Seamless Ethereum Withdrawals: Gasless* Layer 2 to Layer 1 Transactions

Arun Sankar KS[1] and Jeremy Clark[2]

Concordia Institute for Information System Engineering (CIISE)
Concordia University, Montreal, QC, Canada.

Abstract. Rollups are widely used as scalability solutions for Ethereum, functioning as a secondary layer (L2) to the primary blockchain (L1). These systems, while enhancing transaction speed and reducing costs, face challenges in seamless interactions with L1. This paper explores and proposes innovative solutions for facilitating seamless withdrawals from Layer 2 to Layer 1 blockchains without necessitating users to possess native L1 tokens for transaction fees. We highlight the complexities associated with current L2 withdrawal mechanisms and introduce a unique framework for evaluating various strategies aimed at mitigating these challenges. Four main approaches are analyzed: Meta-Transactions with incentives for gas relayers, ERC-4337 Account Abstraction, Triggered Withdrawals, and Sponsored Withdrawals. The latter, focusing on user-friendly transactions, is extensively dissected to demonstrate its practical application in reducing user friction and dependency on L1 assets for gas fees. Our solution uses bundled transactions via Flashbots to ensure atomicity, while also preventing potential security risks associated with individual transaction failures or front-running attacks, offering a substantial contribution to the scalability and efficiency of blockchain ecosystems.

Keywords: Ethereum, layer 2, rollups, bridges, gasless transactions

1 Introduction

Layer 2 solutions, often referred to as "L2s," are technologies developed to enhance the scalability, speed, and cost-effectiveness of base-level blockchain networks, commonly known as "Layer 1" blockchains like Ethereum. These Layer 2s address the limitations of their parent blockchains by handling transactions off the main chain. This approach significantly reduces the burden on the main network, allowing for faster transaction times and lower fees, without compromising the security and decentralization features of the underlying blockchain.

Let's assume, for instance, Alice uses Arbitrum, a popular Layer 2 network, for various activities. She receives airdrops of new tokens directly on Arbitrum, which not only introduces her to new projects but does so without the hefty transaction fees typical of the main Ethereum chain. Additionally, Alice prefers withdrawing her assets like ETH from centralized exchanges directly to Arbitrum. This method is not only cost-effective but also allows her to interact with decentralized applications and smart contracts on Arbitrum with efficiency and lower costs.

Consider that Alice has some ETH on Layer 2 and wants to withdraw it to Layer 1. She deposits the ETH to the bridge in L2, and to withdraw from the L1 bridge, Alice needs to pay gas in L1 ETH. Here arises the issue: Alice cannot withdraw her ETH on L1 without already having some ETH on L1 to pay for gas. This leads to a deadlock scenario and affects the user experience of using L2s.

This problem is not only for L2 to L1 withdrawals but also from L3 to L2 and L2 to different L2s.

Arbitrum, one of the leading Layer 2 scaling solutions for Ethereum, uses Optimistic Rollups to provide a significant increase in transaction throughput and a reduction in transaction fees. As of now, Arbitrum processes more transactions daily than Ethereum itself, demonstrating its efficiency and the trust users and developers place in its capabilities [1].

With a Total Value Locked (TVL) in the billions, Arbitrum is a hotspot for DeFi activities, supporting a wide array of applications ranging from decentralized exchanges to yield farming platforms and beyond [2]. This high TVL indicates strong user engagement and a solid foundation of assets underpinning the network's various decentralized applications (dApps).

1.1 Arbitrum Analytics

Here’s some data, Analyzing the transactions of Arbitrum One (Since it has the highest TVL) L1 Outbox Contract :

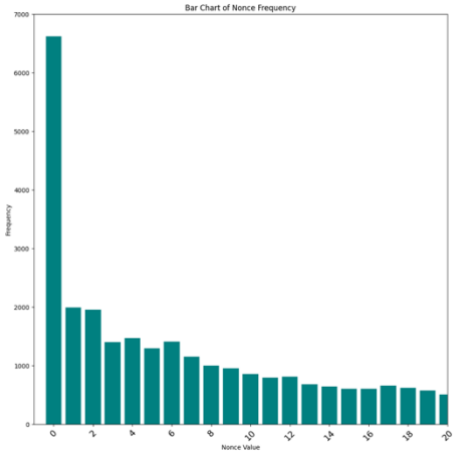

Figure 1. Barplot of Nonce of Transactions to Arbitrum One Outbox contract.

Nonce of Transactions to Arbitrum One Outbox 4 Contract on ETH (L1): 0x0b9…..4840 [13]

Out of the 55,846 transactions as of blockheight 19150536, 6618 have a nonce of 0. This suggests that the initial transactions associated with about 11.85% of the wallets are primarily for funding gas to cover withdrawals from the L1 Outbox, highlighting the interest in gasless withdrawals.

1,992 have a nonce of 1, these wallets primarily interacted with Arbitrum Inbox Contract then the Outbox for withdrawal, meaning these wallets were funded explicitly for bridging funds between layers.

2 Architecture

In this paper, we propose ways to facilitate withdrawals without the need for a user to hold ETH on L1 to pay gas for withdrawals. We will go through four potential solutions to address this problem and select the optimal one.

In most cases, we'll use the `eth_sendbundle()` method from the Flashbots framework to carry out part of the solution.

Flashbots is a research and development organization focused on mitigating the negative externalities and risks of current miner extractable value (MEV) extraction strategies in blockchain ecosystems, particularly Ethereum [11]. MEV refers to the maximum value that can be extracted from block production in excess of the standard block reward and gas fees through methods like reordering, inserting, or censoring transactions within blocks [10].

2.1 Meta Transactions with Incentive for Gas Relayer

A meta transaction is a type of Ethereum transaction where the initiator does not bear the cost of the gas fee. Although they originate the transaction, it is not directly executed from their wallet. Instead, it is sent by another wallet, using a relay smart contract, to act on behalf of the initiator [6].

Meta Transactions operate on a straightforward principle:

1. There is the user who wants to make the transaction.
2. A third party, known as a Relayer, processes transactions for the user and covers the associated gas fees.
3. Trusted Forwarder, a contract trusted by the recipient (bridge) to validate signature and nonces.
4. The recipient / Smart contract who the user interacts with.

Participants:

User: User sign messages (compliant to EIP-712 standard), which contains the payload to be executed, and sign the refund transaction to the sponsor.

Relayer: Gets a signed message off chain from the 'User'. Relayer is responsible for paying the gas fee, signing it into a native transaction, paying gas and submitting it for execution.

Trusted Forwarder: This is a contract that the "recipient" contract relies on to verify nonces and signatures before it forwards the request from the "relayer" to the "recipient."

Recipient Contract: A smart contract that accepts meta transactions via a" trusted forwarder."

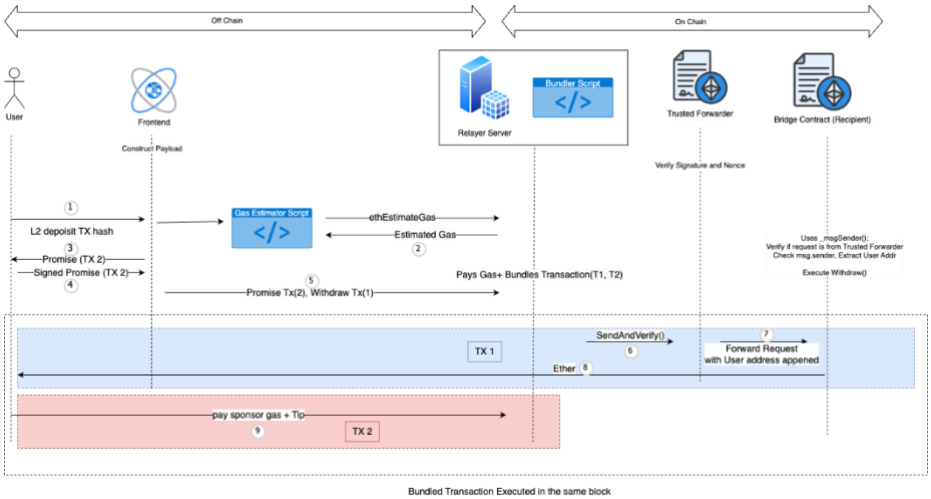

Figure 2. Gasless withdrawal using Meta Transactions

Workflow:

1. User submits the Layer 2 deposit transaction hash, which we will be using to compute the withdrawal proofs and payload for TX 1. The hash is used to construct the payload.

2. We run a gas estimator script, `ethEstimateGas`, and we get the gas required to execute the transaction. Based on the value returned, we calculate the cost by Gas * Current Network gas price.

3. We construct the payload for the Meta Transaction, using the L2 hash and have the user sign it, we will call this TX 1.

4. We craft a payload based on the cost of executing the withdrawal transaction, stating that the user will refund the sponsor for the gas cost + an incentive (10% usually) as the value, and we will call this TX 2, and have the user sign it.

5. Now we have TX 2 (User promise of refund) + TX 1 (Withdrawal transaction which we constructed using L2 Deposit Hash), we bundle them up using flashbots [11]. This will make sure that both the transactions are atomic, either both of them execute or neither of them does, and that the order of the transactions remains the same.

TX 1 (Meta Transaction)

6. TX 1 calls the SendandVerify() function on the Trusted forwarder contract (trusted by bridge contract hence the name), this function verifies the payload and makes sure that the nonces and signature of the original user are valid.
7. The request is forwarded from the trusted forwarder, and `_msgSender()` function is used to calculate the original sender (User address) and complete the withdrawal.
8. The ether is sent to the user account.

TX 2 (Promise TX)

9. The promise transaction (TX 2) is executed, after the withdrawal is complete in the same block. This transaction will refund the sponsor.

2.1.1 High Level Overview

User Frontend Interaction (Off Chain):
User submits the L2 Deposit TX hash (required to compute the withdrawal Tx payload) and the frontend then interacts with a Gas Estimator Script, which calculates the estimated gas required for the transaction. Using these the user creates and signs a “Promise” transaction stating “Send Gas + tip” Ether to relayer server (Gas provider).

Relayer Server Operations (Off Chain to On Chain):
The Relayer Server receives the payload, which includes the promise TX 2 and the withdrawal request TX 1. The Relayer Server uses a Bundler Script to bundle these transactions together and pays for the gas fees for Withdraw TX, preparing them to be sent to the blockchain in one operation [11].

Trusted Forwarder (On-Chain):
This serves as a Trusted Source of Meta Transactions to the Recipient contract (Bridge Contract). The trusted forwarder verifies the signature and nonce and then forwards it to the bridge contract.

Bridge Contract (Recipient):
The bridge contract uses a new method `_msgSender' to get the original transaction signer (User) and then executes the withdrawal.

Execute Withdraw (On-Chain):
Once verification is complete, the bridge contract executes the Withdraw() function, which facilitates the withdrawal of funds (Ether in this case) to the user's address. And if the withdrawal Tx 1 is successful, the Promise TX 2 is executed within the same block after Tx 1.

Assumptions:

- Users should avoid keeping L1 ETH in their wallets as a malicious sponsor could potentially steal these funds without processing their withdrawal requests, they could simply execute the User promise (TX2) without executing the withdrawal.

- While signing the payloads, we must trust the relayer to act in good faith, as users can be duped into signing malicious payloads (e.g. permissions to drain the wallet).

- Sponsor should use 'eth_sendbundle' to send both transactions. Otherwise, they risk losing money because of the user's transaction failure or front-running attacks.

- The amount of ETH that is going to be withdrawn for user on L1 after the execution of withdraw transaction must be greater than or equal to (number of gas used + X% tip) * (block.basefee + 5 Gwei) at the time of execution of transaction. Otherwise sponsoring will fail.

- Users are required to add a 'X' percent tip over the sponsor's execution costs as a reward for handling the transaction.
- Relayers should be satisfied with the tip provided.

- eth_estimateGas method does not return the exact amount of ETH consumed in the transaction. It is generally impossible to guess the exact amount of gas a transaction will use until its execution.

- Block.basefee can fluctuate between the time a user signs the transaction and when the transaction will be confirmed by a builder.

- Modifying the bridge contract can introduce new vulnerabilities or unintended loopholes. Every change needs thorough testing and auditing to prevent security breaches, which can be costly and time-consuming.

2.2 ERC 4337 Account Abstraction

ERC-4337, also known as Account Abstraction, is an Ethereum improvement proposal that aims to fundamentally alter how user accounts and transactions are handled on the Ethereum blockchain [5]. Unlike traditional Ethereum accounts, which require externally owned accounts to initiate transactions, ERC-4337 introduces a new concept where accounts can be abstracted into smart contracts, effectively allowing these accounts to behave like smart contracts themselves.

The ERC4337 System includes several new elements, beyond the usual:

User Operation: Unlike a traditional transaction, a user operation in ERC-4337 encapsulates all transaction-related information in a single package that can be processed by the network. This package includes the intended actions, the max fee willing to be paid, the gas limit, and a signature proving the user's consent to the operation.

EntryPoint: The EntryPoint is a dedicated contract on Ethereum that acts as the gatekeeper for user operations. All user operations are sent to the EntryPoint, which is responsible for validating them, ensuring they conform to the rules set out by the user's smart contract wallet, and executing them accordingly.

Paymaster: One of the features of ERC-4337 is the ability to pay transaction fees with tokens other than ETH, or even have a third party cover these costs. Paymasters are contracts that agree to sponsor the gas costs of transactions in exchange for compensation, potentially in different forms or tokens.

Bundler: After validation, user operations are handed over to bundlers. Bundlers collect multiple user operations, package them into bundles, and submit these to miners or validators in the network. This process ensures efficient block space utilization and can help users save on transaction fees.

Participants:

User: Sends the withdrawal call from the smart wallet, signs the promise transaction.

ERC 4337 Contract: the smart wallet contract, which acts as the EOA and executes the UserOperations.

Bundler: The bundler/builder, who bundles the transactions [12].

Sponsor: Sponsor contract which funds the withdrawal transaction.

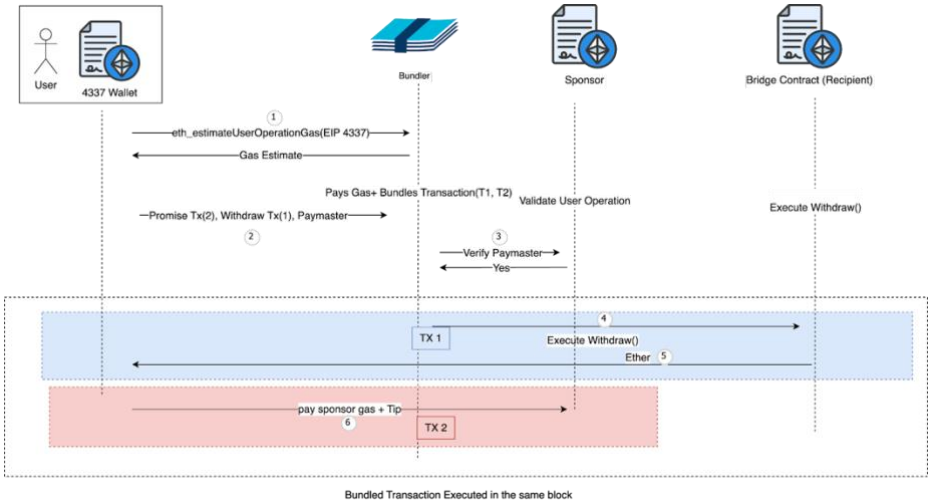

Figure 3. Gasless withdrawal system using ERC 4337

Workflow:

1. The initial process is similar to the previous approach. We get the Layer 2 deposit transaction hash, which we use to compute the withdrawal proofs and payload. In ERC 4337 Compliant transactions, a new method called eth_estimateUserOperationGas() is used to calculate the gas required for the transaction execution.

2. A payload is built based on the cost of executing the withdrawal transaction, stating that the user will refund the sponsor for the gas cost + an incentive (10% usually) as the value. We will call this TX 2, and let the user sign it. At this stage, we also include the Paymaster (sponsor) in the ERC 4337 TX1.

3. We bundle the transactions and send them to the network. It goes through an 'Entry Point Contract,' which acts as the coordinator for managing user operations. After that, the Entry Point Contract interacts with the Paymaster.

   Here, the Paymaster verifies the transaction (TX 1) and ensures that the transaction is valid. 'Valid' can mean anything from 'is the address blacklisted?' to 'does this withdrawal contain enough ETH to make this worthwhile?'

4. After passing all the checks by the Entry Point Contract and the Paymaster, the withdrawal is executed.

5. The ether is successfully withdrawn to the user's account.

6. The promise transaction (TX 2) is executed after the withdrawal is complete in the same block by the bundler. This transaction will refund the sponsor.

2.2.1 High Level Overview

Gas Estimation (Off Chain):
The user performs a gas estimation by calling eth_estimateUserOperationGas(EIP 4337) using Bundler RPC to get a gas estimate of the withdrawal.

Transaction Preparation (Off Chain to On Chain):
The User signs the Promise Tx 2 based on Gas Estimate and the withdrawal TX1. The bundler is responsible for bundling two transactions (T1 and T2). The bundler also includes the paymaster, who sponsors the transaction fees.

Validation (On Chain):
The Tx 1 transaction is sent to the on-chain bundler / entry-point contract, which validates the user operation. The entry point contract checks if the UserOperations are legitimate, the paymaster verifies if they want to sponsor the transaction and the bundler bundles the transactions.

Execution of Withdrawal (On Chain):
Once the parities verify the transaction, the bundled transaction (TX 1) is sent to the Bridge Contract (Recipient) to execute the withdrawal function.

Ether Transfer (On Chain):
The Bridge Contract handles the withdrawal of Ether.

Sponsorship and Tips (Off Chain):
After Transaction 1, Transaction 2, which involves paying sponsor gas and tip flow, is executed.

Bundled Transaction Execution (On Chain):
The bundled transactions are executed in the same block on the blockchain, in the same order.

Assumptions:

- The degree of acceptance and adoption by the broader Ethereum community plays a critical role. For ERC-4337 to become mainstream, it needs not only to be technically sound but also widely recognized and utilized by developers and users in the ecosystem. This is one of the biggest drawbacks of this approach.

- Like the eth_estimateGas method, similarly eth_estimateUserOperationGas() method does not return the exact amount of ETH consumed in the transaction. It is generally impossible to guess the exact amount of gas a transaction will use until its execution.

- Paymaster and builders should have enough incentive to execute the transaction.

- The 4337 Proposal introduces additional complexity to Dapps, as it is relatively new and the associated developer tools and guides are still in the early stages of development.

- Users should avoid keeping L1 ETH in their wallets as a malicious sponsor could potentially steal these funds without processing their withdrawal requests, just by simply executing the User promise (TX2) without executing the withdrawal.

- System should use 'eth_sendbundle' to send both transactions. Otherwise, they risk losing money because of the user's transaction failure or front-running attacks. Alternatively, they can also rely on other builders and bundlers in the network to bundle up their 4337 Tx and payback transaction.

- The amount of ETH that is going to be withdrawn for user on L1 after the execution of withdraw transaction must be greater than or equal to (number of gas used + X% tip) * (block.basefee + 5 Gwei) at the time of execution of transaction. Otherwise sponsoring will fail.

- Users are required to add a 'X' percent tip over the sponsor's execution costs as a reward for handling the transaction.

- Block.basefee can fluctuate between the time a user signed the transaction and when the transaction will be confirmed by a miner.

2.3 Triggered Withdrawals

For this approach, we can change how the withdrawals work in the bridge contract, have one withdrawal trigger multiple other withdrawals, like storing the pending withdrawals in a list and executing them all when one user sends a withdrawal transaction.

The changes to implement this would be to rewrite the bridge, by storing all pending withdrawals in a list, and whenever a user withdraws, clear off the last 'x' pending

withdrawals. An Incentive for the user to trigger these withdrawals should be designed as gas for this method will be a lot higher.

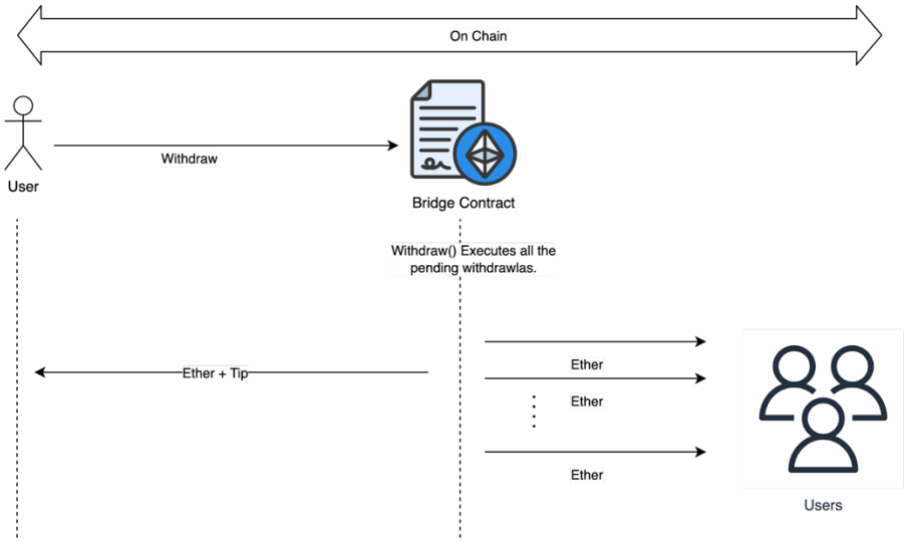

Figure 4. Triggered Withdrawal System design.

There are multiple cons to this approach,

1. *Exceeding Gas Block Limit:* This approach can potentially lead to exceeding the gas block limit if there are too many pending withdrawals. When transactions accumulate without being processed, it increases the demand on the block's capacity. Hence the maximum triggered withdrawals have to be set accordingly.

2. *Security Implications:* Modifying the bridge contract can introduce new vulnerabilities or unintended loopholes. Every change needs thorough testing and auditing to prevent security breaches, which can be costly and time-consuming.

3. *Lack of Incentive Mechanisms in The L1 Outbox:* The L1 Outbox, lacks mechanisms to incentivize gas refunds. Offering such refunds could become prohibitively expensive, potentially leading to inefficiencies where the cost of transactions becomes a barrier to their execution. This lack of incentives would reduce the attractiveness for the initial withdrawer, so an incentive mechanism should be set in place.

4. *High Ether Balance Requirement for Initial Withdrawers*: The initial parties attempting to withdraw might need to maintain a high balance of Ether to cover the gas fees, especially during times of network congestion. This requirement can pose a significant barrier to entry for users with limited resources and can discourage participation. Moreover, it can lead to scenarios where only the financially well-off or large institutions can afford to initiate transactions, thereby limiting the decentralization and widespread usage of the blockchain.

2.4 Sponsored Withdrawals

A user submits a transaction hash, and a signed promise to cover gas costs plus a x% incentive. Using this, a third party EOA / Contract triggers two transactions: a withdrawal and the promised refund. Both are executed simultaneously using flashbots to ensure they process together, transferring Ether to the user and refunding the sponsor.

Users initiating withdrawals from Layer 2 need not be aware of the sponsorship. The withdrawal process is open to execution by any party via the outbox, and the ETH will be directed to the L1 address specified in the withdrawal.

Under this model, a third-party must facilitate the withdrawal. Therefore, the system includes a mechanism to refund gas expenses and provide additional incentives to support this third-party entity.

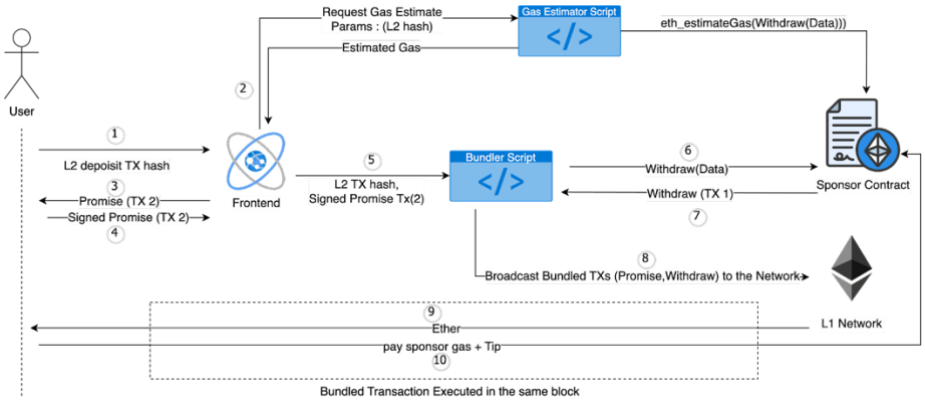

Figure 5. Sponsored withdrawal system design

2.4.1 High Level Overview

1. User submits the Layer 2 deposit transaction hash, which will be used to compute the withdrawal proofs and payload.

2. The hash is used to construct the payload and run a gas estimator script, `ethEstimateGas`, to get the gas required to execute the transaction. Based on the value returned, we calculate the cost by Gas * Current Network gas price.
3. We craft a payload based on the cost of executing the withdrawal transaction, stating that the user will refund the sponsor for the gas cost + an incentive (10% usually) as the value, this will be TX2.
4. User signs the TX 2 on the frontend.
5. Now we have TX 2 (User promise of refund) + L2 Hash.
6. We use the L2 hash to call our Sponsor Contract (Can be a EOA too) to generate our TX 1 (withdrawal TX).
7. TX 1 , which will execute the withdrawal, is generated.
8. TX1 and TX2 are bundled up using flashbots. This will make sure that both the transactions are atomic, either both of them execute or neither of them does + the order of the transactions should be the same and pass them off to the Network.
9. On successful inclusion of the bundle, Ether is sent to the user address.
10. The promise transaction (TX 2) is executed, after the withdrawal is complete in the same block. This transaction will refund the sponsor.

Assumptions:

- Users should avoid keeping L1 ETH in their wallets as a malicious sponsor could potentially steal these funds without processing their withdrawal requests, they could simply execute the User promise (TX2) without executing the withdrawal.

- Sponsor should use eth_sendbundle to send two transactions. Otherwise they risk losing money because of the user’s transaction failure or front-running attacks.

- The amount of ETH that is going to be withdrawn for user on L1 after the execution of withdraw transaction must be greater than or equal to (number of gas used + X% tip) * (block.basefee + 5 Gwei) at the time of execution of transaction. Otherwise sponsoring will fail.

- Users are required to add a X % tip over the sponsor's execution costs as a reward for handling the transaction.

- eth estimateGas() method does not return the exact amount of ETH consumed in the transaction. It is generally impossible to guess the exact amount of gas a transaction will use until its execution.

- Block.basefee can fluctuate between the time a user signed the transaction and when the transaction will be confirmed by a miner, hence it's preferred to set a high base fee.

3 Proposed Approach

Based on the four approaches we analyzed, we will be working on Sponsored Withdrawals (4th Solution).

There are two possible ways to approach it based on how the bridge contract is written. Both of these scenarios need to be executed using `eth_sendbundle()` as an atomic bundle. This can be achieved through Bundlers / Bribing Block builders (flashbots project) [11].

3.1 Scenario 1: msg.sender() must be the User in Bridge Contract

1. *Send ETH to User:* The initial transaction involves transferring Ethereum (ETH) to the user's account.
2. *Withdraw ETH:* The user then withdraws the ETH from the bridge.
3. *User Repays*: Finally, the user repays the ETH to the sponsor who initially sent the ETH.

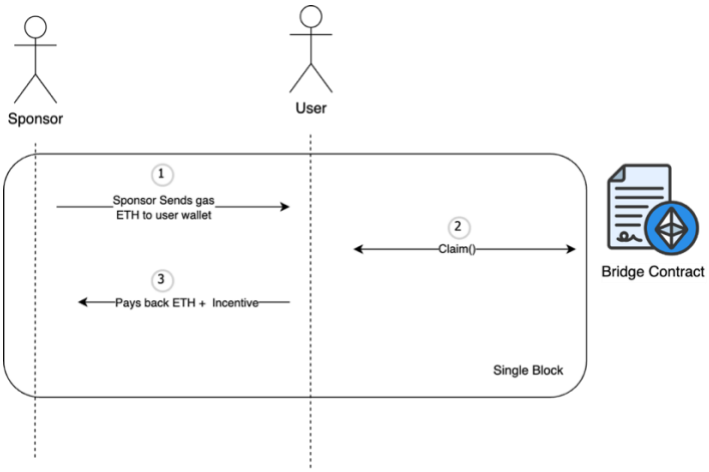

Figure 6. Highlevel flow of a sponsored withdrawal (fig 5.) for contracts which require msg.sender() to be the user

In some contracts, it is essential for the claim() function to be initiated by the user themselves, i.e., msg.sender() should be the user. The solution is just to fund the user with enough gas to cover for the withdrawal and get a promise TX that the user will

repay the sponsored amount. This involves a sequence of three steps: the sponsor transfers payment to the user, the user then claims the funds through the bridge, and finally, repays the sponsor with an added bonus.

It's crucial for these three steps to occur within the same block and to be executed atomically; this means they must either all complete successfully in the specified sequence or not at all. This is important because if one or two of the steps are completed but not the others, it could lead to issues such as the sponsor providing ETH for gas only for the user to abscond with it, or the user withdrawing the funds and fleeing without reimbursing the sponsor.

To achieve this grouped execution, Flashbots can be utilized. Flashbots offer a specialized RPC that includes additional methods beyond those available in ethers.js, which supports the grouping of transactions. These wrappers are utilized by multiple blockbuilders and relayers, so we don't have to rely on a single party.

Furthermore, utilizing multiple blockbuilders endpoints can enhance the likelihood of transactions being included in blocks without the need for expensive bribes [11].

Gas information:

This approach requires the execution of three transactions in total:

- Transaction 1 (Tx1): A simple transfer, consuming 21,000 gas units.
- Transaction 2 (Tx2): Withdrawing ETH from a bridge, consuming 120,000 gas units (for Arbitrum Outbox Withdraw).
- Transaction 3 (Tx3): Another simple transfer, consuming 21,000 gas units.

The sponsor is responsible for the costs associated with Tx1 and Tx2, totaling 141,000 units. The user bears the cost of Tx3, which amounts to 21,000 units [4].

Consequently, the total expenditures are:

- Sponsor: 141,000 units.
- User: The user repays 141,000 units plus a 10% tip (14,100 units) and the gas cost for Tx3 (21,000 units), summing to 176,100 units.

Which means that the user pays 46.75% more than if they had performed the transaction by themself.

3.2 Scenario 2: msg.sender() Does Not Need to Be User in Bridge Contract

In some contracts, the msg.sender() doesn’t have to be the user, any 3rd party account can withdraw the funds on behalf of the user.

Consider user Alice has a pending withdrawal on the outbox (for let's say address X) on L1. In these types of contracts (e.g. Arbitrum), it is possible for other addresses (Y) to execute the withdrawal Tx in the outbox on behalf of Alice.

In bridges constructed this way, only two transactions are necessary for sponsored withdrawals: one where the sponsor withdraws the ETH into the user's account, and another where the user compensates the sponsor for the gas used plus an additional reward.

The execution of this approach is similar to Scenario 1, we bundle up all the transactions and broadcast it as a bundle to the relayers.

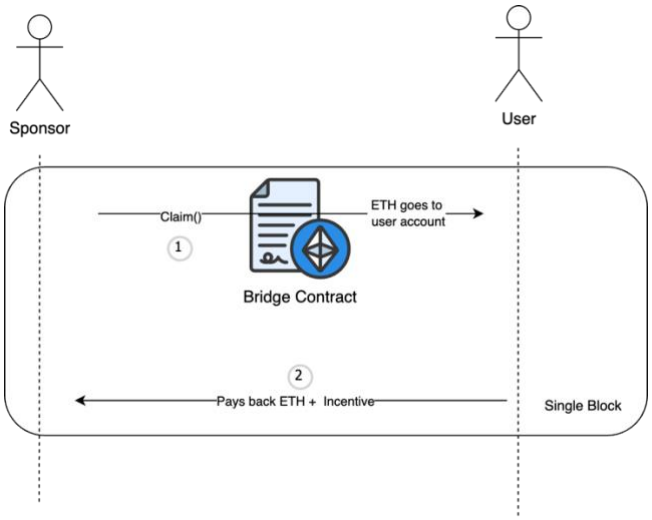

Figure 7. Highlevel flow of a sponsored withdrawal (fig. 5) for contracts which doesn't require msg.sender() to be the user

1. Sponsor Withdraws ETH to User Account: In this scenario, the sponsor directly withdraws ETH to the user's account, eliminating the need for the user to withdraw from the bridge.

2. User Repays ETH to Sponsor: The user then repays the ETH to the sponsor.

Gas information:
This approach requires the execution of two transactions in total:

- Transaction 1 (Tx1): Withdrawing ETH from a bridge, consuming 120,000 gas units (average for Arbitrum Outbox Withdraw).
- Transaction 2 (Tx2): A simple transfer, consuming 21,000 gas units.

The Sponsor would have to pay for TX1, which would be 120,000 Units. The User would have to pay gas for the TX2, which be 21,000 Units

Consequently, the total expenditures are:

- Sponsor: 120,000 units.
- User: Repays 120,000 units, adds a 10% tip (12,000 units), and pays the gas for Tx2 (21,000 units), totaling 153,000 units.

Which means that the user pays 27.5% more than if they had performed the transaction by themself.

3.3 Implementation of Scenario 2

This scenario is implemented on ARB Sepolia L2 -> ETH Sepolia Outbox.

We require the L2 Tx hash of the withdrawal. This hash is used to get the L2 to L1 Message, which in turn is used to construct the payload and proofs for the Claim Transaction on L1 Mainnet.

1. User submits the Layer 2 deposit transaction hash, which will be used to compute the withdrawal proofs and payload.
2. The hash is used to construct the payload and run a gas estimator script, `ethEstimateGas`, which gives the gas required to execute the transaction. Based on the value returned, we calculate the cost by Gas * Current Network gas price.
3. A payload is built based on the cost of executing the withdrawal transaction, stating that the user will refund the sponsor for the gas cost + an incentive (10% usually) as the value, we will call this TX 2.
4. User signs the TX 2 on the frontend.
5. Now we have TX 2 (User promise of refund) + L2 Hash,
6. The L2 hash is used to call the Sponsor Contract (Can be a EOA too) to generate our TX 1 (withdrawal TX)
7. TX 1, which will execute the withdrawal, is generated.
8. We bundle them up using flashbots. This will make sure that both the transactions are atomic, either both of them execute or neither of them does + the order of the transactions should be the same and pass them off to the Network.
9. On successful inclusion of the bundle, Ether is sent to the user address.
10. The promise transaction (TX 2) is executed, after the withdrawal is complete in the same block. This transaction will refund the sponsor.

4 Evaluation Framework

We will now evaluate all the four different approaches that we have analyzed in the paper.

Criteria for Evaluation:

The evaluation will focus on the following key aspects:

1. Gasless Withdrawals means that the protocol allows withdrawals to work without gas (Native L1 token) in the withdrawer's account.

2. ERC 20 as Gas, as the name suggests, indicates whether the standard allows ERC20 tokens to be used for gas payments, instead of Ethereum's native currency, ETH.

3. TX Timing indicates whether the standard supports the ability to schedule transactions for future execution.

4. Non-reliance on a 3rd party indicates whether the standard can be implemented without reliance on a third party, i.e Offchain - Relayers.

5. No Changes to User onboarding Infra: This feature suggests that the standard does not require changes to the infrastructure that supports user onboarding, i.e., Wallet Providers, Dapps.

Table 1. Comparing all the approaches and evaluating them on the said criteria. Full dot satisfies the property fully, Partial satisfies the property partially and No dot means the property is not satisfied. we aim to have full dots on all columns.

	Gasless Withdrawls	ERC20 as GAS	Tx Timing	No Smart Contract Changes	Non reliance on 3rd party	No Changes to user onboarding Infra
ERC 2771	●		●			●
ERC 4337	●	●	●	●	●	
Triggered withdrawals	◑				●	●
Sponsored withdrawls	●		●	●		●

Gasless Withdrawals:

- ERC 2771, ERC 4337, and Sponsored Withdrawals: These protocols allow for gasless function calls using a sponsor that covers the transaction fees.

Essentially, the user executes transactions without paying gas fees directly, instead, the paymaster pays on their behalf.
- Triggered Withdrawals: Unlike the other protocols, this method requires one user to pay the gas fees to initiate all pending withdrawals. This initial cost can be refunded to the user based on certain incentives. This system only partially fits the "gasless" model, as it involves an upfront cost that is later reimbursed.

ERC 20 Tokens as Gas:
- ERC 4337: This standard allows ERC20 tokens to be used instead of Ether for transaction gas fees. It provides greater flexibility in how transaction costs are managed and paid, broadening the usability of ERC20 tokens beyond just value transfer or governance.

Transaction Timing:
- ERC 2771 and Sponsored withdrawal Transactions: These can be pre-scheduled during the relayer phase, allowing transactions to be executed at a predetermined time without direct sender intervention [6].
- ERC 4337: Offers direct control over the wallet, giving users or applications the ability to manage when and how transactions are sent.
- Triggered Withdrawals: The scheduling of these withdrawals depends on the initiator of the withdrawal, making the timing less predictable and dependent on external triggers.

Smart Contract Modifications:
- ERC 2771: Requires existing smart contracts to migrate from using msg.sender to _msgSender() to properly decode the address of the transaction signer. This change helps distinguish between the entity sending the transaction and the one paying for the gas.
- Triggered Withdrawals: Modifications are needed within the bridge contract to accommodate the withdrawal functions, aligning them with the new system that supports gas payment reimbursement.

Dependence on Third Parties:
- ERC 2771 and Sponsored withdrawal: Both rely on offchain components—relayers and scripts, respectively—to manage or fund transactions. This reliance introduces external dependencies and potential points of failure but allows for more sophisticated transaction management.

Changes to User Onboarding Infrastructure:
- ERC 4337 Integration with Wallets Like Metamask: For smooth operation, popular wallet extensions need to support integrations with smart wallet accounts under ERC 4337. Users must be educated on the workings of smart wallet accounts, which differ from traditional wallet setups, emphasizing the need for understanding new functionalities like using ERC20 tokens for gas.

5 Conclusion

In conclusion, we have addressed the significant challenge of facilitating gasless transactions between Ethereum's Layer 1 and Layer 2 networks. By exploring four innovative strategies Meta-Transactions with incentives for gas relayers, ERC-4337 Account Abstraction, Triggered Withdrawals, and Sponsored Withdrawals. we have provided a comprehensive analysis of possible solutions to enhance user experience and reduce dependency on Layer 1 assets for gas fees. Our focus on the Sponsored Withdrawals approach, in particular, demonstrates its potential to streamline the withdrawal process in a user-friendly manner while ensuring transaction security for sponsors through atomic execution.

The proposed solutions are not merely theoretical, they offer practical frameworks that can be implemented to mitigate the complexities associated with Layer 2 withdrawals. Additionally, the principles and methods used can be applied to other problems and contexts, beyond just provisioning gasless bridging between layers.

Further research and development are encouraged to refine these methods, address any emerging challenges, and explore new possibilities in the evolving landscape of blockchain technology. The future of Ethereum’s scalability solutions looks promising, with these innovative approaches leading the charge toward more accessible and efficient blockchain interactions.

References

1. Moosavi, M., Salehi, M., Goldman, D., and Clark, J. 2023. Fast and Furious Withdrawals from Optimistic Rollups. In Proceedings of the 5th Conference on Advances in Financial Technologies (AFT '23), Leibniz International Proceedings in Informatics (LIPIcs), Vol. 282. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 22:1-22:17. https://doi.org/10.4230/LIPIcs.AFT.2023.22

2. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S. M., and Felten, E. W. 2018. Arbitrum: Scalable, private smart contracts. In Proceedings of the USENIX Security Symposium. USENIX Association, 1353-1370.

3. McCorry, P., Buckland, C., Yee, B., and Song, D. 2021. SoK: Validating bridges as a scaling solution for blockchains. Technical Report. Cryptology ePrint Archive.

4. Zarir, A. A., Oliva, G. A., Jiang, Z. M., and Hassan, A. E. 2021. Developing Cost-Effective Blockchain-Powered Applications: A Case Study of the Gas Usage of Smart Contract Transactions in the Ethereum Blockchain Platform. ACM Trans. Softw. Eng. Methodol. 30, 3, Article 28 (July 2021), 38 pages. https://doi.org/10.1145/3431726

5. ERC-4337 Account Abstraction. Retrieved April 20, 2024, from https://www.erc4337.io/

6. ERC-2771: Secure Protocol for Native Meta Transactions. Retrieved April 20, 2024, from https://eips.ethereum.org/EIPS/eip-2771

7. Rethink Digital Transactions with Account Abstraction. Visa. Retrieved April 20, 2024, from https://usa.visa.com/solutions/crypto/rethink-digital-transactions-with-account-abstraction.html

8. What are Meta Transactions? Exploring ERC-2771. Moralis. Retrieved April 20, 2024, from https://moralis.io/what-are-meta-transactions-exploring-erc-2771/

9. Account Abstraction (ERC-4337) vs. Meta Transactions (ERC-2771). Alchemy. Retrieved April 20, 2024, from https://www.alchemy.com/overviews/4337-vs-2771

10. Transaction Simulation. Tenderly. Retrieved April 20, 2024, from https://tenderly.co/

11. Flashbots. Retrieved April 20, 2024, from https://www.flashbots.net/

12. Builders list. GitHub. Retrieved April 20, 2024, from https://github.com/blue-searcher/awesome-block-builders

13. Arbitrum One Outbox 4 Contract on ETH (L1) https://etherscan.io/address/0x0b9857ae2d4a3dbe74ffe1d7df045bb7f96e4840