Blockchain: Research and Applications 1 (2020) 100001

Contents lists available at ScienceDirect

Blockchain: Research and Applications

journal homepage: www.journals.elsevier.com/blockchain-research-and-applications

Application and evaluation of payment channel in hybrid decentralized ethereum token exchange

Xuan Luo [a,1], Zehua Wang [a,b,1,*], Wei Cai [c,d], Xiuhua Li [e], Victor C.M. Leung [a,f]

[a] Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, Canada
[b] Blockchain@UBC, The University of British Columbia, Vancouver, Canada
[c] The Chinese University of Hong Kong, Shenzhen, China
[d] Shenzhen Institute of Artificial Intelligence and Robotics for Society, Shenzhen, China
[e] Chongqing University, China
[f] Shenzhen University, China

ARTICLE INFO

Keywords:
Blockchain
Payment channel
Ethereum
Smart contract
Token exchange
Optimal gas price

ABSTRACT

Traditional centralized token exchange (CEX) has been suffering from hacking due to the centralized management of users' tokens. In contrast, decentralized token exchange (DEX) maintains users' assets by smart contracts in a decentralized manner, but introduces additional overhead in terms of *gas* fee and *transaction confirmation latency*. Hybrid decentralized token exchange (HEX) has been proposed to combine the benefits of CEX and DEX. However, existing HEX is criticized for two issues. First, trading transactions are time-consuming and expensive for frequent token traders. Second, excessive simultaneous transactions might cause the pending transaction congestion in the Ethereum network. In this paper, we propose a payment channel based HEX, which extends existing solutions by adding a new payment channel layer to benefit frequent traders and alleviate the pending transaction congestion. Besides, we propose the very first gas-price vs. transaction-confirmation-latency function to guide Ethereum transaction issuers to choose an optimal gas price that minimizes the overall cost. Extensive simulations are conducted to compare the cost in the proposed HEX with that in the conventional HEX. The results demonstrate the effectiveness of our proposed mechanism in terms of reducing gas fee and transaction confirmation latency for frequent traders as well as the pending transaction congestion in Ethereum.

1. Introduction

A blockchain Nakamoto [1] is a decentralized public ledger that is used to record transactions across a peer-to-peer network so that any involved record cannot be altered retroactively. The possible applications are immense Khan and Salah [2], from cryptocurrency to automated payment without third party's intervention Hong et al. [3], to digital content source tracking Hasan and Salah [4], and to access control and security protection of the Internet of things (IoT) devices Al Breiki et al. [5] and Wu et al. [6], and to privacy preservation Cheng et al. [7] and Wu et al. [6]. As the killer decentralized application hosted by blockchain, cryptocurrencies have been accepted as the digital cash by many investors and consumers nowadays. According to CoinMarket,[2] more than 90% of the top 100 cryptocurrencies are Ethereum Buterin [8] based tokens. Thus, many token exchange platforms are for Ethereum tokens nowadays. Generally, we classify current token exchange into three categories, *i.e.*, centralized token exchange (CEX), decentralized token exchange (DEX), and hybrid decentralized token exchange (HEX). CEX

*Corresponding author. Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, Canada.
E-mail addresses: xuanluo2@ece.ubc.ca (X. Luo), zwang@ece.ubc.ca (Z. Wang), caiwei@cuhk.edu.cn (W. Cai), lixiuhua@cqu.edu.cn (X. Li), vleung@ieee.org (V.C.M. Leung).

Production and Hosting by Elsevier on behalf of KeAi

[1] The first two authors contributed equally to this research.
[2] https://coinmarketcap.com/, accessed on September 10, 2018.

https://doi.org/10.1016/j.bcra.2020.100001
Received 6 July 2020; Received in revised form 27 November 2020; Accepted 30 November 2020

has been the most popular token exchange ever since its prevalence. The core idea of CEX is centralization, which is achieved in the way that users are required to deposit tokens to a CEX-provided address and thus all users' assets are taken care of by the CEX instead of by users themselves. The centralization nature of the CEX brings two benefits to users. First, CEX provides a rapid trading speed since any trading transaction leads to a direct update on the CEX database instead of an update on the blockchain Cai et al. [9]. Second, CEX gives users the option to trade in private, since trading records of a particular user will not be shown publicly in blockchain. However, the centralization nature of CEX is contradictory with the decentralization spirit of the blockchain and distributed ledger. It also makes CEX intrinsically vulnerable to hacking and denial of service attacks.

Different from centralized management of users' tokens in the CEX, DEX lets users manage their own digital currencies and acts as a transparent middle-man for token traders. In the DEX, trading procedures are implemented in smart contracts Álvarez Díaz et al. [10], which are the bytecode programs recorded on the blockchain and executed automatically without third party's intervention. Users conduct their trades by addressing a blockchain transaction to trigger a procedure defined in the smart contract. DEX minimizes the potential security issue in the CEX by leveraging automatically executed smart contracts, but it also introduces two critical issues. First, the lack of order matching service makes it difficult for token traders to find appropriate counterparties. Second, all trading requests will be submitted to blockchain as blockchain transactions, which are constrained by the low transaction throughput in Ethereum due to factors such as the Proof of Work consensus algorithm Back [11] and scalability issues of EVM Chen et al. [12].

HEX is a hybrid approach that combines the benefits of CEX and DEX. HEX addresses the order matching issue by maintaining a centralized order management database, while all trades are still conducted by calling procedures in the smart contract. However, this approach does not fix the issue that all trading requests will be submitted to blockchain as blockchain transactions, which are constrained by the low transaction throughput in Ethereum. This is particularly important for frequent token traders, as more blockchain transactions imply more gas fee and longer transaction confirmation latency. On the other hand, excessive simultaneous transactions might cause the pending transaction congestion in Ethereum.

1.1. State-of-the-art token exchange

Mt.Gox[3] is the first well-known CEX for Bitcoin tradings. Unfortunately, there are not just theoretical risks but disasters that have occurred for thousands of cryptocurrency investors in the past. In 2014, Mt.Gox claimed that 850,000 BTCs belonging to customers but the company were missing, which led to the loss of thousands of customers. However, CEX is still the mainstream of token exchange nowadays. Popular token exchange, including Coinbase,[4] Gemini,[5] Poloniex,[6] Kraken,[7] and Houbi,[8] still adopts the CEX architecture.

Relying on smart contracts, DEX provides more secure trading services to the cryptocurrency users. DEX like KyberNetwork[9] and AirSwap[10] decentralizes the settlement and all related functions by defining them as the on-chain procedures. However, DEX is still unpopular among investors, due to the lack of centralized order management. Since DEX uses on-chain orderbook to manage orders, DEX users have to monitor the on-chain orderbook from time to time in order to find potential matching orders.

Combining the benefits of CEX and DEX, HEX represents the latest version of token exchange. EtherDelta[11] and 0x Project[12] are described as decentralized exchange but they are more like a hybrid design. They decentralize the settlement and use a centralized server to handle the orderbook. However, as one successful trade requires three blockchain transactions, users of EtherDelta and 0x Project will pay for extra blockchain gas fee and have to wait for three blockchain confirmations. Besides, there is no order matching engine provided, so the buyers have to monitor the market all the time. Also, many buyers may compete for one selling order, and the unsuccessful buyers waste their gas fees. IDEX[13] and JOYSO[14] improve the above issues with an automatic order matching engine. Especially, JOYSO claims that traders can continue trading on JOYSO without the successful confirmation of the previous trading transaction on the blockchain. Yet, this causes a potential security issue, where a failure of the broadcast of a matching order to blockchain will invalidate all dependent orders. Moreover, hacking of the token exchange may cause the loss of all users' assets.

1.2. Comparisons of state-of-the-art token exchange

There are some key features which are equally important reasons for a token exchange structure. As shown in Fig. 1, we look at the following key features when comparing state-of-the-art token exchange in this section:

- **Security**: The centralization nature of CEX makes it vulnerable to hacking and denial of service attacks. While, DEX, the conventional HEX and the proposed payment channel based HEX let users manage their own digital currencies by utilizing smart contracts. Thus, the security of user assets in CEX is relatively low compared to that in both DEX and the conventional or payment-channel-based HEX.
- **Order Matching Support**: Order Matching Support refers to the ability that a token exchange is able to match a selling order with an appropriate buying order without interfere of token traders. Order matching is not supported in DEX since it utilizes a decentralized orderbook and token traders have to manually identify suitable trades. While, order matching is supported in CEX, the conventional HEX, as well as the proposed payment-channel-based HEX since they use centralized orderbooks.
- **Gas Fee**: In CEX and the proposed payment-channel-based HEX, frequent traders conduct their trades off-chain in the HEX platform. While, in DEX and the conventional HEX, every successful trade leads to a blockchain transaction. Therefore, frequent traders in DEX and the conventional HEX pay for a higher gas fee compared to users in CEX and the proposed payment-channel-based HEX.
- **Transaction Confirmation Latency**: Frequent traders need to wait for blockchain confirmations to complete their trades when conduct trades in DEX and the conventional HEX, which is not necessary when they trade in CEX and the proposed HEX. Thus, transaction confirmation latency in DEX and the conventional HEX is relatively higher than that in CEX and the proposed HEX.
- **Transaction Congestion**: All trading requests are blockchain transactions in DEX and the conventional HEX. While, trades are performed by sending Internet data packets off-chain in CEX and the proposed HEX. Therefore, DEX and the conventional HEX are more likely to cause the pending transaction congestion than CEX and the proposed payment-channel-based HEX.

[3] https://www.mtgox.com/.
[4] https://www.coinbase.com/.
[5] https://gemini.com/.
[6] https://poloniex.com/.
[7] https://www.kraken.com/.
[8] https://www.huobi.com.
[9] https://developer.kyber.network/docs/ArchitectureOverview.
[10] https://blog.airswap.io/introducing-swap-a-protocol-for-decentralized-peer-to-peer-trading-on-the-ethereum-blockchain-d4058f3179cf.
[11] https://github.com/etherdelta.
[12] https://0xproject.com/pdfs/0x_white_paper.pdf.
[13] https://idex.market/static/IDEX-Whitepaper-V0.7.5.pdf.
[14] https://joyso.io/whitepaper.pdf.

Token Exchange	Security	Order Matching Support	Gas Fee	Transaction Confirmation Latency	Transaction Congestion
CEX	Low	Yes	Low	Low	Low
DEX	High	No	High	High	High
Conventional HEX	High	Yes	High	High	High
Desired HEX (Our Solution)	High	Yes	Low	Low	Low

Fig. 1. Comparison of key features.

As shown in Fig. 1, it is obvious that none of the state-of-art token exchange could provide friendly user experience for frequent token traders. To tackle this issue, we propose to extend the conventional HEX by adding a new payment channel layer to decrease gas fee and transaction confirmation latency for frequency token trader as well as to alleviate transaction congestion in blockchain.

1.3. Payment channel

A payment channel is the technique designed to allow users to make a series of off-chain payments. So, it can facilitate multiple cryptocurrency trades without committing each of them to blockchain one by one. With a typical payment channel, only two on-chain transactions will be recorded on the blockchain, while (nearly) unlimited number of off-chain payments between the participants can happen in the middle of the two on-chain transactions. For instance, Alice creates a channel to Bob by initiating an on-chain deposit transaction to deposit tokens into the deployed smart contract that serves as an escrow account. Alice can then make an arbitrary number of rapid payments to Bob, by signing the digital signatures of the *deposit division agreement* (*i.e.*, an agreement that allocates Alice's deposit in the escrow account between Alice and Bob) and sending the signatures to Bob over the Internet. The signatures sent from Alice and received by Bob are not tokens, but Bob can eventually receive the tokens by closing the payment channel. When Bob closes the payment channel, the latest deposit division signature created by Alice is required to invoke the procedure in the smart contract to distribute the deposit to counterparties' accounts. This is the reason that we call the deposit division signatures sent from Alice to Bob the off-chain payments. Note that the smart contract has nothing to do with the off-chain payments, as the off-chain payments are purely done by sending the Internet packets from Alice to Bob with Alice's signatures. Hence, these intermediate off-chain payments will not be processed by the blockchain miners.

Payment channels can be classified into two types, namely, the uni-directional and bi-directional payment channels. An uni-directional payment channel only allows single directional off-chain payment. In CLTV-style uni-directional payment channel,[15] the security of the payment channel is based on the fact that the payment receiver does not have the incentive of using an old signature issued by the spender to close the payment channel. On the other hand, a bi-directional payment channel allows both counterparties to send off-chain payments. Therefore, one may have the incentive of using an old signature issued by the counterparty to close the payment channel and receive more asset than he/she deserves. Thus, new security models are required to resolve this problem. The duplex payment channel Decker and Wattenhofer [13] uses time-locking transactions to resolve the above problem to an extent. The newest signature is always created with a shortest time-lock, meaning that it is the first one that triggers the channel closing procedure. While, due to the intrinsic mechanism, duplex payment channels have limited lifetime. Poon-Dryja payment channel in the Lightning Network[16] solves the lifetime issue in the duplex payment channel by taking advantage of multi-signature and hashed time-lock contract[17] technologies. The security of Poon-Dryja payment channel is based on the punishment of a malevolent counterparty rather than on time. A challenging-period will be specified by both counterparties when creating the payment channel. When a dishonest counterparty tries to close the payment channel by submitting its opponent's payment signature to blockchain, if the opponent could submit a newer payment signature received from the dishonest counterparty within the challenging-period, the dishonest side would be punished and all deposits in the escrow account could be transferred to the honest side.

Payment channels have been widely used in off-chain peer to peer micro-payments, like in the Lightning Network. While, such application of payment channel in the Lightning Network is not suitable in HEX due to the following two reasons: First, most of the time a token trader will not trade with the same opponent continuously. If a trader needs to establish a new payment channel when there is no existing payment channel to the target trader, the overhead of creating and closing the payment channel could be not economical. Second, Lightning Network only supports exchanging two types of tokens within the entire network, while in practical token exchange, it is common for users to exchange more than two types of tokens within a certain period. Therefore, in order to decrease the overhead of our proposed scheme, users will establish a payment channel with the token exchange instead of with other users directly. Such application of payment channel in HEX also allows the multi-token exchange, as we will present in the rest of the paper.

In this paper, we extend the existing HEX solution by adding a new payment channel[18] layer to decrease gas fee and transaction confirmation latency for frequent token traders. The payment channel is a technique allowing for off-chain transactions with a final on-chain settlement Xiao et al. [14]. We extend the draft design of payment channel application in HEX in Luo et al. [15] by systematically designing and implementing a minimal viable program (MVP) of the proposed solution. We also validate the effectiveness of the payment-channel-based HEX by the

[15] https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki.

[16] https://medium.com/cardstack/scalable-payment-pools-in-solidity-d97e45fc7c5c.

[17] http://www.lightning.network/lightning-network-paper-DRAFT-0.5.pdf.

[18] https://en.bitcoin.it/wiki/Payment_channels.

experimental comparisons with conventional HEX in this paper. The major contributions of the work are as follows:

- We systematically design the very first payment channel based HEX to benefit frequent token traders. Compared with our previous work in Luo et al. [15], our updated solution presented in this paper allows a payment channel to be topped up, designs how the payment channel between HEX users and the HEX could be closed to minimize users' loss when potential attacks targeting to users occur, and introduces security assumptions and potential attacks in the payment channel based HEX. The proposed scheme supports multi-token off-chain exchange, and decreases the total gas fee and transaction confirmation latency for frequent traders.
- We propose the very first gas-price vs. transaction-confirmation-latency function to help blockchain transaction issuers to decide an optimal gas price that minimizes the overall (*i.e.*, monetary and waiting time) cost.
- Based on the proposed gas-price vs. transaction-confirmation-latency function, we quantitatively evaluate the performance of the payment-channel-based HEX. The gas-price vs. transaction-confirmation-latency function can also be used to quantitatively evaluate the performance of payment channel in general applications using the payment channel technique.

In the rest of this paper, the system overview and security analysis of the proposed HEX are presented in Sections 2 and 3, respectively. Mathematical modelling and experiments are in Sections 4 and 5, respectively. Section 6 concludes the paper.

2. System overview

In this section, we will present the design of the proposed payment-channel-based HEX system.

2.1. System architecture

Fig. 2 illustrates the proposed system framework, which consists of three layers, namely, the on-chain, payment channel, and the off-chain layers. The payment channel layer works as a bridge to connect the on-chain layer with the off-chain layer.

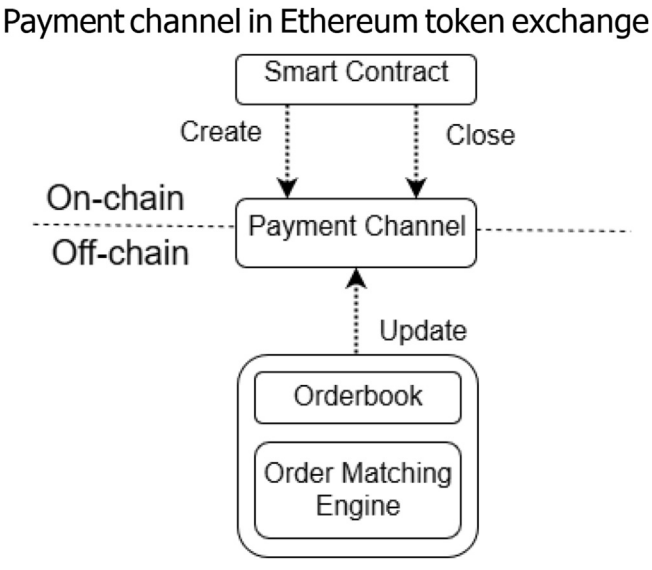

Fig. 2. Architecture of the payment channel based HEX.

2.1.1. On-chain layer

The on-chain layer is the key to securing users' assets as well as creating and closing the payment channels. The deployed smart contract in the on-chain layer acts as a verifiable, open source, and trustworthy escrow account.

Before a HEX user is able to exchange tokens in the HEX platform, both the user and the HEX need to first deposit tokens into the smart contract escrow account. Then, a payment channel creating transaction with both counterparties' signatures has to be submitted to the blockchain in order to create a payment channel. Afterward, the user and the HEX could exchange tokens off-chain by exchanging signatures to update the deposit division agreement in an off-chain manner. When a user wants to withdraw his/her deposit, the user could submit a signed payment channel closing transaction to trigger the payment channel closing procedure. Settlement can thus be done on-chain by calling the corresponding procedure defined in the smart contract, which cannot be altered or interfered with.

To introduce the on-chain layer in details, the main functions of the smart contract are defined as follows:

- **Deposit**: Both the user and the HEX send their deposits to the smart contract. Since smart contract cannot be altered or interfered with, it acts like a trustworthy escrow account.
- **Withdraw**: A user is only allowed to withdraw his/her deposit that is not locked in the payment channel. If a user wants to withdraw the deposit that is locked in the payment channel, the user needs to close the payment channel in order to do so.
- **Create a payment channel**: To create a payment channel with the HEX, a user shall firstly sign a time-locked message to create a payment channel with a maximum lifetime and a challenging-period. After the time specified by the time lock, the message becomes invalid and could not be used to create a payment channel any more. The maximum lifetime of the payment channel determines the duration that the user could exchange tokens in the HEX platform after a payment channel has been created. The challenging-period specifies a time duration after a counterparty requests to close the payment channel, and within this duration, the opponent can dispute. The user sends the payment channel creating signature to the HEX, and if the HEX agrees to create the payment channel with the user, the HEX and the user exchange their initial deposit division agreement signatures. This is because that the HEX and the user should be allowed to close the payment channel even if the user has not traded any token yet (*i.e.*, get their initial deposits back to individual's accounts). The user then needs to send both counterparties' signatures to the blockchain to trigger the creation of a new payment channel in the smart contract before the time specified by time lock.
- **Top up a payment channel**: When a user wants to add some more deposit for exchange while holding a payment channel with the HEX, the user could deposit into the smart contract and lock the new deposit into the existing payment channel. Also, the HEX can top up an existing payment channel by locking some more deposit into the payment channel.
- **Close a payment channel**: There are three scenarios where a payment channel can be closed:
- Both the user and the HEX agree to close a payment channel. In this scenario, no matter if the payment channel reaches its maximum lifetime or not, both counterparties exchange their signatures of the agreement on payment channel closure. Then, the user or the HEX can send the two-party signed payment channel closing message to the smart contract and the locked deposits in the payment channel are divided and saved in both counterparties' accounts on-chain at once.
- Only one counterparty wants to close the payment channel before the maximum lifetime of the payment channel. In this case, the payment channel can also be closed but not so straight forward as above.

Without loss of generality, we assume Alice is a HEX user and wants to close the payment channel with the HEX. Alice can use the off-chain payment signature received from the HEX to *request* to close the payment channel unilaterally by calling a procedure in the smart contract. The smart contract first verifies the signature and then set the status of the payment channel to "requested-to-close". The payment channel enters the challenging-period. Within the challenging-period, if the HEX could send the smart contract a signature created by Alice chronically later than the signature used by Alice to request to close the channel, Alice is proved cheating. The predefined penalty subroutine could be triggered and the payment channel can be closed with bias. Otherwise, if no dispute happens, the payment channel can be closed after the challenging-period. The assets in the payment channel are split according to what revealed by the signature that Alice used when requesting to close the payment channel.

- Only one counterparty wants to close the payment channel after the maximum lifetime of the payment channel. After the payment channel has reached its maximum lifetime, one counterparty could trigger the close of the payment channel with its own signature. Then, the deposits in the locked payment channel will be distributed according to the initial deposit division. Such design is to minimize users' loss when potential attacks targeting to users occur. This will be introduced later with more details when we analyze the security of the proposed HEX.

2.1.2. *Payment channel layer*

Payment channel layer is the bridge to connect the on-chain layer with the off-chain layer. Since creating a payment channel requires deposits from both counterparties, the main usage of the payment channel is to establish the trustworthiness between the counterparties, so that they can conduct off-chain trades by exchanging the signatures of the agreement on the deposit division.

Here, we choose the bi-directional payment channel instead of the uni-directional payment channel because the users exchanging tokens with each other needs the HEX to serve as a transparent hub, so the token exchanges between the HEX and one user is also bi-directional. For example, Alice wants to exchange tokens with any other token traders for three times over the HEX, the detailed process of which is presented in Fig. 3. Both Alice and the HEX need to deposit into the smart contract and sign an initial deposit division agreement in order to create a payment channel. The initial deposit division agreement is the Trade 0 in Fig. 3. Then, Alice exchanges tokens three times. In Trade 1, Alice buys in 5 VERI with 0.5 ETH (*i.e.*, the other token trader sells out 5 VERI for 0.5 ETH). In Trade 2, Alice buys in 50 OMG with 5 VERI (*i.e.*, the other token trader sells out 50 OMG for 5 VERI). In Trade 3, Alice buys in 100 REP with 10 OMG (*i.e.*, the other token trader sells out 10 OMG for 100 REP). Then, Alice wants to stop trading. The payment channel is closed by sending a two-party signed payment channel closing transaction based on the latest deposit division agreement to the blockchain. Then, 0.5 ETH, 40 OMG, and 100 REP are transferred to Alice's account.

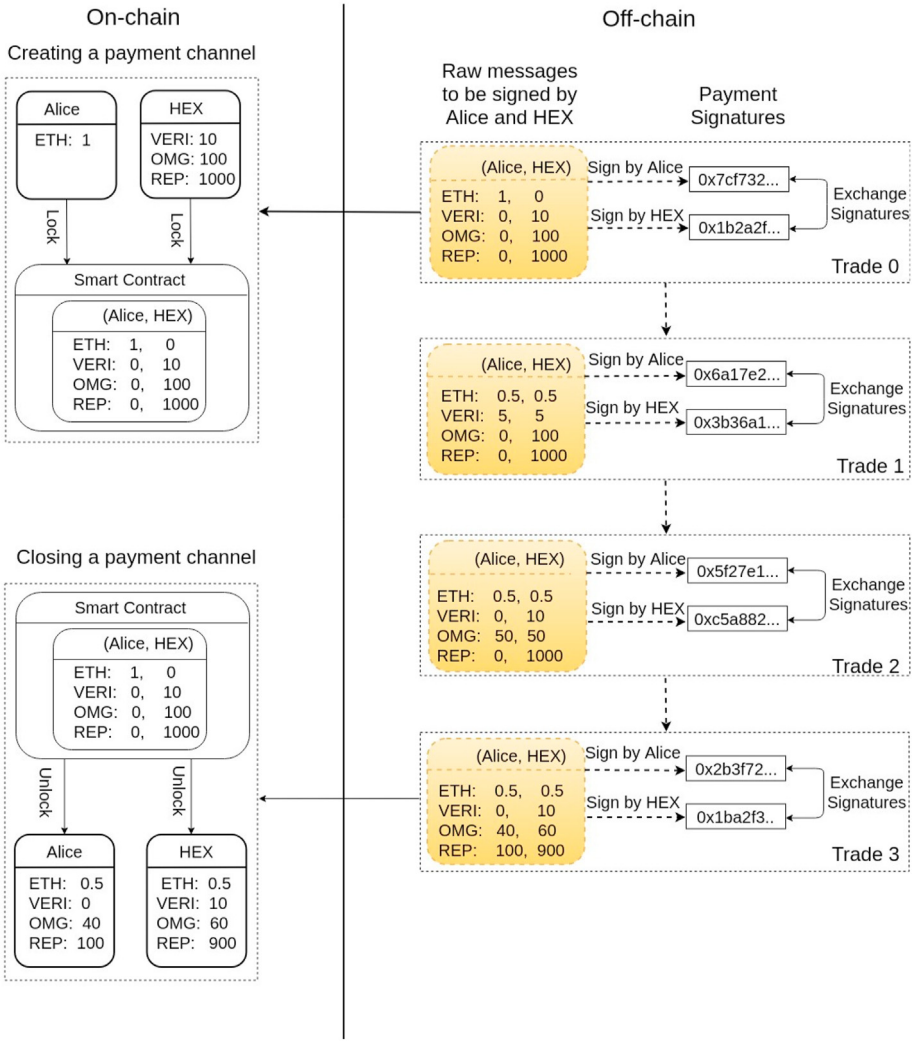

Fig. 3. An illustration of payment channel between Alice and the HEX.

2.1.3. Off-chain layer

The off-chain layer is responsible for order execution, which includes order placement, order canceling, and order matching.

After a payment channel is created between the user and the HEX, the user is able to continuously trade with the HEX off-chain. To start a new trade, the user could place an order by sending a signed buy/sell order to the HEX. The signature of the buy/sell order here is only used to show the user's willingness to trade and not used to update the deposit division agreement in the on-chain escrow account. The new order will then be recorded in the orderbook. When the HEX finds a matching order via order matching engine, the HEX would then notify the user immediately and ask the user to reply within a specific time. Afterward, the user needs to send a payment signature to the HEX before the time specified by the HEX. The HEX then exchanges its payment signature to complete the order. If the user does not reply within the time specified by the HEX, the related order will be cancelled by the HEX. If a user wants to proactively cancel the order, the user needs to sign and send another order-cancel request to the HEX before the HEX finds a matching order for the user's order.

2.2. Comparisons with existing solutions

To help readers better understand the system architecture of our proposed HEX system, we will compare our system with existing solutions in two aspects, *i.e.*, working mechanism of the system and order execution model in token exchange.

2.2.1. Working mechanism

To compare the working mechanism, Fig. 4 and Fig. 5 illustrate the key components and continuous trading workflows of the conventional HEX and the proposed payment channel based HEX, respectively. According to Figs. 4 and 5, it is obvious that the proposed HEX adds a payment channel layer to alter the workflow of trading procedures.

As shown in Fig. 4, a user in the conventional HEX deposits tokens into the smart contract before starting a trade. Then, the user sends a signed token buy/sell order to the HEX. If the HEX finds a matching order, the HEX signs both the buy and sell orders issued by two token traders to approve the trade. Afterward, the signed matching orders will be submitted to the blockchain by the HEX for deposit settlement. For continuous trades, a user needs to repeat Steps 2, 5, and 6 in Fig. 4.

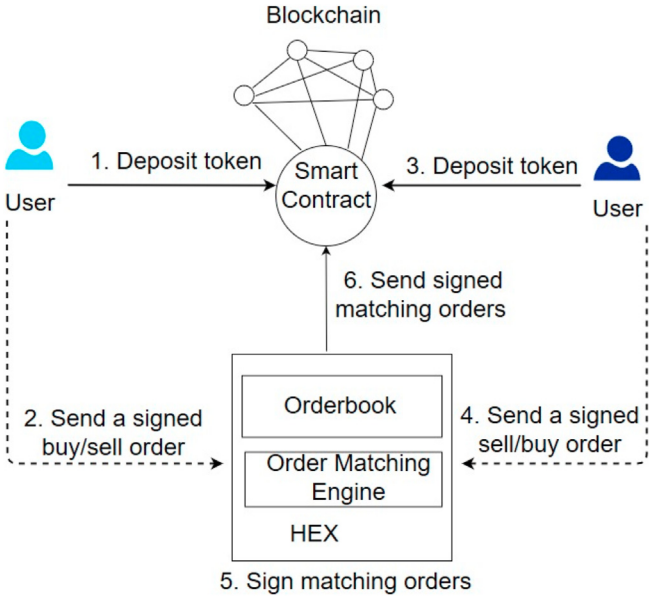

Fig. 4. An example for how a trader exchanges tokens with the HEX without payment channel. For continuous trades, a user needs to repeat Steps 2, 5, and 6.

As shown in Fig. 5, a user in our proposed system also deposits tokens into the deployed smart contract first. While, before the user can conduct the off-chain token exchange, he/she needs to sign and exchange a payment channel creating message with the HEX in order to create a payment channel. After a payment channel is created, the user could place signed buy/sell orders to show his/her willingness to exchange tokens. If a matching order could be found by the HEX, both the seller and the buyer then exchange with the HEX their digital signatures that reveal their agreements on new deposit divisions for their deposits in the on-chain escrow account. For continuous trades, a user needs to repeat order placement and payment signature exchange in Steps 7, 8, 9, 10, and 11 in Fig. 5.

According to the workflows of continuous trades, Step 6 in Fig. 4 is on-chain transaction, while Steps 7, 8, 9, 10, and 11 in Fig. 5 are completely off-chain. Thus, the proposed payment-channel-based HEX eliminates the number of on-chain transactions for frequent token tradings, and achieves better performance over conventional HEX in terms of total gas fee and transaction confirmation latency.

2.2.2. Order execution model

To compare the execution model, we look at the following aspects: whether order placement is on-chain and whether order cancellation and matching are supported.

- **Order placement**: Orders in token exchange could be classified into two types in terms of the impact on cryptocurrency liquidity, *i.e.*, the make order and the take order. In the context of token exchange, cryptocurrency liquidity refers to the ability of a token to be converted into other tokens easily. The make order is a buy/sell order that will not be fulfilled immediately, and thus, adds liquidity to an exchange. While, the take order is a buy/sell order that will be immediately fulfilled, and thus, decreases liquidity to an exchange. Table 1 shows whether make and take orders are placed on-chain in different token exchange.
- **Order cancellation**: As shown in Table 2, we compare how order cancellation takes place in different token exchange. In CEX, the conventional HEX, and the proposed HEX, order cancellation is free since orders are placed off-chain and could be cancelled off-chain. While, users in DEX have to send an order canceling transaction to the DEX smart contract to cancel an order, and thus, it costs extra Ethereum gas fee. To successfully cancel an order, users in CEX and the conventional HEX need to rely on the token exchange, while users in DEX and the proposed HEX could cancel the order at their will before the order is fulfilled.
- **Order matching**: Table 3 shows whether order matching is supported in different token exchange. DEX does not support order matching as it utilizes a decentralized orderbook on the blockchain to record orders. While, order matching is supported in CEX, the conventional HEX, and the proposed payment channel based HEX, since all of them record orders using centralized orderbooks.

3. Security analysis

In this section, we will introduce the security assumptions and potential attacks in the proposed payment-channel-based HEX.

3.1. Security assumptions

We have the following assumptions for the proposed payment-channel-based HEX system:

- The HEX is always online to provide service to the HEX users (*i.e.*, token traders).
- The user needs to be notified when the HEX requests to close the payment channel. Techniques like *watchtower* Osuntokun [16] could

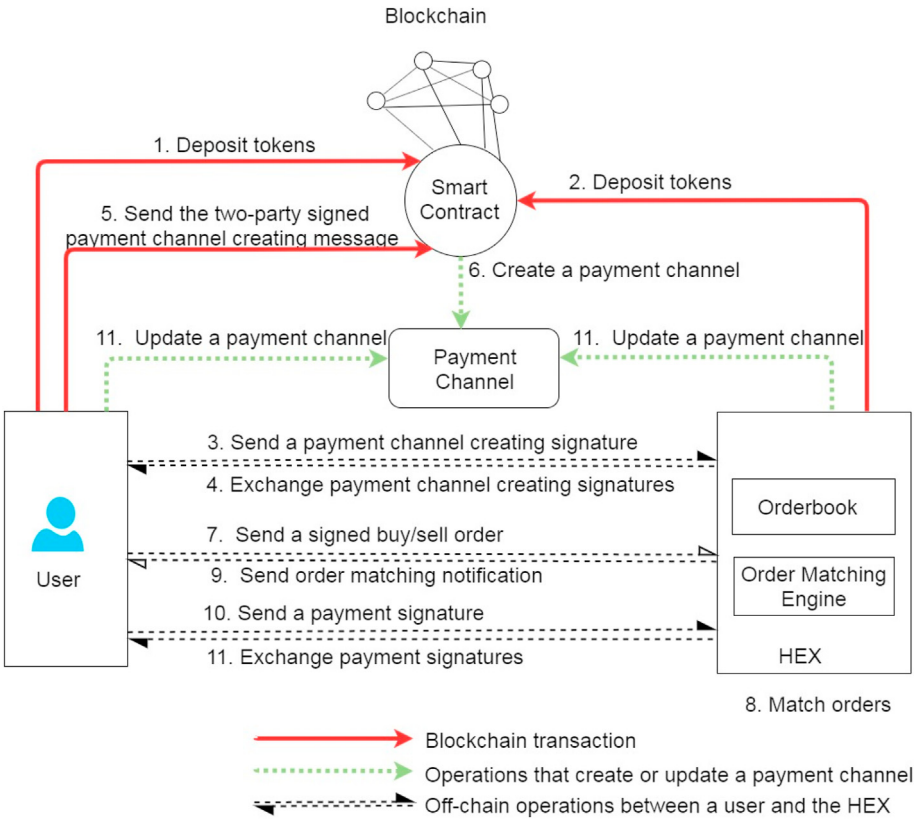

Fig. 5. An example for how a trader exchanges tokens with the payment channel based HEX. For continuous trades, user needs to repeat Steps 7, 8, 9, 10, and 11.

Table 1
Comparison of order placement.

Exchange Type	Make Order	Take Order
CEX	Off-chain	Off-chain
DEX	On-chain	On-chain
Conventional HEX	Off-chain	Off-chain
Proposed HEX	Off-chain	Off-chain

Table 2
Comparison of order cancellation.

Exchange Type	Cost	Trust Model
CEX	Free	Trustful
DEX	Gas fee	Trustless
Conventional HEX	Free	Trustful
Proposed HEX	Free	Trustless

Table 3
Comparison of order matching.

Exchange Type	Order Matching Support
CEX	Yes
DEX	No
Conventional HEX	Yes
Proposed HEX	Yes

help here to alert the user when the HEX tries to close the payment channel unilaterally.

- The user or the HEX is able to close a payment channel before the maximum lifetime of the payment channel.

3.2. Attack vectors

Here, we classify attacks into three types, *i.e.*, attacks toward users, attacks toward the HEX, and attacks toward the system.

3.2.1. Attacks toward users

- **Signature Holding Attack**: During the order process, the HEX could hold the deposit division agreement signature sent from a user. When the HEX sends the order matching notification to the user, the user will send a deposit division agreement signature to the HEX. The HEX could keep the deposit division agreement signature and become unresponsive to the user. In such a situation, the user would immediately know that the HEX might be malicious. Since the user is able to cancel all other orders without the approval of the HEX, the user could simply wait till the maximum lifetime of the payment channel. This will lead to following two possible results:
- The HEX might use the most recently received deposit division agreement signature to close the payment channel before the maximum lifetime of the payment channel. In this situation, the user could get his/her deposit back based on the latest deposit division agreement.
- The HEX might not close the payment channel proactively. In this situation, the user could trigger the close of the payment channel after the payment channel reaches its maximum lifetime. All trades during the lifetime of the payment channel become invalid since the payment channel settles with the initial deposit division agreement as the payment channel is created.

In both situations, the user could either get his/her deposit based on the last deposit division agreement or get his/her initial deposit back. But for the HEX, it will lose the trust of the user, which in long term decreases liquidity of the exchange. Thus, for the HEX side, they will not have

enough incentives to hold the payment signatures issued by the users to cheat.

- **Front-running Attack**: Front-running is an investing strategy that predicts the impact of upcoming trades using prior trading information about their own or others. Front-running attack Eskandari et al. [17] might be launched by the HEX against the users since the HEX could get private knowledge of an upcoming order and strategically place their own orders in such a way that is profitable for the HEX. Even though the HEX could theoretically front-run users by itself, users could detect that orders are not in sequence. The loss of the trust from users would result in bad liquidity of the HEX in long term, which makes it fair to say that the HEX is unlikely to front-run users for a long-term benefit.

3.2.2. Attacks toward HEX

- **Private Key Hacking Attack**: The HEX might lose part of its deposit held by the smart contract if its private keys were stolen. Deposit held by the smart contract could be classified into two types, *i.e.*, deposit locked in the payment channel and deposit not locked in the payment channel. Since the smart contract could not be altered, deposit in the smart contract could only be withdrawn by using private keys. If an attacker steals the private keys of the HEX, he/she could transfer the deposit of the HEX which is not locked in any of the payment channels into his/her private address at once. However, deposits locked in the payment channels could only be withdrawn till the close of the payment channels. To close a payment channel, the attacker also needs to hack the off-chain HEX platform to get the last payment signature issued by the HEX user of the payment channel. Even assuming that the attacker is able to hack both the private keys of the HEX and payment signatures from HEX users, the close of the payment channel is an on-chain operation, which requests confirmations from the blockchain to take the effect and this will cost some time. During this time, the token exchange could have some emergency solutions. For example, when the HEX is hacked, the system administrator of the HEX would be able to update the address of the HEX to a new address for all existing payment channels, thus, all HEX's deposits in these payment channels will be sent to this new address when payment channels are closed. It is worth noting that users' deposits are safe no matter the HEX is hacked or not. An attacker cannot steal users' deposits in the smart contract as long as he/she does not hack the private keys of users.
- **Liquidity Attack**: Malicious users may lock a large amount of tokens from the HEX without placing any order or placing orders that are hard to be fulfilled. As the locked tokens of the HEX could not be used to serve other users who actively exchange tokens, the liquidity of the HEX will decrease in long term. This could be solved by setting KYC (Know-Your-Customer[19]) or an effective incentive-and-punishment mechanism. KYC is the process of evaluating the risks of the potential illegal intentions toward the business relationship. Business with the KYC policy requests a customer to submit their basic personally identifiable information, so that business could create assessment of a customer's profile on the basis of his/her transnational behaviors. This could prevent the potential liquidity attacks by designing a proper incentive-and-punishment mechanism. An example could be that HEX can utilize historical transaction data of a HEX user to decide how to react to the user when creating the payment channel. For instance, if a user continuously locking a large amount of token from the HEX without placing any order, the HEX could refuse to create a payment channel with the same user for a specific period or set a maximum amount of tokens allocated to the user's payment channel next time.

3.2.3. Attacks toward the system

- **Smart Contract Attack:** If there is any vulnerability in smart contract code, the entire system might suffer from potential attacks such as the decentralized autonomous organization (DAO) attack Dupont [18], where a hacker spotted a flaw in the DAO's smart contract and transferred 3.6 million Ether into a personal account. To protect the proposed system against known bugs and vulnerabilities, tools such as *Oyente* Luu et al. [19], *SmartEmbed* Gao et al. [20], *DefectChecker* Chen et al. [21] and *ZEUS* Kalra et al. [22] could be used for smart contract code analysis.
- **Man-in-the-Middle and Replay Attack**: Attackers might secretly replay and alter the off-chain communications between the user and the HEX who believe they are directly communicating with each other. To prevent this kind of attacks, we could encrypt off-chain messages between the user and the HEX with a nonce value and timestamps Almadhoun et al. [23]; Hasan and Salah [24]. By doing so, even if an attacker could replace the user address with his/her own address, he/she would not be able to correctly sign it.

4. Minimizing overall cost for frequent traders

To quantitatively evaluate the performance of payment channel in HEX, we would like to compare the cost (*i.e.*, gas fee and transaction confirmation latency) for users in the conventional HEX and the proposed HEX when the overall cost is minimized. In this section, we will introduce the very first gas-price vs. transaction-confirmation-latency function. Then, we will show how the overall cost could be minimized based on gas-price vs. transaction-confirmation-latency function. In next section we will show how we compare the cost in terms of gas fee and transaction confirmation latency in the conventional HEX and the proposed HEX when the overall cost is minimized.

4.1. Overall cost modelling

There are two major cost for blockchain transaction issuers, *i.e.*, gas fee and transaction confirmation latency, which could be evaluated by the following performance metrics, respectively:

- Gas fee: to describe the cost (unit: Ether) in a blockchain transaction.
- Transaction confirmation latency: to describe the interval (unit: sec) between the submission time when a transaction is submitted to blockchain network, and the confirmation time when the transaction receives 12 block confirmations in blockchain.

For the gas fee, the gas to be used by a blockchain transaction could be regarded as a constant since it could be estimated by calling Ethereum API like web3.eth.estimateGas[20]. Therefore, for a transaction, the gas fee could be expressed by

$$f_1(x) = c \times x / 10^9 \tag{1}$$

where

- $f_1(x)$ denotes gas fee;
- c denotes gas cost, which describes the amount of gas to be used by a blockchain transaction;
- x denotes gas price, which describes the price per unit of gas with the unit 1×10^9 Wei (*i.e.*, GWei).

For the transaction confirmation latency, it is well known that as with the increase of gas price, the transaction confirmation latency will decrease. Based on this, we propose the following gas-price vs. transaction-confirmation-latency function for Ethereum Mainnet

[19] https://en.wikipedia.org/wiki/Know_your_customer.

[20] https://web3js.readthedocs.io/en/1.0/web3-eth.html#estimategas.

$$f_2(x) = a/x + b \tag{2}$$

where

- $f_2(x)$ denotes the estimation of the transaction confirmation latency;
- a denotes the status of network congestion in Ethereum Mainnet: When a is small, noting that the constant b is added on the ratio of a/x, it means that the Ethereum network is not so congested such that changing the gas price x (*e.g.*, doubling or tripling x) will not cause a significant difference of the resultant transaction confirmation latency as b is the dominant item in (2). While, when a is very large and the ratio of a/x becomes the dominant item in (2), it means that the Ethereum network is congested. This is because changing the gas price x (*e.g.*, doubling or tripling x) will lead to a considerable variance of the transaction confirmation latency;
- b denotes the estimation of transaction confirmation latency in an ideal situation where x is large enough so that a transaction will be mined in no time (*i.e.*, the average duration of mining 12 blocks in Ethereum Mainnet);
- x denotes gas price, which describes the price per unit of gas with the unit 1×10^9 Wei (i.e., GWei).

We would like to highlight that we will estimate the parameters (a, b) in the proposed function as well as validate the effectiveness of the proposed function via big data analysis later.

For a blockchain transaction issuer, the objective is to simultaneously minimize gas fee and transaction confirmation latency, which could be regarded as a multi-objective optimization problem Hwang et al. [25]. The multi-objective optimization problem could be turned into a single-objective mathematical optimization problem which is to minimize the overall cost of gas fee and transaction confirmation latency via weighted sum model Fishburn [26]. As gas fee and transaction confirmation latency have different magnitude, normalization is required for gas fee and transaction confirmation latency when utilizing weighted sum model. According to previous study Grodzevich and Romanko [27], there are three major methods to normalize objective functions in weighted sum model, *i.e.*, normalize by the magnitude of the objective function at the initial point, normalize by the minimum of the objective function, and normalize by the differences of optimal function values in the Utopia and Nadir points, where Utopia and Nadir points define the lower and upper bounds of the optimal Pareto set for objective functions, respectively. In our case, it is hard to determine a proper initial point of both gas fee and transaction confirmation latency, and the minimum of the gas fee could be 0, which makes normalization by the initial point and by the minimum of the objective functions impractical and ineffective. Therefore, normalization by Utopia and Nadir points is chosen in our case.

Theoretically, the maximum gas fee and transaction confirmation latency could be huge. While, in real world, users cannot use infinite gas fee in a transaction or wait endlessly for the confirmation of a transaction. Thus, for a given user we could define the maximum gas fee as f_1^{max}, and the maximum transaction confirmation latency as f_2^{max}.

Combining (1) and (2), we could derive the following equations:

$$\begin{aligned} f_1^U &= \frac{a \times c}{10^9 \times \left(f_2^{max} - b\right)}, \\ f_1^N &= f_1^{max}, \\ f_2^U &= \frac{a \times c}{10^9 \times f_1^{max}} + b, \\ f_2^N &= f_2^{max} \end{aligned} \tag{3}$$

where

- f_1^U denotes the lower bound of $f_1(x)$ based on the Utopia point of $f_1(x)$;
- f_1^N denotes the upper bound of $f_1(x)$ based on the Nadir point of $f_1(x)$;
- f_2^U denotes the lower bound of $f_2(x)$ based on the Utopia point of $f_2(x)$;
- f_2^N denotes the upper bound of $f_2(x)$ based on the Nadir point of $f_2(x)$.

Thus, for a blockchain transaction issuer with the maximum gas fee as f_1^{max} and the maximum transaction confirmation latency as f_2^{max}, the minimization of the overall cost in terms of gas fee and transaction confirmation latency could be reformulated as

$$\begin{aligned} &\min_x \left\{ \frac{\alpha \times c}{10^9 \times \left(f_1^{max} - f_1^U\right)} \times x + \frac{(1-\alpha) \times a}{\left(f_2^{max} - f_2^U\right)} \times \frac{1}{x} - \frac{\alpha \times f_1^U}{\left(f_1^{max} - f_1^U\right)} - \frac{(1-\alpha) \times \left(f_2^U - b\right)}{\left(f_2^{max} - f_2^U\right)} \right\}, \\ &\text{subject to}: x > 0 \end{aligned} \tag{4}$$

where

- We use the lower bounds based on Utopia points in (3) to simplify the expression of the single objective function;
- α is a weighting coefficient, which stands for the blockchain issuer's preference of gas fee over transaction confirmation latency, and $0 < \alpha < 1$.

We denote the above single objective function as $f(x)$. To minimize $f(x)$, let us have a look at the first and second derivatives of $f(x)$ as follows:

$$\begin{aligned} f'(x) &= \frac{\alpha \times c}{10^9 \times \left(f_1^{max} - f_1^U\right)} - \frac{(1-\alpha) \times a}{\left(f_2^{max} - f_2^U\right)} \times \frac{1}{x^2}, \\ f''(x) &= \frac{2 \times (1-\alpha) \times a}{\left(f_2^{max} - f_2^U\right)} \times \frac{1}{x^3}. \end{aligned} \tag{5}$$

It is obvious that $f''(x) > 0$ always holds, thus, we could know that $f(x)$ is minimized when $f'(x) = 0$. By solving the equation $f'(x) = 0$, the optimal gas price to minimize the overall cost could be expressed as

$$x^* = \sqrt{\frac{10^9 \times (1-\alpha) \times a \times \left(f_1^{max} - f_1^U\right)}{\alpha \times c \times \left(f_2^{max} - f_2^U\right)}}. \tag{6}$$

However, to make use of the overall cost function, we still need to know the value of parameters (a, b) in gas-price vs. transaction-confirmation-latency function. In next section, we will use big data analysis technique to get qualitative estimates of parameters (a, b).

4.2. Parameter estimation

To estimate the parameters (a, b) in the gas-price vs. transaction-confirmation-latency function, we collect 540,296 transactions from Ethereum Mainnet. Based on these transactions, we could get 540,296 (x, y) pairs, where

- x denotes gas price, which describes the price per unit of gas with the unit 1×10^9 Wei (i.e., GWei);
- y denotes transaction confirmation latency.

To calculate the transaction confirmation latency for a transaction, we need to know the submission time when a transaction is submitted to blockchain, and the confirmation time when a transaction is confirmed by 12 blockchain blocks. The confirmation time is the blocktime of the 12th block that confirms the transaction, which could be easily obtained by calling Ethereum API.[21] As for the submission time, the most accurate way to get it is to use an Ethereum client to submit real transactions to Ethereum Mainnet, which can help us to determine the submission time

[21] https://web3js.readthedocs.io/en/1.0/web3-eth.html.

of each transaction precisely. However, it cost real money to submit real transactions in Ethereum Mainnet. A large number of transactions cost a considerable amount of money until a sufficient large dataset can be collected for the analysis. Moreover, the cost is especially high when investigating the latency for the transactions with high gas prices. Therefore, it is economically infeasible to accurately collect the submission time of transactions by submitting real transactions to Ethereum by ourselves. In this paper, we propose to estimate a transaction's submission time based on the observation time when the transaction is received by an Ethereum full node while taking the transmission latency of the gossip protocol into account. To be specific, we use the observation time when a transaction firstly shows up in the pending transaction pool of an Ethereum full node to predict the submission time of the transaction. Due to the unavoidable latency in gossip protocols, the observation time of a transaction would deviate from the submission time depending on the network distance between the Ethereum full node who observes the transaction and the Ethereum client who submits the transaction. To minimize the deviation caused by the network latency, we make the following assumptions:

4.2.1. Assumptions

- We assume the network latency between two network servers using gossip protocols to exchange information is determined by hop count Hedrick [28]. So, we classify the collected transactions into clusters based on the hop counts between the Ethereum full node who observes transactions and the Ethereum clients who submit these transactions. Let us denote the maximum hop count in the network as K, and we could divide all transactions into K clusters, which is $1, 2, \ldots, K$, respectively. Cluster K means transactions in this cluster are submitted by Ethereum clients whose hop count to the Ethereum full node (*i.e.*, the deployed observer server) is K. It is economically unaffordable to submit transactions directly from our observer server. Thus, the minimal hop count is 1, which means that the cluster number starts from 1 instead of 0. Since transactions from cluster 1 have the minimal network latency, we will use the data fitting result for cluster 1 to approximate the wanted gas-price vs. transaction-confirmation-latency model that we need for hop count equal 0.
- For clusters $1, 2, 3, \ldots, K$, we assume the following

$$\begin{cases} y_1 = a_1/x_1 + b_1 + \varepsilon_1, \\ \cdots \\ y_k = a_k/x_k + b_k + \varepsilon_k, \\ \cdots \\ y_K = a_K/x_K + b_K + \varepsilon_K \end{cases} \tag{7}$$

where

- x_k denotes gas price for a transaction in cluster k, which describes the price per unit of gas with the unit 1×10^9 Wei (i.e., GWei); $x_k > 0$, $k = 1, 2, \ldots, K$;
- y_k denotes transaction confirmation latency for a transaction in cluster k, $k = 1, 2, \ldots, K$;
- $a_k > 0$, $k = 1, 2, \ldots, K$;
- ε_k is random error, under the assumption of normality, where $\varepsilon_k \tilde{} \mathcal{N}(0, \sigma_k^2)$, where σ_k is the standard deviation of the Gaussion distribution of ε_k, $k = 1, 2, \ldots, K$.

Based on what we have explained before, we have

$$b_1 > b_2 > \ldots > b_K. \tag{8}$$

This can be deducted as follows: for all clusters, when gas price goes to infinity, a transaction will be mined in no time, thus, transaction confirmation latency will depend on block time and network latency. Since block time will not change for transactions from different clusters, estimation of transaction confirmation latency will change according to network latency only. As with the increase of hop count, network latency between the Ethereum full node (*i.e.*, our deployed observer server) who observes transactions and the Ethereum clients who submit transactions increases. Therefore, when gas price goes to infinity, as with the increase of hop count, estimation of transaction confirmation latency will decrease due to the increase of network latency. Thus, we have $b_1 > b_2 > \ldots > b_K$.

4.2.2. Data collection and pre-processing

We will introduce how we collect and pre-process the raw data received on the Ethereum full node (*i.e.*, the deployed observer server).

- We set up five Ethereum full nodes through Amazon Web Services[22] in the following locations: West America, East South America, Europe, Northeast Asia Pacific, Southwest Asia Pacific. In each Ethereum full node, we record the observation time of transactions when these transactions firstly show up in the pending transaction pool of the node.
- Data collection starts at 2019-04-12 00:00:00 UTC and ends at 2019-04-14 23:59:59 UTC. To preserve as many transactions with long transaction confirmation latency as possible, we only use transactions which are mined successfully between 2019 and 04–14 00:00:00 UTC and 2019-04-14 23:59:59 UTC in our model fitting process. In this way, we are able to keep these transactions whose transaction confirmation latency is longer than two days.
- Transactions collected by all five nodes are merged in one dataset. The uniqueness of each transaction is preserved by its transaction hash. Specifically, for transactions with the same transaction hash, only the one with the earliest observation time is kept. There are two reasons for this:
- By merging transactions collected from five different areas of the world, the potential bias caused by different network structure in different areas of the world could be minimized.
- By using the earliest observation time of transactions, the weights of transactions from clusters with small hop counts are increasing, which also potentially increases the weight of transactions from cluster 1.
- After above processing, we get $540,296$ (x, y) pairs from transactions mined successfully on 2019-04-14 UTC.

4.2.3. Data modelling

So far, we have a mixed the collected (x, y) pairs from K clusters, and we need to estimate the parameters (a_1, b_1) for cluster 1 and use them to approximate the gas-price vs. transaction-confirmation-latency model in the ideal case (*i.e.*, cluster 0). As shown in (7), our models are finite mixtures of regressions. By replacing x with its inverse $1/x$, we could turn our models into linear regressions. Afterward, we could fit the data with finite mixture of linear regressions Veaux [29], which could estimate the distinct parameters of each regression model within finite mixture models.

Given a set of independent observations $y_1, y_2, \ldots, y_n$, corresponding to values $x_1, x_2, \ldots, x_n$ of the predictor $1/x$, then the problem in (7) could be reformulated as

$$y_i = \begin{cases} a_1 \times x_i + b_1 + \varepsilon_{i1}, & \text{with probability } \pi_1, \\ \cdots \\ a_k \times x_i + b_k + \varepsilon_{ik}, & \text{with probability } \pi_k, \\ \cdots \\ a_K \times x_i + b_K + \varepsilon_{iK}, & \text{with probability } \pi_K \end{cases} \tag{9}$$

where

[22] https://en.wikipedia.org/wiki/Amazon_Web_Services.

- x_i, y_i is the *ith* observation of the collected $(1/x, y)$ pairs;
- π_k is the mixing probability where
- $0 < \pi_k < 1$, for all $k = 1, 2, \ldots, K$;
- $\sum_{k=1}^{K} \pi_k = 1$;
- ε_{ik} is random error, under the assumption of normality, where $\varepsilon_{ik} \sim \mathcal{N}(0, \sigma_k^2)$, where σ_k is the standard deviation of the Gaussion distribution of ε_{ik}, $i = 1, 2, \ldots, n$, $k = 1, 2, 3, \ldots, K$.

The parameters in the mixture model $\theta = (\pi_1, \pi_2, \ldots, \pi_K, a_1, a_2, \ldots, a_K, b_1, b_2, \ldots, b_K, \sigma_1, \sigma_2, \ldots, \sigma_K)$ could be estimated by maximizing the log-likelihood

$$L\left(\theta | x_1, \ldots, x_n, y_1, \ldots, y_n\right) = \sum_{i=1}^{232} \ln\left(\sum_{k=1}^{232} \pi_k \varphi(y_i | a_k, b_k, \sigma_k, x_i)\right) \tag{10}$$

where

$$\varphi(y_i | a_k, b_k, \sigma_k, x_i) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \times \exp\left(-\frac{(y_i - (a_k x_i + b_k))^2}{2\sigma_k^2}\right). \tag{11}$$

The standard tool for maximum likelihood estimation in finite mixture models is the EM algorithm Dempster Laird and Rubin [30]. The EM algorithm is an iterative method to find maximum likelihood estimates of parameters in a two-step process. First, the E-step computes the conditional expectation of the log-likelihood evaluated using the current estimates for the parameters. Second, the M-step maximizes the log-likelihood of the parameter estimates learned in the E-step.

Once we have learned the model parameters based on EM algorithm, we then need to decide an optimal cluster number. A consistent estimator of the cluster number K could be achieved based on Bayesian Information Criteria (BIC) Fraley and Raftery [31], where the value of K is chosen at which the BIC value asymptotically converges.

After getting the optimal number of clusters and related model parameters, we could choose the model of cluster 1 from models for all clusters based on (8).

4.2.4. Results

For the model fitting process using EM algorithm, we use the finite mixture models fitting package mixtools Benaglia, Chauveau, Hunter and Young [32]. For the cluster number, we try K from 2 to 10. According to previous study about how to choose initial values for the EM algorithm for finite mixtures Karlis and Xekalaki [33], the solution of the EM algorithm depends on the initial value and the stopping criterion. Therefore, for each given K, we start the EM iteration with 10 different initial

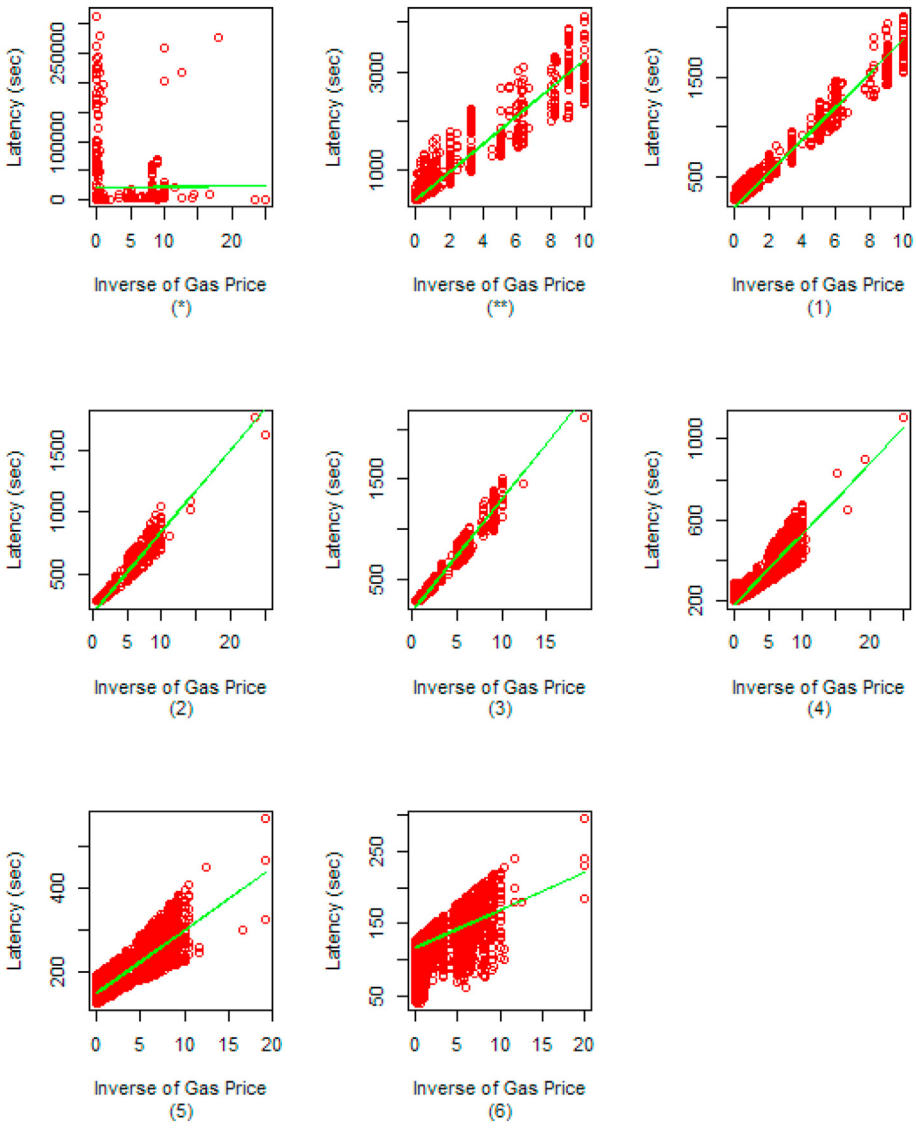

Fig. 6. Regression models for clusters using finite mixture of linear regressions. Models are listed in descending order of the values of b_k.

Table 4
Model parameters and mixing probabilities.

Clusters	Parameters (a, b)	Mixing Probabilities
*	149.35, 20585.11	0.00203055885101554
**	284.30, 394.90	0.00660449752529227
1	165.65, 210.04	0.0353444561749399
2	66.04, 183.90	0.0836724202165458
3	111.35, 183.85	0.0486831288290784
4	34.85, 180.51	0.202374396346127
5	14.89, 150.84	0.374904339750736
6	5.12, 117.34	0.246386202306265

values and choose the stopping criterion as when the difference between two iterations is less than 1×10^{-6}. Afterward, we choose the one with the largest log-likelihood as the solution for the given K within the 10 iterations. Then, we compare the BIC of each solution for K from 2 to 10, the BIC value converges when $K = 8$ in our experiment. Therefore, we choose our optimal cluster number as 8, and the optimal model fitting results of 8 clusters are shown in Fig. 6. In Fig. 6, latency refers to the transaction confirmation latency. Besides, related model parameters and mixing probabilities are shown in Table 4, where numbers in the column Parameters (a, b) are rounded to two decimal points. It is worth noting that in Table 4, the models are listed in descending order of the values of b_k.

In Fig. 6 and Table 4, we do not assign any cluster number for the first two models but instead we use * and ** to refer to these two models, as these two models could not be the model for cluster 1. There are two reasons for this. First, by utilizing data from Etherscan,[23] the expectation of block time in Ethereum Mainnet is 15.46 s based on law of large numbers Grimmett and Stirzaker [34], and the parameter b in the first two models is much higher than that of 12×15.46 as shown in Table 4. Second, if we take a look at mixing probabilities of these two models, sum of the mixing probabilities of these two models are less than 0.009 (*i.e.*, 0.9%), which is way lower than the mixing probability of any other model. Therefore, we could safely say these two models could not be the models for cluster 1. A potential cause for these two models might be that transactions from these two models might have exceptional nonce value. For a transaction from these two models, if any other transaction from the same transaction issuer with a lower nonce value is not mined into blockchain, the transaction from these two models have to wait for a long time before mined into blockchain even when its gas price is high.

After excluding the first two models, there are 6 clusters left. According to (8), we take the third model in Table 4 which has the largest parameter b as the one from the cluster 1. Also, the parameter a for the cluster 1 is 165.65, which is reasonable since it is neither big nor small. For example, for transactions with gas prices 0.1, 1, 10, and 100 GWei, the corresponding transaction confirmation latency will be 1866.54, 375.69, 226.61, 211.7 s, which makes sense based on common knowledge about transaction confirmation latency on Ethereum Mainnet. The rationality of the estimates of parameters (a, b) proves the effectiveness of the gas-price vs. transaction-confirmation-latency model.

Since we are using the model from the cluster 1 to approximate our gas-price vs. transaction-confirmation-latency model, the eventual model that we adopt is drawn in Fig. 7.

4.3. Simulation results

Now we have the value of parameters (a, b) in (4), we could see how the overall cost changes for different users for a given transaction.

Fig. 8 shows how the overall cost changes with gas price for a given transaction, when users' maximum gas fee and maximum transaction confirmation latency change. Here, we take Ethereum token transfer transaction whose gas cost is 21,000 as an example. According to the

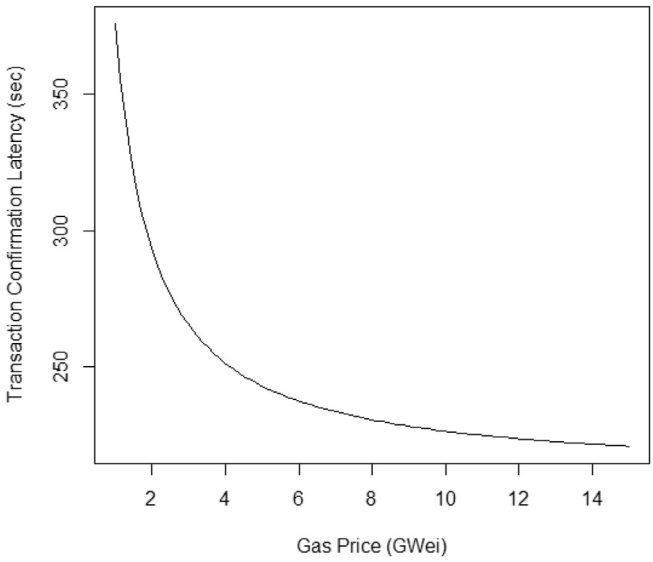

Fig. 7. Gas-price vs. transaction-confirmation-latency model.

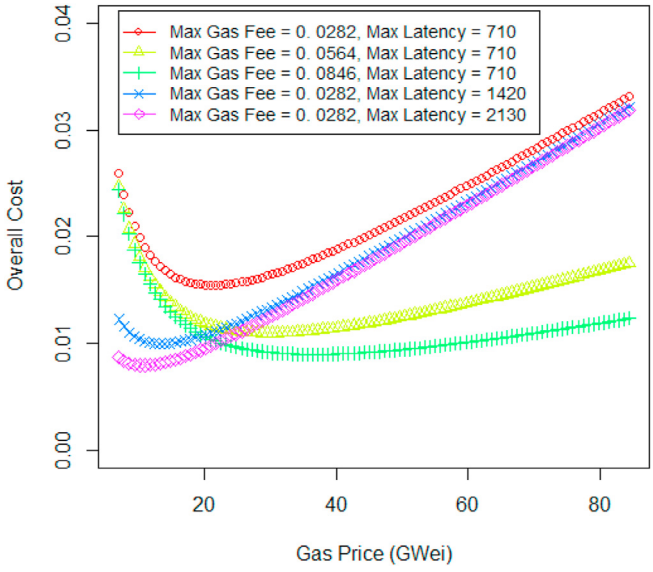

Fig. 8. Overall cost versus gas price when users change the maximum gas fee and transaction confirmation latency for a given transaction. Here we take gas cost = 21,000, $\alpha = 0.5$ as an example.

samples from cluster 1 in Fig. 6, the median gas fee is 0.0282 Ether and the median transaction confirmation latency is 710 s. Therefore, in the experiment, we choose the following settings: when maximum gas fee is 0.0282 Ether, the maximum transaction confirmation latency is set as 710, 1420, 2130 s, respectively; when maximum transaction confirmation latency is set as 710 s, the maximum gas fee is set as 0.0282, 0.0564, 0.0846 Ether, respectively. As shown in Fig. 8, when maximum gas fee and maximum transaction confirmation latency are fixed, the overall cost will decrease with the increase of gas price before reaching the optimal gas price, and then the overall cost will increase with the increase of gas price after passing the optimal gas price. Particularly, when maximum transaction confirmation latency is fixed, the optimal gas price increases with the increase of maximum gas fee. While, when maximum gas fee is fixed, the optimal gas price decreases with the increase of maximum transaction confirmation latency.

Fig. 9 shows how the value of cost function changes with gas price for a given transaction when users' preferences between gas fee and

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

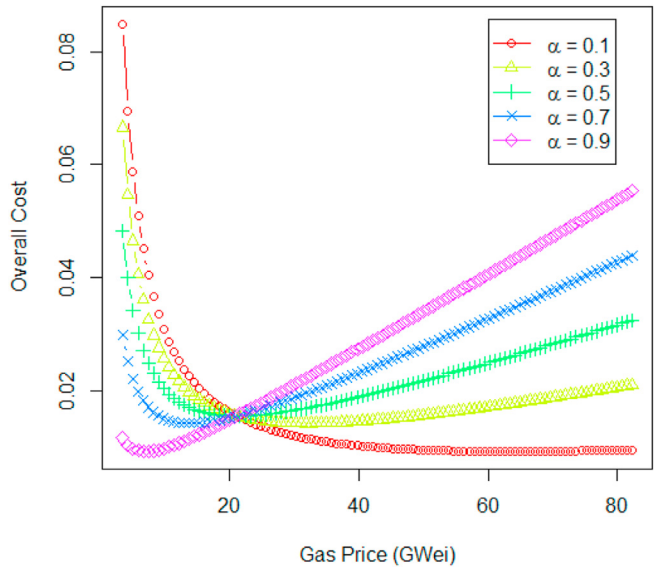

Fig. 9. Overall cost versus gas price when users' preference between gas fee and transaction confirmation latency changes for a given transaction. Here we take gas cost = 21,000, maximum gas fee = 0.0282 Ether, and maximum transaction confirmation latency = 710 s as an example.

transaction confirmation latency change. For a given α, the overall cost will decrease with the increase of gas price before reaching the optimal gas price, and then the overall cost will increase with the increase of gas price after passing the optimal gas price. When α increases, the optimal gas price to minimize the overall cost decreases. The corresponding optimal gas price is 63.28056, 32.22088, 21.09352, 13.80895, 7.031173 GWei for $\alpha = 0.1, 0.3, 0.5, 0.7, 0.9$, respectively. The reason why the increase of α leads to the decrease of the optimal gas price is that the larger α is, the less users want to spend on gas fee, and thus, the less optimal gas price is. Particularly, among all users, the minimal overall cost is the maximum for users with $\alpha = 0.5$, and the same minimal overall cost is shared for two users where the summation of the two users' α equals to 1.

5. Experiment

In this section, for evaluating the proposed payment channel based HEX framework, we carry out some experiments by simulating different scenarios in practical token exchange. Firstly, to prove how our proposed scheme decreases gas fee and transaction confirmation latency for frequent token traders, we will show the percentage of gas fee and the percentage of transaction confirmation latency could be saved, when the overall cost in terms of gas fee and transaction confirmation latency for a user is minimized when replacing the conventional scheme with the proposed scheme. Secondly, to prove how our proposed scheme alleviate transaction congestion in blockchain, we will show the percentage of the number of blockchain transactions could be saved by replacing the conventional payment channel with the proposed scheme.

5.1. Experimental setup

5.1.1. Experimental configuration

Hardware used in the experiment is listed in Table 5.

[24] https://github.com/dexDev/DEx.top/blob/master/whitepaper/DEx-Whitepaper-Short-Version.pdf

Table 5
Summary of hardware settings.

Device	OS	Memory	CPU
DELL Inspiron 15	Ubuntu 18.04	8G	Intel i5-7300

5.1.2. Baseline schemes and performance metrics

For performance comparison, the used baseline scheme is a conventional HEX similar to DEx.top,[24] which does not consider the payment channel. Besides, to evaluate the schemes, the following performance metrics would be used:

- *Gas fee saving ratio*: to describe the percentage of gas fee could be saved by replacing the conventional scheme with the proposed scheme, when the overall cost for a user is minimized in both schemes.
- *Latency saving ratio*: to describe the percentage of transaction confirmation latency could be saved by replacing the conventional scheme with the proposed scheme, when the overall cost for a user is minimized in both schemes.
- *Blockchain transaction saving ratio*: to describe the percentage of the number of blockchain transactions could be saved by replacing the conventional payment channel with the proposed scheme.

Particularly, to approximate the gas costs of blockchain transactions in both schemes, we implement two MVPs for both schemes. Based on the MVPs, we have simulated the function calls to smart contracts via web3.js to estimate the gas costs of blockchain transactions. Since the gas cost of a blockchain transaction might fluctuate in a small range, we trigger 30 function calls to each target smart contract method and take the average gas cost in 30 simulations as the estimate of the gas cost. Here are the estimated gas costs of transactions to be used in our experiments:

1. In the conventional scheme, the gas cost in a trading transaction is 84,841.
2. In the proposed scheme, the gas costs in payment channel creation and closure increase linearly with the increase of the number of types of locked tokens in a payment channel. The related gas costs are shown in Table 6.

5.2. Experimental cases

As proposed before, the proposed scheme could help save gas fee and transaction confirmation latency for frequent traders, and at the same time ease potential Ethereum network congestion caused by large amounts of trading transactions. Therefore, we consider three types of use cases:

1. How gas fee saving ratio changes with the number of trading transactions per payment channel in terms of different preferences over gas fee and transaction confirmation latency, different limitations for gas fee and transaction confirmation latency, and different gas costs in payment channel creation and closure for different users.
2. How latency saving ratio changes with the number of trading transactions per payment channel in terms of different preferences over gas

Table 6
Gas cost in payment channel creation and closure.

Number of Token Types	Gas Cost in Payment Channel Creation	Gas Cost in Payment Channel Closure
2	266,047	73,936
5	387,892	112,216
10	591,329	231,830
15	794,704	359,374
20	998,020	486,756

fee and transaction confirmation latency, different limitations for gas fee and transaction confirmation latency, and different gas costs in payment channel creation and closure for different users.

3. How blockchain transaction saving ratio changes when the average number of trading transactions per payment channel changes for the proposed payment channel based HEX.

5.2.1. How gas fee saving ratio changes

Fig. 10 (a) shows how gas fee saving ratio changes with the number of trading transactions per payment channel, when users' preferences over gas fee and transaction confirmation latency, maximum gas fee, and maximum transaction confirmation latency change. Here we take gas costs in payment channel creation and closure as 266,047 and 73,936 as an example. As shown in Fig. 10 (a), the gas fee saving ratio increases when the number of tradings per payment channel increases. Particularly, for users whose number of tradings per payment channel is small, they will spend more gas fee in the proposed scheme than in the conventional scheme (*i.e.*, the saving ratio is negative). This is caused by the overhead of gas fees in creating and closing a payment channel in the proposed scheme. But when users do more tradings per payment channel, the gas fee saving ratio will be larger than 0. Besides, for users with the same amount of trading transactions per payment channel, the gas fee saving ratio does not change when they have different preferences over gas fee and transaction confirmation latency, different maximum gas fee, or different maximum transaction confirmation latency. This proves the fairness of our overall cost function with regards to gas fee saving ratio, which helps different users to obtain the same gas fee saving ratio as long as they have the same number of trading transactions per payment channel.

Fig. 10 (b) shows how gas fee saving ratio changes with the number of trading transactions per payment channel, when gas costs in payment channel creation and closure increase. In the figure, G1 stands for gas cost in payment channel creation and G2 stands for gas cost in payment channel closure. Here we take $\alpha = 0.5$, maximum gas fee $= 0.0282$ Ether, and maximum transaction confirmation latency $= 710$ s as an example. As shown in Fig. 10 (b), the gas fee saving ratio will increase when the number of tradings per payment channel increases. Particularly, for users whose number of tradings per payment channel is small, they will spend more gas fee in the proposed scheme than in the conventional scheme. This is caused by the overhead of gas fees in creating and closing a payment channel in the proposed scheme. But, when users do more tradings per payment channel, the gas fee saving ratio will be greater than 0. Besides, the higher the summation of the gas costs in payment channel creation and closure is, the higher the number of trading transactions per payment channel is to compensate for the overhead of gas fees in payment channel creation and closure. Particularly, for users with the same amount of trading transactions in the payment channel, the

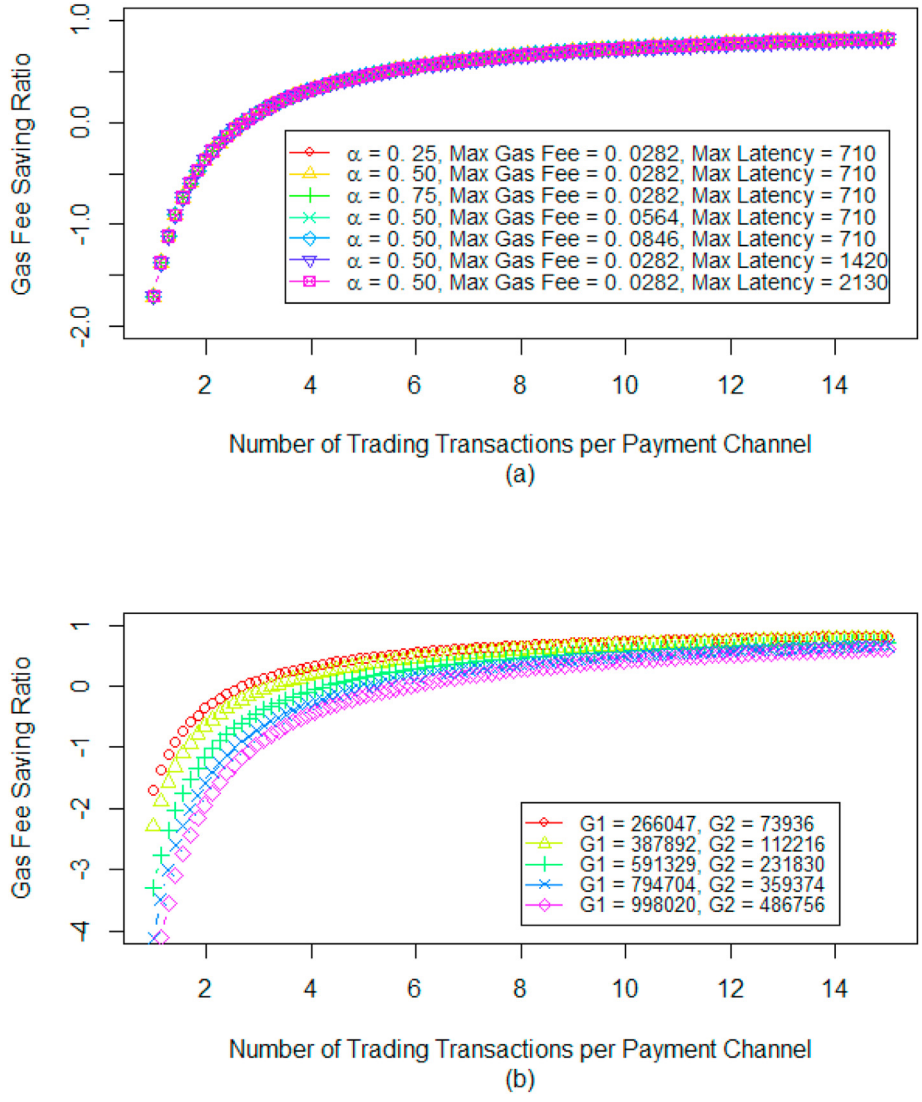

Fig. 10. Gas fee saving ratio versus number of trading transactions per payment channel.

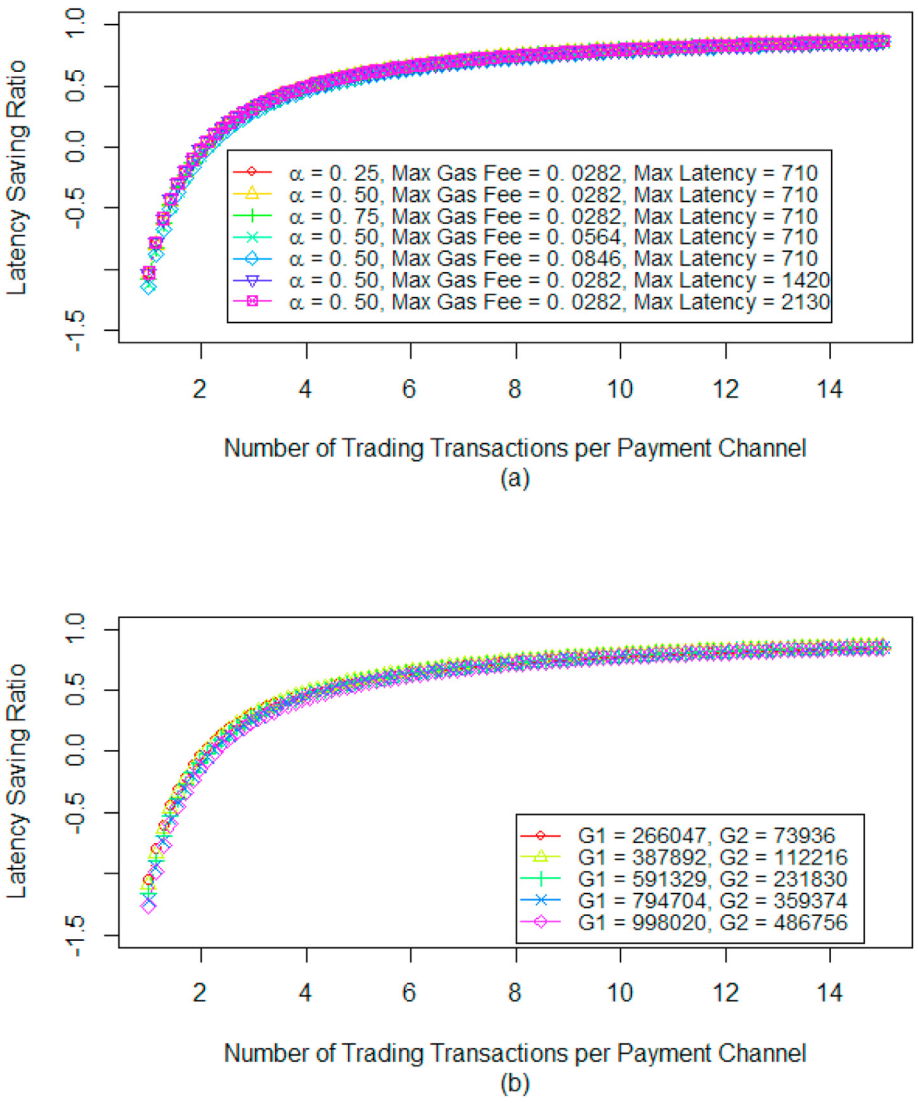

Fig. 11. Latency saving ratio versus number of trading transactions per payment channel.

higher the summation of the gas costs in payment channel creation and closure is, the lower the gas fee saving ratio is.

5.2.2. How latency saving ratio changes

Fig. 11 (a) shows how latency saving ratio changes with the number of trading transactions per payment channel, when users' preferences over gas fee and transaction confirmation latency, maximum gas fee, and maximum transaction confirmation latency change. Here we take gas costs in payment channel creation and closure as 266,047 and 73,936 for an example. As shown in Fig. 11 (a), the latency saving ratio will increase with the increase of number of tradings per payment channel. Particularly, for users whose number of tradings per payment channel is less than 2, they will expect longer transaction confirmation latency in the proposed scheme than in the conventional scheme. This is because there are at least two blockchain transactions in the proposed scheme due to payment channel creation and closure. However, as long as users trade more than twice in the payment channel, the latency saving ratio will be larger than 0. Besides, for users with the same amount of tradings in the payment channel, there is only a minor difference in the latency saving ratio when they have different preferences over gas fee and transaction confirmation latency, different maximum gas fee, or different maximum transaction confirmation latency. This proves the fairness of our overall cost function with regards to latency saving ratio, which helps different users to obtain similar latency saving ratio as long as they have the same number of trading transactions per payment channel.

Fig. 11 (b) shows how latency saving ratio changes with the number of trading transactions per payment channel, when gas costs in payment channel creation and closure increase. In the figure, G1 stands for gas cost in payment channel creation and G2 stands for gas cost in payment channel closure. Here we take $\alpha = 0.5$, maximum gas fee $= 0.0282$ Ether, and maximum transaction confirmation latency $= 710$ s as an example. As shown in Fig. 11 (b), the latency saving ratio will increase when the number of tradings per payment channel increases. Particularly, for users whose number of tradings per payment channel is less than 2, they will expect longer transaction confirmation latency in the proposed scheme than in the conventional scheme. This is because there are at least two blockchain transactions in the proposed scheme due to payment channel creation and closure. However, as long as users trade more than twice in a payment channel, the latency saving ratio will be larger than 0. Besides, for users with the same amount of tradings in the payment channel, the higher the summation of gas costs in payment channel creation and closure is, the lower the latency saving ratio is, though the difference is quite minor.

5.2.3. How blockchain transaction saving ratio changes

As shown in Fig. 12, the blockchain transaction saving ratio increases when the average number of tradings per payment channel for all users

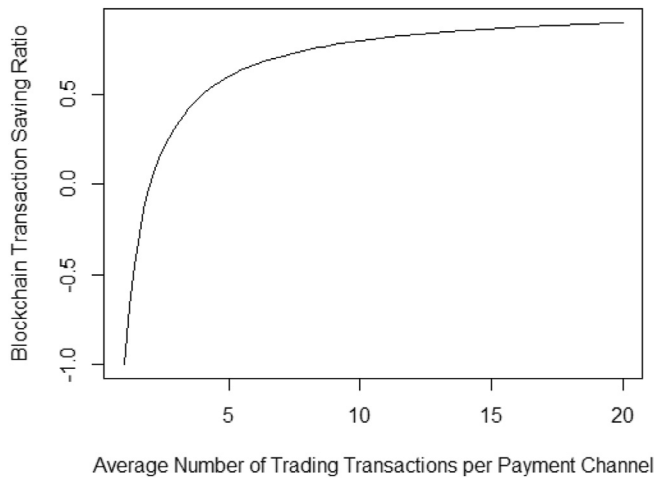

Fig. 12. Blockchain transaction saving ratio versus average number of trading transactions per payment channel.

increases in the proposed scheme. Particularly, when the average number of tradings per payment channel for the proposed scheme is less than 2, the blockchain transaction saving ratio will be less than 0. This is because there are at least two blockchain transactions in the proposed scheme due to payment channel creation and closure. However, as long as users trade more than twice in average, the more often users choose the proposed scheme over the conventional scheme, the more onchain transactions could be saved, which helps ease potential network congestion in Ethereum caused by trading transactions in HEX.

6. Conclusions and discussions

In this paper, we finished a systematic design of the proposed payment channel based HEX which benefits frequent Ethereum token traders and alleviates potential Ethereum network congestion. Afterward, we propose the very first gas-price vs. transaction-confirmation-latency model to help blockchain transaction issuers to minimize the overall cost of gas fee and transaction confirmation latency. Based on the proposed gas-price vs. transaction-confirmation-latency function, we quantitatively evaluate the performance of our payment-channel based HEX. Experimental results have been presented to show that our proposed solution incurs a low overhead while achieving a high efficiency.

While the proposed platform is promising, this paper reveals several limitations that we intend to address in our future work to refine this platform:

- Due to the nature of token lock in creating payment channel, malicious HEX users may initiate attacks to HEX by creating a large number of channels, or creating a large number of cancelled orders. Therefore, the widely used KYC technique in many token exchanges or an effective incentive-and-punishment mechanism might be needed to prevent these attacks.
- Implementation of the proposed system needs to be optimized. As shown from our experimental results, gas fees in payment channel creation and closure increase linearly with the number of locked tokens' types. An optimized implementation is required to lower down gas fees on payment channel creation and closure when the number of locked tokens' types is huge.
- More data may be helpful to get more accurate quantitative estimates of parameters in the gas-price vs. transaction-confirmation-latency model. Due to high cost in terms of money in setting up Ethereum full nodes in Amazon Web Services, we are only able to collect Ethereum transactions for three days for now. If we could afford to collecting more data in the future, we might be able to compare the results from data sets in different dates and get more accurate parameter estimates in the gas-price vs. transaction-confirmation-latency model.
- We focused on the payment-channel based HEX design in Ethereum. This is because more than 90% of the top 100 cryptocurrencies are Ethereum based tokens. While, the business logic of our proposed payment-channel based HEX can be applied to other blockchain systems which support smart-contract-like functions (*e.g.*, Hyperledger Fabric, EoS). As long as a blockchain system support smart contract, the deployed smart contract (or chaincode in Hyperledger Fabric) can serve as a trustful escrow account and verifies the signatures before splitting the deposit and sending different portions to different accounts.
- We could extend our proposed solution to other blockchain-based systems, like Bitcoin, BitCash, etc. To achieve this goal, we need to set up gateway peers as the bridge connecting Ethereum and other blockchain-based systems, where the gateway works as the middleman to talk with our proposed payment-channel based HEX system and other blockchain-based systems. However, a great deal of security issues need to be addressed when synchronizing two blockchains. This could be a good research direction for the future work.
- Order matching engine might be implemented in a decentralized way. Existing order matching engines are implemented in a centralized way, which might make the off-chain HEX platform a single point of failure. For future work, it might be possible to implement the order matching engine in a decentralized way for further security improvement Al Breiki et al. [5].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was supported by Blockchain@UBC, Natural Sciences and Engineering Research Council of Canada (RGPIN-2019-06348), National Natural Science Foundation of China (Project No. 61902333), and Shenzhen Institute of Artificial Intelligence and Robotics for Society (AIRS).

References

[1] S. Nakamoto, Bitcoin: a peer-to-peer electronic cash system, URL, http://www.bitcoin.org/bitcoin.pdf, 2009.
[2] M.A. Khan, K. Salah, Iot security: review, blockchain solutions, and open challenges, Future Generat. Comput. Syst. 82 (2018) 395–411.
[3] Z. Hong, Z. Wang, W. Cai, V.C. Leung, Connectivity-aware task outsourcing and scheduling in d2d networks, in: Proc. Of the 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, Canada, 2017.
[4] H.R. Hasan, K. Salah, Combating deepfake videos using blockchain and smart contracts, IEEE Access 7 (2019) 41596–41606.
[5] H. Al Breiki, L. Alqassem, K. Salah, M. Habib ur Rehman, D. Svetinovic, Decentralized access control for iot data using blockchain and trusted oracles, in: Proc. of IEEE International Conference on Industrial Internet, Orlando, FL, 2019.
[6] Y. Wu, H. Dai, H. Wang, K.R. Choo, Blockchain-Based Privacy Preservation for 5g-Enabled Drone Communications, 2020, p. 3164. CoRR abs/2009.03164. URL, https://arxiv.org/abs/2009.03164. arXiv:2009.
[7] L. Cheng, J. Liu, G. Xu, Z. Zhang, H. Wang, H. Dai, Y. Wu, W. Wang, Sctsc: a semicentralized traffic signal control mode with attribute-based blockchain in iovs, IEEE Trans. Comput. Soc. Syst. 6 (2019) 1373–1385.
[8] V. Buterin, A next-generation smart contract and decentralized application platform, URL, https://github.com/ethereum/wiki/wiki/White-Paper, 2013.
[9] W. Cai, Z. Wang, J.B. Ernst, Z. Hong, C. Feng, V.C. Leung, Decentralized applications: the blockchain-empowered software system, IEEE Access 6 (2018) 53019–53033.
[10] N. Álvarez Díaz, J. Herrera-Joancomartí, P. Caballero-Gil, Smart contracts based on blockchain for logistics management, in: Proc. of 1st International Conference on Internet of Things and Machine Learning, 2017.
[11] A. Back, Hashcash — a denial of service counter-measure, URL, http://www.hashcash.org/hashcash.pdf, 2002.
[12] J. Chen, X. Xia, D. Lo, J. Grundy, X. Yang, Maintaining Smart Contracts on Ethereum: Issues, Techniques, and Future Challenges, 2020, 00286 arXiv:2007.

[13] C. Decker, R. Wattenhofer, A fast and scalable payment network with bitcoin duplex micropayment channels, in: Proc. of 17th International Symposium on Stabilization, Safety, and Security of Distributed Systems, Edmonton, Canada, 2015.
[14] B. Xiao, X. Fan, S. Gao, W. Cai, Edgetoll: a blockchain-based toll collection system for public sharing of heterogeneous edges, in: Proc. of IEEE International Conference on Computer Communications Workshops, Paris, France, 2019.
[15] X. Luo, W. Cai, Z. Wang, X. Li, V.C. Leung, A payment channel based hybrid decentralized ethereum token exchange, in: Proc. of IEEE International Conference on Blockchain and Cryptocurrency, Seoul, Korea, 2019.
[16] O. Osuntokun, Hardening lightning network, URL, https://diyhpl.us/wiki/transcripts/blockchain-protocol-analysis-security-engineering/2018/hardening-lightning/, 2018.
[17] S. Eskandari, S. Moosavi, J. Clark, Sok: Transparent Dishonesty: Front-Running Attacks on Blockchain, 2019. URL, https://arxiv.org/pdf/1902.05164.pdf.
[18] Q. Dupont, Experiments in Algorithmic Governance: A History and Ethnography of " the DAO, " a Failed Decentralized Autonomous Organization, 2017.
[19] L. Luu, D.H. Chu, H. Olickel, P. Saxena, A. Hobor, Making smart contracts smarter, in: Proc. of ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 2016.
[20] Z. Gao, L. Jiang, X. Xia, D. Lo, J. Grundy, Checking smart contracts with structural code embedding, IEEE Trans. Software Eng. 1 (2020), https://doi.org/10.1109/TSE.2020.2971482.
[21] J. Chen, X. Xia, D. Lo, J.C. Grundy, X. Luo, T. Chen, DEFECTCHECKER: automated smart contract defect detection by analyzing EVM bytecode, CoRR abs/2009.02663. URL, https://arxiv.org/abs/2009.02663, 2020b. arXiv:2009.02663.
[22] S. Kalra, S. Goel, M. Dhawan, S. Sharma, Zeus: analyzing safety of smart contracts, in: NDSS, 2018.
[23] R. Almadhoun, M. Kadadha, M. Alhemeiri, M. Alshehhi, K. Salah, A user authentication scheme of iot devices using blockchain-enabled fog nodes, in: IEEE/ACS 15th International Conference on Computer Systems and Applications, Aqaba, Jordan, 2018.
[24] H.R. Hasan, K. Salah, Proof of delivery of digital assets using blockchain and smart contracts, IEEE Access 6 (2018) 65439–65448.
[25] Ching-Lai Hwang, S.M. Masud, Abu, Multiple Objective Decision Making — Methods and Applications: A State-Of-The-Art Survey, first ed., Springer-Verlag, 1979.
[26] P.C. Fishburn, Additive utilities with incomplete product sets: application to priorities and assignments, Oper. Res. 15 (1967) 537–542.
[27] O. Grodzevich, O. Romanko, Normalization and other topics in multi-objective optimization, in: Proc. of the 1st Fields-MITACS Industrial Problems Workshop, Toronto, Canada, 2006.
[28] C. Hedrick, Routing information protocol, URL, https://tools.ietf.org/html/rfc1058, 1988.
[29] R.D.D. Veaux, Mixtures of linear regressions, Comput. Stat. Data Anal. 8 (1989) 227–245.
[30] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the em algorithm, J. Roy. Stat. Soc. B 39 (1977) 1–38.
[31] C. Fraley, A.E. Raftery, Model-based clustering, discriminant analysis, and density estimation, J. Am. Stat. Assoc. 97 (2002) 611–631.
[32] T. Benaglia, D. Chauveau, D. Hunter, D. Young, mixtools: an R package for analyzing finite mixture models, J. Stat. Software 32 (2009) 1–29.
[33] D. Karlis, E. Xekalaki, Choosing initial values for the EM algorithm for finite mixtures, Comput. Stat. Data Anal. 41 (2003) 577–590.
[34] G.R. Grimmett, D.R. Stirzaker, Probability and Random Processes, third ed., Oxford University Press, 2003.