Collateral-Efficient Instant Contingent Payments: The Promise of a Hardware-Driven Off-Chain Payment System

Anxin Zhou, Yuefeng Du, and Xiaohua Jia(✉)

City University of Hong Kong, Hong Kong, People's Republic of China
anxin.zhou@my.cityu.edu.hk, {yf.du,csjia}@cityu.edu.hk

Abstract. As the cryptocurrency universe continues to expand at an unprecedented pace, efficient and routine transactions have become a critical necessity. Yet, the current transaction processing capabilities of many blockchains fail to meet the burgeoning demands of retail payments. While off-chain scaling solutions present promising alternatives to augment blockchain throughput and uphold compatibility with established blockchains, they often impose impractical burdens on users or experience significant payment latency. This study introduces a hardware-driven off-chain payment system designed to accommodate contingent payments. Merchants can accept payments instantly without waiting for transaction confirmation. Our assessment reveals that this system can achieve a peak throughput of approximately 10000 payments per second, representing an 84-fold improvement over traditional Ethereum transactions. This hardware-driven solution holds significant promise for instant and collateral-efficient transactions, effectively unlocking the untapped potential of instant payments with high throughput.

Keywords: Off-chain transaction · Contingent payment · Trusted hardware

1 Introduction

With the cryptocurrency market witnessing unprecedented growth, the interest in utilizing cryptocurrencies for everyday transactions is steadily increasing. It's estimated that nearly 18% of the adult population in the US [7], approximately 46 million consumers, are considering making retail purchases with cryptocurrencies. Furthermore, both large retailers (85% of 202 surveyed) [11] and small businesses (32% of owners and top-level executives surveyed) [1] have begun accepting cryptocurrencies as a method of payment. Major payment providers like PayPal [12], Visa [15], and leading crypto exchanges such as Binance [3] and Coinbase [6] have also broadened their financial services to accommodate cryptocurrency payments.

J. Liu et al. (Eds.): TridentCom 2023, LNICST 523, pp. 21–37, 2024.
https://doi.org/10.1007/978-3-031-51399-2_2

Despite the original vision of a decentralized payment network [4], retail cryptocurrency payments remain predominantly controlled by companies. For example, payments made through PayPal [12] or Binance [3] occur between accounts managed by these companies. Consumers of Visa [15] and Starbucks [13] must first convert their cryptocurrencies into fiat currencies or gift cards. However, cryptocurrency markets may not be well regulated, leading to potential misrepresentation of financial positions [8] and insufficient client protection in the event of company bankruptcy [14]. Moreover, companies may unilaterally ban clients from certain regions due to political considerations [10].

Direct cryptocurrency use for retail payments could eliminate centralized control over consumers' funds. Yet, the transaction processing capabilities of popular blockchains, such as Bitcoin and Ethereum, which together constitute 60% of the global crypto market, are inadequate for handling the demands of retail payments. Their blockchains can only process around 7 and 15 transactions per second, respectively, with confirmation times reaching up to an hour and 15 min. This stands in stark contrast to Visa, which handles an average of 1700 transactions per second with minimal confirmation latency. Therefore, it is crucial to devise solutions that provide comparable throughput and latency for cryptocurrency payments.

Off-chain scaling [20,23,28,34,39] seeks to enhance blockchain throughput while preserving compatibility with existing blockchains. This approach transitions the processing of transactions from the blockchain to off-chain components, typically relying on the inherent security mechanisms of the blockchain to protect these off-chain processes. However, existing solutions such as payment channels [39] and commit-chains [32] suffer from usability issues for retail payments, as they necessitate periodic online activity from customers. Sidechains [28] are believed to lack security, as evidenced by real-world incidents when using a small number of nodes for performance. Rollups [20] have high latency as transaction data must be stored back on-chain for transactions to be committed.

Our Contribution. In this work, we introduce a hardware-driven off-chain payment system designed for practical throughput and low latency without requiring continuous online involvement from customers. An untrusted server processes payments off-chain, similar to rollups, and is required to periodically commit processed payments on-chain. This allows users to retrieve off-chain funds even in the event of server misbehavior. We innovatively enable merchants to accept unconfirmed payments without waiting for on-chain commitment, facilitating prompt goods or services delivery.

Our unique approach allows unconfirmed payments to be re-executed on the blockchain if the server fails to commit payments on-chain promptly. To ensure successful re-execution, we leverage the integrity protection offered by trusted execution environments (TEEs)-secure hardware that has found widespread application and support from leading cloud service providers. The untrusted server is required to operate off-chain funds within a TEE and thus cannot revert a payment that has been re-executed.

We implement a prototype and evaluate our system on Ethereum. The average gas cost for a payment is around 250 (compared to 21000 for a standard

Ethereum transfer). Based on this, we estimate the maximum throughput and transaction fee of a payment. Using the current block gas limit (30M) and block interval (12.11 s) as well as the gas price (14.3 gwei) and ETH price (1919.55 USD), we find the average transaction fee of a payment to be approximately $6.78 * 10^3$. The maximum payment throughput stands at about 10000 payments per second, an 84-fold improvement over Ethereum transfers. Payment can be immediately accepted once processed by the server, with an average processing time of 0.21 ms.

In summary, our contributions in this paper are threefold:

1. We design an off-chain payment system that offers low latency and practical throughput.
2. We propose a hardware-driven solution that allows merchants to safely accept unconfirmed payments without using collateral.
3. We implement and evaluate a system prototype. The experiment shows promising performance and low transaction fees.

2 Preliminaries

Distributed Ledger. A distributed ledger, also known as a blockchain system, involves transactions that typically encompass simple transfers or calls to on-chain smart contracts. Clients initiate transactions by broadcasting them to peer nodes within the blockchain network. These nodes leverage a consensus protocol, for instance, proof of work, to agree on the sequence for transaction execution. To optimize efficiency, nodes batch multiple transactions into a single block, forming a chain of these blocks as a blockchain. Depending on the consensus protocol, a block is either deterministically finalized [33] when added to the blockchain, or considered irreversible with overwhelming probability [27].

Trusted Execution Environment. We instantiate TEE with Intel Software Guard Extensions (SGX), which are a set of security-related instruction codes integrated into certain modern Intel CPUs. These instructions enable developers to create protected enclaves within the processor that safeguard sensitive data and code from unauthorized access or modification, even by the operating system or hypervisor. A key feature of Intel SGX is remote attestation, which lets a remote entity confirm the integrity and authenticity of an enclave operating on a different system. Remote attestation is a procedure that allows a trusted remote entity to verify the identity and secure environment of an enclave on a remote system. The process usually involves the following steps: 1) the remote enclave generates a key pair and securely stores the private key within the enclave; 2) the enclave creates a report containing its identity, measurement, and the newly generated public key; 3) the report is signed with the private key and transmitted to the remote party; 4) the remote party verifies the report using the public key, and checks the signature, ensuring that a genuine SGX enclave created it.

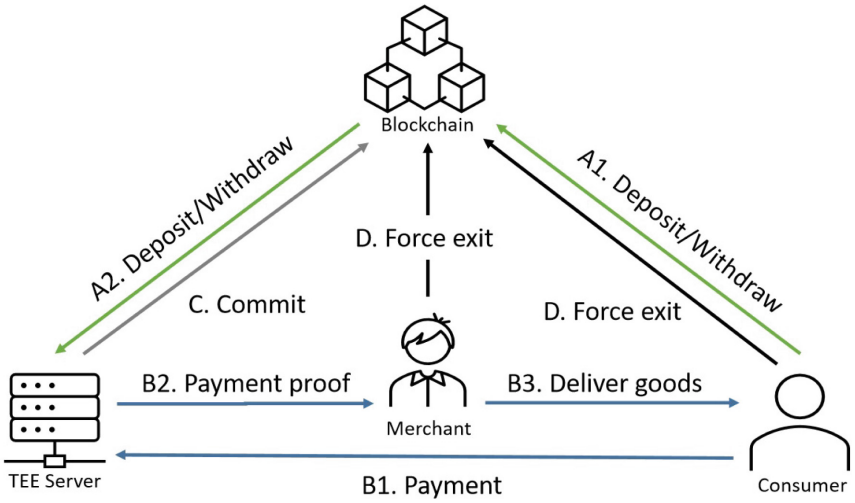


Fig. 1. System overview.

3 Problem Statement

3.1 System Model

Our system is made up of four roles. The server runs an off-chain payment service designed to expedite and economize cryptocurrency transactions. In real-world applications, the server might be a major payment processor or an online retail platform, both intent on attracting consumers and merchants eager to transact using cryptocurrencies.

Consumers and merchants are users of our system. They have the capability to deposit cryptocurrencies on the blockchain, thereby accumulating funds in their off-chain accounts. These funds can be transferred back to the blockchain when needed. Our primary objective is to enable instantaneous payments for merchants who typically ship goods after payment has been received. By reducing this delay, we can make crypto payments more competitive compared to traditional methods. However, our system does not take into account merchants who accept payments but fail to deliver goods.

Our infrastructure employs a public blockchain that supports smart contracts. In our design, we utilize a smart contract to ensure that users can withdraw funds when the server acts improperly, such as in the event of denial of service. We integrate a TEE that offers integrity protection and attested execution functions. The server, through a TEE, administers off-chain funds, ensuring that balance tampering is not possible and that other roles can verify the server's operations. While we use the Ethereum blockchain and SGX enclaves in this work as examples, it's worth noting that other blockchains or TEEs with similar capabilities could also be used.

Threat Model. While maintaining the off-chain payment service, the server is inherently restricted in its potential for malfeasance due to the employment of a Trusted Execution Environment (TEE). Thanks to the integrity protection offered by the TEE, we can assure that the off-chain balance updates correctly and exclusively by predefined program code. Falsely reporting a related message to other roles in the system is ineffective, as the authenticity of the message can be verified through the TEE's attested execution. However, it should be noted that the server retains control over the TEE's network stack. This means the server can potentially block, delay, or tamper with messages sent to or from the TEE. In this context, we dismiss the likelihood of tampering as it can be easily detected through an authentication mechanism [38].

Given that the TEE already mitigates many forms of misbehavior, we mainly focus on two threats in this work. Firstly, we consider the scenario where a malicious consumer colludes with the server to double-spend. For instance, if a merchant accepts an unconfirmed payment and proceeds with delivery, the server could then refuse to confirm the payment. Secondly, we consider the possibility of the server censoring a user's request, resulting in the user's inability to transfer funds back to the blockchain. It's worth noting that we consider it beyond the scope of our work to address situations where merchants fail to deliver goods or services post-payment. Our primary objective is to enable cryptocurrency payments in a retail setting without the need for trusted custodians, not to ensure the secure exchange between merchants and customers.

3.2 Design Goals

We aim to design an off-chain payment system where customers can transfer cryptos to their off-chain accounts for efficient payment, merchants can safely accept instant (unconfirmed) payments, and both customers and merchants can transfer their funds back to the blockchain on demand. The system should also meet the following properties:

- *Correctness.* On-chain and off-chain funds should be updated according to transactions confirmed on the blockchain.
- *Censorship resistance.* A user eventually can withdraw off-chain funds.
- *Finality.* A confirmed transaction cannot be reverted.
- *Secure instant payment.* An instant payment accepted by an honest merchant should eventually be confirmed.

The first two properties jointly secure users' funds. The last two properties ensure double spending is not possible.

4 Design

Figure 1 highlights the main interaction between the roles. In the normal case when the server behaves honestly, there are three kinds of operations. A walk-through of payment is shown as step B1-B3. After receiving a consumer's payment, the server processes the payment and notifies the merchant, which then

delivers goods (or services) to the consumer. Deposit or withdraw (step1 A1-A2) allows a user to transfer on-chain funds to its off-chain account or vice versa. The user first submits the request on-chain. The blockchain and the server in turn update the corresponding on-chain and off-chain accounts.

Periodically, the server submits transaction data to the blockchain (step C) so that the blockchain can serve as the last resort for a user to withdraw its off-chain funds even when the server rejecting processing the user's withdrawal request. As the blockchain is transparent and public available, a user can accumulate the transaction data and re-construct the off-chain states. In this way, a user can prove its off-chain balance to the blockchain and withdraw the funds during the exit phase (step D). In this phase, the merchants can also submit unconfirmed payments to the blockchain which confirm the transactions by re-excuting them.

In the following, we present our system design. The functionalities of blockchain and TEE are shown in Fig. 2 and 3 respectively. For simplicity, we assume the roles by default can verify the authenticity and integrity of the messages from each other. However, this does not include the interaction between the TEE server and the blockchain because the malicious server, as the TEE host, can feed fake blockchain data to the TEE. Unfortunately, exist solutions are only applicable to proof-of-work blockchains [19,31] since the solutions rely on the consensus's computational difficulty. As recent blockchains are using more energy-efficient consensus, our system uses a general approach which relies on the related roles to verify if the server is using fake blockchain data.

4.1 Normal-Case Operation

In the normal case, the server will honestly collect and process the deposit, withdrawal and payment requests from users. In each epoch (the interval is up to configuration), the server submits a commitment commt including the transaction data, to the blockchain. When the commitment is finalized on-chain, the included transactions are also committed. The commitment later can also be used for the users to retrieve off-chain funds when the server misbehaves.

Initialize. Initially, the server generates a key pair $(\mathsf{pk}_{\mathsf{tee}}, \mathsf{sk}_{\mathsf{tee}})$. The private key $\mathsf{sk}_{\mathsf{tee}}$ is stored inside the TEE while the public key $\mathsf{pk}_{\mathsf{tee}}$ is submitted to the blockchain. A user verifies that the public key $\mathsf{pk}_{\mathsf{tee}}$ is indeed generated by the TEE via remote attestation before using the system.

Deposit. A deposit transaction aims to transfer a consumer $\mathcal{C}$'s on-chain funds to its off-chain account. A consumer first submits on-chain a request depReq, including the on-chain account onBal and amount amnt. On receiving the $\mathsf{Deposit}$ message, the blockchain debits the account first to ensure that the request can be finished. Then the blockchain marks the request's status as $\mathsf{pending}$. The status will be turned into $\mathsf{finished}$ when the transaction is later committed on-chain.

Next on receiving the $\mathsf{Deposit}$ message from the blockchain, the server credits the customer's off-chain account offBal. However, the new funds cannot be used for payment until the deposit transaction is committed on-chain. Otherwise, in the *exit* phase, a consumer may not be able to re-execute a payment depending on the deposit. Thus, the server will treat the new funds as pending by recording

Initialize($\mathsf{pk}_{\mathsf{tee}}$)**:**
set $\mathsf{pk}_{\mathsf{tee}}$ as the TEE's public key and stopBlockHeight := exitBlockHeight := ∞
On receive ("Deposit", depReq) **from** $\mathcal{C}$
assert not SysStopped(), parse depReq as ($\mathcal{C}$, amnt)
set $\mathcal{C}$.onBal −= amnt and depReq.status := pending
On receive ("Withdraw", wdReq) **from** $\mathcal{U}$
assert not SysStopped()
set wdReq.status := pending and wdReq.expiry := blockHeight + δ
On receive ("Commit", commt, σ_{tee}) **from** $\mathcal{S}$
assert not SysStopped() and Σ_{tee}.Verify($\mathsf{pk}_{\mathsf{tee}}, \sigma_{\mathsf{tee}}$, commt)
assert commt.Epoch = epoch, set epoch += 1
for all depReq in commt.depReqs **do**
assert depReq is the first pending deposit, set depReq.status := finished
for all wdReq in commt.wdReqs **do**
assert wdReq is the first pending withdrawal, parse wdReq as ($\mathcal{U}$, amnt)
set $\mathcal{U}$.onBal += amnt and wdReq.status := finished
On receive ("StopSys") **from** $\mathcal{U}$
assert not SysStopped(), let wdReq be the earliest pending withdrawal
assert curBlockHeight >= wdReq.expiry
set stopBlockHeight = curBlockHeight and exitBlockHeight = stopBlockHeight + ϵ
On receive ("CancelDep", depReq) **from** $\mathcal{C}$
assert SysStopped() and depReq.status = pending
parse depReq as ($\mathcal{C}$, amnt), set $\mathcal{C}$.onBal += amnt and depReq.status = aborted
On receive ("ComLeftPay", payReq, payProof) **from** $\mathcal{M}$
assert payReq.status! = finished, parse payProof as (σ_{tee}, epoch, comEpoch)
assert SysStopped() and not ExitStarted() and comEpoch $\leq$ epoch $\leq$ payProof.epoch
assert Σ_{tee}.Verify($\mathsf{pk}_{\mathsf{tee}}, \sigma_{\mathsf{tee}}$, payReq, payProof.epoch, comEpoch)
parse payReq as ($\mathcal{C}$, $\mathcal{M}$, amnt), set $\mathcal{C}$.onBal −= amnt
set $\mathcal{M}$.onBal += amnt and payReq.status := finished
On receive ("Exit", bal, merklePath) **from** $\mathcal{U}$
assert SysStopped() and ExitStarted()
let commt be the recent commitment and merkleRoot := commt.merkleRoot
assert MerkleVerify(merkleRoot, ($\mathcal{U}$, bal), merklePath), set $\mathcal{U}$.onBal += bal
Function SysStopped()
Return curBlockHeight >= stopBlockHeight
Function ExitStarted()
Return curBlockHeight >= exitBlockHeight

Fig. 2. Blockchain functionalities.

it in the variable balToConfirm, which keep a record of pending funds in each epoch. Those pending funds will be set as confirmed when the corresponding epoch is committed on-chain. Finally, the server adds the deposit request depReq to the commitment commt as the preparation to commit the transaction.

Withdraw. A withdrawal transaction aims to transfer a user's off-chain funds to its on-chain account. Similarly, a user first submits on-chain a request wdReq.

Initialize:
$\mathsf{pk}_{\mathsf{tee}}, \mathsf{sk}_{\mathsf{tee}} = \mathsf{TEE.GenKey}()$, output $\mathsf{pk}_{\mathsf{tee}}$
On receive ("Deposit", depReq) **from** $\mathcal{B}$
parse depReq as $(\mathcal{C}, \mathsf{amnt})$, set $\mathcal{C}$.offBal += amnt
add $(\mathcal{C}, \mathsf{amnt})$ to balToConfirm[epoch] and depReq to commt.depReqs
On receive ("Withdraw", wdReq) **from** $\mathcal{B}$
parse wdReq as $(\mathcal{U}, \mathsf{amnt})$
if $\mathcal{U} = \mathcal{C}$ **then** Set $\mathcal{U}$.comOffBal −= amnt and $\mathcal{U}$.offBal −= amnt
else if $\mathcal{U} = \mathcal{M}$ **then** set $\mathcal{M}$.offBal −= amnt
add wdReq to commt.wdReqs
On receive ("Payment", payReq) **from** $\mathcal{C}$
parse payReq as $(\mathcal{C}, \mathcal{M}, \mathsf{amnt})$, set $\mathcal{C}$.comOffBal −= amnt, $\mathcal{C}$.offBal −= amnt
set $\mathcal{M}$.offBal += amnt and $\sigma_{\mathsf{tee}} := \mathsf{TEE.Sign}(\mathsf{payReq}, \mathsf{epoch}, \mathsf{comEpoch})$
add payReq to commit.payReqs, output $(\sigma_{\mathsf{tee}}, \mathsf{epoch}, \mathsf{comEpoch})$ to $\mathcal{C}$
Function Commit()
set commit.merkleRoot := merkleTree.root and $\sigma_{\mathsf{tee}} := \mathsf{TEE.Sign}(\mathsf{commit})$
set oldCommit := commit, reset commit, set epoch += 1
set commit.epoch := epoch, return (oldCommit, σ_{tee})
Function ConfirmEpoch()
1: Set comEpoch += 1
2: **for all** $(\mathcal{C}, \mathsf{amnt})$ in balToConfirm[comEpoch] **do** set $\mathcal{C}$.comOffBal += amnt

Fig. 3. TEE server functionalities.

On receive the Withdraw message, the blockchain then marks the request's status as pending and specially sets the request's expiry time to $\mathsf{blockHeight} + \delta$, which is the current block height plus a hyper-parameter. This timeout ensures that a user can retrieve its off-chain funds given an untrusted server.

Next on receiving the Withdraw message from the blockchain, the server starts to debut the user's off-chain balance. If the user is a merchant $\mathcal{M}$, the server debits the merchant's off-chain account offBal. If the user is a consumer $\mathcal{C}$, the server debits the account comOffBal, in which the funds received are from confirmed deposits. This ensures that the consumer only use funds from confirmed deposits for withdrawal. Meanwhile, the server deducts the equivalent amount from offBal, which records the consumer's total off-chain balance, including the funds from unconfirmed deposits. Finally the server adds the request wdReq to the commitment commt to commit the transaction later.

Payment. A payment transaction aims to transfer a consumer's off-chain funds to a merchant. A consumer $\mathcal{C}$ first submits a request payReq to the server. On receiving the Payment message, the server debits the consumer's account comOffBal and offBal to ensure that the consumer only uses funds from confirmed deposits for payment. Then the server credits the merchant's account offBal accordingly. Next the server adds the request to the commit to commit the transaction later. Finally, the server uses the TEE's private key $\mathsf{sk}_{\mathsf{tee}}$ (unknown to the server host for the TEE's privacy ensurance) to sign the request together

with the current epoch and the confirmed epoch comEpoch in the TEE. The generated σ_{tee} and the signed epoch variables can serve as a payment proof payProof that a consumer later can use to re-execute unconfirmed payments in the *exit* phase.

Next the server notifies the merchant with the payment proof payProof attached. First, the merchant verifies the signature σ_{tee} to ensure that the payment proof is indeed generated by the TEE. Besides, the merchant has to verify that the epoch comEpoch has been committed on-chain. The reason, as we stated before is that the TEE's network stack is controlled by the untrusted server host, and there is no authenticated channel between the TEE and the blockchain. So, the TEE may not know when an epoch's commitment would be finalized on-chain. Our design is to let the server itself decides when to confirm an epoch in the TEE while let the merchant itself verify the genuinity. With an accurate comEpoch, the TEE guarantees that the consumer only used the funds from confirmed deposits for payment. After verifying the payment proof, the merchant can deliver goods or services immediately to the consumer without waiting for the payment transaction to be committed.

Commit. At the start of each epoch, the server commits the executed transactions on the blockchain. The server first executes the function Commit which adds the current merkleRoot to the commitment commt and uses the TEE's private key to sign the commitment.

Next the server submits the function outputs, including the commitment commt and the TEE's signature σ_{tee}, to the blockchain. On receiving the Commit message, the blockchain first verifies the signature and checks that the commitment's epoch matches the expected epoch. Then the blockchain retrieves pending deposits and withdrawals to check they match the requests included in the commitment in order. For each withdrawal, the blockchain also credits the corresponding account. Afterwards, the requests' statuses are marked as finished, indicating that the transactions cannot be reverted once the commitment is finalized on the blockchain.

Finally, after observing that the commitment commt has been finalized on-chain, the server invokes the function Confirm to proceed the epoch in the TEE. The function releases the pending funds accumulated dring the earliest unconfirmed epoch to each corresponding consumer's account comOffBal. A consumer thereafter can use the funds for instant payment.

4.2 Force Exit

When the server misbehaves, a user can submit a withdrawal request to the blockchain to withdraw off-chain funds. If the server honestly responds to the request before the timeout, users can get back the funds; this will also help a merchant confirm unconfirmed payments since a commitment will finalize all the previously executed transactions in the TEE. Otherwise, as in rollups and commit-chains, any user can stop the system on the blockchain. After that, the server and blockchain no longer accept new requests. Specifically, the merchants

should submit unconfirmed payments before consumers can withdraw off-chain funds. This ensures that the consumers have enough funds to cover the payments.

Stop System. Once a user notice that the earliest pending withdrawal request wdReq times out, i.e. the current block height surpasses wdReq.expiry, the user sends the StopSys message to the blockchain. After verifying that the request indeed has timed out, the blockchain halts the system by setting stopBlockHeight to the current block height. The exit time exitBlockHeight is set to the stop time plus the hyper-parameter epsilon so that merchants have enough time to submit unconfirmed payments before users withdraw their off-chain funds. The blockchain will no longer accept deposits, withdrawals and commitments after the system is stopped. Thereafter, a consumer can first cancel an unfinished deposit by sending the CancelDep message to the blockchain, which credits the on-chain account onBal and sets the deposit's status to aborted. As introduced below, a consumer also and withdraw its off-chain account after merchants confirming unfinished payments.

Confirm Unfinished Payments. A merchant has time to submit unconfirmed payments before the exit block height is reached. For each payment, the merchant submits the request payReq together with the payment proof payProof on-chain. On receiving the ComLeftPay message, the blockchain first verifies the payment proof as the merchant has done, i.e., checking the TEE's signature and verifying that the epoch comEpoch attached to the payment proof has been confirmed. Besides, the blockchain verifies that the epoch attached to the proof is less or equal than the current on-chain epoch. This makes ensure that the merchant only submit unconfirmed payments. Finally, the blockchain executes the payment according to the request. It's acceptable if the consumer's account onBal has a balance below zero since it is compensated by the consumer's off-chain funds.

Withdraw Off-chain Funds. A user can withdraw all its off-chain funds after the block height surpasses exitBlockHeight. Since all transaction data have been posted on-chain along with the commitments, a user can re-construct the Merkle tree that records everyone's balance. To withdraw the funds, the user submits the balance bal and the corresponding Merkle tree path merklePath to the blockchain. On receiving the Exit message, the blockchain uses the tree path and the Merkle root in the recent commitment to verify that the balance is indeed a part of the tree leaf node. Finally, the balance is credited to the user's account onBal.

4.3 Analysis

Without instant payments, our design has the same security property as zkRollups since we majorly replace zero knowledge proofs with the a TEE on the server side. We next show our system is still secure after bringing in instant payments.

It is easy to verify that instant payments can correctly update the funds in the system and cannot be reverted once finalized on the blockchain. We mainly argue that an honest merchant can finally confirm an instant payment once

receiving the valid payment proof. In the first case when the server submits the next commitment on the blockchain in time, the instant payment will be confirmed once the commitment is finalized on-chain. Due to the TEE's integrity, the instant payment should have been stored in that commitment or a commitment before. In the other case when the server fails to submit a commitment in time, the honest merchant or other users can stop the system by invoking stopSys function. After that, the merchant can make the instant payment confirmed by submitting and re-executing it on the blockchain.

Note that an honest merchant should not accept an instant payment before receiving a valid payment proof. In this case, the payment is treated as non-instant payment and will be confirmed later when the server submits the next commitment on-chain. Besides, an honest user should not accept an instant payment after the system has stopped but the time for submitting instant payments has passed. Otherwise, the user cannot submit and confirm the payment on-chain.

5 Evaluation

Our experiments aim to answer the following questions: *1) What is the maximum throughput of the transactions that users may frequently make? 2) What are the transaction fees of the transactions involving interaction with the blockchain?*

A transaction's maximum throughput in our system may depend on several factors. If an transaction is solely on-chain, then its throughput depends on the transaction's gas cost, which decides the largest number of transactions that can be included in a block. If an transaction also needs the server's assistance, then its throughput also depends on the server's capacity. Besides throughput, another aspect impacts user experience is transaction fees. We mainly focus on on-chain transaction fees since these fees cannot be easily avoided in practice. A transaction's fee also can be computed from the transaction's gas cost.

In summary, our measurement includes two parts. First, we measure the gas cost of the transactions involving the interaction with the blockchain. Based on that, we estimate the frequently-used operations' maximum on-chain throughput. Here, we use the term *maximum* because we assume the ideal case when blocks in the blockchain can be fully used by the transactions in our system. Finally, we measure the frequently-used operations off-chain throughput.

Setup. We use Ethereum as the underlying blockchain. The blockchain functionalities are implemented with a smart contract using Solidity 0.8.18. The TEE functions are implemented with SGX SDK using C++. The server and users use the Ethereum's native signature scheme ECDSA to sign messages. We assume users sign their transactions before the transactions are sent to the server.The Merkle tree that stores users' off-chain funds uses the Ethereum's native hash function keccak256. To fully test the system's capacity, we use a static Merkle tree pre-filled with the maximum user numbers. We will test the impact of different numbers on the system performance.

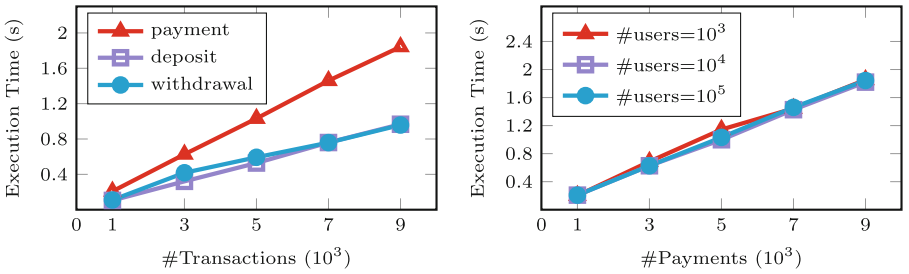
5.1 Off-Chain Throughput

Deposits, withdrawals and payments are the operations that the server may need to execute most of the time. Figure 4(a) shows the execution time of the operations on the server side. It can be seen that the time grows linearly with the number of transactions. In one second, the server can process around 4845 payments, 9307 deposits and 9395 withdrawals. The performance gap between payments and deposits/withdrawals comes from the fact that an payment operation needs to generate a payment proof. Otherwise, the operations have similar performance. Another factor that may impact performance is the number of users, which decides the depth of the Merkle tree. The three kinds of operations also need to update the Merkle tree when updating off-chain balance. But as suggested by Fig. 4(b), the number of users have little impact. The performance is majorly capped by verifying user signatures on transactions. We expect that the performance can be further improved by parallelizing the signature verification.

5.2 On-Chain Throughput and Transaction Fee

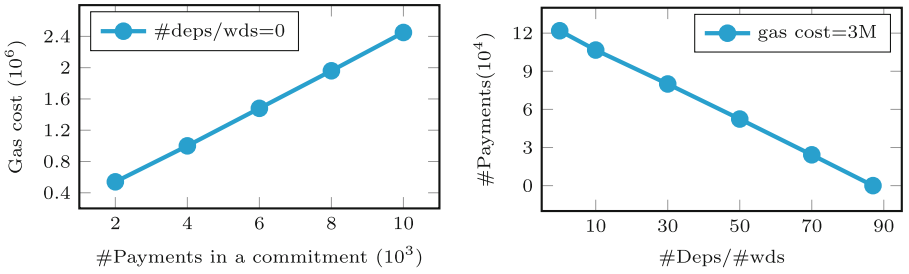
To estimate the throughput and transaction fee, we need to first measure the gas cost. Table 1 shows the gas cost of the smart contract functions that users or the server needs to invoke multiple times. Payments do not incur gas cost until being committed on the blockchain. Figure 5(a) shows the gas cost of committing payments when there are no deposits or withdrawals to be committed at the same time. It can be seen that the gas cost grows linearly with the payment numbers. On average, one payment only incurs 250 gas. In contrast, one transfer incurs around 21000 gas in Ethereum. One reason that off-chain payment is more efficient is that off-chain payment can be expressed in a concise form. As in rollups, an account can be expressed as 4-bytes integer instead of 20-bytes address in Ethereum, and similarly an payment amount can be expressed as 4-bytes float instead of 32bytes unsigned integer. Another reason is that payments only need to be submitted as transaction payload and do not have to be executed.

Deposits and withdrawals first need to start the operation on the blockchain. As shown in Table 1, the two corresponding smart contract functions incurs 124.4k and 146.39k gas respectively. Then the two kinds of operations need to be committed on the blockchain. As suggested by Fig. 6(a), the cost of the two operations grow linearly with the number of transactions. The average gas cost of committing a deposit or a withdrawal is around 34.33k. Overall, deposits or withdrawals are not faster than plain transfers in Ethereum since both operations need to operate on-chain balance. The extensive use of the two operations will slow down payments since a commitment comitts payments as well. Figure 5(b) shows the impact on payments. For a commitment that consumes 3M gas, the number of max payments drops linearly from 12000 to 0 when the number of deposits and withdrawals grows from 0 to 90. (Here, we assume there are equal number of deposits and withdrawals). Thus, users are expected to use deposits and withdrawals minimally.



(a) Execution time vs. #transactions (b) Execution time vs. #transactions

Fig. 4. (a): execution time of payment/deposit/withdrawal transactions. **(b):**: payment execution time under different number of users

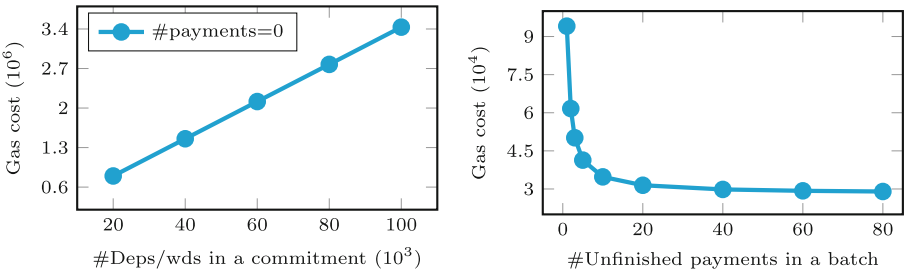


(a) Gas cost vs. #payments (b) #Payments vs. #deps/wds

Fig. 5. (a): #payments and #deposits/withdrawals in a commitment that consumes 3M gas. **(b)**: gas cost of a commitment without deposits and withdrawals.

When the server misbehaves, users need to confirm unfinished instant payments and retrieve their off-chain funds. The gas cost of committing one single instant payment is k. This cost can be further reduced by batching the payments to be confirmed. In the normal case, instead of returning a payment proof proving the current payment, the TEE server returns the proof proving the payments in an epoch. Figure 6(b) shows the impact of such optimization. The gas cost is reduced from 94.07k to 28.85k when number of payments in a batch grows from 1 to 80. This optimization basically saves the cost of verifying the payment proof for every payment. Besides, users need to retrieve their off-chain funds with the exit function, which verifies a Merkle proof proving the balance. The number of users decides the depth of the Merkle tree, thus impacting the gas cost of the exit function. Fortunately, the impact is minor. The gas cost of one call only increases from 63.44k to 74.7k as the number of users increases from 100 to 10k. Finally, users need to invoke the function CancelDep to revert pending deposits. One function call incurs the gas cost of 63.44k.

Maximum On-chain Throughput. Based on the gas costs, we can estimate the throughput of each function on the blockchain. As per May 7, the maximum gas of a block in Ethereum has been increased to 30M, the block time is



(a) Gas cost vs. #deps/wds (b) Gas cost vs. #unfinished payments

Fig. 6. (a): gas cost of a commitment without payments. **(b)**: gas cost per unfinished payment under batching.

Table 1. Performance of smart contract functions.

Function	Deposit	Withdraw	Commit	CancelDep	ComLeftPay	Exit
Gas Cost (k)	124.4	146.39	0.25 (payment) or 34.33 (dep/wd)	60.24	28.85	63.44
Tx Fee ($)	3.41	4.02	$6.78 * 10^{-3}$ (payment) or 0.94 (dep/wd)	1.65	0.79	1.74
Max TPS	19.9	16.84	9909.16 (payment) or 72.09 (dep/wd)	41.123	85.8	38.98

Note that the gas cost of a plain transfer is around 21k on Ethereum.

12.11 s on average. In the extreme case, our system can fully utilize each block in the blockchain. The maximum transactions per second (TPS) is decided by the following equation:

$$\text{TPS} = \frac{\text{Block Gas Limit}}{\text{Transaction Gas Cost * Block Time}} \tag{1}$$

As shown in Table 1, all the operations but payments have similar slow throughput to Ethereum, while the max throughput of payments are quite appealing, which is as high as 9909 TPS. The throughput is comparable to the reported statistics from some rollups in industry. But we also have addressed the latency issue. So far, the off-chain throughput of deposits (9307 TPS) and withdrawals (9395 TPS) is already fast enough. Although that of payments (4845 TPS) is around half of the maximum on-chain throughput, we consider it acceptable as our system cannot fully use every block in the blockchain, while we stress that the off-chain throughput still have much room for improvement.

Transaction Fees. We also can estimate the transactions fees given the gas costs. As per May 7 2023, we use the standard gas price 14.3 gwei and use the ETH price 1919.55$, as suggested by [9]. The transaction fees are shown as in Table 1. Since all the operations except payments are expected to be used infrequently, we consider their transaction fees acceptable. For payments, the transaction fees are relatively cheap when the server commit multiple payments at the same time. The average fee is $6.78{*}10^{-3}$ for 10000 payments and $7.32{*}10^{-3}$ for 2000 payments, far more cheaper than the transfer cost 0.58$ in Ethereum.

6 Related Work

On-chain Scaling. On-chain scaling methods aim to boost performance by reimagining the blockchain protocol. These strategies include replacing the proof of work mechanism with more efficient consensus protocols [25,29,33] and leveraging parallelism by proposing multiple blocks at the same time [17,35,41]. However, these on-chain scaling solutions often face compatibility issues with pre-existing blockchains. Furthermore, they typically exhibit high latency and limited throughput, or their real-world performance hasn't been fully evaluated and verified. Nonetheless, if a high-performance blockchain were to exist, it would be beneficial to off-chain protocols, which typically rely on the underlying blockchain for security. Off-chain protocols also prove beneficial in distributing the workload of the blockchain.

Off-chain Scaling. Off-chain scaling instead processes transactions outside the blockchain. Existing works are trustless (e.g., [20–22,24,30,32] or trust-based(e.g., [16,18,26,28,34,36,40]). The trust-based solutions need to totally trust the off-chain components, usually a committee, for processing transactions. The committee may be compromised as evidenced by real-world incidents [2,5]. Trustless solutions on the other hand use the blockchain as a trust anchor to verify off-chain operations. However, payment channels [22] and commit chains [24,32] require users to be periodically online to secure their funds and also have high collateral requirement as analyzed in [37]. Rollups [20,24,32] and Ekiden [21] have high transaction latency for committing transactions on the blockchain. While snappy [37] allows fast transaction confirmation by compensating potential loss with collateral, their design does not improve the transaction throughput and the collateral amount scales with the system's transaction volume.

7 Conclusion

We introduce an off-chain payment system characterized by low latency and practical throughput. Uniquely, our system incorporates a hardware-driven solution that permits merchants to securely accept unconfirmed payments without requiring collateral. However, a limitation of our current design is the lack of privacy assurance, as transaction data posted to the blockchain remains in plaintext. Looking forward, we plan to integrate TEE and cryptography to simultaneously achieve off-chain efficiency and on-chain privacy.

Ackowledgement. This work was fully supported by the Research Grants Council of Hong Kong under GRF Grant CityU 11213920, and RIF Grant R1012-21.

References

1. Accepting bitcoin payments. https://www.skynova.com/blog/accepting-bitcoin, Accessed 27 Aug 2022
2. Are blockchain bridges safe? why bridges are targets of hacks. https://www.coindesk.com/learn/are-blockchain-bridges-safe-why-bridges-are-targets-of-hacks/, Accessed 27 Aug 2022
3. Binance. https://pay.binance.com/en, Accessed 27 Aug 2022
4. Bitcoin: a peer-to-peer electronic cash system. https://www.ussc.gov/sites/default/files/pdf/training/annual-national-training-seminar/2018/Emerging_Tech_Bitcoin_Crypto.pdf, Accessed 27 Aug 2022
5. Bitcoin SV suffers a new 51% attack. https://forkast.news/headlines/bitcoin-sv-bsv-suffers-new-51-attack/, Accessed 27 Aug 2022
6. Coinbase. https://commerce.coinbase.com/, Accessed 27 Aug 2022
7. Cryptocurrency payments report: how consumers want to use it to shop and pay. https://www.pymnts.com/wp-content/uploads/2021/05/PYMNTS-Cryptocurrency-Payments-Report-May-2021.pdf, Accessed 27 Aug 2022
8. Cryptocurrency tether is fined $41 million for lying about reserves. https://fortune.com/2021/10/15/tether-crypto-stablecoin-fined-reserves/, Accessed 27 Aug 2022
9. Eth gas station. https://ethgasstation.info, Accessed 27 Aug 2022
10. Mastercard, visa, Paypal suspend Russian operations - no love for Russia? https://bitcoinist.com/mastercard-visa-paypal-suspend-russian-operations/, Accessed 27 Aug 2022
11. Paying with cryptocurrency: What consumers and merchants expect from digital currencies. https://www.pymnts.com/study/paying-with-cryptocurrency-shopping-consumer-finance-digital-wallets/, Accessed 27 Aug 2022
12. Paypal. https://www.paypal.com/us/digital-wallet/manage-money/crypto, Accessed 27 Aug 2022
13. Starbucks now accepts bitcoin as payment (kind of...). https://www.foodandwine.com/news/starbucks-bitcoin-frequent-flyer-miles-payment-gift-cards, Accessed 27 Aug 2022
14. Tech embattled crypto lender celsius files for bankruptcy protection. https://www.cnbc.com/2022/07/19/what-happens-to-my-funds-if-a-crypto-exchange-goes-bankrupt.html, accessed 27 Aug 2022
15. Visa. https://www.visa.com.hk/content/VISA/usa/englishlanguagemaster/en_US/home/solutions/crypto.html, Accessed 27 Aug 2022
16. Avarikioti, Z., Kokoris-Kogias, E., Wattenhofer, R., Zindros, D.: Brick: asynchronous incentive-compatible payment channels. In: Proceedings of FC (2021)
17. Bagaria, V.K., Kannan, S., Tse, D., Fanti, G.C., Viswanath, P.: Prism: deconstructing the blockchain to approach physical limits. In: Proceedings of ACM CCS (2019)
18. Baudet, M., Danezis, G., Sonnino, A.: Fastpay: high-performance byzantine fault tolerant settlement. In: Proceedings of ACM AFT (2020)
19. Bentov, I., Ji, Y., Zhang, F., Breidenbach, L., Daian, P., Juels, A.: Tesseract: real-time cryptocurrency exchange using trusted hardware. In: Proceedings of ACM CCS (2019)
20. Buterin, V.: On-chain scaling to potentially 500 tx/sec through mass tx validation. https://ethresear.ch/t/on-chain-scalingto-potentially-500-tx-sec-through-mass-tx-validation/3477, Accessed 27 Aug 2022

21. Cheng, R., et al.: Ekiden: a platform for confidentiality-preserving, trustworthy, and performant smart contracts. In: Proceedings of EuroS&P (2019)
22. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Proceedings of SSS (2015)
23. Dziembowski, S., Fabiański, G., Faust, S., Riahi, S.: Lower bounds for off-chain protocols: exploring the limits of plasma. In: Proceedings of ITCS (2021)
24. Erwig, A., Faust, S., Riahi, S., Stöckert, T.: Commitee: an efficient and secure commit-chain protocol using tees. Cryptology ePrint Archive (2020)
25. Eyal, I., Gencer, A.E., Sirer, E.G., van Renesse, R.: Bitcoin-ng: a scalable blockchain protocol. In: Proceedings of NSDI (2016)
26. Gai, F., Niu, J., Tabatabaee, S.A., Feng, C., Jalalzai, M.: Cumulus: a secure BFT-based sidechain for off-chain scaling. In: Proceedings of IWQoS (2021)
27. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: analysis and applications. In: Proceedings of Eurocrypt (2015)
28. Gaži, P., Kiayias, A., Zindros, D.: Proof-of-stake sidechains. In: Proceedings of S&P (2019)
29. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: scaling byzantine agreements for cryptocurrencies. In: Proceedings of SOSP (2017)
30. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: scalable, private smart contracts. In: Proceedings of USENIX Security (2018)
31. Kaptchuk, G., Green, M., Miers, I.: Giving state to the stateless: augmenting trustworthy computation with ledgers. In: Proceedings of NDSS (2019)
32. Khalil, R., Zamyatin, A., Felley, G., Moreno-Sanchez, P., Gervais, A.: Commit-chains: secure, scalable off-chain payments. Cryptology ePrint Archive (2018)
33. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Proceedings of CRYPTO (2017)
34. Kiayias, A., Zindros, D.: Proof-of-work sidechains. In: Proceedings of FC (2019)
35. Li, C., et al.: A decentralized blockchain with high throughput and fast confirmation. In: Proceedings of ATC (2020)
36. Lind, J., Naor, O., Eyal, I., Kelbert, F., Sirer, E.G., Pietzuch, P.: Teechain: a secure payment network with asynchronous blockchain access. In: Proceedings of SOSP (2019)
37. Mavroudis, V., Wüst, K., Dhar, A., Kostiainen, K., Capkun, S.: Snappy: fast on-chain payments with practical collaterals. In: Proceedings of NDSS (2020)
38. Pass, R., Shi, E., Tramer, F.: Formal abstractions for attested execution secure processors. In: Proceedings of Eurocrypt (2017)
39. Poon, J., Dryja, T.: The bitcoin lightning network: scalable off-chain instant payments (2016). https://lightning.network/lightning-network-paper.pdf
40. Wüst, K., Matetic, S., Egli, S., Kostiainen, K., Capkun, S.: Ace: asynchronous and concurrent execution of complex smart contracts. In: Proceedings of CCS (2020)
41. Zamani, M., Movahedi, M., Raykova, M.: Rapidchain: scaling blockchain via full sharding. In: Proceedings of CCS (2018)