Fast and Furious Withdrawals from Optimistic Rollups

Abstract. Optimistic rollups are in wide use today as a scalability add-on 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. We modify a popular rollup, *Arbitrum*, to enable withdraws to be traded on L1 before they are finalized. We also study the combination of a tradeable exit with a prediction market on L1 that effectively provides insurance in the event that the withdraws 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 As a result, anyone (including contracts) on L1 can safely accept withdrawn tokens while the dispute period is open despite having no knowledge of what is happening on L2. 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), 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, and finds that two optimistic rollups, *Arbitrum* and *Optimism*, respectively account for 50% and 30% of 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 [8], that can trustlessly move ETH (and other tokens and even arbitrary messages) between L1 and its own

[1] L2 Beat, accessed Oct. 2022.

L2. It does this by locking the ETH in the L1 contract and later releasing it, on request, according to who its new owner is on L2 at the time of the request. If finalized, the ETH will be destroyed on L2 and will be released by the bridge on L1.

Owing to how optimistic rollups convince the L1 bridge contract of who the current owner of withdrawn ETH is on L2 (explained below), the bridge has to first wait 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 withdraw. We concentrate in the task of drawing down ETH specifically but the solution works for any fungible ERC20 (or related) token. While these techniques work easily between human participates 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. Finally, we modify the current version, *Nitro*, of the most popular optimistic rollup, *Arbitrum*, made open source[2] by *Offchain Labs* to implement our solution and provide measurements. *Arbitrum* is a commercial product with academic origins [7].

2 Background

Inbox. Roll-ups have emerged as a workable approach to reducing fees and latency for Ethereum-based decentralized applications. In a roll-up, 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 depositing them in 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 will operate by different rules designed to reduce fees, increase throughput, and decrease latency.

Outbox. Occasionally (*e.g.,* every 30 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 in a contract on L1 called the outbox. Note that anyone with a view of L1 can validate that the

[2] GitHub: Nitro

sequence of transactions recorded in the inbox produces the asserted checkpoint in the outbox, however asking Ethereum 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 checkpoint is correct.

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 the assertion can be disputed by anyone on Ethereum and Ethereum itself can decide between two (or more) disputes for far less cost than running all of the transactions. It will then reallocate the fidelity bonds to the whoever is making correct assertions. If an assertion is undisputed for a window of time (*e.g.,* 7 days), the assertion is considered final.

Bridge. A final piece of layer 2 infrastructure is a bridge, which can move ETH, tokens, NFTs, and even arbitrary messages, between layer 1 and layer 2. In this paper, we consider the case of a bridge for ETH (while discussing other kinds much later in Section **??**). 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 layer 2 environment. Since any deviation from this will result in an incorrect checkpoint which is ultimately checkable by Ethereum, the bridge does not need to be trusted. A set of transactions on layer 2 might see the ETH move from Alice's account to Bob's account on Layer 2. Bob is now entitled to draw down from layer 2 to layer 1. He submits a withdraw request, which reduces his Layer 2 balance and ends up in the next asserted checkpoint on Layer 1. Optimistically, the checkpoint is undisputed and 7 days later it is finalized. Bob can now ask the bridge to release the ETH by demonstrating his withdraw is contained in the finalized checkpoint.

2.1 Related Work

Arbitrum is described at *USENIX Security* [7]. 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 roll-ups and validating bridges [8]. Some papers implement research solutions on Arbitrum for improved performance: decentralized order books [9] and secure multiparty computation [2]. Further academic work on optimistic rollups and bridges is largely missing at this time, but 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

[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.

Type	*Example*	No trusted third party	Within a L1 transaction	Within a 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	•		•	•	•	•	
Hedged Tradeable Exits	This Work	•		•	•	•	•	

Table 1. Comparing alternatives for fast withdrawals where • satisfies the property fully, ○ partially satisfies the property, and no dot means the property is not satisfied.

of the area (and a new theoretical result) [11]. Decentralized prediction markets proposals predate Ethereum and include Clark *et al.* [1] and Truthcoin [10]. Early Ethereum projects Augur and Gnosis began as prediction markets.

The idea of tradeable exits predates our work. It is hard to pinpoint a source but it is discussed: examples. In this paper, we do a comparative analysis of tradeable tickets with other approaches (atomic swaps), actually implement it, and suggest parameters. Further, our hedging protocol with prediction markets is, to our knowledge, novel.

3 Proposed Solution

For simplicity, we will describe a fast exit system for ETH but it is designed for any fungible token native to L1, available for exchange on L1. 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. If we admit a centralized trusted party, such as an exchange (*e.g., Coinbase*, *Binance*), then a market for $\text{ETH}_{\mathsf{L2}}/\text{ETH}_{\mathsf{L1}}$ can be established with liquidity on both sides. Liquidity

might come from a decentralized set of providers, but relaying the outcome of a L2 action to L1 requires a trusted entity. This entity can be distributed to an established set of trustees (called proof of authority) but is not decentralized (*i.e.,* not a large, *open* set of participants).

Atomic Swap. 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) a L1 transaction moving 99.95 ETH_{L1} from Bob to Alice. By binding, we mean either both execute or both fail. A hash-timelocked-contract (HTLC) is a standard way of achieving this that works between Ethereum and an EVM-based L2. 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 griefing). Bob also needs to monitor both chains so he cannot be a smart contract.

Conditional Transfers. Alice uses a L1 contract to register a request for payment of 99.95 ETH_{L1} (from anyone) with ID number 1337. On L2, she provides Bob with a signed conditional transaction that withdraws 100 ETH_{L2} to Bob (in 7 days) *if and only if* a payment has been made on L1 in the registry for 1337 (otherwise it becomes invalid after one hour). A more powerful rollup can check the L1 contract state before deciding to include the L2 transaction or not. As in atomic swaps, Bob needs to monitor L2 for the conditional transaction before deciding to act on L1 and submit payment. Also like atomic swaps, Bob can grief Alice.

Bridge Token. A third party creates a bridge on L2 that converts ETH_{L2} into a custom ticket that serves as a claim for ETH_{L2}, and an equivalent bridge on L1. Alice burns 100 tickets on L2. Bob notices and generates a claim for ETH_{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 semi-trusted as he can 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_{XX} from Alice for ETH_{L1} and receiving his ETH_{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.

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

3.2 Tradeable Exits

In this paper, we implement at tradeable exit. An 'exit' is the proof submitted by the user after the dispute period to the bridge (which references the outbox)

that a token withdrawal has been finalized. Currently in major deployed rollups, an exit can only be redeemed by the account (*i.e.,* using `msg.sender`) that was specified by the user 7 days earlier in the L2 withdrawal request. With tradeable exits, the authorized user can transfer the exit to a new address. The outbox tracks transfers and will offer a function to the bridge that answers queries about the current owner of an exit.

This functionality can be approximated by a third party L1 contract without changing the bridge. In this scenario, a two-stage withdrawal would occur. The user would specific 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. So why modify the bridge/outbox? 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 withdraws normally to their own account cannot change their mind and opt-in to a fast withdrawal—it is too late. With tradeable exits, a user can decide at anytime to expedite the withdrawal. This allows flexibility for users who are not initially aware of the 7 day dispute window (antidotally, this is a common concern on support channels for optimistic rollups), whose reasons for needing a fast exit emerge sometime within the 7 days, or they are simply waiting on a lower price to exit fast.

In our protocol so far, Alice wants to fast withdraw 100 ETH_{L2}. Bob has 100 ETH_{L1} that he 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. In this case, Alice will withdraw 100 ETH_{XX} and swap it for Bob's 100 ETH_{L1}, while paying a fee to Bob (we compute fees in Section 5.1).

3.3 Hedged Tradeable Exits

One remaining issue with this method is how specialized Bob is: he must have liquidity in ETH_{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 ETH_{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 account or contract.

The primary risk to 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 L1 contract. We model it 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 amounts would be smaller but harder to read) and receive two 'shares': one that is a bet that the RBlock will finalize, called FINAL_{PM}, and one that is a bit that the RBlock will fail called FAIL_{PM}. These shares can be traded on any platform. At any time while the prediction market is open, any user can redeem 1 FINAL_{PM} and 1 FAIL_{PM} for 1 ETH_{L1}. Once the dispute period is over, any user can request that the market closes. The market checks the rollup's outbox for the status of the RBlock—since both contacts are L1, this can be done directly without oracles or governance. If the RBlock finalizes, it offer 1 ETH_{L1} for any 1 FINAL_{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 (1) the price of one share matches the probability (according to the collective wisdom of the market) that its winning condition will occur, and (2) the prices of 1 FINAL_{PM} and 1 FAIL_{PM} will sum up to 1 ETH_{L1}. For example, if FAIL_{PM} trades for 0.001 ETH_{L1}, then (1) the market believes the RBlock will fail with probability 0.1% and (2) FINAL_{PM} will trade for 0.999 ETH_{L1}. These arguements do not assume market friction: if the gas cost for redeeming shares is D (for delivery cost), the share's price will incorporate D as discount in the price. Note that since D is a fixed cost for any number of shares, how much it reduces the price of a single share depends on the scale of the market. We model this in Section 5.1. Last, prediction markets are flexible for traders who can enter and exit positions—at a profit, if they correctly identify over- and/or under-valued forecasts. This is in contrast to an insurance-esque arrangement where the insurer is committed to holding their position until completion of the arrangement.

Hedging exits. Given a prediction market, Alice can hedge 100 ETH_{XX} by obtaining 100 FAIL_{PM} as insurance. Any autonomous L1 contract ('Carol') should be willing to accept a portfolio of 100 ETH_{XX} and 100 FAIL_{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 is narrow. At first, it might seem we have successfully separated Carol—the L1 contract that is L2-agnostic—from the L2 validator David. However if Alice is able to obtain 100 FAIL_{PM}, that means someone else is holding 100 FINAL_{PM}.

Consider Path 1 through the protocol. Alice withdrawals 100 ETH_{L2} from L2 and obtains 100 ETH_{XX}. Bob creates 100 FAIL_{PM} and 100 FINAL_{PM} for a cost of 100 ETH_{L1}. Alice buys 100 FAIL_{PM} from Bob for a small fee. Alice gives Carol 100 ETH_{XX} and 100 FAIL_{PM} and is credited as if she deposited 100 ETH_{L1}. In seven days, Bob gets 100 ETH_{L1} for his 100 FINAL_{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 FAIL_{PM}. In both cases, Alice has a balance of 100 ETH_{L1} with Carol.

In path 2, Alice withdrawals 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 seven 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 just uses basic tradeable exits. So do prediction markets actually add nothing to tradeable exits? 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 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 $\text{FINAL}_{\mathsf{PM}}$. Skeptics of rollups might bet against the technology with $\text{FAIL}_{\mathsf{PM}}$. Speculators add liquidity to the prediction market which reduces transactional fees for Alice. However it also brings externalities to the rollup system, which relies on incentives, where the side-bets on an RBlock could exceed the staking requirements for posting an RBlock. However this at least makes the externalities visible—there is no way to prevent them and they exist in any decentralized incentive-based system [3].

4 Implementation and Performance Measurements

To implement our solutions, we get an *Arbitrum Nitro* test-net up and running locally and use Hardhat [4] to run our experiments and obtain our performance metrics using TypeScript.

4.1 Tradeable Exits

Trading the exit directly through the bridge/outbox. We modify the *Arbitrum Nitro* outbox to add a native support for the fast exit solution. 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 very 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 very efficiently. The key of the mapping is the exit number and the value is the current owner of the exit.

[4] GitHub: link is removed for anonymity.

- Modifying the `executeTransactionImpl()` function. Once the dispute period is passed and the withdrawal transaction is confirmed, anyone can before we try to execute out message, anyone can call this function from the outbox 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 ETH_{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 the most recent asserted RBlock. If the exit has been transferred, the `msg.sender` is cross checked against the current owner of the exit (remember we keep track of exit owners using 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. An optional solution for Alice is to transfer her exit through a market that resides on L1 (instead of directly through the bridge/outbox). To support this solution, we implement and deploy an L1 market which allows users to trade their exits on L1 even though the dispute window in not passed. In addition, we add a new function to the *Arbitrum Nitro* outbox, the `checkExitOwner()`, which 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 are open source and written in 125 and 294 lines (SLOC) of Solidity respectively. The solidity code for these con-

[5] A typescript library for client-side interactions with Arbitrum.

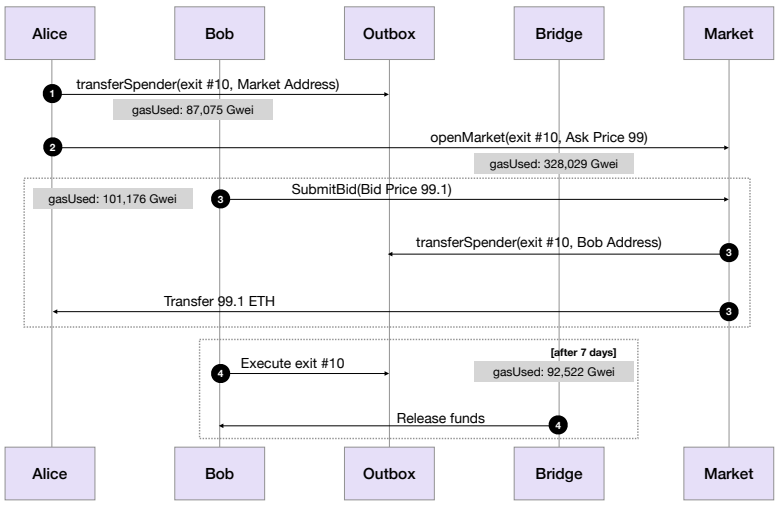

Fig. 1. Overview of trading the exit through an L1 market.

tracts 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. To implement this, we offer one can use an existing decentralized prediction market (*e.g., Augur* or *Gnosis*) however we further modify *Arbitrum Nitro* 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 very efficiently. The key of the mapping is the exit number and the value is the user-defined data type `state` that restrict 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.

[6] GitHub: link is removed for anonymity.

- 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 Discussion

5.1 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 [**?**], 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 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 [**?**] that determine the price of a forward contract (F_0) and translate what they mean for ETH_{XX}:

- *Spot price of ETH_{L1} (S_0):* the price today of what will be delivered in the future. ETH_{XX} is the future delivery of ETH_{L1}, which is by definition worth 100 ETH_{L1} today.
- *Settlement time (Δt):* the time until the exit can be traded for ETH_{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 ETH_{XX} and securing ETH_{L1} is identical in normal circumstances, so not having to take possession of ETH_{L1} for Δt time does not reduce costs for a ETH_{XX} holder.

[7] GitHub: link is removed for anonymity.

- *Delivery cost (D):* the cost of delivery of the asset, which in our case will encompass gas costs. Exchanging ETH_{L1} for ETH_{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 ETH_{L1} seller should be compensated for these costs in the price of ETH_{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 ETH_{L1} for future delivery of ETH_{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 ETH_{L1} and ETH_{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 ETH_{L1} that cannot be earned by ETH_{XX}, while y is the opposite: yield earned from ETH_{XX} and not ETH_{L1}. Initially $y > 1$ and $r = 0$ however with ETH_{XX} becoming mainstream, it is possible $r = y$ (especially hedged ETH_{XX}).
- *Settlement risk (R):* the probability that ETH_{L1} will fail to be delivered for ETH_{XX} discounts the price of ETH_{XX}. We will deal with this separately.

Put together, the price of ETH_{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.

Last, 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 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 $FINAL_{PM}$ and $FAIL_{PM}$. 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.

5.2 RBlock Failures

If Alice's exit is contained in a given RBlock and the RBlock is finalized, so is Alice's exit. However if the RBlock fails, should Alice's exit fail along with it? There is first the question of why the RBlock fails? 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 relating 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 pros of tying an exit to an RBlock are: (i) it is consistent with the current implementation of *Arbitrum*; (2) a prediction market can be run for an entire RBlock instead of a fragmentation of markets for each specific exit. The cons are: (i) Alice has to validate the entire RBlock (including transactions after her withdrawal request) before establishing her own confidence in her ETH_{XX}.

5.3 Withdrawal Format

As implemented, transferable exits can only be transferred in their entirety. If Alice wants to withdrawal 100 ETH_{L2} and give 50 ETH_{XX} to one person and 50 ETH_{XX} to another, she cannot change this once she has withdrawn (if she anticipates it, she can request two separate withdrawals for the smaller amounts). We could implement divisible exits and for ETH, it there are no forseen challenges since the semantics of ETH_{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 behaviour of the tokens is the same as L1 (or that any inconsistencies are not meaningful). Even if a token implements are 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, 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.

5.4 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 ETH_{XX} and Bob is a willing buyer with ETH_{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 ETH_{XX} but no ETH_{L1} that she is willing to lock up (hence why she is trying to fast exit). Bob has ETH_{L1} but to be an LP, he would also have to have ETH_{XX} from another user. However this just pushes the problem to how Bob got ETH_{XX} from that user. The first user to sell ETH_{XX} cannot use an AMM without locking up ETH_{L1}, which is equivalent to selling ETH_{XX} to herself for ETH_{L1}. The second challenge of an AMM is the unlikely case that an RBlock fails and ETH_{XX} is worthless—then the LPs have to race to withdrawal their collateral before it is other users extract it with worthless ETH_{XX}. It is better to use a traditional order-based market, however these are expensive to run on L1 [9]. One could do the match-making on L2 and then have the buyer and sell execute of 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.

5.5 Low Liquidity or Non-Fungible Tokens

For tokens that have low liquidity on L1, or in the extreme case, do not exist yet on L1 (*e.g.,* NFT), fast exits do not seem feasible. All the fast exit methods we examined do not actually withdrawal the original tokens faster, this more indirectly substitute a functionally equivalent token that is already on L1. However we can still help out 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 a 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 where the seller can just sell to the holder of $\text{FINAL}_{\mathsf{PM}}$.

Another reason to fast withdrawal might be to use the token. 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 to not want to sell their tokens for a tradable exit which would prevent them from voting. There is no way around this without modifying the DAO, however it 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 FAIL_{PM}.

6 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. Tradeable exits provide users with flexibility when they withdrawal. If they decide 7 days is too long, they can seek to trade their exit for ETH_{L1}. They can decide anytime after the

References

1. Clark, J., Bonneau, J., Felten, E.W., Kroll, J.A., Miller, A., Narayanan, A.: On decentralizing prediction markets and order books. In: Workshop on the Economics of Information Security (WEIS). vol. 188 (2014)
2. Demirag, D., Clark, J.: Absentia: Secure multiparty computation on ethereum. In: Workshop on Trusted Smart Contracts (WTSC). pp. 381–396. Springer (2021)
3. Ford, B., Böhme, R.: Rationality is self-defeating in permissionless systems (2019)
4. Foundation, N.: Hardhat. `https://hardhat.org` (October 2022), (Accessed on 10/18/2022)
5. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Layer-two blockchain protocols. In: Bonneau, J., Heninger, N. (eds.) Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers. Lecture Notes in Computer Science, vol. 12059, pp. 201–226. Springer (2020). `https://doi.org/10.1007/978-3-030-51280-4_12`, `https://doi.org/10.1007/978-3-030-51280-4_12`
6. Harris, L.: Trading and exchanges: market microstructure for practitioners. Oxford (2003)
7. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: USENIX Security Symposium. pp. 1353–1370 (2018)
8. McCorry, P., Buckland, C., Yee, B., Song, D.: Sok: Validating bridges as a scaling solution for blockchains. Cryptology ePrint Archive (2021)

9. Moosavi, M., Clark, J.: Lissy: Experimenting with on-chain order books. In: Workshop on Trusted Smart Contracts (WTSC) (2021)
10. Sztorc, P.: Truthcoin. peer-to-peer oracle system and prediction marketplace. (2015)
11. Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, E., Moreno-Sanchez, P., Kiayias, A., Knottenbelt, W.J.: Sok: Communication across distributed ledgers. In: International Conference on Financial Cryptography and Data Security. pp. 3–36. Springer (2021)