Fast and Furious Withdrawals from Optimistic Rollups

Abstract. Optimistic rollups are in wide use today as an opt-in scalability layer for blockchains like Ethereum. In such systems, Ethereum is referred to as L1 (Layer 1) and the rollup provides an environment called L2, which reduces fees and latency but cannot instantly and trustlessly interact with L1. One practical issue for optimistic rollups is that trustless transfers of tokens and ETH, as well as general messaging, from L2 to L1 is not finalized on L1 until the passing of a dispute period (aka withdrawal window) which is currently 7 days in the two leading optimistic rollups: *Arbitrum* and *Optimism*. In this paper, we explore methods for sidestepping the dispute period when withdrawing ETH from L2 (called an exit), even in the case when it is not possible to directly validate L2. We fork the most-used rollup, *Arbitrum Nitro*, to enable exits to be traded on L1 before they are finalized. We also study the combination of tradeable exits and prediction markets to enable insurance for withdrawals that do not finalize. As a result, anyone (including contracts) on L1 can safely accept withdrawn tokens while the dispute period is open despite having no knowledge of what is happening on L2. Our scheme also allows users to opt-into a fast withdrawal at any time. All fees are set by open market operations.

Keywords: Ethereum · layer 2 · rollups · bridges · prediction markets

1 Introductory Remarks

Ethereum-compatible blockchain environments, called Layer 2s (or L2s) [5], have demonstrated an ability to reduce transaction fees by 99–99.9% while preserving the strong guarantees of integrity and availability in the underlying blockchain. The subject of this paper concerns one subcategory of L2 technology called an optimistic rollup. The website *L2 Beat* attempts to capitalize all tokens of known value across the top 25 L2 projects. It finds that the top two L2s are both optimistic rollups, *Arbitrum* and *Optimism*, which respectively account for 50% and 30% of all L2 value—$4B USD at the time of writing.[1]

We will describe the working details of optimistic rollups later in this paper but here are the main takeaways: currently, rollups are faster and cheaper than Ethereum itself. However, each L2 is essentially an isolated environment that cannot instantly and trustlessly interact with accounts and contracts that are running on either L1 or other L2s. An optimistic rollup project will typically provide a smart contract, called a validating bridge [9], that can trustlessly move

[1] L2 Beat, accessed Oct. 2022.

ETH (and other tokens and even arbitrary messages) between L1 and its own L2. It does value transfers by locking ETH in an L1 contract and minting the equivalent ETH (more precisely, it is a L2 claim on L1 ETH) on L2 and assigning it to the user's L2 address. Later when the user requests a withdrawal, the ETH will be destroyed on L2 and released by the bridge back onto L1 according to who its new owner is on L2 at the time of the request. This requires the rollup to convince the L1 bridge contract of who the current owner of withdrawn ETH is on L2. We provide details later but this process takes time: the bridge has to wait for a period of time called the dispute window. The current default is 7 days in *Arbitrum* and *Optimism*, however the filing of new disputes can extend the window. The bottom line is that users have to wait at least 7 days to draw down ETH from an optimistic rollup.

Contributions. In this paper, we compare several methods—atomic swaps and tradeable exits—for working around this limitation. While we argue workarounds cannot be done generally, some circumstances allow it: namely, when the withdrawn token is liquid, fungible, and available on L1 and the withdrawer is willing to pay a fee to speed up the withdrawal. While these techniques work easily between human participants that have off-chain knowledge, such as the valid state of the L2, it is harder to make them compatible with L1 smart contracts that have no ability to validate the state of L2. We propose a solution using tradeable exits and prediction markets to enable an L1 smart contract to safely accept withdrawn tokens before the dispute period is over. We fork the current version, *Nitro*, of the most popular optimistic rollup, *Arbitrum*, made open source[2] by *Offchain Labs*. We implement our solution and provide measurements. *Arbitrum* is a commercial product with academic origins [8]. Finally, we provide an analysis of how to price exits and prediction market shares.

2 Background

While we describe optimistic rollups as generally as possible, some details and terms are specific to *Arbitrum*.

Inbox. Rollups have emerged as a workable approach to reducing fees and latency for Ethereum-based decentralized applications. In a rollup, transactions to be executed on L2 are recorded in an L1 smart contract called the inbox. Depending on the system, users might submit to the inbox directly, or they might submit to an offchain service, called a sequencer, that will batch together transactions from many users and pay the L1 fees for posting them into the inbox. Transactions recorded in the inbox (as `calldata`) are not executed on Ethereum, instead, they are executed in a separate environment off the Ethereum chain, called L2. This external environment is designed to reduce fees, increase throughput, and decrease latency.

[2] GitHub: Nitro

Outbox. Occasionally (*e.g.,* every 30–60 minutes), validators on L2 will produce a checkpoint of the state of all contracts and accounts in the complete L2 according to the latest transactions and will place this asserted state (called an RBlock) in a contract on L1 called the outbox. Note that anyone with a view of L1 can validate that the sequence of transactions recorded in the inbox produces the asserted RBlock in the outbox. This includes Ethereum itself, but asking it to validate this be equivalent to running the transactions on Ethereum. The key breakthrough is that the assertion will be posted with *evidence* that the RBlock is correct so Ethereum does not have to check completely.

Optimistic vs. zk-rollups. In practice, two main types of evidence are used. In zk-rollups,[3] a succinct computational argument that the assertion is correct is posted and can be checked by Ethereum for far less cost than running all of the transactions. However the proof is expensive to produce. In optimistic rollups, the assertions are backed by a large amount of cryptocurrency acting as a fidelity bond. The correctness of an RBlock can be challenged by anyone on Ethereum and Ethereum itself can decide between two (or more) RBlocks for far less cost than running all of the transactions (by having the challengers isolate the exact point in the execution trace where the RBlocks differ). It will then reallocate the fidelity bonds to whoever made the correct RBlock. If an RBlock is undisputed for a window of time (*e.g.,* 7 days), it is considered final.

Bridge. A final piece of the L2 infrastructure is a bridge, which can move ETH, tokens, NFTs, and even arbitrary messages, between L1 and L2. For now, we limit the discussion to bridging ETH but the ideas extend to other tokens. If Alice has ETH on Ethereum, she can submit her ETH to a bridge smart contract on Ethereum which will lock the ETH inside of it, while generating the same amount of ETH in Alice's account inside the L2 environment. The bridge does not need to be trusted because every bridge operation is already fully determined by the contents of the inbox. Say that Alice transfers this ETH to Bob's address on L2. Bob is now entitled to draw down the ETH from L2 to L1 by submitting a withdrawal request using the same process as any other L2 transaction—*i.e.,* placing the transaction in the inbox on L1, having it executed on L2, and seeing it finalized in an RBlock on L1. Optimistically, the RBlock is undisputed for 7 days and is finalized. Bob can now ask the bridge on L1 to release the ETH to his address by demonstrating his withdrawal (called an exit) is included in the finalized RBlock (*e.g.,* with a Merkle-proof).

2.1 Related Work

Arbitrum is first described at *USENIX Security* [8]. Gudgeon *et al.* provide a systemization of knowledge (SoK) of Layer 2 technology (that largely predates rollups) [5], while McCorry *et al.* provide an SoK that covers rollups and

[3] zk stands for zero-knowledge, a slight misnomer: succinct arguments of knowledge that only need to be complete and sound, not zero-knowledge, are used [10].

validating bridges [9]. Some papers implement research solutions on Arbitrum for improved performance: decentralized order books [11] and secure multiparty computation [2]. The idea of tradeable exits predates our work but is hard to pinpoint a source (our contribution is implementation and adding hedges). Further academic work on optimistic rollups and bridges is nascent—we anticipate it will become an important research area.

Other related topics are atomic swaps and prediction markets. Too many papers propose atomic swap protocols to list here but see Zamyatin *et al.* for an SoK of the area (and a new theoretical result) [14]. Decentralized prediction markets proposals predate Ethereum and include Clark *et al.* [1] and Truthcoin [12]. Early Ethereum projects *Augur* and *Gnosis* began as prediction markets.

3 Proposed Solution

For simplicity, we will describe a fast exit system for withdrawing ETH from L2, however it works for any L1 native fungible token (*e.g.,* ERC20) that is available for exchange on L1. We discuss challenges of fast exits for non-liquid/non-fungible tokens in Section 6.4. Consider an amount of 100 ETH. When this amount is in the user's account on L1, we use the notation 100 ETH_{L1}. When it is in the bridge on L1 and in the user's account on L2, we denote it 100 ETH_{L2}. When the ETH has been withdrawn on L2 and the withdrawal has been asserted in the L1 outbox, but the dispute window is still open, we refer to it as 100 ETH_{XX}. Other transitionary states are possible but not needed for our purposes.

3.1 Design Landscape

Centralized. Consider Alice who has 100 ETH_{L2} and wants (something like) 99.95 ETH_{L1} for it. We describe a set of solutions for Alice. A centralized exchange (*e.g., Coinbase*, *Binance*) can open a market for $\text{ETH}_{\mathsf{L2}}/\text{ETH}_{\mathsf{L1}}$. Alternatively, a bridge might rely on an established set of trustees to relay L2 actions to L1. This is called proof of authority; it is distributed but not decentralized (*i.e.,* not an *open* set of participants).

Hash Time Locked Contracts (HTLCs). Assume Bob has 99.95 ETH_{L1} and is willing to swap with Alice. An atomic swap binds together (i) an L2 transaction moving 100 ETH_{L2} from Alice to Bob and (ii) an L1 transaction moving 99.95 ETH_{L1} from Bob to Alice. Either both execute or both fail. HTLC is a blockchain-friendly atomic swap protocol. Its main drawback is that it also has a time window where Alice (assuming she is the first mover in the protocol) must wait on Bob, who might abort causing Alice's ETH_{L2} to be locked up while waiting (called the griefing problem), or might watch price movements before deciding to act (called free option problem). Bob needs to monitor both chains so he cannot be an autonomous smart contract. HTLCs work between two L2s.

Type	*Example*	No trusted third party	Within an L1 transaction	Within an L2 rollup	No griefing	No free option	Opt-in anytime	L2-to-L2
Normal Exit (baseline)	Arbitrum	•			•	•		
Centralized	Coinbase		•	•	•	•		•
HTLC Swaps	Celer	•	○	•				•
Conditional Transfers	StarkEx	•	•	•				
Bridge Tokens	Hop	○	•	•		•		•
Tradeable Exits	This Work	•	$\sim$	•	•	•	•	
Hedged Tradeable Exits	This Work	•	$\sim$	•	•	•	•	

Table 1. Comparing alternatives for fast withdrawals from optimistic rollups for liquid and fungible tokens where • satisfies the property fully, ○ partially satisfies the property, and no dot means the property is not satisfied. For our work, $\sim$ means we propose how to fully achieve the property but do not by default (see caveats in Section 6.1).

Conditional Transfers. In this contract-based atomic swap, Alice uses an L1 contract (called a registry) to record a request for payment of 99.95 ETH$_{\mathsf{L1}}$ (from anyone) with ID number 1337. Off-chain, she provides Bob with a signed L2 transaction (called a conditional transfer or CT) that (slow) withdraws 100 ETH$_{\mathsf{L2}}$ to Bob *if and only if* payment 1337 has been received on the L1 registry at the time the CT is added to the inbox; otherwise the CT reverts. The CT also expires (always reverts) after one hour. CTs have similar properties to an atomic swap except Alice gets paid on L1 before anything happens on L2. The registry check cannot work quickly between different L2s.

Bridge Token. A third party creates a bridge on L2 that converts ETH$_{\mathsf{L2}}$ into a custom ticket that serves as a claim for ETH$_{\mathsf{L2}}$ [13]. It creates an equivalent bridge on L1. Alice burns 100 tickets on L2. Bob notices and generates a claim for ETH$_{\mathsf{L1}}$ on L1 (assuming sufficient supply) in the equivalent L1 bridge. To prevent Bob from maliciously minting tokens on L1 that were not burned on L2, he must post a fidelity bond of equal or greater value (otherwise Bob is trusted to not cause insolvency). After the 7 day dispute period, the L1 bridge can verify Bob's actions are consistent with L2 and release his fidelity bond. Note that when you collapse this functionality, it is equivalent to Bob buying ETH$_{\mathsf{XX}}$ from Alice for ETH$_{\mathsf{L1}}$ and receiving his ETH$_{\mathsf{L1}}$ back 7 days later. The extra infrastructure is necessary because today native bridges do not support tradeable exits. As in atomic swaps, Bob can fail to act (griefing) which is worst in this case if Alice cannot 'unburn' her tokens, but there is no free option because Bob is a relay and not a recipient.

Comparative Evaluation. These solutions are compared with (hedged) tradeable exits—described next in the paper—in Table 1.

3.2 Tradeable Exits

Alice wants to withdraw 100 $\mathrm{ETH}_{\mathsf{L2}}$. Bob has 99.95 $\mathrm{ETH}_{\mathsf{L1}}$ that will not use until after the dispute window. Bob also runs an L2 validator so he is assured that if Alice withdraws, it is valid and will eventually finalize. With a tradeable exit, the outbox allows Alice to change the recipient of her withdraw from herself to Bob. Thus Alice swaps her pending exit of 100 $\mathrm{ETH}_{\mathsf{L1}}$ (which we call 100 $\mathrm{ETH}_{\mathsf{XX}}$) for Bob's 99.95 $\mathrm{ETH}_{\mathsf{L1}}$ on L1 (note we discuss the actual difference in price in Section 5). After 7 days, Bob can ask the bridge to transfer the $\mathrm{ETH}_{\mathsf{L1}}$ to his address, and the bridge checks the outbox to validate that Bob's address is the current owner of the exit.

In our forked bridge, Alice can transfer any of her exits that are in an RBlock (*i.e.,* an asserted L2 state update registered in the outbox). Technically Bob can check the validity of the withdrawal as soon as it is in the inbox, and not wait 30-60 minutes for an RBlock. However for implementation reasons, it is easier to track an exit based on its place (*i.e.,* Merkle path) in an RBlock, rather than its place in the inbox. When we say a withdrawal is 'fast,' we mean 30-60 minutes (*i.e.,* one L2 rollup).

Like bridge tokens, tradeable exits can be approximated by a third party L1 contracts that do not modify the rollup. In this scenario, a two-stage withdrawal would occur. The user would specify the contract as the recipient of the exit, and the contract would specify the user as the recipient (initially). The user could then transfer ownership to a new account within the contract. Given this option, why modify the bridge/outbox of the rollup? We have two main arguments: (1) it is more efficient for the user to have the functionality natively in the bridge/outbox, which they have to interact with anyways; and (2) a user who initially request a 'normal' withdrawal cannot change their mind and opt-in to a fast withdrawal—it is too late. A tradeable exit can bail out a user who withdrawals without realizing there is a 7 day dispute window (anecdotally, this is a common concern on support channels for optimistic rollups). It also lets a user who is aware decide if and when to expedite a withdrawal.

3.3 Hedged Tradeable Exits

One remaining issue with tradeable exits is how specialized Bob is: he must have liquidity in $\mathrm{ETH}_{\mathsf{L1}}$, be an active trader who knows how to price futures, and be an L2 validator. While we can expect blockchain participants with each specialization, it is a lot to assume of a single entity. The goal of this subsection is to split Bob into two distinct participants: one that has $\mathrm{ETH}_{\mathsf{L1}}$ liquidity but does not know about L2 (Carol) and one that knows about L2 but is not necessarily an active trader on L1 (David). The main impact of this change is that Carol can be an autonomous L1 smart contract.

Recall that Alice wants ETH_{L1} quickly in order to do something on L1 with it; Carol can be that destination contract. The primary risk for Carol accepting ETH_{XX} as if it were ETH_{L1} is that the RBlock containing the ETH_{XX} withdrawal fails and the exit is worthless. If Alice can obtain insurance for the ETH_{XX} that can be verified via L1, then Carol's risk is hedged and she could accept ETH_{XX}. The insurance could take different forms but we propose using a prediction market.

Prediction markets. A decentralized prediction market is an autonomous (*e.g.,* vending machine-esque) third party contract. Since we are insuring L1 ETH_{XX}, we need to run the market on L1 (despite the fact that it would be cheaper and faster on L2). Consider a simple market structure based on [1]. A user can request that a new market is created for a given RBlock. The market checks the outbox for the RBlock and its current status (which must be pending). Once opened, any user can submit 1 ETH_{L1} (for example, the actual amount would be smaller but harder to read) and receive two 'shares': one that is a bet that the RBlock will finalize, called $\text{FINAL}_{\mathsf{PM}}$, and one that is a bet that the RBlock will fail, called $\text{FAIL}_{\mathsf{PM}}$. These shares can be traded on any platform. At any time while the prediction market is open, any user can redeem 1 $\text{FINAL}_{\mathsf{PM}}$ and 1 $\text{FAIL}_{\mathsf{PM}}$ for 1 ETH_{L1}. Once the dispute period is over, any user can request that the market close. The market checks the rollup's outbox for the status of the RBlock—since both contacts are on L1, this can be done directly without oracles or governance. If the RBlock finalizes, it offers 1 ETH_{L1} for any 1 $\text{FINAL}_{\mathsf{PM}}$ (and conversely if it fails). The market always has enough ETH_{L1} to fully settle all outstanding shares.

It is argued in the prediction market literature [1] that (i) the price of one share matches the probability (according to the collective wisdom of the market) that its winning condition will occur, and (ii) the price of 1 $\text{FINAL}_{\mathsf{PM}}$ and 1 $\text{FAIL}_{\mathsf{PM}}$ will sum up to 1 ETH_{L1}. For example, if $\text{FAIL}_{\mathsf{PM}}$ trades for 0.001 ETH_{L1}, then (i) the market believes the RBlock will fail with probability of 0.1% and (ii) $\text{FINAL}_{\mathsf{PM}}$ will trade for 0.999 ETH_{L1}. These arguments do not assume market friction: if the gas cost for redeeming shares is D (for delivery cost), both share prices will incorporate D (see Section 5). Lastly, prediction markets are flexible and traders can enter and exit positions at any time—profiting when they correctly identify over- or under-valued forecasts. This is in contrast to an insurance-esque arrangement where the insurer is committed to hold their position until completion of the arrangement.

Hedging exits. Given a prediction market, Alice can hedge 100 ETH_{XX} by obtaining 100 $\text{FAIL}_{\mathsf{PM}}$ as insurance. Any autonomous L1 contract (Carol) should be willing to accept a portfolio of 100 ETH_{XX} and 100 $\text{FAIL}_{\mathsf{PM}}$ as a guaranteed delivery of 1 ETH_{L1} after the dispute period, even if Carol cannot validate the state of L2.

Perhaps surprisingly, this result collapses when withdrawing ETH_{L2}—consider Path 1 through the protocol. Alice withdraws 100 ETH_{L2} from L2 and obtains 100 ETH_{XX}. Bob creates 100 $\text{FAIL}_{\mathsf{PM}}$ and 100 $\text{FINAL}_{\mathsf{PM}}$ for a cost of 100 ETH_{L1}.

Alice buys 100 $\text{FAIL}_{\mathsf{PM}}$ from Bob for a small fee. Alice gives Carol 100 ETH_{XX} and 100 $\text{FAIL}_{\mathsf{PM}}$ and is credited as if she deposited 100 ETH_{L1}. In seven days, Bob gets 100 ETH_{L1} for his 100 $\text{FINAL}_{\mathsf{PM}}$ and Carol gets 100 ETH_{L1} for her 100 ETH_{XX}. If the RBlock fails, Bob has 0 ETH_{L1} and Carol has 100 ETH_{L1} from the 100 $\text{FAIL}_{\mathsf{PM}}$. In both cases, Alice has a balance of 100 ETH_{L1} with Carol.

In path 2, Alice withdraws 100 ETH_{L2} from L2 and obtains 100 ETH_{XX}. Alice sells 100 ETH_{XX} to Bob for 100 ETH_{L1}. Alice gives Carol 100 ETH_{L1} and is credited with a balance of 100 ETH_{L1}. In 7 days, Bob gets 100 ETH_{L1} for his 100 ETH_{XX} and Carol has 100 ETH_{L1}. If the RBlock fails, Bob has 0 ETH_{L1}, Carol has 100 ETH_{L1}, and Alice has a balance of 100 ETH_{L1} with Carol.

Modulo differing gas costs and market transaction fees, paths 1 and 2 are equivalent. Path 2 does not use a prediction market at all, it only uses basic tradeable exits. Given this, do prediction markets add nothing to tradeable exits? We argue prediction markets still have value for a few reasons. (1) Speculators will also participate in the prediction market which gives Alice a chance for a fast exit even without Bob (an L2 validator). (2) If Alice withdraws a token other than ETH, the prediction market should still be setup to payout in ETH (otherwise you end up with 50 separate prediction markets for the 50 different kinds of tokens in any given RBlock). In this case, Alice can obtain $\text{FAIL}_{\mathsf{PM}}$ when Bob has no liquidity or interest in the token she is withdrawing (however Carol needs to incorperate an exchange rate risk when accepting an exit in one token and the insurance in ETH). (3) The PM can also help with NFTs and other non-liquid tokens (see Section 6.4).

Three of the most common types of traders are utility traders, speculators, and dealers [6]. With a prediction market, Alice is a utility trader and Bob is a dealer. However, there might exist speculators who want to participate in the market because they have forecasts about rollup technology, a given RBlock, the potential for software errors in the rollup or in the validator software, *etc.* Executives of rollup companies could receive bonuses in $\text{FINAL}_{\mathsf{PM}}$. Quick validators might profit from noticing an invalid RBlock with $\text{FAIL}_{\mathsf{PM}}$ or they might be betting on an implementation bug or weeklong censorship of the network. Speculators add liquidity to the prediction market which reduces transactional fees for Alice. However, speculation also brings externalities to the rollup system where the side-bets on an RBlock could exceed the staking requirements for posting an RBlock, breaking the crypo-economic arguments for the rollup. In reality, these externalities can never be prevented in any decentralized incentive-based system [3].

4 Implementation and Performance Measurements

We run *Arbitrum Nitro* test-net locally and use Hardhat [4] for our experiments. We obtain our performance metrics using TypeScripts.

4.1 Tradeable Exits

Trading the exit directly through the bridge/outbox. We fork the *Arbitrum Nitro* outbox to add native support for tradeable exits. The modified outbox is open source, written in 294 lines (SLOC) of Solidity, and a bytecode of 6,212 bytes (increased by 1,197 bytes). The solidity code and Hardhat scripts are available in a GitHub repository.[4] Our modifications include:

- Adding the `transferSpender()` function which allows the exit owner to transfer the exit to any L1 address even though the dispute period is not passed.
- Adding the `isTransferred()` mapping which stores key-value pairs efficiently. The key of the mapping is the exit number and the value is a boolean.
- Adding the `transferredToAddress` mapping which stores key-value pairs efficiently. The key of the mapping is the exit number and the value is the current owner of the exit.
- Modifying the `executeTransactionImpl()` function. Once the dispute period is passed and the withdrawal transaction is confirmed, anyone can call the `executeTransaction()` function from the outbox (which internally calls the `executeTransactionImpl()`) and release the funds to the account that was specified by the user 7 days earlier in the L2 withdrawal request. With our modifications, this function is now enabled to release the requested funds to the current owner of the exit.

To execute the `transferSpender()` function; Alice (who has initiated a withdrawal for 100 $\mathrm{ETH}_{\mathsf{L2}}$) has to provide variables related to her exit (*e.g.,* exit number), which she can query using the Arbitrum SDK[5], as well as the L1 address she wants to transfer her exit to. The `transferSpender()` function then checks (1) if the exit is already spent, (2) it is already transferred, and (3) the exit is actually a leaf in any unconfirmed RBlock. If the exit has been transferred, the `msg.sender` is cross-checked against the current owner of the exit (recall exit owners are tracked in the `transferredToAddress` mapping added to the outbox). Once these tests are successfully passed, the `transferSpender()` function updates the exit owner by changing the address in the `transferredToAddress` mapping. This costs 85,945 Gwei in L1 gas. Note that the first transfer always costs more as the user has to pay for initializing the `transferredToAddress` mapping. `transferSpender()` costs 48,810 and 48,798 Gwei in L1 gas for the second and third transfer respectively. The `gasUed` for executing the new `executeTransactionImpl()` function is 91,418 Gwei in L1 gas.

Trading the exit through an L1 market. We also implement and deploy an L1 market that allows users to trade their exits on L1 even though the dispute window is not passed (see Section 6.3 for why *Uniswap* is not appropriate). In addition, we add a new function to the *Arbitrum Nitro* outbox, the `checkExitOwner()`, which

[4] GitHub: link is removed for anonymity.
[5] A typescript library for client-side interactions with Arbitrum.

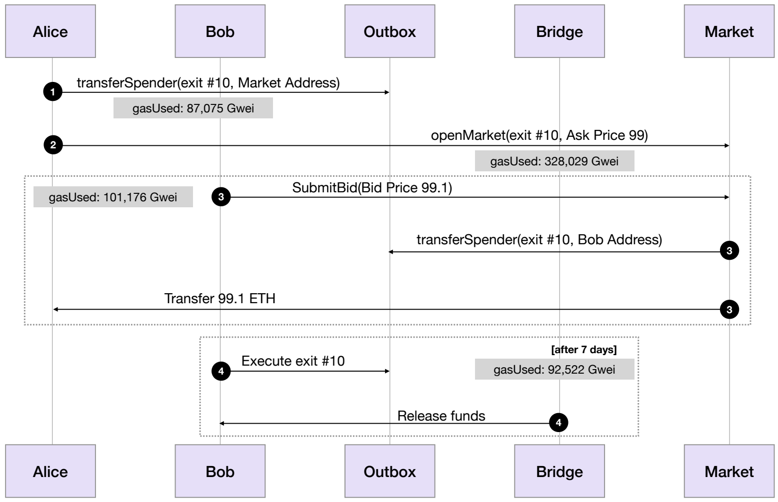

Fig. 1. Overview of trading the exit through an `L1` market.

returns the current owner of the exit. Figure 1 illustrates an overview of participant interactions and related gas costs. To start trading, Alice needs to lock her exit up in the market by calling the `transferSpender()` function from the outbox. Next, she can open a market on this exit by calling the `openMarket()` from the market contract and providing the ask price. The market checks if Alice has locked her exit (by calling the `checkExitOwner()` from the outbox) and only in that case a listing is created on this exit. The market would be open until a trade occurs or Alice calls the `closeMarket()` on her exit. Bob, who is willing to buy Alice's exit, calls the payable `submitBid()` function from the market contract. If the `msg.value` is equal or greater than Alice's ask price, the trade occurs; (1) the market calls the `transferSpender()` from the outbox providing Bob's address. Note that market can only do that since it is the current owner of the exit being traded, and (2) the `msg.value` is transferred to Alice.

The market and modified outbox are open source and written in 125 and 294 lines (SLOC) of Solidity respectively. The solidity code for these contracts in addition to the Hardhat scripts are available in a GitHub repository.[6] Once deployed, the bytecode of the market and outbox is 5,772 and 6,264 bytes respectively.

4.2 Prediction Market

As described in Section 3.3, a prediction market can be used to hedge the exit. We do not implement this as one can use an existing decentralized prediction market (*e.g., Augur* or *Gnosis*). However, we further modify *Arbitrum Nitro*

[6] GitHub: link is removed for anonymity.

to make it friendly to a prediction market that wants to learn the status of an RBlock (pending, confirmed). More specifically, we modify the *Arbitrum Nitro* outbox and RollupCore smart contracts, modifications include:

- Adding the `assertionAtState` mapping to the outbox which stores key-value pairs efficiently. The key of the mapping is the exit number and the value is the user-defined data type `state` that restricts the variable to have only one of the `pending and confirmed` predefined values.
- Adding the `markAsPending` function to the outbox which accepts an RBlock and marks it as pending in the `assertionAtState` mapping.
- Adding the `markAsConfirmed` function to the outbox which accepts an RBlock and marks it as confirmed in the `assertionAtState` mapping.
- Modifying the `createNewNode()` function in the RollupCore contract. To propose an RBlock, the validator acts through the RollupCore contract by calling a `createNewNode()` function. We modify this function to call the `markAsPending()` from the outbox which marks the RBlock as pending.
- Modifying the `confirmNode()` function in the RollupCore contract. Once an RBlock is confirmed, the validator acts through the RollupCore contract via `confirmNode` to move the now confirmed RBlock to the outbox. We modify this function to call the `markAsConfirmed()` from the outbox which marks the RBlock as confirmed.

The modified outbox and RollupCore are open source and written in 297 and 560 lines (SLOC) of Solidity respectively. The solidity code for these contracts in addition to the Hardhat scripts are available in a GitHub repository.[7] Once deployed, the bytecode of the outbox and RollupCore is 6,434 and 3,099 bytes respectively.

5 Pricing

Pricing ETH_{XX}. Consider how much you would pay for 100 ETH_{XX} (finalized in 7 days = 168 hours) in ETH_{L1} today. Since ETH_{XX} is less flexible than ETH_{L1}, it is likely that you do not prefer it to ETH_{L1}, so our intuition is that it should be priced less (*e.g.,* 100 ETH_{XX} = 99 ETH_{L1}). However, our solution works for any pricing and we can even contrive corner cases where ETH_{XX} might be worth more than ETH_{L1} by understanding the factors underlying the price.

In traditional finance [7], forward contracts (and futures, which are standardized, exchange traded forwards) are very similar to ETH_{XX} in that they price today the delivery of an asset or commodity at some future date. One key difference is that with a forward contract, the price is decided today but the actual money is exchanged for the asset at delivery time. When ETH_{XX} is sold for ETH_{L1}, both price determination and the exchange happen today, while the delivery of ETH_{L1} for ETH_{XX} happens in the future. The consequence is

[7] GitHub: link is removed for anonymity.

that we can adapt pricing equations for forwards/futures, however, the signs (positive/negative) of certain terms need to be inverted.

We review the factors [7] that determine the price of a forward contract (F_0) and translate what they mean for $\mathrm{ETH}_{\mathsf{XX}}$:

- *Spot price of ETH_{L1} (S_0):* the price today of what will be delivered in the future. $\mathrm{ETH}_{\mathsf{XX}}$ is the future delivery of $\mathrm{ETH}_{\mathsf{L1}}$, which is by definition worth 100 $\mathrm{ETH}_{\mathsf{L1}}$ today.
- *Settlement time (Δt):* the time until the exit can be traded for $\mathrm{ETH}_{\mathsf{L1}}$. In *Arbitrum*, the time depends on whether disputes happen. We simplify by assuming Δt is always 7 days (168 hours) from the assertion time. A known fact about forwards is that F_0 and S_0 converge as Δt approaches 0.
- *Storage cost (U):* most relevant for commodities, receiving delivery of a commodity at a future date relieves the buyer of paying to store it in the short-term. Securing $\mathrm{ETH}_{\mathsf{XX}}$ and securing $\mathrm{ETH}_{\mathsf{L1}}$ is identical in normal circumstances, so not having to take possession of $\mathrm{ETH}_{\mathsf{L1}}$ for Δt time does not reduce costs for a $\mathrm{ETH}_{\mathsf{XX}}$ holder.
- *Delivery cost (D):* the cost of delivery of the asset, which in our case will encompass gas costs. Exchanging $\mathrm{ETH}_{\mathsf{L1}}$ for $\mathrm{ETH}_{\mathsf{XX}}$ requires a transaction fee and also creates a future transaction fee to process the exit (comparable in cost to purchasing a token from an automated market maker). An $\mathrm{ETH}_{\mathsf{L1}}$ seller should be compensated for these costs in the price of $\mathrm{ETH}_{\mathsf{XX}}$.
- *Exchange rate risk:* a relevant factor when the asset being delivered is different than the asset paying for the forward. In our case, we are determining the price in $\mathrm{ETH}_{\mathsf{L1}}$ for future delivery of $\mathrm{ETH}_{\mathsf{L1}}$, thus, there is no exchange risk at this level of the transaction. However, the price of gas (in the term D) is subject to ETH/gas exchange rates. For simplicity, we assume this is built into D.
- *Interest / Yield ($-r+y$):* both $\mathrm{ETH}_{\mathsf{L1}}$ and $\mathrm{ETH}_{\mathsf{XX}}$ have the potential to earn interest or yield (compounding over Δt), while for other tokens, there might be an opportunity to earn new tokens simply by holding the token. Let r be the (risk-free) interest (yield) rate for $\mathrm{ETH}_{\mathsf{L1}}$ that cannot be earned by $\mathrm{ETH}_{\mathsf{XX}}$, while y is the opposite: yield earned from $\mathrm{ETH}_{\mathsf{XX}}$ and not $\mathrm{ETH}_{\mathsf{L1}}$. Initially $y > 1$ and $r = 0$, however, with $\mathrm{ETH}_{\mathsf{XX}}$ becoming mainstream, it is possible $r = y$ (especially hedged $\mathrm{ETH}_{\mathsf{XX}}$).
- *Settlement risk (R):* the probability that $\mathrm{ETH}_{\mathsf{L1}}$ will fail to be delivered for $\mathrm{ETH}_{\mathsf{XX}}$ discounts the price of $\mathrm{ETH}_{\mathsf{XX}}$. We will deal with this separately.

Put together, the price of $\mathrm{ETH}_{\mathsf{XX}}$ (F_0) is:

$$F_0 = (S_0 + U - D) \cdot e^{(-r+y)\cdot\Delta t} \cdot R$$

This value, F_0, is an expected value—the product of the value and the probability that the RBlock fails to finalize. However, the trader is informed because they have run verification software and checked that the RBlock validates.

$$R = (1 - \mathbf{Pr}[\mathsf{rblock}\text{ fails to finalize}|\mathsf{rblock}\text{ passes software verification}])$$

Working Example. We start with R. The promise of an optimistic rollup is that it is very costly to post an RBlock that will not finalize. Assume the probability an RBlock fails for any reason is 1 in a billion. Assume the probability of inattention—that no one challenges a bad RBlock—is 1 in a million. Assume the validation software is wrong (false positive) also with 1 in a million. Using Bayes theorem, $R = (1 - 10^{-15})$; a near-certain probability. Next, consider the resulting price of F_0. Alice starts with 100 ETH_{XX} and Bob purchases it from her. Bob can hold ETH_{XX} with no cost ($U = 0$). Alice pays the transaction fee for the deposit, however the cost for the contract for exiting ETH_{XX} into ETH_{L1} after the dispute period is expected to be $D = 0.008$ ETH (D). Assume a safe-ish annual percent yield (APY) on ETH deposits is 0.2%. Assume ETH_{XX} expires in 6 days (0.0164 years). ETH_{XX} earns no yield ($y = 0$). Plugging this into the equation, $F_0 = 99.665$ ETH.

As a second example, consider a smaller amount like 0.05 ETH_{XX} (less than $100 USD at time of writing). Now the gas costs are more dominating. $F_0 = 0.04186$ ETH_{L1} which is only 83.7%. This demonstrates that fast exits are expensive for withdrawals of amounts in the hundreds of dollars.

Lastly, could ETH_{XX} ever be worth more than ETH_{L1}? The equation says yes: with a sufficiently high U or y. A contrived example would be some time-deferral reason (*e.g.,* tax avoidance) to prefer receiving ETH_{L1} in 7 days instead of today. However, in order to purchase ETH_{XX} at a premium to ETH_{L1}, it would have to be cheaper to trade for it than to simply manufacture it. Someone holding ETH_{L1} and wanting ETH_{XX} could simply move it to L2 and then immediately withdraw it to create ETH_{XX}. The gas cost of this path will be one upper bound on how much ETH_{XX} could exceed ETH_{L1} in value.

Pricing $\text{FINAL}_{\mathsf{PM}}$ and $\text{FAIL}_{\mathsf{PM}}$. It might appear surprising at first, but one of the main results of this paper is that the price of 100 ETH_{XX} and the price 100 $\text{FAIL}_{\mathsf{PM}}$ are essentially the same. Both are instruments that are redeemable at the same future time for the same amount of ETH_{L1} (either 100 if the RBlock finalizes and 0 if the RBlock fails) with the same probability of failure (that the RBlock fails). The carrying costs of both are identical. There may be slight differences in the gas costs of redeeming ETH_{L1} once the dispute period is over, however the operation (at a computational level) is largely the same process.

6 Discussion

6.1 Prediction Market Fidelity

If Alice's exit is contained in a given RBlock that is finalized, so is Alice's exit. However if the RBlock fails, should Alice's exit fail along with it? First, there is the question: why would the RBlock fail? If it is because of a bad transaction that impacts Alice's exit directly (*e.g.,* she never asked for a withdrawal) or indirectly (*e.g.,* a double spend that eventually flows into Alice's account), then Alice's exit should fail. But if the RBlock fails for no reason related to Alice's exit, and the successful challenging RBlock causing it to fail includes Alice's exit, should

Alice be able to trade her exit specifically, regardless of the RBlock it appears in? The advantages of tying an exit to an RBlock are: (1) it is consistent with the current implementation of *Arbitrum*, and (2) a prediction market can be run for an entire RBlock instead of a fragmentation of markets for each specific exit. The disadvantage is: Alice has to validate the entire RBlock (including transactions after her withdrawal request) before establishing her own confidence in her $\mathrm{ETH}_{\mathsf{XX}}$.

6.2 Withdrawal Format

As implemented, transferable exits can only be transferred in their entirety. If Alice wants to withdraw 100 $\mathrm{ETH}_{\mathsf{L2}}$ and give 50 $\mathrm{ETH}_{\mathsf{XX}}$ to one person and 50 $\mathrm{ETH}_{\mathsf{XX}}$ to another, she cannot change this once she has initiated the withdraw (if she anticipates it, she can request two separate withdrawals for the smaller amounts). We could implement divisible exits and for ETH; there are no foreseen challenges since the semantics of $\mathrm{ETH}_{\mathsf{L1}}$ are specified at the protocol-level of Ethereum. However for custom tokens, the bridge would need to know how divisible (if at all) a token is. In fact, a bridge should ensure that the L2 behavior of the tokens is the same as L1 (or that any inconsistencies are not meaningful). Even if a token implementation is standard, such as ERC20, this only ensures it realizes a certain interface (function names and parameters) and does not mean the functions themselves are implemented as expected (parasitic ERC20 contracts are sometimes used to trick automated trading bots []). The end result is that bridges today do not allow arbitrary tokens; they are built with allowlists of tokens that are human-reviewed and added by an authorized developer. In this case, ensuring divisible exits are not more divisible than the underlying token should be feasible, but we have not implemented it.

6.3 Markets

At the time of writing, the most common way of exchanging tokens on-chain is with an automated market maker (AMM) (*e.g., Uniswap*). If Alice withdraws $\mathrm{ETH}_{\mathsf{XX}}$ and Bob is a willing buyer with $\mathrm{ETH}_{\mathsf{L1}}$, an AMM is not the best market type for them to arrange a trade. AMMs use liquidity providers (LPs) who provide both token types: Alice has $\mathrm{ETH}_{\mathsf{XX}}$ but no $\mathrm{ETH}_{\mathsf{L1}}$ that she is willing to lock up (hence why she is trying to fast exit). Bob has $\mathrm{ETH}_{\mathsf{L1}}$ but to be an LP, he would also need to have $\mathrm{ETH}_{\mathsf{XX}}$ from another user. However, this only pushes the problem to how Bob got $\mathrm{ETH}_{\mathsf{XX}}$ from that user. The first user to sell $\mathrm{ETH}_{\mathsf{XX}}$ cannot use an AMM without locking up $\mathrm{ETH}_{\mathsf{L1}}$, which is equivalent to selling $\mathrm{ETH}_{\mathsf{XX}}$ to herself for $\mathrm{ETH}_{\mathsf{L1}}$. The second challenge of an AMM is the unlikely case that an RBlock fails and $\mathrm{ETH}_{\mathsf{XX}}$ is worthless—then the LPs have to race to withdraw their collateral before other users extract it with worthless $\mathrm{ETH}_{\mathsf{XX}}$. It is better to use a traditional order-based market; however, these are expensive to run on L1 [11]. One could do the matchmaking on L2 and then have the buyer and seller execute on L1, but this reintroduces the griefing attacks we have tried to avoid. For now, we implement a very simple one-sided market where Alice can deposit her ETH_{XX} and an offer price, and Bob can later execute the trade against. If Alice is unsure how to price ETH_{XX}, an auction mechanism could be used instead.

6.4 Low Liquidity or Non-Fungible Tokens

For tokens that have low liquidity on L1, or in the extreme case, are unique (*e.g.,* an NFT), fast exits do not seem feasible. All the fast exit methods we examined do not actually withdraw the original tokens faster; they more indirectly substitute a functionally equivalent token that is already on L1. However, we can still help out with low-liquidity withdrawals. First, we should consider *why* the user wants a fast exit. If it is to sell the token, they can sell the exit instead of the token to any buyer that is L2-aware. To sell to an L2-agnostic buyer, the seller can insure the claim with enough $\text{FAIL}_{\mathsf{PM}}$ to cover the purchase price. Note that in this case, a prediction market is necessary: the claim is one kind of token, while the $\text{FAIL}_{\mathsf{PM}}$ can be redeemed for ETH_{L1}, so it is not the case that the seller can just sell to the holder of $\text{FINAL}_{\mathsf{PM}}$.

Another reason a user may want a fast withdrawal would be to use the token directly. For example, a user might have bought governance tokens on L2 because it was cheaper, but when a snap election over an emergency measure is proposed on L1, they realize they cannot get their tokens out of L2 fast enough to vote. Because the vote is contentious, other holders do not want to sell their tokens for a tradeable exit which would prevent them from voting. There is no way around this without modifying the DAO.

An example modification could require voters to transfer their exit, returnable immediately after the election, and stake ETH_{L1} as a fidelity bond, returnable after the dispute period if the RBlock containing the exit finalizes.

Last, an L2-to-L1 transfer could be a general message rather than a token transfer. For example, an oracle system might operate on L2 because it is cheaper, and then a need arises on L1 for the oracle's data. Again, a prediction market can provide insurance for the data in an exit being incorrect through $\text{FAIL}_{\mathsf{PM}}$.

7 Concluding Remarks

This paper addresses a common 'pain point' for users of L2 rollups on Ethereum. The 7 day dispute period prevents users from withdrawing ETH, tokens, and data quickly. Tradable exits provide users with flexibility after they request a withdrawal. If they decide 7 days is too long, they can seek to trade their exit for ETH_{L1} or they can ask a contract to accept their ETH_{XX} by bundling it with insurance against the failure of the RBlock—this way the contract does not have to be L2-aware.

References

1. J. Clark, J. Bonneau, E. W. Felten, J. A. Kroll, A. Miller, and A. Narayanan. On decentralizing prediction markets and order books. In *Workshop on the Economics of Information Security (WEIS)*, volume 188, 2014.
2. D. Demirag and J. Clark. Absentia: Secure multiparty computation on ethereum. In *Workshop on Trusted Smart Contracts (WTSC)*, pages 381–396. Springer, 2021.
3. B. Ford and R. Böhme. Rationality is self-defeating in permissionless systems. Technical Report cs.CR 1910.08820, arXiv, 2019.
4. N. Foundation. Hardhat. `https://hardhat.org`, October 2022. (Accessed on 10/18/2022).
5. L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais. Sok: Layer-two blockchain protocols. In *Financial Cryptography*, 2020.
6. L. Harris. *Trading and exchanges: market microstructure for practitioners*. Oxford, 2003.
7. J. Hull, S. Treepongkaruna, D. Colwell, R. Heaney, and D. Pitt. *Fundamentals of futures and options markets*. Pearson Higher Education AU, 2013.
8. H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium*, pages 1353–1370, 2018.
9. P. McCorry, C. Buckland, B. Yee, and D. Song. Sok: Validating bridges as a scaling solution for blockchains. Technical report, Cryptology ePrint Archive, 2021.
10. S. Meiklejohn. An evolution of models for zero-knowledge proofs. In *EUROCRYPT (invited talk)*, 2021.
11. M. Moosavi and J. Clark. Lissy: Experimenting with on-chain order books. In *Workshop on Trusted Smart Contracts (WTSC)*, 2021.
12. P. Sztorc. Truthcoin. Technical report, 2015.
13. C. Whinfrey. Hop: Send tokens across rollups. Technical report, 2022.
14. A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt. Sok: Communication across distributed ledgers. In *Financial Cryptography*, 2021.