


Fast and Furious Withdrawals from Optimistic Rollups

Mahsa Moosavi ✉ 🏠 

Concordia University, Canada

OffchainLabs, United States

Mehdi Salehi ✉

OffchainLabs, United States

Daniel Goldman ✉

OffchainLabs, United States

Jeremy Clark ✉ 🏠 

Concordia University, Canada

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 As a result, anyone (including contracts) on L1 can safely accept withdrawn tokens while the dispute period is open despite having no knowledge of what is happening on L2.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.

2012 ACM Subject Classification Security and privacy; Security and privacy → Cryptography

Keywords and phrases Ethereum, layer 2, rollups, bridges, prediction markets

Digital Object Identifier 10.4230/LIPIcs...

Acknowledgements @jc tk. Ed Felten, Rachel Bousfield, a16z crypto, Devcon 6, reviewers.

1 Introductory Remarks

Ethereum-compatible blockchain environments, called Layer 2s (or L2s) [Gudgeon et al.(2020)], 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

¹ L2 Beat: <https://l2beat.com/scaling/tv1/>, accessed Oct. 2022.



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 [McCorry et al.(2021)], that can trustlessly move 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 whom its new owner is on L2 at the time of the request. This requires the rollup to convince the L1 bridge contract of whom 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 source² by *Offchain Labs*. We implement our solution and provide measurements. *Arbitrum* is a commercial product with academic origins [Kalodner et al.(2018)]. 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 reduce 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 <https://github.com/OffchainLabs/nitro>

81 Outbox.

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

90 Optimistic vs. zk-rollups.

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

101 Bridge.

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

115 2.1 Related Work

116 Arbitrum is first described at *USENIX Security* [Kalodner et al.(2018)]. Gudgeon *et al.*
 117 provide a systemization of knowledge (SoK) of Layer 2 technology (that largely predates
 118 rollups) [Gudgeon et al.(2020)]. McCorry *et al.* provide an SoK that covers rollups and
 119 validating bridges [McCorry et al.(2021)], while Thibault *et al.* provide a survey specifically
 120 about rollups [Thibault et al.(2022)]. Some papers implement research solutions on Arbitrum

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 [Meiklejohn(2021)].

<i>Type</i>	<i>Example</i>	No trusted third party	Within an L1 transaction	Within an L2 rollup	No grieving	No free option	Opt-in anytime	L2-to-L2	L1 gasUsed	L2 gasUsed	Compensation			
Normal Exit (baseline)	Arbitrum	•		•	•				≈200K	≈80K	None			
Centralized	Binance		•	•	•	•	•	•	≈400K	≈21K	Exchange			
HTLC Swaps	Celer	•	◦	•				•	≈625K	≈92K	Protocol			
Conditional Transfers	StarkEx	•	•	•					⊥	⊥	None			
Bridge Tokens	Hop	◦	•	•		•	•	•	≈1.8M	≈300K	Bonder			
Tradeable Exits	This Work	•	~	•	•	•	•		≈200K	≈80K	None			
Hedged Tradeable Exits	This Work	•	~	•	•	•	•		≈265K	≈80K	None	test	test	test

■ **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, ~ means we propose how to fully achieve the property but do not by default (see caveats in Section 6.1).

for improved performance: decentralized order books [Moosavi and Clark(2021)] and secure multiparty computation [Demirag and Clark(2021)]. 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) [Zamyatin et al.(2021)]. Decentralized prediction markets proposals predate Ethereum and include Clark *et al.* [Clark et al.(2014)] and Truthcoin [Sztorc(2015)]. 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 $100 \text{ ETH}_{\text{L1}}$. When it is in the bridge on L1 and in the user’s account on L2, we denote it $100 \text{ ETH}_{\text{L2}}$. 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 $100 \text{ ETH}_{\text{XX}}$. Other transitional states are possible but not needed for our purposes.

3.1 Design Landscape

Centralized.

Consider Alice who has $100 \text{ ETH}_{\text{L2}}$ and wants (something like) $99.95 \text{ ETH}_{\text{L1}}$ for it. We describe a set of solutions for Alice. A centralized exchange (*e.g.*, *Coinbase*, *Binance*) can

144 open a market for $\text{ETH}_{\text{L2}}/\text{ETH}_{\text{L1}}$. Alternatively, a bridge might rely on an established set
 145 of trustees to relay L2 actions to L1 . This is called proof of authority; it is distributed but
 146 not decentralized (*i.e.*, not an *open* set of participants). The gas costs consists of Alice
 147 transferring her ETH_{L2} onto the exchange (withdraw to L1 is paid for by the exchange).
 148 Compensating the exchange for this is mandatory to the primitive.

149 Hash Time Locked Contracts (HTLCs).

150 Assume Bob has 99.95 ETH_{L1} and is willing to swap with Alice. An atomic swap binds together
 151 (i) an L2 transaction moving 100 ETH_{L2} from Alice to Bob and (ii) an L1 transaction moving
 152 99.95 ETH_{L1} from Bob to Alice. Either both execute or both fail. HTLC is a blockchain-
 153 friendly atomic swap protocol. Its main drawback is that it also has a time window where
 154 Alice (assuming she is the first mover in the protocol) must wait on Bob, who might abort
 155 causing Alice's ETH_{L2} to be locked up while waiting (called the grieving problem), or might
 156 watch price movements before deciding to act (called free option problem). Bob needs to
 157 monitor both chains so he cannot be an autonomous smart contract. HTLCs work between
 158 two L2 s. The fee structure consists of two parts: the gas fee, which covers the gas cost for
 159 sending tokens on the destination chain, and the protocol usage fee which is paid to liquidity
 160 providers and stakers for their services.

161 Conditional Transfers.

162 In this contract-based atomic swap, Alice uses an L1 contract (called a registry) to record a
 163 request for payment of 99.95 ETH_{L1} (from anyone) with ID number 1337. Off-chain, she
 164 provides Bob with a signed L2 transaction (called a conditional transfer or CT) that (slow)
 165 withdraws 100 ETH_{L2} to Bob *if and only if* payment 1337 has been received on the L1
 166 registry at the time the CT is added to the inbox; otherwise the CT reverts. The CT also
 167 expires (always reverts) after one hour. CTs have similar properties to an atomic swap except
 168 Alice gets paid on L1 before anything happens on L2 . The registry check cannot work quickly
 169 between different L2 s. A practical deployment of this protocol for experimental purposes was
 170 not identified within the available resources. We denoted this situation in the table using a
 171 \perp symbol.

172 Bridge Token.

173 A third party creates a bridge on L2 that converts ETH_{L2} into a custom ticket that serves
 174 as a claim for ETH_{L2} [Whinfrey(2022)]. It creates an equivalent bridge on L1 . Alice burns
 175 100 tickets on L2 . Bob notices and generates a claim for ETH_{L1} on L1 (assuming sufficient
 176 supply) in the equivalent L1 bridge. To prevent Bob from maliciously minting tokens on L1
 177 that were not burned on L2 , he must post a fidelity bond of equal or greater value (otherwise
 178 Bob is trusted to not cause insolvency). After the 7-day dispute period, the L1 bridge can
 179 verify Bob's actions are consistent with L2 and release his fidelity bond. Note that when
 180 you collapse this functionality, it is equivalent to Bob buying ETH_{XX} from Alice for ETH_{L1}
 181 and receiving his ETH_{L1} back 7 days later. The extra infrastructure is necessary because
 182 today native bridges do not support tradeable exits. As in atomic swaps, Bob can fail to act
 183 (grieving) which is worst in this case if Alice cannot 'unburn' her tokens, but there is no free
 184 option because Bob is a relay and not a recipient. The fee here is composed of the gas fee
 185 which accounts for the expense of destination chain gas required to transmit tokens to the
 186 user, and the bonder fee is used to compensate bonders for managing the liquidity of the
 187 transfer on the L1 .

188 **Comparative Evaluation.**

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

191 **3.2 Tradeable Exits**

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

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

206 Like bridge tokens, tradeable exits can be approximated by a third party L1 contract
 207 that does not modify the rollup. In this scenario, a two-stage withdrawal would occur. The
 208 user would specify the contract as the recipient of the exit, and the contract would specify
 209 the user as the recipient (initially). The user could then transfer ownership to a new account
 210 within the contract. Given this option, why modify the bridge/outbox of the rollup? We
 211 have two main arguments: (1) it is more efficient for the user to have the functionality
 212 natively in the bridge/outbox, which they have to interact with anyways; and (2) a user
 213 who initially request a ‘normal’ withdrawal cannot change their mind and opt-in to a fast
 214 withdrawal—it is too late. A tradeable exit can bail out a user who withdraws without
 215 realizing there is a 7-day dispute window (anecdotally, this is a common concern on support
 216 channels for optimistic rollups). It also lets a user who is aware decide if and when to
 217 expedite a withdrawal. However, we understand that opting for an optimal default setting is
 218 generally more favorable than requiring user-initiated actions. Thus, our intention here is
 219 not to advocate for a specific perspective, but rather to address the issue from a technical
 220 standpoint. For instance, Arbitrum could incorporate a third-party L1 contract to facilitate
 221 withdrawals without altering the bridge and encourage its users to use it consistently. If
 222 this becomes the norm for withdrawals, the associated costs should remain fairly similar.
 223 Although users who have already withdrawn using the old method might face some challenges,
 224 these issues would likely be temporary if the transferable withdrawal approach becomes the
 225 standard. While transitioning to this default option could be challenging due to social and
 226 educational factors, moving to a new optimistic rollup that supports tradable exits would
 227 present similar challenges.

228 **3.3 Hedged Tradeable Exits**

229 One remaining issue with tradeable exits is how specialized Bob is: he must have liquidity in
 230 ETH_{L1} , be an active trader who knows how to price futures, and be an L2 validator. While
 231 we can expect blockchain participants with each specialization, it is a lot to assume of a
 232 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 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 [Clark et al.(2014)]. 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 FINAL_{PM} , and one that is a bet 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 close. The market checks the rollup's outbox for the status of the RBlock—since both contracts are on L1, this can be done directly without oracles or governance. If the RBlock finalizes, it offers 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 [Clark et al.(2014)] 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 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 (i) the market believes the RBlock will fail with probability of 0.1% and (ii) FINAL_{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 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 100 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 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 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 set up 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 FAIL_{PM} when Bob has no liquidity or interest in the token she is withdrawing (however Carol needs to incorporate 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 [Harris(2003)]. 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 FINAL_{PM}. Quick validators might profit from noticing an invalid RBlock with FAIL_{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 crypto-economic arguments for the rollup. In reality, these externalities can never be prevented in any decentralized incentive-based system [Ford and Böhme(2019)].

4 Implementation and Performance Measurements

We run *Arbitrum Nitro* test-net locally and use Hardhat [Foundation(2022)] for our experiments. We obtain our performance metrics using TypeScript scripts.

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:

⁴ GitHub:Nitro, Fast-Withdrawals: <https://github.com/MadibaGroup/nitro/tree/fast-withdrawals>

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

330 To execute the `transferSpender()` function; Alice (who has initiated a withdrawal for
331 100 ETH_{L2}) has to provide variables related to her exit (*e.g.*, exit number), which she can
332 query using the Arbitrum SDK⁵, as well as the L1 address she wants to transfer her exit
333 to. The `transferSpender()` function then checks (1) if the exit is already spent, (2) it is
334 already transferred, and (3) the exit is actually a leaf in any unconfirmed RBlock. If the
335 exit has been transferred, the `msg.sender` is cross-checked against the current owner of the
336 exit (recall exit owners are tracked in the `transferredToAddress` mapping added to the
337 outbox). Once these tests are successfully passed, the `transferSpender()` function updates
338 the exit owner by changing the address in the `transferredToAddress` mapping. This costs
339 85,945 units of L1 gas. Note that the first transfer always costs more as the user has to
340 pay for initializing the `transferredToAddress` mapping. `transferSpender()` costs 48,810
341 and 48,798 units of L1 gas for the second and third transfer respectively. The `gasUsed` for
342 executing the new `executeTransactionImpl()` function is 91,418 units of L1 gas.

343 Trading the exit through an L1 market.

344 We also implement and deploy an L1 market that allows users to trade their exits on L1 even
345 though the dispute window is not passed (see Section 6.3 for why *Uniswap* is not appropriate).
346 In addition, we add a new function to the *Arbitrum Nitro* outbox, the `checkExitOwner()`,
347 which returns the current owner of the exit. Figure 1 illustrates an overview of participant
348 interactions and related gas costs. To start trading, Alice needs to lock her exit up in the
349 market by calling the `transferSpender()` function from the outbox. Next, she can open a
350 market on this exit by calling the `openMarket()` from the market contract and providing the
351 ask price. The market checks if Alice has locked her exit (by calling the `checkExitOwner()`
352 from the outbox) and only in that case a listing is created on this exit. The market would be
353 open until a trade occurs or Alice calls the `closeMarket()` on her exit. Bob, who is willing
354 to buy Alice's exit, calls the payable `submitBid()` function from the market contract. If the
355 `msg.value` is equal or greater than Alice's ask price, the trade occurs; (1) the market calls
356 the `transferSpender()` from the outbox providing Bob's address. Note that market can
357 only do that since it is the current owner of the exit being traded, and (2) the `msg.value` is
358 transferred to Alice.

359 The market and modified outbox are open source and written in 125 and 294 lines (SLOC)
360 of Solidity respectively. The solidity code for these contracts in addition to the Hardhat

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

XX:10 Fast and Furious Withdrawals from Optimistic Rollups

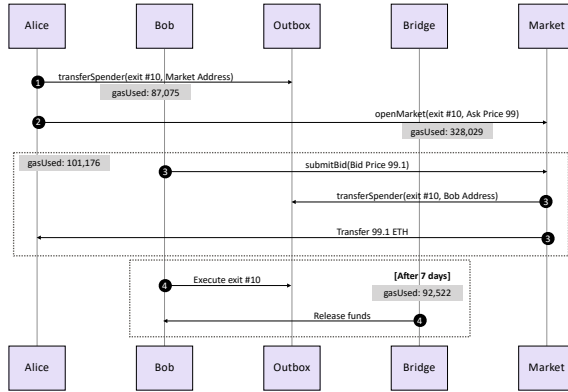


Figure 1 Overview of trading the exit through an L1 market.

361 scripts are available in a GitHub repository.⁶ Once deployed, the bytecode of the market
362 and outbox is 5,772 and 6,264 bytes respectively.

363 4.2 Prediction Market

364 As described in Section 3.3, a prediction market can be used to hedge the exit. We do not
365 implement this as one can use an existing decentralized prediction market (*e.g.*, *Augur* or
366 *Gnosis*). However, we further modify *Arbitrum Nitro* to make it friendly to a prediction
367 market that wants to learn the status of an RBlock (pending, confirmed). More specifically,
368 we modify the *Arbitrum Nitro* outbox and RollupCore smart contracts, modifications include:

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

385 The modified outbox and RollupCore are open source and written in 297 and 560 lines
386 (SLOC) of Solidity respectively. The solidity code for these contracts in addition to the

⁶ GitHub:Nitro, Fast-Withdrawals: <https://github.com/MadibaGroup/nitro/tree/fast-withdrawals>

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 [Hull et al.(2013)], 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 [Hull et al.(2013)] 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.
- *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)

⁷ GitHub:Nitro, Fast-Withdrawals: <https://github.com/MadibaGroup/nitro/tree/fast-withdrawals>

XX:12 Fast and Furious Withdrawals from Optimistic Rollups

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.

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

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 $D = 0.008 \text{ ETH}$ (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, $F_0 = 99.665 \text{ ETH}$.

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. $F_0 = 0.04186 \text{ ETH}_{\text{L1}}$ 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 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 FAIL_{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

A prediction market that covers a larger event should attract more interest and liquidity. For example, betting on an entire RBlock will have more market interest than betting on Alice's specific exit. On the other hand, if markets are exit-specific, the market can be established immediately after Alice's withdrawal hits the inbox instead of waiting for an RBlock (hence \sim in Table 1 to indicate it could be done within one L1 transaction). Another consideration arises when tokens other than ETH are being withdrawn—if the payout of the market matches the withdrawn token, FAIL_{PM} will perfectly hedge the exit. Otherwise the hedge is in the equivalent amount of ETH which could change over 7 days. Our suggestion is to promote the most traders in a single market and avoid fragmentation—so we suggest one market in one payout currency (ETH) for one entire RBlock.

6.2 Withdrawal Format

As implemented, transferable exits can only be transferred in their entirety. If Alice wants to withdraw 100 ETH_{L2} and give 50 ETH_{XX} to one person and 50 ETH_{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 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 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.⁸ 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 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 need to have ETH_{XX} from another user. However, this only 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 withdraw their collateral before 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 [Moosavi and Clark(2021)]. One could do the

⁸ “Bad Sandwich: DeFi Trader ‘Poisons’ Front-Running Miners for \$250K Profit.” *CoinDesk*, Mar 2021.

512 matchmaking on L2 and then have the buyer and seller execute on L1, but this reintroduces
513 the grieving attacks we have tried to avoid. For now, we implement a very simple one-sided
514 market where Alice can deposit her ETH_{XX} and an offer price, and Bob can later execute
515 the trade against. If Alice is unsure how to price ETH_{XX} , an auction mechanism could be
516 used instead.

517 **6.4 Low Liquidity or Non-Fungible Tokens**

518 For tokens that have low liquidity on L1, or in the extreme case, are unique (*e.g.*, an NFT),
519 fast exits do not seem feasible. All the fast exit methods we examined do not actually
520 withdraw the original tokens faster; they substitute a functionally equivalent token that is
521 already on L1. However, we can still help out with low-liquidity withdrawals. We should
522 consider *why* the user wants a fast exit. If it is to sell the token, they can sell the exit instead
523 of the token to any buyer that is L2-aware and willing to wait 7 days to take actual possession.
524 To sell to an L2-agnostic buyer, the seller can insure the exit with enough FAIL_{PM} to cover
525 the purchase price. In this case, the buyer does not get the NFT if the RBlock fails but they
526 get their money back.

527 **7 Concluding Remarks**

528 This paper addresses a common ‘pain point’ for users of L2 optimistic rollups on Ethereum.
529 The 7-day dispute period prevents users from withdrawing ETH, tokens, and data quickly.
530 Tradeable exits provide users with flexibility after they request a withdrawal. If they decide
531 7 days is too long, they can seek to trade their exit for ETH_{L1} or they can ask a contract to
532 accept their ETH_{XX} by bundling it with insurance against the failure of the RBlock—this way
533 the contract does not have to be L2-aware. While some users might still prefer the features of
534 other withdrawal methods (centralized exchanges or solution like *Hop*), it is useful to make
535 the native rollup functionality as flexible as possible, especially for users who do not realize
536 that a withdrawal induces a 7-day waiting period until it is too late.

References

- Clark et al.(2014)** Jeremy Clark, Joseph Bonneau, Edward W Felten, Joshua A Kroll, Andrew Miller, and Arvind Narayanan. 2014. On decentralizing prediction markets and order books. In *Workshop on the Economics of Information Security (WEIS)*, Vol. 188.
- Demirag and Clark(2021)** Didem Demirag and Jeremy Clark. 2021. Absentia: Secure Multiparty Computation on Ethereum. In *Workshop on Trusted Smart Contracts (WTSC)*. Springer, 381–396.
- Ford and Böhme(2019)** Bryan Ford and Rainer Böhme. 2019. *Rationality is Self-Defeating in Permissionless Systems*. Technical Report cs.CR 1910.08820. arXiv.
- Foundation(2022)** Nomic Foundation. 2022. Hardhat. <https://hardhat.org>. (Accessed on 10/18/2022).
- Gudgeon et al.(2020)** Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. 2020. SoK: Layer-Two Blockchain Protocols. In *Financial Cryptography*. https://doi.org/10.1007/978-3-030-51280-4_12
- Harris(2003)** Larry Harris. 2003. *Trading and exchanges: market microstructure for practitioners*. Oxford.
- Hull et al.(2013)** John Hull, Sirimon Treepongkaruna, David Colwell, Richard Heaney, and David Pitt. 2013. *Fundamentals of futures and options markets*. Pearson Higher Education AU.
- Kalodner et al.(2018)** Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. 2018. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium*. 1353–1370.
- McCorry et al.(2021)** Patrick McCorry, Chris Buckland, Bennet Yee, and Dawn Song. 2021. *Sok: Validating bridges as a scaling solution for blockchains*. Technical Report. Cryptology ePrint Archive.
- Meiklejohn(2021)** Sarah Meiklejohn. 2021. An Evolution of Models for Zero-Knowledge Proofs. In *EUROCRYPT (invited talk)*.
- Moosavi and Clark(2021)** Mahsa Moosavi and Jeremy Clark. 2021. Lissy: Experimenting with on-chain order books. In *Workshop on Trusted Smart Contracts (WTSC)*.
- Sztorc(2015)** Paul Sztorc. 2015. *Truthcoin*. Technical Report.
- Thibault et al.(2022)** Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. 2022. Blockchain Scaling Using Rollups: A Comprehensive Survey. *IEEE Access* 10 (2022), 93039–93054.
- Whinfrey(2022)** Chris Whinfrey. 2022. *Hop: Send Tokens Across Rollups*. Technical Report.
- Zamyatin et al.(2021)** Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J Knottenbelt. 2021. Sok: Communication across distributed ledgers. In *Financial Cryptography*.