that he owns his private key without giving away any information about the key itself. The rider is also supposed to use a new key pair for each ride in order to prevent any tracking. The work also does some evaluation of the costs per trip of their implementation on the Ethereum blockchain. The review was done with the prices at the time of September 2019. They take a gas price of 0.5 Gwei and Ether price of USD 208 and conclude that a trip would cost around USD 0.02 at that time.

Baza et al. [22] also built upon their previous work in a new prototype called B-Ride. They introduce small deposits for both the rider and the driver. Those deposits are sent to a smart contract when both parties agree on a ride. If the driver does not show up at the defined location at the defined time, the deposits are sent to the rider; otherwise, they are sent to the driver. They also discuss how zero-knowledge proofs can be used to prove that the driver arrived at the defined location without exposing any private location data on the blockchain. This is done in the same contracts to which the deposits are sent.

Their work proposes that during the ride, the payment is made periodically. Upon the beginning of the ride, the rider calls a smart contract with his funds, where the previous mentions deposits act as a down payment. Subsequently, the driver publishes data on the driven distance. If the rider agrees with the data, it is sent to the contract signed by both party members. The smart contracts then send funds from the rider to the driver. This entire process can be seen in Figure 4.

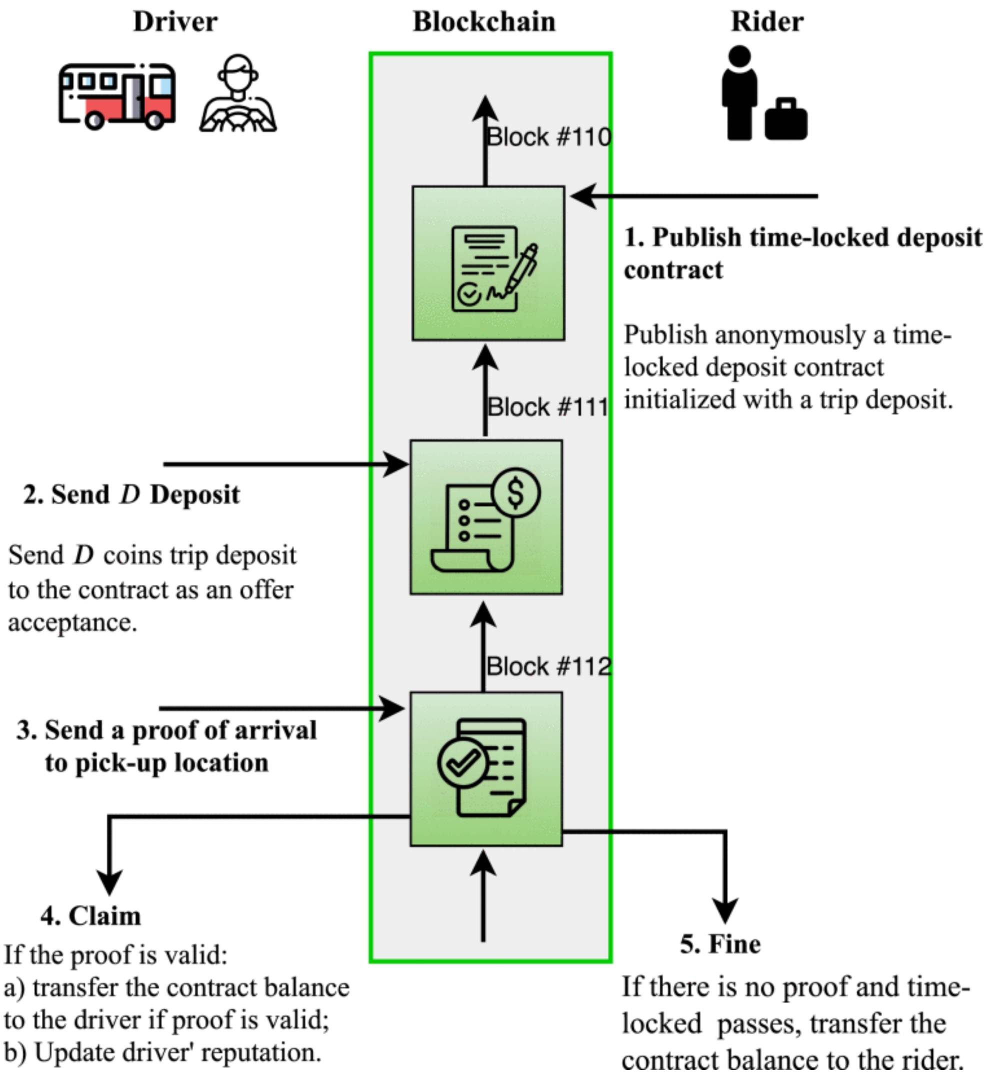
Driver
Blockchain
Rider
Block #110
Block #111
Block #112
1. Publish time-locked deposit contract
Publish anonymously a time-locked deposit contract initialized with a trip deposit.
2. Send D Deposit
Send D coins trip deposit to the contract as an offer acceptance.
3. Send a proof of arrival to pick-up location
4. Claim
If the proof is valid:
a) transfer the contract balance to the driver if proof is valid;
b) Update driver' reputation.
5. Fine
If there is no proof and time-locked passes, transfer the contract balance to the rider.

Figure 4. Visualization of the concept of B-Ride, from [22].

They also discuss how a reputation system can be implemented on a decentralized platform. The idea is the reputation of a driver increases each time he arrives at the pick-up location in time as well as each time he arrives at the destination. In order to prevent drivers from creating new contracts when getting a negative reputation, they have to deposit some funds, which can be used to refund drivers if they run away before or during the ride.

In their evaluation of the improvement platform, they conclude that while decentralized ride-sharing is definitely possible, there are still many unsolved problems, such as transactions privacy, throughput on proof of work blockchains, and legal enforcement. Moreover, sustainability in proof of work based blockchains remains a general problem, as Rieger at al explain in [25].

Pal et al. [26] discuss the implementation of BlockV, a concept of how a ride-sharing platform could be implemented on the Ethereum blockchain. It focuses on a fair distribution of rides and how disagreements between driver and rider could be handled fairly within smart contracts. They discuss an algorithm in which both the driver and rider store the destination in a smart contract as well as send location data continuously during the ride to a smart contract. The rider also sends funds equal to the agreed price to the smart contract upfront. The rider can call a function in the smart contracts if a disagreement happens before reaching the agreed location. The smart contract then uses the saved locations to resolve the conflict. If the data from the ride and driver show differences beyond a certain threshold, it is assumed that the ride ended. The driver can also call the contract if the rider leaves during the ride. In this case, the driver receives funds equal to the distance driven, and the rest of the funds are refunded to the rider.

They also evaluated the cost of the ride of their implemented smart contracts on Ethereum. They used the price of April 2019, at which time one Ether was around USD 141 for their calculations. They only calculate the price for individual transactions, ranging from USD 0.02 to USD 0.45 rather than the cost for an actual ride.

Bathen et al. [27] discuss a decentralized ride-sharing environment for self-driving cars. In their work, they propose a new blockchain with a conscience mechanism called proof of matching, which is targeted at the hardware in autonomous vehicles (AVs), which are specialists in computer vision and machine learning. Proof of matching is similar to the proof of personhood protocols [28]. The idea is that users can register themselves by taking multiple selfies. These images are then transformed multiple times, and a machine learning model is trained to identify the person. When a rider enters an AV, it validates that the rider is registered by taking a picture and running the trained machine learning model. If it matches it, then it verifies the result as well as GPS data of the endorsers. Endorsers will verify both the identification as well as the GPS data. By doing so, they receive a small reward if the transaction makes it into a block. A validator is chosen to propose a proof of match, which is a signed hash of the root Merkle tree storing the transactions as well as a random nonce. The other validators then validate that block by comparing the hash to the one calculated. If it is valid, the endorsers and validators are rewarded, and the AV is paid.

4. Implementation

This chapter describes the implementation of a ride-sharing platform prototype which will later be used for evaluation. The platform consists of smart contracts and two frontend applications. It also explains which requirements those components have to meet, as well as how they are implemented.

4.1. Requirements

4.1.1. Functional Requirements

Functional requirements describe functions or features which have to be implemented in order for the prototype to be usable.

- **Registration** of a driver, including the vehicle used.
- **Registration check** required for all drivers and users.
- **Ride request** from a user’s current location to a particular target.
- **Ride offering** and pricing is done by drivers after viewing all requests.
- **Payment** is done by the user upfront to the driver’s smart contract.
- **Pull over Push** principle defines that the driver has to withdraw the funds manually from his contract.
- **Accessing the user’s wallet** by connecting it to the platform frontend.

4.1.2. Nonfunctional Requirements

- **Efficient Smart contracts**: Transactions, storage, and deployment of smart contracts are expensive. Therefore, smart contracts must be implemented as efficiently as possible.
- **Security**: Especially since one smart contract stores funds of the user, the implementation must be secure. The security requirements with regard to blockchains has been thoroughly discussed [29,30]. Therefore, possible vulnerabilities in the implementation of smart contracts will be exploited. Unit tests must be used to make sure the transaction works as intended and ensure no such exploits are possible.

4.1.3. Chain of Use Cases

1. **Register a driver** including car and phone information.
2. **Request a ride** from a user's current location to a given target, including the number of seats required.
3. **Offer a ride** The driver gets notified of a ride request and offers a ride at his price.
4. **Accept a ride** The rider can accept a ride offer. The price for the ride has to be transferred into the smart contract of the driver.
5. **Complete a ride** The rider completes a ride. The driver has access to the price paid by the user through his smart contract.
6. **Refund rider** The driver can refund the rider.

4.2. Smart Contracts

4.2.1. Design

The smart contracts were designed to be as efficient as possible while providing all the required functionality and security. This approach means storing as little data as possible in each smart contract. The smart contracts were also designed in a way that additional functionality, such as on-chain conflict resolution, could be added to implementation without changing the complete architecture. Figure 5 shows the three designed smart contracts:

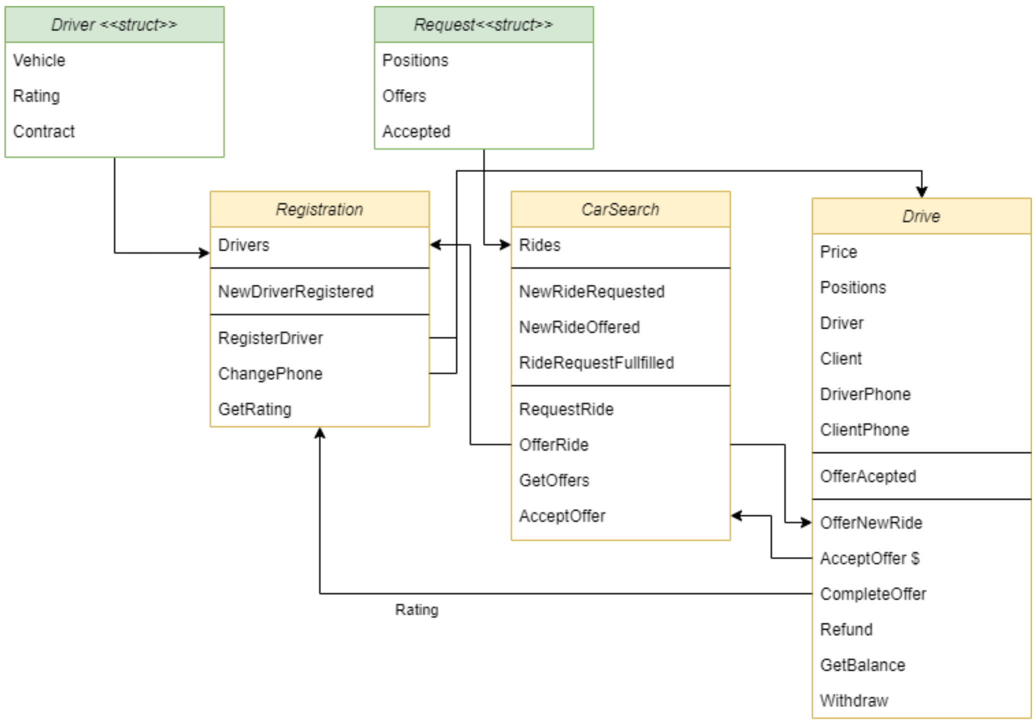

Figure 5. The architecture of decentralized ride-sharing prototype smart contracts.

- The `Registration` contract handles the registration of drivers. A driver can register by calling the `RegisterDriver` function and providing information about the vehicle he is driving, such as the number of seats and his phone number. Doing so creates a new struct entry in the `Registration` contract, and a new `Drive` contract is deployed for the driver. The struct also stores the rating information of the driver, which can easily be accessed via a function.
- The `CarSearch` contract handles the distribution of rides on the platform. Riders can call the `RequestRide` function with location data for their planned ride. Doing so creates a new `Request` struct and adds to the smart contract, and the `NewRideRequested` event gets fired. Drivers can subscribe to that event and create an offer for the request. This results in their `Drive` contract being parameterized for the drive, the offer being added to the request struct, and the `NewRideOffered` event firing. If the rider misses

the event, they can still call the `GetOffers` function to receive all offers for their current request. The Offers field of the request structs contains the address of a `Drive` contract.

- The `Drive` contract handles the logic during a drive as well as the payment. A rider can accept an offer by calling the `AcceptOffer` function, which also requires an upfront payment equal to the defined price. By doing so, the Drive contract also calls the `CarSearch` contract and sets the `Accepted` flag in the request struct of the rider so that other drivers know that the rider is no longer looking for offers. If everything goes as planned, the rider can call `CompleteOffer` at the end of the ride and give the driver a rating. Doing so completes the ride. The Refund function can be used to handle simple conflicts and rides which are canceled midway through.

4.2.2. Privacy

While the implementation focuses on efficiency, it provides some privacy options based on the ideas from [24], especially regarding the user's GPS and phone data.

While the rider has to provide some GPS data when requesting a ride, it is possible to only provide rough data instead of an exact location. The driver can be notified at a later point in time about the exact location via a different communication channel that exists off-chain.

The `Drive` contract stores information which contains both the phone number of the rider and the driver. To provide extra privacy, the application and/or users can choose to encrypt the phone number stored in the smart contract with the public key of the opposite party. This ensures only the other party can actually decrypt the phone number.

4.2.3. Implementation and Deployment

The first step when implementing smart contracts is to actually choose a blockchain for which the contracts should be implemented.

In this case, Ethereum was chosen as the primary blockchain because it provides Turing Complete smart contracts, which are a requirement. It has also proven in the last decade to be one of the most stable and decentralized blockchains out there. Writing the smart contracts for Ethereum also brings the advantage that since the smart contracts run in the EVM, they can also relatively easily be deployed to the other blockchain which supports the EVM. For this reason, Polygon and Harmony One are chosen as alternative blockchains to deploy the contracts to. The previously described architecture for the ride-sharing platform is implemented in Solidity.

The Truffle development framework was used to group the smart contracts into a project. Since both the Registration and `CarSearch` contract need to communicate with each other, this creates a sort of cyclic dependency. In order to resolve this cyclic dependency, a deployment script has to be implemented. This deployment script first deploys the `Registration` contract and then saves the address of the deployed Registration contract. In the next step, it deploys the `CarSearch` contract and parses the saved address of the deployed `Registration` contract in the constructor. Lastly, the script sets the address of the `CarSearch` address in the `Registration` contract via a function. The function in the `CarSearch` contract, which sets the address to the `Registration` contract, is secured so that only the address which deployed the contract can call this function and therefore change the address. This functionality can also be used to redeploy a single contract of the decentralized application and connect it with the other smart contracts.

In order to deploy the smart contract to different blockchains and test environments, the configuration file of the truffle project has to be set up. Since key-value pairs with funds are required to pay the deployment cost, this sensitive information must be parsed for the Truffle project. This is done by storing the mnemonic phrase, which is used to create the key pairs in a hierarchical deterministic wallet in a separate .secret file. The content from the file is read and parsed to the HDWalletProvider, which is part of the Truffle suite. The configuration file also contains additional settings for the deployment target.

Confirmations The number of confirmations between the deployment of smart contracts. Using a value of two is very common for the test environment because it is highly likely that there is no other longer chain. When deploying to a blockchain's main network, choosing a higher value might make sense. The drawback is, of course, that the deployment will take more time.
TimeoutBlocks The number of blocks waited on to mint the transaction containing the deployment of the smart contract. The value of 200 means that after 200 blocks, the deployment attempt is canceled. A total of 200 blocks would take around 50 min on Ethereum. This configuration option ensures that if the gas prices change, we can reattempt deployment with a higher gas price.
SkipDryRun Whether the migration should be executed locally before deploying the actual network. Since we deploy to a test network and we already test our application via unit tests, we can set this flag to yes. When deploying to the main network, this might be useful as an extra layer of security that deployment configuration actually works.

Lastly, the Truffle config file also contains the information about the language version of Solidity, which the compiler should target. The smart contract can now be deployed via `truffle deploy -network [Network]` where `Network` is the key of the defined config block.

4.3. Web Application Design

This section describes the implementation of a web application allowing the user of the decentralized ride-sharing platform prototype to interact with the smart contract. The idea is that the platform should feel and behave similarly to a traditional web application.

The web application for the decentralized ride-sharing prototype is designed as straightforwardly as possible while still providing all the required possibilities to interact with the smart contracts in a user-friendly way. The flow of the web application is structured as follows: The user enters the application on the home page. From there, it is possible to choose whether the user is a rider or a driver. If the user is a driver, it checks if the address they are using is already registered. If the driver is not registered, they are navigated to the registration page. If they are already registered, they are navigated to the driver page. Here they can create offers for riders. The driver is navigated to the driver page if an offer is accepted. Once the drive is completed, the rider is navigated to the account page.

If the user is a rider, they are navigated to the rider page. On this page, the rider can pick the location from which they want to get picked up and where they want to go. Once the ride request is sent, offers are also shown on the page. Once an offer is taken and completed, the rider is also taken to the drive page. On this page, the rider can see all the essential information, and they can also complete the ride on this page. Figure 6 shows the entire navigation flow of the web application prototype.

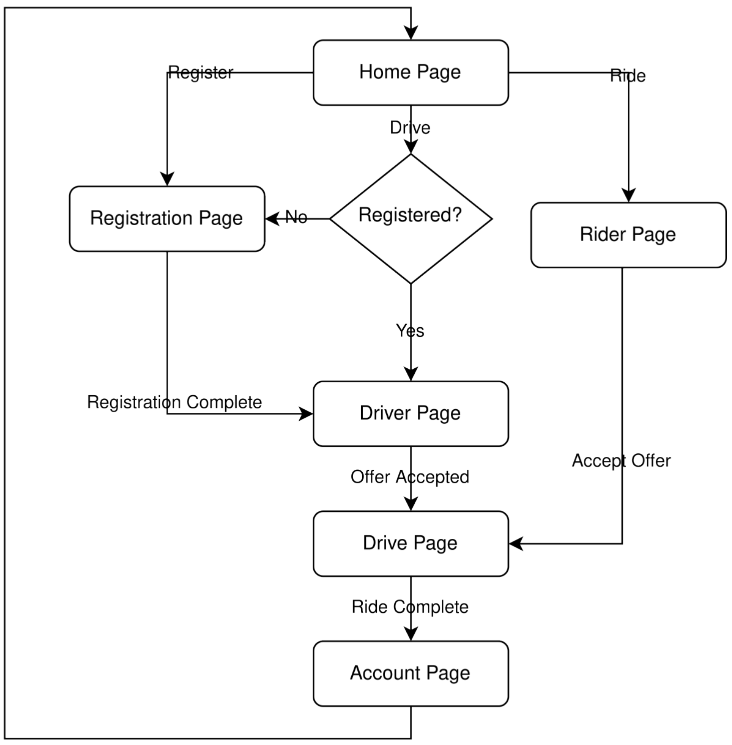

Figure 6. Navigation flow through the pages of the web application.

The pages themselves provide the following functionality:

- The **Home Page** shows a logo if the user's wallet setup is correct and the application is successfully connected to it. Otherwise, an error message is shown. The header displays the address of the user with whom he is connected to and the blockchain network.
- The **Registration Page** checks if the user is already registered at the Registration contract. If that is the case, his information is displayed, and the user has the possibility to change his information. Otherwise, allow registration.
- The **Driver Page** allows the driver to browse open ride requests in a list, including location information. The driver can create an offer by clicking on one of the requests.
- The **Drive Page** page shows relevant information for the pickup and ride, especially phone number and the exact location marked on a map.
- The **Account Page** shows the driver's information about his `Drive` smart contract. It displays the contract's balance stored in the contract in native coin and current fiat currency values. The driver has the option to withdraw the funds from his contract to his wallet.
- The **Rider Page** allows to search for a ride. It displays the driver's information as well as the cost. The page displays a map where the current location is already marked as the source location, but that may be changed by the user. Ride requests can be confirmed, and ride offers created and displayed. The rider can choose to accept an offer by clicking on it.

4.4. Implementation

Various implementation details differ from implementing a web application communicating with a blockchain compared to a traditional centralized backend web service.

Connecting to the wallet

The web application relies on a valid connection to the blockchain as well as wallet addresses, which are used to sign and pay for transactions. Therefore, the application must ensure that those prerequisites are met, because if they are not met, the web application is not functional. The root component of the application handles this checking. When the root component of the web application is initialized, it checks if there is a wallet that supports the standard Ethereum object, such as Metamask. If there is an Ethereum object, a message is displayed to the user that they require such a wallet to use the web application. Next, it is also required that the user unlocks the wallet for the specific web application in order for the application to access the account of the user. A similar message is displayed to the user if the wallet is not unlocked. Furthermore, an event is registered to listen when the user unlocks his wallet to change the state of the application automatically. A react hook is used to run the logic only a single time when the application is loaded. The `detectEthereumProvider` function is used to locate the Ethereum JavaScript object, which is called provider in this case if available. If that is not the case, a component with a corresponding message is displayed. The display of this message is handled through the `isWeb3` state. In the next step, we try to access the accounts via the Ethereum object. If it is impossible because the wallet is locked, the `isWeb3Locked` state is set, and another message is rendered again.

Creating smart contract objects

The TruffleContract objects representing the deployed smart contracts are the core concept of interaction with the smart contracts. They are required for most of the functionality the web application provides. Therefore, they need to be created, initialized, and stored within the web application. In order to create a smart contract object through which the application can call the smart contract, it is necessary to load the ABI of the deployed contract. The ABI in JSON form is then parsed to the TruffleContract function, creating a proxy object that can be used to call the actual smart contract. On the created object, the provider and default account from the funds taken are required. Whenever a smart contract call requires a transaction, the proxy object prepares this transaction and parses it to the provider, which signs it if the user confirms the transaction. In our case, where Metamask or a similar browser extension is the provider, it shows a popup to the user whenever a transaction requires signing.

Listening to events

The events provided by EVM smart contracts are a clever way for the application to get notified whenever something changes without polling. This feature is handy when waiting for the other party to act and get a notification when they do so. With the truffle contract, this is done by calling the event on the proxy object and providing a callback function. This function is invoked whenever the smart contract triggers its event. The function takes an error and data as parameters, which allows the logic to process the parameter of the event fired by the smart contract. Calling the event function also returns an object representing the subscription itself. It is crucial to properly unregister that callback function once the logic is no longer needed because the component is no longer visible, for example. Not doing so can lead to memory leaks and potentially the whole application not working correctly. In ReactJs this can be done with cleanup functions, which can optionally be returned from a react hook. This function is then called when the component is unmounted. In this cleanup function, the subscription to the events is simply canceled. Figure 7 shows how the event is used by the application to receive new ride requests for the driver when a rider sends a new request.

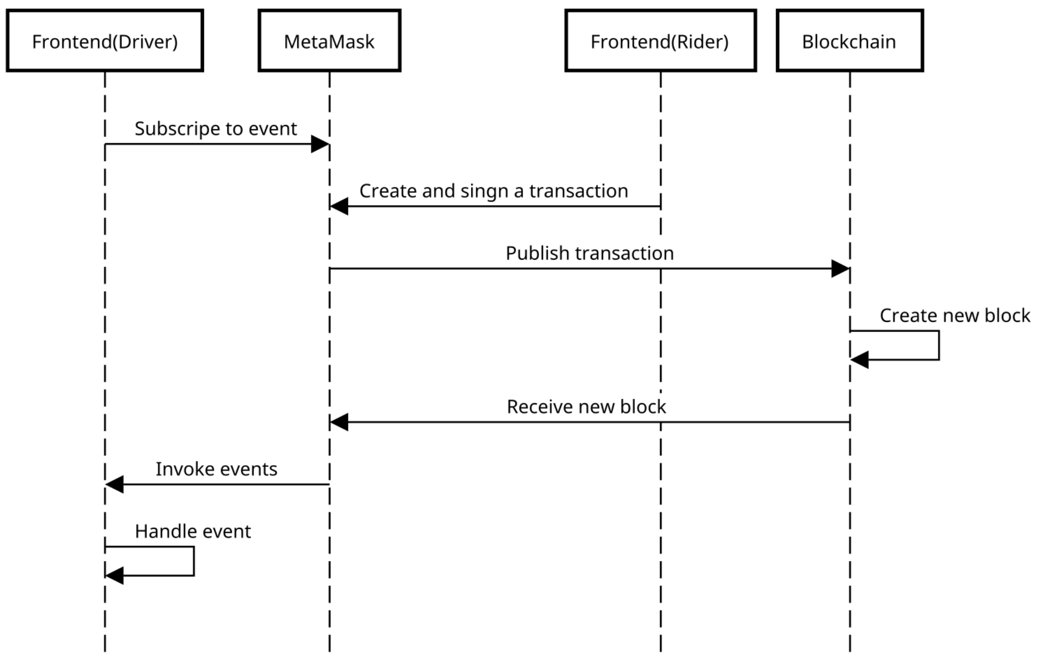

Figure 7. Illustration of how the events are used in order to receive new ride requests in the frontend application without polling.

We plan to release the source code of the entire project using an open source, free software license once the research project is finished. Until then, the code of the core contracts described in this chapter can be downloaded separately for further analysis (https://drive.google.com/file/d/1lKk37tjOO0JXwDLik7O9HRkmuDWxZPSm, accessed on 3 April 2023).

5. Gathering Data

This chapter explains how data are gathered in order to evaluate the defined research questions. The significant steps of collecting the data and how the data are transformed to be able to make comparisons are explained.

5.1. Transaction Costs

Each transaction in Ethereum and most other EVM-compatible blockchains costs money in the form of the native currency of that blockchain. Transaction costs for smart contract calls can be described as follows:

$$\text{gas fee} = \text{gas required for the transaction} \times \text{gas price}$$

In order to calculate the price in fiat currency, such as USD, we need to multiply it with the cost per native currency.

$$\text{fiat price} = \text{gas fee} \times \text{fiat price per unit}$$

The following describes how the values of these formulas can be collected for the implemented prototype.

5.1.1. Gas Required for Smart Contract Transactions

In order to calculate the gas fee, it is essential first to determine the gas required for each transaction invoking a smart contract call. The amount of gas a smart contract transaction will consume is 100% deterministic and can be determined by two factors. The first factor is the number of CPU cycles required to execute the smart contract code, and the second factor is the number of changes in state in the storage scope. This means the gas required for a smart contract transaction could be calculated by looking at the code and the parameters invoking a specific function of the smart contract. The simplest way to determine the gas required for smart contract transactions is to use the EVM. For each transaction, the caller receives a receipt of the transaction, which also includes the amount of gas used. In order to prevent spending real currency, this can also be done on a local test blockchain environment, such as Ganache (https://trufflesuite.com/docs/ganache/index.html (accessed on 3 April 2023)).

Unit tests are one of the simplest ways to create transactions that can be rerun and collect the receipt. So, an additional unit test suit was created in the Truffle project. This unit test suit exports the gas used field stored in the transaction receipt.

5.1.2. Effects of Different Parameters on the Required Gas

While this approach allows for the collection of the value of the `gasUsed` field in transactions, there is still a problem with this data. Since the `gasUsed` depends on the CPU cycles required to execute the smart contract code as well as the state changes in the storage scope, this means not every call of the same function will use the same amount of gas (i.e., the parametrization used during contract invocation may lead to significantly varying requirements of gas and, hence, total cost). However, even just calling a function with a different parameter and executing the same code will result in a different amount of gas used. Branches will cause completely different results depending on which branch is executed. Fortunately, our implementation of the smart contracts described in this proposal does not rely on branches.

The total amount of gas consumed varies based on parsed parameters and the smart contracts themselves. If it has a noticeable effect on the described implementation, they were called with the different parameters. Table 1 shows the gas used by the `registerDriver` function with different parameters. There we can see that parsing an empty string results in the transaction consuming almost 20,000 Wei less than the other descriptions because this is the default value, and thus the storage state is not changed. Among the other descriptions, the difference is only 280 Wei, which is less than 0.02 % of the gas used. This small difference means that the difference in the amount of gas used for useful inputs for the smart contract call is irrelevant in this case. One reason that might be the case here is that since the smart contract call creates a new smart contract and therefore requires much gas, the difference made by the input is small in comparison.

Table 1. Amount of gas used by registerDriver function with different values for the vehicleDescription parameter.

Vehicle Description	Gas Used
Empty String	1,484,028
SUV	1,503,411
SUV-4 Door	1,503,519
1999 Toyota RAV 4 Oceanblue	1,503,699

In order to figure out whether or not the difference is also insignificant for other smart contract calls, a similar experiment was performed on a cheaper contract call. In order to do this, we selected the `searchRide` function of the **CarSearch** smart contract for further testing. This call takes four arguments representing the coordinates, either encrypted or not. For the test, the same value was applied for all four arguments in order to show the maximum variance. Since the EVM does not support floating values, the coordinates are represented as a 256 bit integer. The application uses the factor of 100,000,000 to convert it into an actual coordinate value. Table 2 shows the amount of gas used for the different values. The value 0 is noticeably cheaper than everything else. This is because 0 is the default value of an integer, and thus, the smart contract call does not actually alter the positions stored within it and requires significantly less gas. While 0 is technically a valid value, it is highly unlikely that a coordinate actually is 0; therefore, this case will be ignored in further consideration. The difference between the remaining values is about 1500 gas or about 1.4% of the amount of gas. The flat difference is higher than `registerDriver` because it has four parameters. The percentage is higher because of the flat difference and the fact that the transaction itself is more than ten times cheaper than the `registerDriver` call.

Table 2. Amount of gas used by searchRide function with different values for the position parameter.

Position	Gas Used
0	32,595
1	109,443
150,000,000	109,587
−150,000,000	110,931
$2^{255} - 1$	110,931

While 1.4% is notable, it is worth mentioning that this is the function with the highest difference in gas usage when being executed with different parameters. Therefore, for this and all other calculations of gas used by a smart contract, calling the mean value among the valid, realistic values are used. Table 3 shows the amount of gas used by each smart contract function in the decentralized ride-sharing platform in an EVM environment. Functions not listed in the table are either views or pure views and can thus be called and evaluated for free. It shows that `registerDriver` required by far the most gas, mainly because it deploys a new smart contract for the driver. `OfferRide` requires the second most gas but still six times less than `registerDriver`. This is because it changes a lot of state information in the storage scope of both the **CarSearch** contract as well as the **Drive** contract. The withdraw and refund calls are very close to the base cost of a transaction which is 21,000 Wei, because they only do marginally more than send native tokens from the smart contract to a user's address. Figure 8 shows the adapted approach for estimating the gas usage of all smart contract functions of the implemented prototype for a decentralized ride-sharing platform. Only plausible parameters are used as inputs, as while using an empty value for a value is possible and might drag the average down, it is not a realistic situation.

Table 3. Amount of gas used by smart contract functions.

Smart Contract	Function	Gas Used
Registration	registerDriver	1,503,000
CarSearch	searchRide	110,000
CarSearch	offerRide	204,000
Drive	acceptOffer	93,000
Drive	completeOffer	28,000
Drive	withdraw	22,000
Drive	refund	23,000

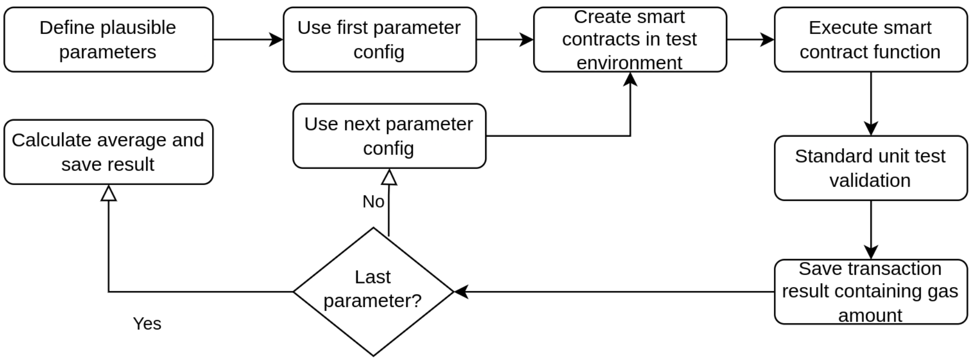
Define plausible parameters
Use first parameter config
Create smart contracts in test environment
Execute smart contract function
Standard unit test validation
Save transaction result containing gas amount
Last parameter?
No
Use next parameter config
Yes
Calculate average and save result

Figure 8. Final approach to collection data for gas usage of smart contract functions.

5.1.3. Gas Price

The next piece required to calculate the actual transaction cost is the gas price. The gas price paid in a transaction determines the priority of the transaction in the network. Therefore, a higher gas price compared to other transactions means a higher priority, while a low gas price might mean that the transaction takes a long time to be accepted or might never be accepted. Since the gas prices of Ethereum and other blockchains tend to vary a lot, it was decided to use the period from 1 January 2022 to 31 March 2022 to estimate the gas price.

- **Ethereum**: For each day, the average gas price paid per transaction, which was successfully written in blocks on the Ethereum blockchain, was analyzed. Table 4 (a) (Data taken from https://ycharts.com/indicators/ethereum_average_gas_price (accessed on 3 April 2023)) shows statistics of daily gas cost on Ethereum on that time span. We can see that the maximum of 218.55 is more than eight times higher than the minimum of 26.29. Therefore, the gas price is considered highly volatile.
- **Polygon**: For each day, the average gas price paid per transaction, which was successfully written in blocks on Polygon 4Matic blockchain, was analyzed. Table 4 (b) (Data taken from https://polygonscan.com/chart/gasprice (accessed on 3 April 2023)) shows statistics of daily gas cost on the Polygon 4Matic network on that time span. The volatility on the Polygon network is even worse than on Ethereum. The maximum on Polygon 4Matic is 763.63, which is more than 15 times higher than the minimum of 44.64.
- **Harmony One**: It is hard to find reliable data on the gas price of Harmony One. While they claim their gas prices to be as low as 10 Gwei (https://docs.harmony.one/home/general/technology/transactions (accessed on 3 April 2023)), because of their higher throughput lately, most of the recent transactions show a higher gas price in the range of 30–70 Gwei. Therefore, a simple experiment was done, where ten transactions from each day form the average for that day. While this is not 100% accurate, this gives a pretty good estimation of the gas prices of Harmony One. Furthermore, only shard 0 was used for this experiment. The gas price on other shards, especially shard 2, seems to be more expensive. Table 4 (c) shows the statistics from the obtained data. The maximum is only 2.5 times higher than the minimum, meaning the gas prices on Harmony One are much less volatile than the other two blockchains in the defined period.

Table 4. Statistics for the daily gas prices from 1 January 2022 to 31 March 2022 of (a) the Ethereum network, (b) the Polygon 4Matic network, and (c) shard 0 on the Harmony One network.

Statistic Operation	(a) Ethereum	(b) Polygon 4Matic	(c) Harmony One
Mean	94.84	136.28	30.89
Median	84.05	87.88	30.00
Min	26.29	44.64	19.00
Max	218.55	763.63	51.00
StdDev	47.00	143.48	3.67

5.1.4. Price of Native Assets

The price of the native assets is required to convert the gas prices into a fiat currency. For each of the three chosen blockchains, the price of the native asset was analyzed over the same time span (Ether data taken from https://ycharts.com/indicators/ethereum_price; Polygon 4Matic data taken from https://www.investing.com/crypto/polygon/historical-data; Harmony One data are taken from https://www.investing.com/crypto/harmony/historical-data (all accessed on 3 April 2023)). Table 5 (a)–(c) shows the prices of those assets in USD. We can see that while the price of the native assets is not as volatile as the gas prices, the prices themselves are still volatile. We can also see that because Ethereum is the most popular smart contract blockchain at the moment, is by far the most expensive of the three native assets, being on average 1500 times more expensive than Harmony One and 15,000 times more expensive than Polygon.

Table 5. Statistics for the daily token prices in USD from 1 January 2022 to 31 March 2022 for the (a) Ethereum, (b) Polygon 4Matic, and (c) Harmony One networks.

Statistic Operation	(a) Ethereum	(b) Polygon 4Matic	(c) Harmony One
Mean	2942.47	0.19	1.73
Median	2924.93	0.18	1.63
Min	2407.38	0.12	1.36
Max	3835.40	0.35	2.57
StdDev	346.35	0.07	0.31

5.1.5. Gas Prices in USD

The gas prices on the chosen networks, as well as the price of native assets, have proven to be highly volatile. There is some consent that there is a correlation between gas prices and native assets because high asset prices mean lower gas prices, and high gas prices mean lower asset prices. Therefore, the average daily gas price was calculated and analyzed. Table 6 (a)–(c) shows the gas price in USD for the three chosen blockchains. We can see that while the minimum and maximum gas price and price of native assets definitely did not occur on the same days, the volatility is still way higher than the gas price and price of native assets themselves. This means the price in USD paid for a transaction can heavily vary depending on the day of the transaction. We can also see that in terms of gas price in USD, Ethereum is a lot more expensive than both Polygon 4Matic and Harmony One.

Table 6. Statistics for the daily gas prices in USD from 1 January 2022 to 31 March 2022 for the (a) Ethereum, (b) Polygon 4Matic, and (c) Harmony One networks.

Statistic Operation	(a) Ethereum	(b) Polygon 4Matic	(c) Harmone One
Mean	0.00028	2.34×10^{-8}	5.32×10^{-8}
Median	0.00024	0.00024	5.02×10^{-8}
Min	6.73×10^{-5}	6.87×10^{-9}	3.99×10^{-8}
Max	0.00069	1.23×10^{-7}	1.08×10^{-7}
StdDev	0.00015	1.94×10^{-8}	1.02×10^{-8}

5.1.6. Transaction Prices in USD

With the previously gathered information, it is now possible to calculate the actual prices of each transaction in USD. This calculation is done by simply multiplying the gas price in USD with the gas required for each transaction. The Tables 7–9 show the calculated prices in USD for each transaction on Ethereum and Polygon 4Matic, as well as Harmony One. We can see that the transaction prices of the Polygon 4Matic and Harmony One blockchain are relatively similar, with Harmony One being around twice as expensive on average. For Ethereum, on the other hand, the average transaction is more than 5000 times more expensive than on Harmony One.

Table 7. Transaction prices on Ethereum from 1 January 2022 to 31 March 2022.

Smart Contract	Function	Mean Price USD	Min Price USD	Max Price USD
Registration	registerDriver	425.1189	101.2673	1037.0036
CarSearch	searchRide	31.1132	7.4114	75.8951
CarSearch	offerRide	57.7008	13.7449	140.7510
Drive	acceptOffer	26.3048	6.2660	64.1659
Drive	completeOffer	7.9197	1.8866	19.3188
Drive	withdraw	6.2226	1.4823	15.1790
Drive	refund	6.5055	1.5497	15.8690

Table 8. Transaction prices on Polygon 4Matic from 1 January 2022 to 31 March 2022.

Smart Contract	Function	Mean Price USD	Min Price USD	Max Price USD
Registration	registerDriver	0.0353	0.0103	0.1858
CarSearch	searchRide	0.0026	0.0008	0.0136
CarSearch	offerRide	0.0048	0.0014	0.0252
Drive	acceptOffer	0.0022	0.0006	0.0115
Drive	completeOffer	0.0007	0.0002	0.0035
Drive	withdraw	0.0005	0.0002	0.0027
Drive	refund	0.0005	0.0002	0.0028

Table 9. Transaction prices on Harmony One from 1 January 2022 to 31 March 2022.

Smart Contract	Function	Mean Price USD	in Price USD	Max Price USD
Registration	registerDriver	0.0801	0.0601	0.1627
CarSearch	searchRide	0.0059	0.0044	0.0119
CarSearch	offerRide	0.0109	0.0037	0.0101
Drive	acceptOffer	0.0050	0.0006	0.0115
Drive	completeOffer	0.0015	0.0011	0.0030
Drive	withdraw	0.0012	0.0009	0.0025
Drive	refund	0.0012	0.00029	0.0024

5.1.7. Summary

A special unit test suit was used to gather data about the amount of gas for each transaction, invoking a smart contract call for the implemented decentralized ride-sharing platform. Different parameters affect the amount of gas used by a smart contract call. These effects are minor, and additionally, the average of multiple realistic parameters is used to obtain a realistic estimate. Data about the gas price and price of native assets were gathered in order to calculate the price per transaction in USD. The data show that Ethereum transactions are way more expensive than on Polygon and Harmony One. The data also show that the price of a successful transaction varies a lot depending on the data the transaction would be created on.

5.2. Transaction Prices on Centralized Ride-Sharing Platforms

In order to compare the costs of a centralized and decentralized ride-sharing platform, it is required to also gather data on the prices of centralized services such as Uber and Lyft.

- **Uber**: Ref. [31] shows that while Uber claims to only charge a 25% fee depending on the region, they charge up to 42.75%. However, this percentage is based on the total cost of the ride service, not the markup on the base price of the ride. This difference is significant because in order to make a proper comparison with the decentralized ride-sharing platform, extract the base price and the fee as a markup percentage. If we calculate the fee depending on the base price of the service, it comes up to a mark up of 33.3% to 74.6%.

- **Lyft**: Lyft is currently the main competitor to Uber, attempting to offer a cheaper alternative. Ref. [32] shows that Lyft, on the other hand, charges a commission fee of 20%. This fee equals a markup of 25%.

Centralized ride-sharing platforms usually charge a percentage fee of the total price of the ride. Uber charges a markup of 33.3% to 74.6% of the base price per ride. Lyft, on the other hand, only charges a markup of 25%.

6. Evaluation and Results

This chapter explains how the defined research questions defined in Section 1 are evaluated. Furthermore, this chapter also shows the results of each of those questions and discusses them.

6.1. Research Question 1: How do the Transaction Cost of a Decentralized Ride-Sharing Platform Implemented on Ethereum and Other EVM-Compatible Blockchains Compare to Fees Charged on a Centralized Platform Such as Uber on a Per-Ride Basis?

6.1.1. Scenarios

In a centralized ride-sharing platform, fees are typically charged as a percentage of the ride's total cost. In a decentralized ride-sharing platform, such as the one implemented, fees depend on the number and complexity of the transaction involved in completing the ride. Therefore, in order to use that data, we need first to create a scenario that defines which and how many transactions are typically used in a ride on the decentralized ride-sharing platform. The decentralized ride-sharing platform also requires a registration transaction for the driver, for which they must pay fees. The fee for this registration transaction is relatively high, meaning that depending on the number of rides the driver completes, this transaction alone might cost them more than the number of the fees of drives themselves. Table 10 shows how much the amount of gas, and thus the price per ride, changes depending on how many rides the driver does after completing their registration. We can see that if the driver only does a single ride, the registration transaction would be the cause of 78% of the total cost of the fees. After 100 rides, the registration becomes only 3% of the total cost of the fees, which is less than most other transactions and therefore acceptable. Because not every driver will provide a service for 100 or more riders on a decentralized ride-sharing platform, two scenarios are created: an optimistic and pessimistic one.

Table 10. Amount of gas used per ride depending on the number of rides since the registration of the driver.

Number of Rides	Gas per Ride	Used by Registration
1	1,938,000	78%
10	585,300	26%
100	450,030	3%
1000	436,503	0.3%

6.1.2. Optimistic Scenario

In this scenario, we assume everything goes pretty well for both the driver and rider. We assume that the rider does 100 rides after registration. Moreover, we also assume that every request of a driver gets at least one acceptable offer from a driver and that 50% of the offers created by the driver are accepted by a rider. We further assume that no refund is required and that the driver only withdraws their funds every 50 rides. Table 11 shows the transaction required for a single ride in this scenario and the amount of gas required for those transactions. Adding the amounts of gas of those transactions together equals 654,470 gas, which is required for a single successful ride in this scenario.

Table 11. Transaction required for a successful ride in the optimistic scenario and the gas cost.

Smart Contract	Function	Transaction per Ride	Gas per Ride
Registration	registerDriver	0.01	15,030
CarSearch	searchRide	1	110,000
CarSearch	offerRide	2	408,000
Drive	acceptOffer	1	93,000
Drive	completeOffer	1	28,000
Drive	withdraw	0.02	440
Drive	refund	0	0

6.1.3. Pessimistic Scenario

In this scenario, we assume people act a lot less ideally than in the optimistic scenario. We assume that the rider only does ten rides after registration, and that 50% of the requests of the driver get no actual acceptable offer by a rider. We further assume that only 20% of the offers that drivers create are accepted by a rider and that every tenth ride requires a refund and that the rider withdraws his fund every ten rides. Table 12 shows the transaction required for a single successful ride in this scenario and the amount required for the transactions. The transaction sum equals 1,515,800 gas, which is required for a single ride. We can see that the amount of gas in the pessimistic scenario is more than double the amount in the optimistic scenario.

Table 12. Transaction required for a successful ride in the pessimistic scenario and the gas cost.

Smart Contract	Function	Transaction per Ride	Gas per Ride
Registration	registerDriver	0.1	150,300
CarSearch	searchRide	2	220,000
CarSearch	offerRide	5	1,020,000
Drive	acceptOffer	1	93,000
Drive	completeOffer	1	28,000
Drive	withdraw	0.1	2,200
Drive	refund	0.1	2,300

6.1.4. Results

The fees of a centralized, decentralized ride-sharing platform cannot be compared directly. The reason for this is that they typically have different fee structures. Centralized ride-sharing platforms such as Uber and Lyft charge a percentage fee depending on the ride's total cost. Decentralized ride-sharing platforms deployed on the blockchain, on the other hand, charge fees per transaction. Since a ride typically consists of the same number of transactions, the fee for a ride is always the same and is therefore considered a flat fee.

Uber charges a markup fee of around 50%, while Lyft charges a markup of around 25%. The fees of the decentralized ride-sharing platforms depend on both the previously described scenarios as well as the blockchain that the platform is using. Table 13 shows the prices of the fees per ride of the three chosen platforms for the described scenarios. We can see that even in the optimistic scenario, the fees for a ride on the decentralized ride-sharing platform hosted on Ethereum would cost USD 283, which is multiple times higher than the average total cost of a ride.

Table 13. Fees per ride on the three different blockchains.

Blockchain	Optimistic Scenario	Pessimistic Scenario
Ethereum	USD 283.25	USD 424.42
Polygon 4Matic	USD 0.02	USD 0.04
Harmony One	USD 0.03	USD 0.08

Ethereum

Figure 9 compares the fees of the centralized ride-sharing platforms Uber and Lyft with both scenarios for the decentralized ride-sharing platform deployed on Ethereum. The comparison uses the base price of a ride, which is the price of the ride itself that the driver will receive before any markups are applied. We can see that even the fees in the optimistic scenario become competitive with Uber above a base price of USD 500. If we compare the pessimistic scenario with the cheaper alternative, Lyft, it would require the base price to be over USD 1700 for the fee to be equal. Very few rides are done with a base price of over USD 100. Thus, we can clearly say that currently, a decentralized ride-sharing platform deployed on Ethereum is way more expensive for an average ride with a base price of fewer than USD 100 than any centralized ride-sharing platform.

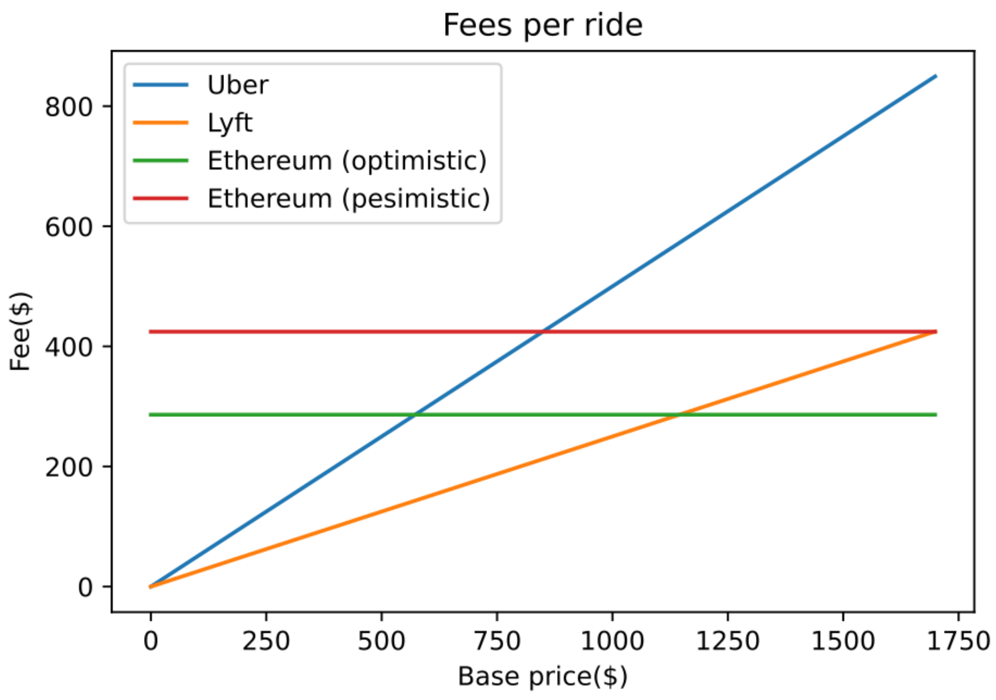

Figure 9. Comparison of fees per ride depending on the base price of a ride for Uber, Lyft, and the decentralized ride-sharing platform on Ethereum.

Polygon 4Matic

Figure 10 shows a comparison of the fee of Uber and Lyft with the two described scenarios for the decentralized ride-sharing platform deployed on Polygon 4Matic. Even at a base price of under USD 1 per ride in the pessimistic scenario, we can see that the fees are lower than a centralized platform. Since the fee on the decentralized platform is flat, the higher the ride's base price, the more significant the difference in fees between the centralized platform and the decentralized one on Polygon. Since any realistic ride has a base price of over USD 1, we can say that the decentralized ride-sharing platform on Polygon 4Matic offer cheaper fees than Uber and Lyft.

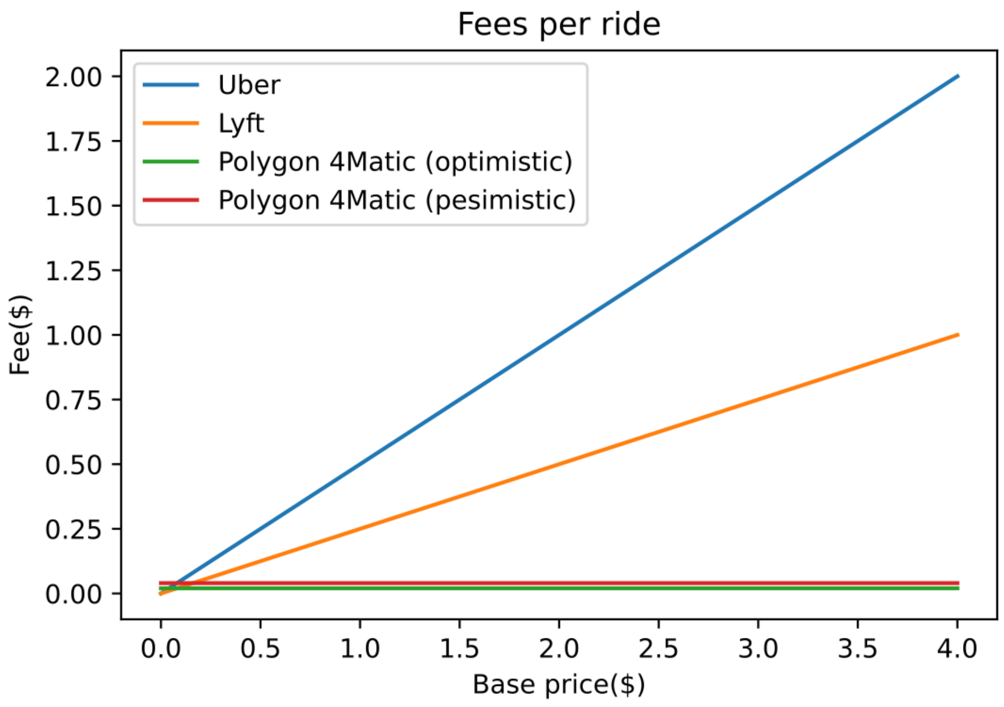

Figure 10. Comparison of fees per ride depending on the base price of a ride for Uber, Lyft, and the decentralized ride-sharing platform on Polygon 4Matic.

Harmony One

Figure 11 shows a comparison of the fees of Uber and Lyft with the two described scenarios for the decentralized ride-sharing platform deployed on Harmony One. We can see that the fees are slightly higher than on Polygon 4Matic, but still cheaper even for rides with a base price of under USD 1 than the centralized platforms. Therefore, it is also cheaper for any realistic ride than the centralized platform Uber and Lyft.

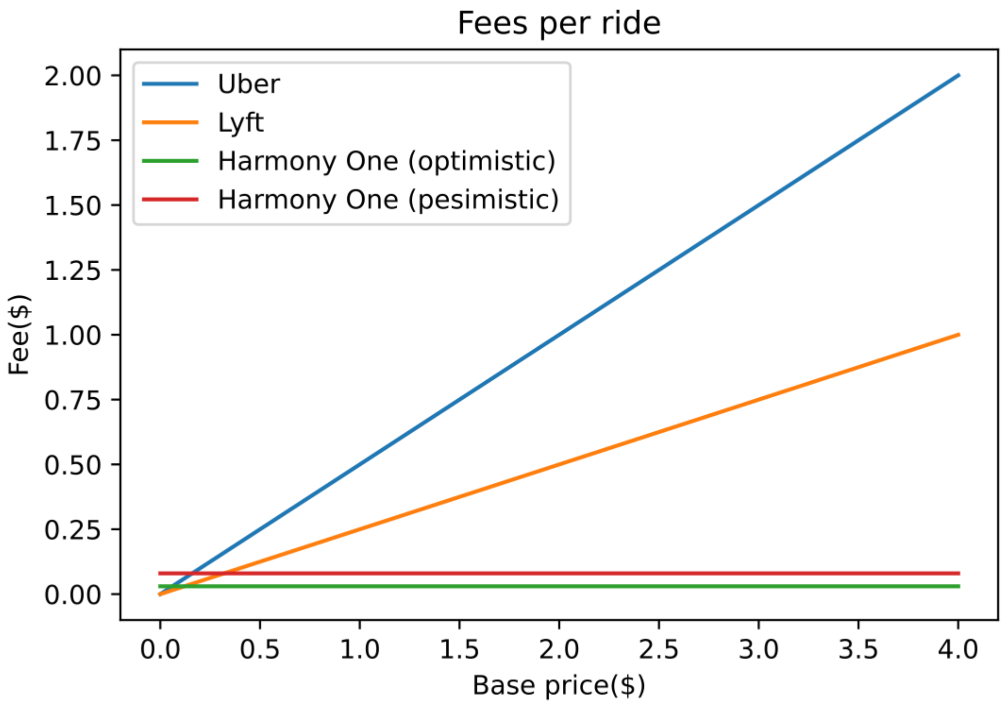

Figure 11. Comparison of fees per ride depending on the base price of a ride for Uber, Lyft, and the decentralized ride-sharing platform on Harmony One.

6.1.5. Discussion

The results show that the ride-sharing platform deployed onto the Ethereum blockchain is by far the most expensive in terms of fees of the three chosen blockchains. It only competes with Uber for rides with a base price of USD 500 and more. Since very few rides have a base price of over USD 500, we can safely say that, at this time, fees for the Ethereum blockchain are nowhere near competitive with the fees on centralized platforms such as Uber and Lyft for most scenarios. This result also shows how much the situation has changed since 2019. Back then, refs. [24,26] claimed that their implementation with similar or higher complexity as the implemented prototype offered fees under a few USD for a ride. On the other hand, the platforms deployed onto Polygon 4Matic and Harmony One offer fees of under one USD, which are not only much cheaper than the fees on Ethereum, but far cheaper than the fees on a centralized platform in every realistic scenario at this time, as seen in Figure 12. This shows that those two blockchains, purely in terms of fees paid to the platform, are way cheaper than the centralized platforms. This would allow drivers to increase their profits while at the same time still offering a lower price to the riders. While the price of fees on Polygon 4Matic and Harmony One is great, it is also worth mentioning that this might not stay that way in the future. While those two platforms offer a higher base throughput of transactions and use a different consensus algorithm than Ethereum, this still does not guarantee that the fees on those blockchains might rise in the future. This change might happen when more and more applications are built on these blockchains, and therefore, the network has to deal with more transactions; thus, the fee increase, similar to Ethereum in the past. A temporary surge in the gas prices on Polygon 4Matic due to a higher demand for the network already happened in December 2021 (https://coingape.com/polygon-network-gas-fee-skyrockets-amid-heavy-nft-gaming-will-matic-rally-continue (accessed on 3 April 2023)). A decentralized Sunflower game using NFTs created many transactions on the Polygon network, this caused the gas price to spike at over 700 Gwei. Because both Polygon 4Matic and Harmony One are still relatively new blockchains compared to Ethereum, it might also be worth mentioning that they do not have the same level of trust in terms of stability and decentralization.

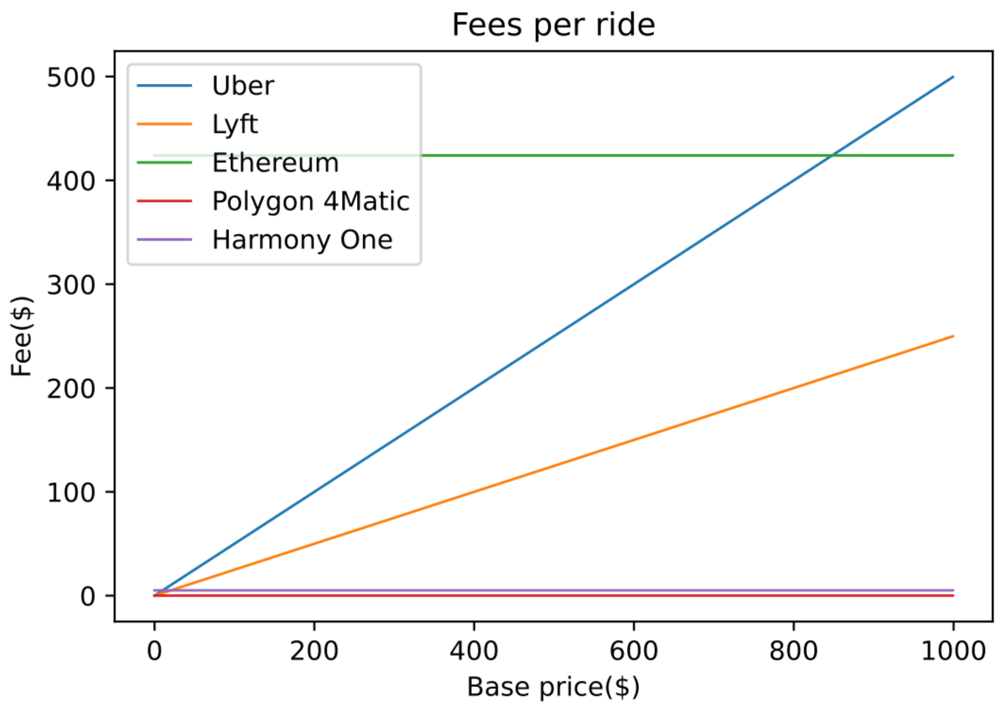

Figure 12. Comparison of fees per ride depending on the base price of a ride for Uber, Lyft, and the decentralized ride-sharing platform on Ethereum, Polygon, and Harmony One.

6.1.6. Summary

The implemented version of the ride-sharing prototype charges a flat fee per ride, while centralized platforms such as Uber and Lyft charge a percentage fee. Therefore, two scenarios were constructed to compare the prices of the fees in those scenarios. The results show that the platform deployed to Ethereum is way more expensive than centralized ride-sharing platforms in the most realistic situation. The platforms deployed on Polygon 4Matic and Harmony One, on the other hand, are cheaper than centralized platforms in every way, with a base price of USD 1 or more. The price for fees on Polygon 4Matic and Harmony One might rise as more applications are built on those blockchains, similar to how the prices on Ethereum rose in the past.

6.2. Research Question 2: Are There Any Noticeable Effects on User Experience When Using a Frontend Application That Interacts with Smart Contracts Compared to Centralized Services (Compared to Centralized Services)?

6.2.1. Results

A few key differences were identified which affect the user experience of the decentralized ride-sharing platform. The two most notifiable ones affecting the user experience negatively are the payment process, and the time it takes a transaction to be confirmed by a blockchain.

Payment

Since the payment on the decentralized ride-sharing platform is made directly via the blockchain, it differs significantly from a traditional platform. It requires the user to have funds in the form of native coins of the blockchain the platform is running on in order to pay the fees and the ride itself. (Theoretically, another token on the blockchain could also be used for the payment of the ride).

The confirmed interactions of a transaction are also different from a traditional platform. In web applications, the confirmation is mostly done via browser extensions or built-in browser features, which behave similarly to browser extensions. This process can irritate the user since the confirmation dialog does not directly come from the web application but the extension or browser.

6.2.2. Response Time

Another significant difference compared to traditional application communication with a backend is the higher response time. While most requests to modern backends are completed within 10 to 100 milliseconds or more/less, a transaction calling smart contracts takes much longer. It requires the transaction to be placed in a new block, which is periodically created. On Ethereum, this takes 15 s on average, but can vary greatly depending on how fast a new valid block is mined. On Polygon 4Matic and Harmony One, this takes around 2 s, and because of the proof of stake, this period is very consistent. In most situations, it is also common to wait for an additional block to be confirmed in order to be sure that the block containing the transaction is part of the longest chain. This security further increases the response time. Handling those requests to a smart contract similar to a request to the traditional backend results in a bad user experience since just waiting multiple seconds or longer is just not acceptable now. Fortunately, with some changes in the implementation, this can be prevented by handling the requests asynchronously and letting the user continue after a request. Once a request is completed, the result can be processed accordingly, and the user can be notified that the transaction succeeded or failed.

6.2.3. Discussion

The results show that while most of the user experience of a decentralized ride-sharing platform is similar to that of a centralized one, they differ in the payment process and response time of requests. Users will likely adapt to the new payment process as it becomes more streamlined over time, by, for example, the operating system providing a native wallet, as already mentioned in RQ1. This improvement would enable the user to confirm payment by clicking on a push notification similar to the existing payment application. On the other hand, the response time requires a different approach to implementing those applications to allow the user to use the application while the transaction progresses. This change in approach is especially important for Ethereum, with a block time of 15 s. However, it also affects Polygon 4Matic and Harmony One with a block time of around 2 s, as even that amount of time is over 40 times higher than the response time of traditional web services.

6.2.4. Summary

The two most notifiable factors affecting the user experience of the decentralized ride-sharing platform are the payment on the blockchain and the response times of transactions invoking smart contracts. The payment on the web application relies on the user configuring and using a specific browser extension which affects the user experience when using the web application. The response times of transactions invoking smart contracts require different implementation in the application in order for the user to not just wait for the transaction to be complete.

7. Conclusions

Our proposal evaluates the possibility of decentralized ride-sharing platforms on blockchains. Previous related work was reviewed, showing some attempts at creating a decentralized ride-sharing platform on blockchains. All of those implementations were focused on the smart contracts themselves and did not include implementing a frontend application for users to interact with the platform. Multiple related works also mentioned the costs of transactions on such a platform, but prices have changed since their evaluation. Therefore, a decentralized ride-sharing platform including multiple frontend applications was implemented, and the cost and usability were evaluated.

7.1. Implementation

We outlined how a centralized ride-sharing platform can be implemented on the blockchain as well as how smart contracts are implemented and deployed to different blockchains that support EVM bytecode. Those smart contracts handle all the ride-sharing platform's logic and the payment. The Truffle suit is used to organize the developed smart contract in a project and provide an easy way to deploy them to the different blockchains.

Unit tests are essential for implementing smart contracts because they are immutable once deployed. Therefore, it is essential to ensure that every function of a smart contract functions correctly. The unit tests are also integrated into the Truffle project.

Additionally, a web application was implemented, allowing end-users to use the centralized ride-sharing platform by providing a user interface for interacting with the deployed smart contract. The web application utilizes the common browser extensions called Metamask to access the user's wallet. The implemented frontend application provides all the required functionality of a basic ride-sharing platform, while not relying on a central server or entity. This working implementation proves a decentralized ride-sharing platform on blockchains is possible and shows what such a platform's implementation could look like.

7.2. Practical and Research Limitations

The costs of a transaction consist of gas, which depends on the complexity of the code invoked by the transaction, as well as the gas price, which is dependent on the utilization of the blockchain. The amount of gas for each transaction of implemented ride-sharing platform was evaluated using a special set of unit tests that log the gas after the transaction. The amount of gas was multiplied by the gas price and the price of the corresponding native asset in USD. Centralized ride-sharing platforms charge a percentage mark of the price of the ride. This mark is around 50% in the case of Uber and 25% in the case of Lyft. In order to compare the fees of the decentralized platform with the centralized one, two scenarios were created: an optimistic one and a pessimistic one. These scenarios consist of a different number of transactions for a single ride. The results show that the fees for a single ride on Ethereum are USD 283.25 for the optimistic scenario and USD 424.42 for the pessimistic scenario. These prices would require a base price of over USD 500 per ride to be competitive with centralized ride-sharing platforms. The fees on the platform deployed on Polygon 4Matic are USD 0.02 for the optimistic scenario and USD 0.04 for the pessimistic scenario. On the platform deployed on Harmony One, the price is USD 0.03 for the optimistic scenario and USD 0.08 for the pessimistic scenario. Both of these platforms offer cheaper fees than centralized platforms, even for rides with a base price of USD 1. Their fees can, therefore, be considered more than competitive compared to centralized ride-sharing platforms.

From a research perspective, all of our scenarios have been tested on actual test-networks for the corresponding blockchains, but not on the main-nets (i.e., where actual transactions are made and token transfer takes place). This is mostly due the real-world cost involved, since deploying contracts and issuing transactions to execute them results in expenses to be balanced in Ether. Aside from that, our research did not take into account the average waiting times that would be involved until transactions are actually being processed by a miner and manifested into a block on the blockchain. This, however, would be required for the driver to assure receiving the price offered.

7.3. User Experience

The implementation of a decentralized ride-sharing platform brings some challenges with it in terms of user experience. Since the payment has to be done on the blockchain, it has to be integrated safely and transparently. The Metamask browser extension is used in the web application to access the user's wallet. Unfortunately, there is no such extension of API for mobile applications, which, therefore, has to handle the user wallet itself. This lack of an API forces the application to implement some way of showing the user when a transaction is created, asking the user to confirm the transaction, or offering the possibility for the user to allow the app to sign certain transactions automatically. The user has to trust such a mobile application with his wallet completely. Therefore, this solution is not ideal.

Another challenge is the response times of transactions invoking smart contracts. Depending on the blockchain and the number of confirmations, these can be between 2 and 30 s or even longer. This problem can be solved by implementing a mobile application different from an application communicating with a traditional web service. The mobile application allows the user to continue after a transaction is sent and shows the user the transaction is complete or takes specific actions when the transaction fails.

7.4. Future Work

While this work shows that the concept and implementation of a decentralized ride-sharing platform are possible today, there are a few promising developments in the near future that would significantly improve it.

- **Standardized API for accessing user wallets on mobile phones**: A standardized API on the mobile operating systems, which would allow applications to access the wallet of users with their consent, would significantly improve the development process of a mobile app using blockchains while improving the user experience. Such standardized API would also allow apps to securely access the user's funds while at the same time reducing the amount of code written in the application.
- **Layer 2 scaling solutions for Ethereum**: While Ethereum does not offer the cheapest fees at the moment, the Ethereum network is still considered one of the most stable and secure blockchains today. Those considerations are because Ethereum has proven reliable in the last decade. Layer 2 scaling solutions [33], such as ZK rollups [34], as described by Lavaur et al. in [35], could allow an application to benefit from stability and security while at the same time paying much cheaper fees.

Author Contributions: Methodology, G.A.; Writing—original draft, G.A.; Writing—review & editing, E.S.; Supervision, E.S.; Project administration, E.S.; Resources, M.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data available in a publicly accessible repository that does not issue DOIs Publicly available datasets were analyzed in this study. This data can be found here: https://drive.google.com/file/d/1lKk37tjOO0JXwDLik7O9HRkmuDWxZPSm.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Smith, J.W. The Uber-All Economy of the Future. *Indep. Rev.* **2016**, *20*, 383–390.
2. Great Learning. What Data Scientists at Uber Are Doing With Your Data. 2018. Available online: https://medium.datadriveninvestor.com/what-data-scientists-at-uber-are-doing-with-your-data-c82ead10326c?gi=7c9a65b7cdc5 (accessed on 3 April 2023).
3. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. *Decentralized Bus. Rev.* **2008**, 21260. Available online: https://assets.pubpub.org/d8wct41f/31611263538139.pdf (accessed on 3 April 2023).
4. Pilkington, M. Blockchain Technology: Principles and Applications. In *Research Handbook on Digital Transformations*; Olleros, F.X., Zhegu, M., Eds.; Edward Elgar Publishing: Cheltenham, UK, 2015.
5. Wood, G. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Proj. Yellow Pap.* **2014**, *151*, 1–32.
6. Buterin, V. A next-generation smart contract and decentralized application platform. *White Pap.* **2014**, *3*, 13–17.
7. Kasireddy, P. How Does Ethereum Work, Anyway. *Medium*. **2017**. Available online: http://www.easygoing.pflog.eu/32_blockchain_P2P/ethereum_blockchain.pdf (accessed on 3 April 2023).
8. Bez, M.; Fornari, G.; Vardanega, T. The scalability challenge of ethereum: An initial quantitative analysis. In Proceedings of the 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), San Francisco East Bay, CA, USA, 4–9 April 2019; pp. 167–176.
9. Kanani, J.; Nailwal, S.; Arjun, A. Matic Whitepaper. 2021. Available online: https://academy.bit2me.com/wp-content/uploads/2021/07/polygon-whitepaper-en.pdf (accessed on 3 April 2023).
10. Tse, S. Technical Whitepaper. Harmony One. 2022. Available online: https://harmony.one/whitepaper.pdf (accessed on 3 April 2023).
11. Pandey, A.A.; Fernandez, T.F.; Bansal, R.; Tyagi, A.K. Maintaining scalability in blockchain. In *Proceedings of the Intelligent Systems Design and Applications: 21st International Conference on Intelligent Systems Design and Applications (ISDA 2021), Online, 13–15 December 2021*; Springer: Berlin, Germany, 2022; pp. 34–45.
12. Szabo, N. Smart Contracts: Building Blocks for Digital Markets. *Extropy J. Transhumanist Thought* **1996**, *18*, 28.
13. Szabo, N. The Idea of Smart Contracts. *Nakamoto Inst.* **1997**. Available online: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html (accessed on 3 April 2023).
14. Christidis, K.; Devetsikiotis, M. Blockchains and Smart Contracts for the Internet of Things. *IEEE Access* **2016**, *4*, 2292–2303. [CrossRef]
15. Mokalusi, O.; Kuriakose, R.; Vermaak, H. A Comparison of Transaction Fees for Various Data Types and Data Sizes of Blockchain Smart Contracts on a Selection of Blockchain Platforms. In *ICT Systems and Sustainability: Proceedings of ICT4SD 2022*; Springer: Berlin, Germany, 2022; pp. 709–718.
16. Drescher, D. *Blockchain Basics*; Apres: Berkeley, CA, USA, 2017.
17. Ethereum Foundation. Solidity Language Documentation. 2022. Available online: Soliditylang.org (accessed on 3 April 2023).
18. Iyer, K.; Dannen, C. The Ethereum Development Environment. In *Building Games with Ethereum Smart Contracts: Intermediate Projects for Solidity Developers*; Apress: Berkeley, CA, USA, 2018; pp. 19–36. [CrossRef]
19. Metamask. Official Documentation. 2021. Available online: Metamask.io (accessed on 3 April 2023).
20. Wüst, K.; Gervais, A. Do you Need a Blockchain? In Proceedings of the 2018 Crypto Valley Conference on Blockchain Technology (CVCBT), Zug, Switzerland, 20–22 June 2018; pp. 45–54. [CrossRef]
21. Sánchez, D.; Martínez, S.; Domingo-Ferrer, J. Co-utile P2P ridesharing via decentralization and reputation management. *Transp. Res. Part Emerg. Technol.* **2016**, *73*, 147–166. [CrossRef]
22. Baza, M.; Lasla, N.; Mahmoud, M.M.E.A.; Srivastava, G.; Abdallah, M. B-Ride: Ride Sharing With Privacy-Preservation, Trust and Fair Payment Atop Public Blockchain. *IEEE Trans. Netw. Sci. Eng.* **2021**, *8*, 1214–1229. [CrossRef]
23. Semenko, Y.; Saucez, D. Distributed Privacy Preserving Platform for Ridesharing Services. In *Security, Privacy, and Anonymity in Computation, Communication, and Storage*; Springer: Berlin, Germany, 2019; pp. 1–14. [CrossRef]
24. Baza, M.; Mahmoud, M.; Srivastava, G.; Alasmary, W.; Younis, M. A Light Blockchain-Powered Privacy-Preserving Organization Scheme for Ride Sharing Services. In Proceedings of the 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), Antwerp, Belgium, 25–28 May 2020; pp. 1–6. [CrossRef]

25. Rieger, A.; Roth, T.; Sedlmeir, J.; Fridgen, G. We need a broader debate on the sustainability of blockchain. *Joule* **2022**, *6*, 1137–1141. [CrossRef]
26. Pal, P.; Ruj, S. BlockV: A Blockchain Enabled Peer-Peer Ride Sharing Service. In Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain), Atlanta, GA, USA, 14–17 July 2019; pp. 463–468. [CrossRef]
27. Bathen, L.A.D.; Flores, G.H.; Jadav, D. RiderS: Towards a Privacy-Aware Decentralized Self-Driving Ride-Sharing Ecosystem. In Proceedings of the 2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS), Oxford, UK, 3–6 August 2020; pp. 32–41. [CrossRef]
28. Siddarth, D.; Ivliev, S.; Siri, S.; Berman, P. Who Watches the Watchmen? A Review of Subjective Approaches for Sybil-Resistance in Proof of Personhood Protocols. *Front. Blockchain* **2020**, *3*, 590171. [CrossRef]
29. Guo, H.; Yu, X. A Survey on Blockchain Technology and its security. *Blockchain Res. Appl.* **2022**, *3*, 100067. [CrossRef]
30. Zou, W.; Lo, D.; Kochhar, P.S.; Le, X.B.D.; Xia, X.; Feng, Y.; Chen, Z.; Xu, B. Smart contract development: Challenges and opportunities. *IEEE Trans. Softw. Eng.* **2019**, *47*, 2084–2106. [CrossRef]
31. Gogol, F. How Much Does Uber Pay? JG Wentworth. 2021. Available online: https://www.stilt.com/blog/2020/02/how-much-does-uber-pay/ (accessed on 3 April 2023).
32. Helling, B. How Much Do Lyft Drivers Make, Really? Factors That Eat Into Lyft Driver Salary. Ridester. 2021. Available online: https://www.ridester.com/how-much-do-lyft-drivers-make/ (accessed on 3 April 2023).
33. Kshetri, N. Blockchain basic: Definitions, key concepts and characteristics. In *The Rise of Blockchains*; Edward Elgar Publishing: Cheltenham, UK, 2022; pp. 3–30.
34. Garoffolo, A.; Kaidalov, D.; Oliynykov, R. Zendoo: A zk-SNARK verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains. In Proceedings of the 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), Singapore, 29 November–1 December 2020; pp. 1257–1262.
35. Lavaur, T.; Lacan, J.; Chanel, C.P. Enabling Blockchain Services for IoE with Zk-Rollups. *Sensors* **2022**, *22*, 6493. [CrossRef] [PubMed]