# TOM: The Open City Distribution Framework for Supply Chain Management on Ethereum


Paolo-Arash Kazemi Koohbanani*
*Department of Intelligent Systems*
*Delft University of Technology*
Delft, The Netherlands
a.kazemikoohbanani@student.tudelft.nl

Mantas Makelis*
*Department of Intelligent Systems*
*Delft University of Technology*
Delft, The Netherlands
m.makelis@student.tudelft.nl

Nick Tehrany*
*Department of Intelligent Systems*
*Delft University of Technology*
Delft, The Netherlands
n.tehrany@student.tudelft.nl

Jorrit van Assen*
*Department of Intelligent Systems*
*Delft University of Technology*
Delft, The Netherlands
j.s.vanassen@student.tudelft.nl

Rick de Boer*
*Department of Intelligent Systems*
*Delft University of Technology*
Delft, The Netherlands
r.e.j.deboer@student.tudelft.nl

Erik de Graaf
*Data Ecosystems*
*TNO*
The Hague, The Netherlands
erik.degraaf@tno.nl

Tianyu Li
*Department of Intelligent Systems*
*Delft University of Technology*
Delft, The Netherlands
tianyu.li@tudelft.nl

Zekeriya Erkin
*Department of Intelligent Systems*
*Delft University of Technology*
Delft, The Netherlands
z.erkin@tudelft.nl

* Authors contributed equally.



***Abstract*—Over the past decade an abundance of online marketplaces have been designed for applications such as food delivery, eCommerce, and property rental markets. However, these marketplaces are often governed by for-profit corporations, requiring fees for every transaction. In addition, platforms requiring physical interaction such as food delivery, are only providing services in large cities, such that profits can be maximized. To this end, we propose the design and implementation of a generic framework to show the viability of using blockchain technology for three-sided marketplaces. The protocol is implemented using Ethereum smart contracts, where data sharing is enabled between parties through off-chain data linking using IPFS. The platform allows clients to directly purchase goods from sellers and delivery service providers to handle these orders, without the need for an intermediary and security is ensured with asymmetric encryption schemes.**

**Performance evaluations of the deployed protocol show that the estimated cost of a complete transaction for an order on the public Ethereum network is equivalent to 85 euros with a gas price of 30 gwei. Although blockchain scalability is yet to be solved, our presented system showcases the possibility of building blockchain-based three sided marketplaces.**



***Index Terms*—Blockchain, Ethereum, Smart Contract, City Distribution, Supply Chain Management, Multi-Sided Marketplace**


## I. Introduction

With the increasing popularity of online marketplaces, the market share of industries such as eCommerce have steadily been expanding. It is estimated that by 2025 eCommerce will account for 24.5% of the global market share for retail sales [1]. This vast online presence has given rise to a plethora of platforms, ranging from food delivery services, such as Uber Eats and Deliveroo, to payment systems such as PayPal and American Express, and property rental systems such as AirBnB. Common amongst all these platforms is that the connection between parties is achieved purely through the platform.

They provide the means for parties to transact between each other, be it physical goods as in the case of delivery platforms, or simply for utilizing payments, as is the case with PayPal. Reputable platforms often provide inherent trust in the system by for instance insuring transferred goods, ensuring parties will not be cheated in the system. A fee is also included, which corresponds to some percentage of the whole transaction, increasing the price of the goods for the customer. While the convenience of delivery platforms is appealing to customers, it is only profitable in densely populated areas such as larger cities. Centralized platforms additionally suffer from privacy concerns regarding data collection [2].

With the increasing adoption of blockchain systems and cryptocurrencies [3], [4], which provide decentralized and transparent storage through distributed ledgers, they present an appealing foundation for building decentralized applications. Through blockchains such as Ethereum [5], applications can run decentralized on the blockchain by making use of its Solidity [6] programming language by creating so called smart contracts. The programmed contracts can then be deployed on the blockchain, and its external functions can be called in an Ethereum transaction. Transactions are being processed by miners on the Ethereum network, whose responsibility is to secure and verify transactions through Proof-of-Work [7]. With

this, any code specified in the contract can be executed by a miner and be included in the next block on the blockchain. As a reward for executing the contract code, the miners receive a reward in the Ethereum token native to its network, which is measured in the form of gas and gas fees associated for the transactions.

Building decentralized applications would move from the prior trust placed in third-party applications to placing trust into the protocols and the blockchain network. In the field of blockchain-based marketplaces, recent developments introduced numerous platforms, such as energy-trading [8], [9], [10], and eCommerce [11]. Additionally, by replacing the monopolized corporate marketplaces with decentralized marketplaces, data privacy concerns can be diminished.

While this approach provides considerable benefits, it introduces numerous challenges. Namely, how trust can be placed into the system if physical goods are involved. With third-party platforms, trust can be offered by insuring goods or handling disputes among parties. Implementing these features with decentralized systems becomes non-trivial. To allow for the transfer of physical goods, addresses of clients and sellers need to be securely provided. Can decentralized applications provide the same scalability and performance in terms of additional transaction costs compared to third-party platforms? Lastly, how can clients and sellers, or any additional party involved in the transaction, be matched to each other on automated two-sided and multi-sided platforms?

With the possibilities of blockchain-based decentralized marketplaces, and the prior mentioned obstacles in mind, we aim to build a prototype framework to explore the viability of a blockchain-based multi-sided marketplace. The main applications of such a system are meant for the use case of city distribution and supply chain management. In particular, in this paper we make the following contributions:

- We design and implement a protocol to support multi-sided transactions for the application of city distribution and supply-chain management.
- We deploy the developed smart contract for multi-sided marketplaces on the Ethereum Kovan Test network.
- We provide a prototype application, connected to the deployed smart contract to showcase the feasibility of multi-sided blockchain-based marketplaces.
- We measure the performance in terms of time and cost implications for transactions within the decentralized blockchain-based marketplace.
- We identify limitations of the current state of blockchains for decentralized markets and propose future work for the improvement of decentralized marketplaces for city distribution and any blockchain-based marketplaces.
- We make all source code for the smart contract and the prototype platform publicly available [1] under the MIT License.

[1]https://github.com/The-Open-Market/tom

## II. Related Work

The idea of building decentralized marketplaces on blockchain systems is not new, there have been several platforms built for two-sided marketplaces. [8] propose a system for energy trading, for speculating on goods such as oil and gas over P2P networks, which could be deployed through smart contracts. Although the authors only present the idea and functionality behind energy trading with blockchain systems, there is no implementation available. Similarly, Kang et al. [9] design and implement an energy trading platform on the Ethereum network, using a smart contract. While the authors present a novel approach at a two-sided platform, it is not generic enough to be applied to additional use cases unlike the system proposed in this work. Lastly, Esmat et al. [10] present a decentralized energy trading platform that provides a full marketplace with P2P trading of energy assets and a blockchain layer for settlement of trades, implemented on Hyperledger Burrow. The system presents an innovative approach at building an extensive P2P trading platform; however, it similarly is limited and application domain dependent. [12] aims to eliminate the corporate middleman in eCommerce applications with a two-sided marketplace on the Ethereum network. Much like the prior mentioned systems, it also is designed to only provide transactions between two parties.

While there are numerous platforms for decentralized marketplaces, a common non-trivial obstacle among most decentralized marketplaces is the matching of parties. Funk et al. [13] present a blockchain-based matching market that utilizes deferred acceptance [14] to match clients to sellers, in for instance labor markets, where there is no direct selection of associating parties. Deferred acceptance is conventionally used with a central entity that provides the matching; however, the authors employ a smart contract on Ethereum to implement a prototype that maintains data privacy and implements decentralized matching. [15] present another two-sided matching mechanism for blockchain-based crowdsourcing platforms, where unlike conventional task assignment on crowdsourcing platforms that only accounts for the requirements of the task publisher, the proposed mechanism also matches based on the workers' preferences. Similarly, Wu et al. [16] show another task matching mechanism for crowdsourcing platforms that focuses on improving reliability by using smart contracts on the Ethereum blockchain and additionally maintaining confidentiality and proficient anonymity for task workers. Unlike the prior discussed systems, which focus purely on optimizing task publishing on existing crowdsourcing platforms, CrowdBC [17] implements a framework that can be used to build decentralized crowdsourcing platforms on the Ethereum blockchain.

While previous developments focused on specific business use cases, a generalized framework has not been designed yet. In particular, the necessity for a common base contract structure arises from the wide range of applications it could be used in. The framework could be then consolidated and audited to provide better security guarantees. Moreover, most

studies focused on two-sided marketplaces, however, in this paper three-sided transactions are considered.

## III. Design

In this section, we present the design and decisions for the developed three-sided open marketplace where anyone can be a participant. The marketplace must have three participating party types, namely a client, a seller, and a delivery service provider (deliverer). All parties must cooperate, so that the seller is able to sell their goods, the client receives the ordered goods, and the deliverer can build a business around connecting the other two types of parties.

The designed system allows the client to select a seller and choose from an array of goods that the seller offers for purchase. Naturally, the client can place an order as in every online marketplace by entering their delivery information and making a transaction. The seller gets notified that a new order has been placed, approves the order, and proceeds to preparing the goods. The deliverer can listen for newly approved orders from the sellers and choose to commit to make the delivery.

The money paid by the client for the goods is first sent to the smart contract address and frozen until the delivery is completed successfully. Such an approach assures the client that their money is safe until the goods are in. Moreover, sellers have the ability to request a deliverer to put up collateral before accepting the order. It is a built-in safe guard for theft that sellers can opt-in. By using collateral, a seller can reduce the risk of a deliverer running away with the goods instead of delivering them to the client. At the time of accepting an order, the paid collateral is locked within the smart contract and it is released to the deliverer upon successful delivery.

Instead of setting a standard fee for the delivery, the fee is adjustable per order basis by the seller. This allows us to fine-tune the fee for individual orders, for example, a small order could be cheaper to deliver than a large order. Moreover, this approach creates a dynamic market where participants can decide what is best for their business and adjust accordingly.

When the physical goods are transferred from one party to another, the system requires a multi-signature to proceed. It is needed, so the system knows that both parties consented and agreed on the transfer. Such a situation happens at the goods pick up with the deliverer and seller, as well as at the delivery time between the deliverer and the client.

An order must have a status, either it is waiting for a pickup or is en-route to the receiver. There must be a way to differentiate between orders that are being prepared and orders that have been delivered. Hence, the developed system implements an order state machine that enables an order to traverse states by execution of a transaction.

### A. Order life-cycle

Figure 1 shows the states an order can be in. Starting from the beginning, a client would like to order goods from a seller. The client places an order and pays into the smart contract which creates the order in a `pending` state. As long as the order is in the `pending` state, the client can cancel the order, which puts the order in the `cancelled` state and refunds the money to the client's account.

Meanwhile, the seller sees the `pending` order and makes a choice. If the seller places the order in the `rejected` state or similarly to the `cancelled` state, the client receives a refund. On the other hand, if the seller wishes to approve the order, they can set a collateral requirement for the order along with the delivery fee that the seller is ready to pay for the delivery. Moreover, the business model of the seller might require delaying the preparation of the goods. Therefore, the seller can flag that a particular order is not immediately ready for pick-up and that notification will be given when it is.

Upon reaching the `approved` state, the order becomes visible for all the deliverers. Anyone who wishes to make the delivery can accept the order by paying the requested collateral in the smart contract. If an acceptance transaction is made, the smart contract checks if the seller has set the *wait-until-ready* flag, and if so, the order is placed in the `accepted` state and waits for the seller to put it in the `ready` state. Otherwise, the order is moved directly to the `ready` state.

When an order is in the `ready` state, the deliverer physically comes to the seller to pick up the goods. Now, the first multi-signature is required to proceed to the `in transit` state. Both parties, i.e. the seller and the deliverer, must sign off on the transfer of the goods to the delivery service's care. The order of signatures does not matter, which is represented by a fork from the `ready` state to either `transferred` or `picked up` states in Figure 1. Right after the second party signs, the order state is set to the `in transit` state.

A similar multi-signature is required at the delivery. Both the deliverer and the client sign and set the order state to the `completed` state. After the last transaction, the money paid by the client and the collateral paid by the deliverer are now unlocked. The deliverer receives a refund of the collateral along with the delivery fee set by the seller, while the seller receives the rest of the amount.

### B. Matching of parties

Optimal matching between two sets was solved for the stability marriage problem [14], in this case the parties would be the order and the deliverer. Although some blockchain-based solutions have been proposed, such as for task matching [15], [16], an equally optimal algorithm has not yet been discovered. Therefore, the chosen solution is to allow system users to assign themselves to their preferred orders. The deliverer is provided with the information of the origin and the destination zip codes of an order, to make a decision on whether to accept it. Although this depends on the number of participants and their location, it is assumed that a deliverer can be found for any order approved by a seller.

### C. Data sharing

Sellers are public businesses with publicly available information, such as delivery address and contact information, required for the city distribution system. The deliverers are not required to share any private information that should be

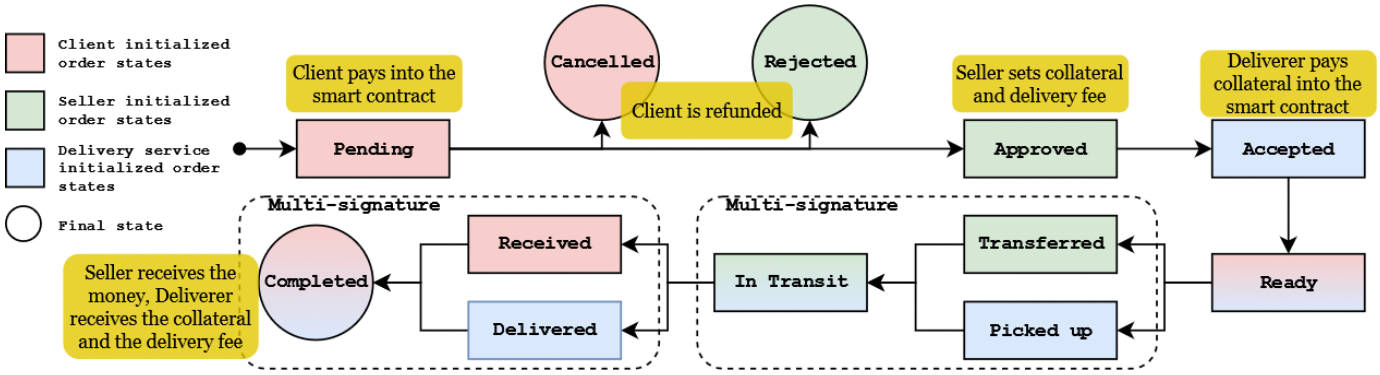


Fig. 1. State machine diagram showing the states an order can be in. Moreover, inputs and outputs of the money are shown near appropriate states in bright yellow color.

protected. However, clients do enter their private information, such as address, into the system to let the seller know where to deliver the ordered goods. Hence, the client privacy aspect is the most important in the application.

The application uses cryptographic privacy techniques, explained in Section III-D, to securely store the client information and keep it private. However, the deliverer must get the destination address to complete the delivery. Therefore, information sharing must happen at an appropriate time i.e. not before the deliverer is committed to the job and not after the deliverer has the goods. The only moment is at the pickup of the goods. Therefore, the system is designed so that the seller physically provides the destination address to the deliverer together with the goods.

### D. Cryptography

The order details, such as the client delivery address or the list of ordered goods, are communicated in encrypted form to the seller. The seller's public-key information is known, as that can be shared along with his service offering. This allows the client to use asymmetric encryption to encrypt his personal details and cart information, required to complete the order. The encrypted data can then be publicly shared, and the seller is able to verify it before accepting an incoming order. Symmetric encryption is used to store the same data, ensuring that the client can retrieve the order information at a later time. Moreover, the client creates a hash of the delivery address, and the seller physically communicates this along with the salt used during hashing. The salt is used to prevent brute-force attacks that, given the limited search space of physical addresses, could deanonymize any client. Moreover, this improves the privacy of the user because the same address hash is not recognizable over a list of orders within the system.

### E. Native currency

Cryptocurrencies are usually volatile assets and businesses might not be interested in investing in them due to their risk [18], [19]. Specifically, if the value of the currency rapidly changes while the order is in the delivery process it will cause losses to the merchant. Hence, using Ethereum's native token `ETH` is not ideal. To accommodate the issue and help keep the paid value stable, a stablecoin must be used. Thus, the system implements `EURT`, a currency that represents the euro. It is not a real stablecoin that is pegged to the euro, but it does implement the same functionality. Hence, in a production system, it could be traded for the contract of a real stable coin.

## IV. Implementation

Choosing a blockchain to build on is a crucial step to ensure future extensibility of the platform. The Bitcoin [7] blockchain allows users to build simple smart contracts with the Script [20] language, such as multi-signature transactions [21] or payment channels [22]. However, its Turing-incompleteness [20] limits future improvements that can be applied, and therefore it is not the right choice to design a generic framework.

Hyperledger Fabric is a permissioned blockchain [23], it has a modular architecture which allows one to build any kind of blockchain application under this model. In a permissioned blockchain, all nodes in the network are known and have been previously authorized. This provides a blockchain that is scalable and can process large amounts of transactions per second. There is no default support for tokens; however, this could be added with the different modules that are provided out of the box [23]. Moreover, it is possible to choose a custom consensus model to validate the transactions. The scalability advantage and reduced costs make Hyperledger Fabric a candidate for the blockchain of choice. However, we decided to not use it because the permissioned model is not compatible with the assumption that anyone can join and participate in the network.

Cardano and the underlying non-spend transaction output (UTxO) architecture model [24] uses Proof-of-Stake [25] as the consensus model. Additionally, the model is extended with the ability to write scalable smart contracts. Contracts are written using Haskell programming language that is compiled into Plutus [26], a subset of Haskell using Template Haskell. The contract code can be formally verified using functional programming. The blockchain custodians are releasing Hydra [27] that promises an isomorphic state-channels as a second

layer solution for scalability. Hydra is very similar to the Bitcoin Lightning network [22] and offers scalability on demand. Although Cardano satisfies all requirements, unfortunately, the programming tools are currently still in the early development stage. Moreover, the high learning curve does not make it the ideal candidate for a generic framework that could be easy to understand and adapt.

Although many blockchains have been designed and deployed over the years, Ethereum [5] has been a stable choice to deploy smart contracts, both for tokens and decentralized applications. It allows us to deploy smart contracts written in the Solidity [6] language, which is Turing-complete by itself [28]. The permissionless model enables any user to interact with the platform and extend the contract for their specific use case. Moreover, the popularity of Ethereum made it so that standards have been proposed for common contracts [29], [30]. More importantly, they have been audited by different independent parties for security vulnerabilities [31]. While the Ethereum blockchain might be seen as expensive due to its increasing gas costs [32], [33], we decided to use it for the aforementioned reasons.

### A. Smart contract

The smart contract is written in Solidity version 0.8.0. The contract is split up in four files: the factory, the manager, the helper, and the main contract to combine all the previous into one. The correctness of the contract is tested using the Truffle Suite [34].

*1) TOM:* TOM is the main contract that inherits from every partial contract and combines them. The contract provides the constructor which initializes the system and sets the token contract address. Additionally, it has a function to allow the owner of the contract to change the token contract address in case it needs to be changed.

*2) Order Factory:* The Order Factory contains data structures used throughout the contracts. These include the definition of an order, order states, and events that are sent in response to a change of state. Moreover, a global list of orders that is stored on the blockchain resides here. The order struct is defined as follows:

- A unique identifier of an order;
- A client, seller and deliverer Ethereum addresses;
- An order total cost, a delivery fee, and a collateral amount;
- The status of the order;
- An address, that points to the order content located off-chain;
- Zip codes of the client and the seller;
- A `waitForReady` flag to determine if the seller needs to approve before the ready state.

Initially, whenever the smart contract receives a client request to place an order, it uses the function to create an order and populate it with the data posted by the client. Upon an order's creation, it is appended to the global list, emitting an event containing the order information. A seller can listen to the event and react accordingly.

*3) Order Manager:* The Order Manager defines external contract functions that can be called through transactions to push the order through the state machine, as described in Section III-A. After a successful execution of any of these functions, an event is emitted, notifying the listening parties about the status change. The contract contains the following functions:

- `placeOrder`: This function is executed by the client at the time of order placement. The method creates an order with the address of the seller, the total amount of the order, and a link, which points to the order details off-chain (see Section IV-C). The transaction amount is transferred from the client account to the contract account.
- `approveOrder`: This function is executed by the seller upon order approval. The seller provides its own and the client zip codes along with the delivery fee and the collateral amount, along with an indication whether the order is ready directly or requires an additional transaction to set it to `ready` state.
- `acceptOrder`: This function is executed by the third address that cannot be neither the client nor the seller address. The address is registered as the delivery service provider, and the collateral is transferred from the address to the contract. Depending on the `waitForReady` flag the order goes to the `accepted` state or `ready` state.
- `preparedOrder`: This function is executed by the seller when the order is ready for pick-up. It is relevant only if the `waitForReady` flag is set. The order is changed to the `ready` state.
- `transferOrder`: This function is executed by the seller and the deliverer separately. The function handles the first multi-signature representing the pick-up of the goods by the deliverer. After both parties sign, the order state is changed to `in transit`.
- `completeOrder`: This function is executed by the client and deliverer separately. The function handles the second multi-signature representing the act of receiving the goods by the client. After both parties sign, the order state is changed to `completed`. Additionally, the delivery fee is paid to the delivery with the collateral and the seller receives the rest.
- `cancelOrder` and `rejectOrder`: These functions can be executed by the client and the seller, respectively. In both cases, the client is reimbursed.

*4) Order Helper:* The Order Helper defines external views to read data from the blockchain. The contract adds functionality to query orders without having to pay for gas. This makes it possible to retrieve orders specifying the Ethereum address of a specific party. Additionally, it allows deliverers to query orders that have been available for acceptance before they started listening to new events. Finally, the contract contains the function to query the number of completed and cancelled orders by address.

### B. Token integration

We use a token (EURT) for payments on the marketplace. The security of token contracts is vital; therefore, we use the ERC-20 contract standard [29], that is implemented by OpenZeppelin [35]. The token contract defines a fungible token and provides the interface that enables TOM to transfer tokens between addresses, i.e., to subtract from an address and put into the TOM contract. In particular, the client can approve an allowance amount with a transaction. The allowance is the maximum amount of money the TOM contract is allowed to subtract from the client account.

### C. Off-chain

The blockchain persistent storage needs to be replicated by every full node in the network, that makes memory space a valuable and expensive resource. To avoid these costs, TOM uses the InterPlanetary File System (IPFS) [36] to store data in a decentralized manner and off-chain, namely, the order details upon order placement. Any distributed storage system can be employed as long as it is possible to link resources using a hash reference; however, IPFS was chosen because it was freely available. The order details are encrypted using the seller's public key. The IPFS link is then stored inside the order struct within the smart contract so that the seller can retrieve the data. Asymmetric and symmetric encryption schemes as well as the hashing scheme are implemented using the Networking and Cryptography library [37].

To accommodate situations where the given address is misspelled or an incorrect address is maliciously given to the delivery person, the hash of the address is publicly available and accessible to the deliverers. On pickup, the deliverer receives from the seller an address and a salt which can be compared to the hash.

### D. Frontend Setup

To create a demonstration of the final system, we provide the web-interface[2] for the client, seller, and delivery service to interact with the contract. The interface is created with the Vue.js [38] framework and uses the Ethers.js library [39] to interact with the blockchain. It requires a web3 enabled browser or a wallet extension such as MetaMask [40] or Coinbase Wallet [41], to authorize transactions.

The dashboard allows for running through the full life-cycle of the orders from each party's perspective. The client or the deliverer, additionally, can set and view the allowance reserved in the application. Moreover, it is possible to add custom sellers to choose from by inputting their business information, i.e. Ethereum address and contact details.

## V. Evaluation

To determine the platform feasibility, we measured the cost in terms of gas needed to interact with the smart contract and the time overhead required to fully complete an order.

[2]https://the-open-market.github.io/tom/

TABLE I
COST TO DEPLOY THE SMART CONTRACTS (GASPRICE = 30 GWEI AND 1 ETH = €3000).

| | Gas used | ETH cost | EUR cost |
|---|---|---|---|
| Deploy EURT | 1336469 | 0.04009407 | €120.28 |
| Deploy TOM | 5387316 | 0.16161948 | €484.86 |
| Total deploy cost | 6723785 | 0.20171355 | €605.14 |

TABLE II
EURT SMART CONTRACT COST (GASPRICE = 30 GWEI AND 1 ETH = €3000).

| EURT Contract | Gas used | ETH cost | EUR cost |
|---|---|---|---|
| Approve allowance | 46954 | 0.00140862 | €4.23 |
| Reset allowance | 24922 | 0.00074766 | €2.24 |
| Send token | 35317 | 0.00105951 | €3.18 |

### A. Gas cost

We first analyze the deployment cost; this is a one-time charge the platform owner incurs. Further, we analyze the cost of all functions in the EURT and TOM contracts, concluding with an overview of the cost breakdown for each party in the transaction. We assume the price of gas for each transaction to be 30 gwei [42], equivalent to $30 \cdot 10^{-9}$ ETH. To calculate the fiat cost of the transaction, the value of 1 ETH is fixed to €3000, which is close to the price of ETH in April 2022[43]. The measured values are taken from Etherscan [43] and MetaMask [40], in case a transaction gas cost varies we take the maximum value we have measured. The analysis focuses on the worst-case scenario to assess feasibility, since this functions as an upper bound of the cost.

Table I shows the cost to deploy the EURT and TOM contract independently, along with the total deployment cost. The EURT is a standard ERC-20 contract costing €120.28 to deploy, however, deploying the platform has a price of €484.86 instead. The total cost is then €605.14, assuming the platform owner would also control the token that is used to conduct transactions.

For completeness, the EURT functions used by the TOM contract were measured as well and the results are displayed in Table II. In particular, setting the allowance that an address can spend can cost up to €4.23, however, in case the value being set is zero the gas cost reduces to €2.24. Sending tokens between users using the ERC-20 contract costs €3.18 in transaction fees, for example when tipping the deliverer.

The measurements of all TOM contract functions can be found in Table III. The highest price is the creation of an order with an estimated cost of €27.77. In case either client or seller decide to refund the order, they incur a cost of €7.50 and €7.06, respectively. Intermediate transitions have the lowest cost with a value less than €6. When completing an order, the transition to `completed` state releases the locked funds to the seller and deliverer. In this case, the gas cost is higher due to the interaction with the token contract; therefore, it is displayed as a separate value. Depending on the confirmation order, the client or deliverer has to pay an extra €2.78 to finalize the order.

TABLE III
TOM SMART CONTRACT COST (GASPRICE = 30 GWEI AND 1 ETH = €3000).

| TOM Contract | Gas used | ETH cost | EUR cost |
|---|---|---|---|
| placeOrder | 308585 | 0.00925755 | €27.77 |
| approveOrder | 158856 | 0.00476568 | €14.30 |
| acceptOrder | 129208 | 0.00387624 | €11.63 |
| preparedOrder | 63858 | 0.00191574 | €5.75 |
| transferOrder | 65503 | 0.00196509 | €5.90 |
| completeOrder | 64798 | 0.00194394 | €5.83 |
| completeOrder [finalize] | 95699 | 0.00287097 | €8.61 |
| cancelOrder | 83314 | 0.00249942 | €7.50 |
| rejectOrder | 78465 | 0.00235395 | €7.06 |

TABLE IV
COST FOR EACH PARTY (GASPRICE = 30 GWEI AND 1 ETH = €3000).

| TOM Contract | Gas used | ETH cost | EUR cost |
|---|---|---|---|
| Client | 404284 | 0.01212852 | €36.39 |
| Seller | 288217 | 0.00864651 | €25.94 |
| Delivery Service | 259509 | 0.00778527 | €23.36 |

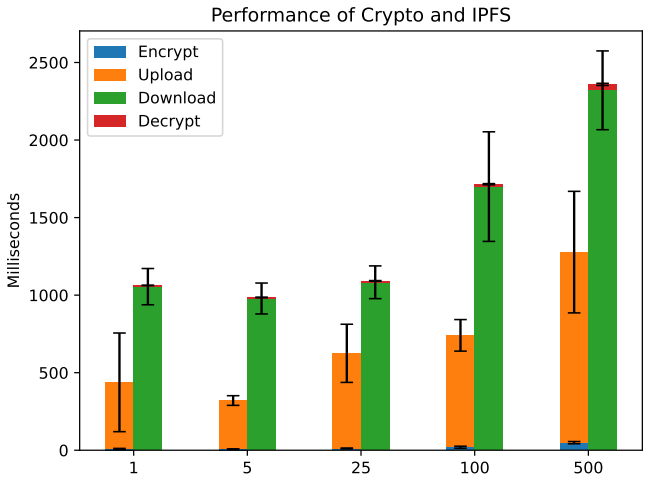


Fig. 2. Performance measurements in milliseconds for the cryptographic operations and interactions with IPFS.

Table IV shows the total cost incurred by each party involved in the transaction. In this scenario, the order needs time to be prepared, consequently, the seller pays for the prepared order transition too. The client cost is the highest with the total of €36.39, assuming they are finalizing the order without including a possible tip transaction. Although the number of transitions is higher, the seller and the deliverer have lower and similar costs of €25.94 and €23.36, respectively.

### B. Transaction time

For each state machine transition, a block on the blockchain should be mined to confirm it. Multi-signature transitions could be placed in the same block with two consecutive transactions. This means, an order going from `pending` to `completed` state takes from 7 to 10 blocks. In the best case, the *accepted* state is also skipped, and the order is ready to be picked up instantly. Moreover, two additional transactions are needed to approve the allowance on the `EURT` contract for the client and deliverer. Considering the average block time on the Ethereum blockchain to be 13 seconds [44] and between each transition 10 confirmations are needed to reach consensus irreversibly [45], the lower bound on time, required to complete an order, can be calculated to be between 910 seconds and 1300 seconds.

Since small read requests for IPFS can incur high resolving overheads [46], we additionally measure the overhead time needed to share the order information between the client and the seller. In particular, the time it takes to encrypt the order information, upload it to IPFS, then download it from IPFS and decrypt it on the seller side. Measurements are done using the JavaScript Performance API [47] from a browser instance connected to an Infura [48] IPFS node. We select five different order sizes, depicting the number of different items contained in the cart, and estimate the bytes needed to store them in plaintext and encrypted form. The results of the byte lengths depending on order size can be found in Table V. For each order size, we measure the milliseconds duration of each step 10 times and calculate mean and standard deviation. The results can be found in Figure 2, the client and the seller measurements are shown as two different bars. For orders smaller than 5, the client overhead is below 500 ms while the seller side is above 1000 ms. The download and decryption phase surpasses 1500 ms once the order size becomes larger than 100. In the case of 500 different items, the total measured time is close to 4000 ms. The majority of the time is used by the upload and download steps required to communicate with an IPFS node, and therefore, bound to the internet latency and speed.

TABLE V
ESTIMATED BYTE LENGTHS FOR DIFFERENT ORDER SIZES.

| Order size | Plaintext size | Encrypted IPFS size |
|---|---|---|
| 1 | 340 | 1344 |
| 5 | 1180 | 3584 |
| 25 | 5380 | 14784 |
| 100 | 21130 | 56784 |
| 500 | 105130 | 280784 |

## VI. DISCUSSION

Overall, the total cost of an order is €85.69, this is a fixed amount that takes into consideration both the payment processing and platform fees. The EUR cost is subject to change due to fluctuations in the price of `ETH` as well as changes in the average gas price. While the current fee would be prohibitive for small value transactions, we showed a multi-sided marketplace could technically be deployed on the Ethereum blockchain. Moreover, the smart contract was not designed to use the least amount of gas and therefore optimizations could be made in this regard. For example, reducing the number of transitions made by the state machine would reduce the required transactions and thus the gas usage too. A layer-two solution could also be built on top of the

smart contract to specifically handle micro-transactions using Zero-knowledge rollups [49], maintaining a more expensive but robust implementation on the first layer [50].

### A. Reputation and dispute systems

One of the initial ideas was to implement a reputation system that could determine the collateral automatically for the delivery service provider based on their reputation. Given an issue with an order, a dispute system could be implemented on top of the reputation system where any participating party could dispute an order. However, the realization that the reputation would be based on the publicly available data on the blockchain made it superficial. Either the reputation system is on-chain and users must pay even higher gas fees for its calculation, or the system can be off-chain but then anyone can build their own system and counter the original.

Furthermore, the reputation would be usable in only limited parts of the system. Firstly, on the rejection to check if the seller is rejecting a user with a high reputation and penalizing the seller for rejecting a reputable client. Nevertheless, this is not an issue in itself because the seller is a public business, and a rejected customer can go to publicly available portals to leave a negative review. Secondly, at a first glance, a user cancelling a lot of orders could seemingly cause some issues for the sellers. Nevertheless, a cancellation is a transaction which costs money to make, and the fee is already a good deterrent to not cancel maliciously because nothing is gained from it.

The aforementioned idea for the reputation use was to determine the collateral for a deliverer. The higher the reputation - the lower collateral is required. However, the goods that the deliverer would need to handle could be very valuable, so an argument could be made that a higher collateral than the value itself could be requested by the seller.

This leaves only a single part of the order life cycle where reputation would be used, namely, a successfully completed order. Consequently, the reputation would only increase and never decrease. However, since a seller has access to the delivery address, it can look up the client's order history on the blockchain to decide on approving or rejecting an order.

Given that the envisioned reputation system is not suitable and the conclusion to use the number of completed orders, the dispute system cannot work based on a single number. Handling disputes is a separate difficult problem that needs to be researched [51]. One way to build a reputation system is off-chain using mediators [52], [53]. However, creating an external dispute system was not the ambition of this work. Nevertheless, a system to handle disputable situations should be implemented for use in the real world.

### B. Time lock

Currently, orders do not enforce an expiration time; therefore, without the aforementioned dispute system, orders could get stuck indefinitely. This problem has been solved for payment channel networks [22] with the use of hash time-locked contracts. In this case, both parties sign a transaction to withdraw all their funds which will become valid only after a certain time. This has the effect of avoiding funds getting stuck for an indefinite period, and forces all parties in the transaction to complete their tasks in a timely manner. In particular, time-locks could be incorporated in various stages of the TOM contract, and they could even be set by the parties of the transaction during the initial approval phase. By choosing an appropriate expiration, the multi-signature part of the state machine could be enforced. Moreover, a larger time-lock could be placed on the entirety of the order, as an example, this could ensure fast deliveries for food delivery platforms. However, the business use case should be defined to appropriately determine which sections of the order stages should be time locked. Therefore, different contracts inheriting from the TOM main framework could be used instead. Additionally, a subset of possible disputes could then be replaced by the use of the time-locked contracts, nevertheless a proper dispute resolution system should be deployed to handle disputes not solved by time-locked contracts.

### C. ERC-20 Permit

At the moment, a user needs to send two different blockchain transactions when placing an order, the first one to authorize the token allowance, and then the actual TOM place order function. This is forced by the ERC-20 standard that does not allow a contract to transfer tokens unless the user has sent an approved allowance transaction first. This could already be solved by having the user set a large allowance before placing the order, which would then be depleted over time with subsequent orders. However, EIP-2612 [54] enables the user to sign an authorization off-chain, which could then be used by a smart contract to transfer the funds, similarly as if the user created an allowance on the blockchain. In this case, the EIP-712 [55] standard is used to create a valid signature that can be verified inside the smart contract.

## VII. Conclusion

In this paper we present the design and implementation of an Ethereum smart contract to support three-sided transactions for the application of city distribution and supply-chain management. Additionally, we provide a prototype application, connected to the deployed smart contract to showcase the working blockchain-based marketplace. Moreover, we presented the performance in terms of time and cost implications for transactions. With the current state of blockchain technology, it is not yet possible to create basic smart contracts without running into high gas costs. Even though TOM is held back by these gas costs when conducting transactions, this work has shown that the creation of such a decentralized application is feasible. We additionally proposed improvements that could be made to optimize the transactions cost, as well as introduce a dispute system with a layer-two solution.

## References

[1] Oberlo. Ecommerce Share of Retail Sales (2019–2025). https://www.oberlo.com/statistics/ecommerce-share-of-retail-sales. Accessed: 08-04-2022.

[2] A. Muneer, S. Razzaq, and Z. Farooq, "Data privacy issues and possible solutions in e-commerce," *Journal of Accounting & Marketing*, vol. 7, no. 3, pp. 1–3, 2018.

[3] Finder. (2021, May) Cryptocurrency adoption Rates. https://dvh1deh6tagwk.cloudfront.net/finder-us/wp-uploads/sites/5/2021/06/Crypto_Adoption_final-compressed-1.pdf. Accessed: 08-04-2022.

[4] Chainalysis. (2021, Oct.) The 2021 Geography of Cryptocurrency Report - Analysis of Geographic Trends in Cryptorrency Adoption and Usage. https://go.chainalysis.com/2021-geography-of-crypto.html. Accessed: 08-04-2022.

[5] V. Buterin, "Ethereum white paper: A next generation smart contract & decentralized application platform," 2013. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[6] C. Dannen, *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*, 1st ed. USA: Apress, 2017.

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[8] S. J. Pee, E. S. Kang, J. G. Song, and J. W. Jang, "Blockchain based smart energy trading platform using smart contract," in *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2019, pp. 322–325.

[9] E. S. Kang, S. J. Pee, J. G. Song, and J. W. Jang, "A Blockchain-Based Energy Trading Platform for Smart Homes in a Microgrid," in *2018 3rd International Conference on Computer and Communication Systems (ICCCS)*, 2018, pp. 472–476.

[10] A. Esmat, M. de Vos, Y. Ghiassi-Farrokhfal, P. Palensky, and D. Epema, "A novel decentralized platform for peer-to-peer energy trading market with blockchain technology," *Applied Energy*, vol. 282, p. 116123, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261920315373

[11] A. K. Shrestha, S. Joshi, and J. Vassileva, "Customer Data Sharing Platform: A Blockchain-Based Shopping Cart," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2020, pp. 1–3.

[12] V. P. Ranganthan, R. Dantu, A. Paul, P. Mears, and K. Morozov, "A decentralized marketplace application on the ethereum blockchain," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, 2018, pp. 90–97.

[13] F. Funk and J. Franke, "Matching in decentralized two-sided markets via Blockchain-based deferred acceptance," in *2021 3rd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*, 2021, pp. 89–92.

[14] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[15] M. Kadadha, H. Otrok, S. Singh, R. Mizouni, and A. Ouali, "Two-sided preferences task matching mechanisms for blockchain-based crowdsourcing," *Journal of Network and Computer Applications*, vol. 191, p. 103155, 2021.

[16] Y. Wu, S. Tang, B. Zhao, and Z. Peng, "BPTM: Blockchain-Based Privacy-Preserving Task Matching in Crowdsourcing," *IEEE Access*, vol. 7, pp. 45 605–45 617, 2019.

[17] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang, and R. H. Deng, "Crowdbc: A blockchain-based decentralized framework for crowdsourcing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1251–1266, 2019.

[18] T. Klein, H. P. Thu, and T. Walther, "Bitcoin is not the new gold–a comparison of volatility, correlation, and portfolio performance," *International Review of Financial Analysis*, vol. 59, pp. 105–116, 2018.

[19] R. K. Lyons and G. Viswanath-Natraj, "What keeps stablecoins stable?" National Bureau of Economic Research, Tech. Rep., 2020.

[20] Bitcoin Wiki. (2021, Apr.) Script. https://en.bitcoin.it/wiki/Script. Accessed: 08-04-2022.

[21] ——. (2021, Jul.) Multi-signature. https://en.bitcoin.it/wiki/Multi-signature. Accessed: 08-04-2022.

[22] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[23] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[24] L. Brünjes and M. J. Gabbay, "Utxo-vs account-based smart contract blockchain programming paradigms," in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2020, pp. 73–88.

[25] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual international cryptology conference*. Springer, 2017, pp. 357–388.

[26] M. Chakravarty, R. Kireev, K. MacKenzie, V. McHale, J. Müller, A. Nemish, C. Nester, M. P. Jones, S. Thompsona, R. Valentine *et al.*, "Functional blockchain contracts," 2019.

[27] M. M. Chakravarty, S. Coretti, M. Fitzi, P. Gazi, P. Kant, A. Kiayias, and A. Russell, "Hydra: Fast isomorphic state channels," *Cryptology ePrint Archive*, 2020.

[28] M. Jansen, F. Hdhili, R. Gouiaa, and Z. Qasem, "Do smart contract languages need to be turing complete?" in *International Congress on Blockchain and Applications*. Springer, 2019, pp. 19–26.

[29] F. Vogelsteller and V. Buterin, "Eip 20: Erc-20 token standard," *Ethereum Improvement Proposals*, vol. 20, 2015.

[30] W. Entriken, D. Shirley, J. Evans, and N. Sachs, "Eip-721: Non-fungible token standard," *Ethereum Improvement Proposals*, vol. 721, 2018.

[31] OpenZeppelin. (2022, Apr.) Security Audits - OpenZeppelin blog. https://blog.openzeppelin.com/security-audits. Accessed: 08-04-2022.

[32] Y. Faqir-Rhazoui, M.-J. Ariza-Garzón, J. Arroyo, and S. Hassan, "Effect of the gas price surges on user activity in the daos of the ethereum blockchain," in *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–7.

[33] G. A. Pierro and H. Rocha, "The influence factors on ethereum transaction fees," in *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 2019, pp. 24–31.

[34] ConsenSys Software Inc. (2022, Apr.) Truffle Suite. https://trufflesuite.com. Accessed: 08-04-2022.

[35] OpenZeppelin. (2022, Apr.) OpenZeppelin — Contracts. https://openzeppelin.com/contracts. Accessed: 08-04-2022.

[36] J. Benet, "Ipfs - content addressed, versioned, p2p file system."

[37] D. J. Bernstein, "Cryptography in nacl," *Networking and Cryptography library*, vol. 3, no. 385, p. 62, 2009.

[38] E. You. (2022, Apr.) Vue.js - The Progressive JavaScript Framework. https://vuejs.org/. Accessed: 08-04-2022.

[39] R. Moore. (2022, Apr.) The Ethers Project. https://github.com/ethers-io/ethers.js. Accessed: 08-04-2022.

[40] Metamask. (2022, Apr.) The crypto wallet & gateway to Web3 blockchain apps — MetaMask. https://metamask.io. Accessed: 08-04-2022.

[41] Coinbase. (2022, Apr.) Coinbase Wallet. https://www.coinbase.com/wallet. Accessed: 08-04-2022.

[42] Etherscan. (2022, Apr.) Ethereum Average Gas Price Chart. https://etherscan.io/chart/gasprice. Accessed: 08-04-2022.

[43] ——. (2022, Apr.) The Ethereum Blockchain Explorer. https://etherscan.io. Accessed: 08-04-2022.

[44] ——. (2022, Apr.) Ethereum average block time chart. https://etherscan.io/chart/blocktime. Accessed: 08-04-2022.

[45] V. Buterin. (2015, Sep.) On Slow and Fast Block Times. https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times. Accessed: 08-04-2022.

[46] J. Shen, Y. Li, Y. Zhou, and X. Wang, "Understanding i/o performance of ipfs storage: A client's perspective," in *Proceedings of the International Symposium on Quality of Service*, ser. IWQoS '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/3326285.3329052

[47] mdn. (2022, Mar.) Performance API. https://developer.mozilla.org/en-US/docs/Web/API/Performance_API. Accessed: 08-04-2022.

[48] Infura. (2022, Apr.) Ethereum API — IPFS API & Gateway — ETH Nodes as a Service — Infura. https://infura.io. Accessed: 08-04-2022.

[49] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 201–226.

[50] O. Marukhnenko and G. Khalimov, "The overview of decentralized systems scaling methods," *COMPUTER AND INFORMATION SYSTEMS AND TECHNOLOGIES*, 2021.

[51] D. W. Allen, A. M. Lane, and M. Poblet, "The governance of blockchain dispute resolution," *Harv. Negot. L. Rev.*, vol. 25, p. 75, 2019.

[52] Bisq. (2022, Apr.) Bisq - A decentralized bitcoin exchange network. https://bisq.network. Accessed: 08-04-2022.

[53] C. Lesaege and F. Ast, "Short paper v1. 0.7 clément lesaege, federico ast, and william george september 2019," 2019.
[54] M. Lundfall, "Eip-2612: permit – 712-signed approvals," *Ethereum Improvement Proposals*, vol. 2612, 2020.
[55] R. Bloemen, L. Logvinov, and J. Evans, "Eip-712: Ethereum typed structured data hashing and signing," *Ethereum Improvement Proposals*, vol. 712, 2017.