Fast and Furious Withdrawals from Optimistic Rollups

Mahsa Moosavi
Concordia University, Montreal, Canada
OffchainLabs, Princeton, NJ, USA

Mehdi Salehi
OffchainLabs, Princeton, NJ, USA

Daniel Goldman
OffchainLabs, Princeton, NJ, USA

Jeremy Clark
Concordia University, Montreal, 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 As a result, anyone (including contracts) on L1 can safely accept withdrawn tokens while the dispute period is open despite having no knowledge of what is happening on L2.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.AFT.2023.22

Supplementary Material *Software (Source Code)*: `https://github.com/MadibaGroup/nitro/tree/fast-withdrawals` archived at `swh:1:dir:5e160d45808e5387baa1e65eff88f6eed0e37c97`

Funding *Jeremy Clark*: acknowledges support for this research project from (i) the National Sciences and Engineering Research Council (NSERC), Raymond Chabot Grant Thornton, and Catallaxy Industrial Research Chair in Blockchain Technologies (IRCPJ/545498-2018), (ii) the Autorité des Marchés Financiers, and (iii) a NSERC Discovery Grant (RGPIN/04019-2021).

Acknowledgements This paper includes useful comments from the reviewers, discussions with Edward W. Felten and Rachel Bousfield, and feedback from presentations at Devcon 6 and a16z crypto research.

1 Introductory Remarks

Ethereum-compatible blockchain environments, called Layer 2s (or L2s) [5], have demonstrated an ability to reduce transaction fees by 99–99.9% while preserving the strong guarantees of integrity and availability in the underlying Layer 1 (or L1) blockchain. The subject of this paper concerns one subcategory of L2 technology called an optimistic rollup. The website *L2 Beat* attempts to capitalize all tokens of known value across the top 25 L2 projects. It finds that the top two L2s are both optimistic rollups, *Arbitrum* and *Optimism*, which respectively account for 50% and 30% of all L2 value – $4B USD at the time of writing. [1]

We will describe the working details of optimistic rollups later in this paper but here are the main takeaways: currently, rollups are faster and cheaper than Ethereum itself. However, each L2 is essentially an isolated environment that cannot instantly and trustlessly interact with accounts and contracts that are running on either L1 or other L2s. An optimistic rollup project will typically provide a smart contract, called a validating bridge [9], that can trustlessly move ETH (and other tokens and even arbitrary messages) between L1 and its own L2. It implements a transfer by locking the ETH in an L1 contract and minting the equivalent ETH on L2 and assigning it to the user's L2 address. More precisely, L2 ETH is a transferrable claim for L1 ETH from the L1 bridge at the request of the current owner of the L2 claim. 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 (*e.g.,* for NFTs, function outputs, or arbitrary messages), 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 used optimistic rollup, *Arbitrum*, maintained as open source software[2] by *Offchain Labs*. *Arbitrum* is a commercial product with academic origins [8]. We implement our solution and provide measurements. We also 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*.

2.1 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

[1] L2 Beat: `https://l2beat.com/scaling/tvl/`, accessed Oct. 2022.
[2] GitHub: Nitro `https://github.com/OffchainLabs/nitro`

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.

2.2 Outbox

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

2.3 Optimistic vs. zk-rollups

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

2.4 Bridge

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

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

2.5 Related Work

Arbitrum is first described at *USENIX Security* [8]. Gudgeon *et al.* provide a systemization of knowledge (SoK) of Layer 2 technology (that largely predates rollups) [5]. McCorry *et al.* provide an SoK that covers rollups and validating bridges [9], while Thibault *et al.* provide a survey specifically about rollups [13]. Some papers implement research solutions on Arbitrum for improved performance: decentralized order books [11] and secure multiparty computation [2]. The idea of tradeable exits predates our work but is hard to pinpoint a source (our contribution is implementation and adding hedges). Further academic work on optimistic rollups and bridges is nascent – we anticipate it will become an important research area.

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

3 Proposed Solution

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

3.1 Design Landscape

In Table 1, we compare our solution to alternatives in industry and the blockchain (academic and grey) literature that could be used for fast withdrawals.

3.1.1 Properties

We are interested in solutions that do not require a trusted third party. If trust is acceptable, a centralized exchange that has custody of its users funds is a fast and user-friendly solution. We consider anything faster than the 7-day dispute period as "fast" but take measurements of solutions that can settle within a fully confirmed "L1 transaction" (*e.g.,* minutes) and within a unconfirmed L2 RBlock (*e.g.,* hours). This assumes that all counterparties perform instantly upon request. Settlement is from the perspective of the withdrawer, Alice, only and does not necessarily mean other counterparties will complete within the same timeframe. For example, in many solutions, Alice will have her withdrawn ETH quickly at the expense of a counterparty waiting out the dispute period.

Some solutions require one party to act, followed by an action of the counterparty in a follow-up transaction. This creates the risk that the counterparty aborts the protocol before taking their action. Since it is unknown if the counterparty will act or not, these protocols establish a window of time for the counterparty to act and if the window passes without action, the initial party has to begin the protocol again with a new counterparty. The protocols ensure that funds are never at risk of being lost, stolen, or locked up forever, however the

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. ⊥ was not measured. For our work, ∼ means we propose how to fully achieve the property but do not by default (see caveats in Section 6.1).

Type	*Example*	No trusted third party	Within an L1 transaction	Within an L2 rollup	No griefing	No free option	Opt-in anytime	Crosschain or L2-to-L2	L1 gasUsed	L2 gasUsed	Other Fees
Normal Exit (baseline)	Arbitrum	●			●	●			200K	80K	–
Centralized	Binance		●	●	●	●		●	400K	21K	Operator
HTLC Swaps	Celer	●	○	●				●	625K	92K	
Conditional Transfers	StarkEx	●	●	●					⊥	⊥	Operator
Bridge Tokens	Hop	○	●	●		●		●	1.8M	300K	Operator
Tradeable Exits	This Work	●	∼	●	●	●	●		200K	80K	Discount
Hedged Tradeable Exits	This Work	●	∼	●	●	●	●		265K	80K	FAIL_{PM}

protocols admit two smaller issues. The first issue is that a malicious counterparty could accept to participate with no intention of completing the protocol just to "grief" the party taking the action – wasting their time and possibly gas fees for setting up and tearing down the conditions of the trade. The second issue is that a strategic counterparty can accept to participate and then selectively choose to complete or abort, as well as timing exactly when they choose to complete (within the window), based on price movements or other market information. This is called (somewhat cryptically) a "free option;" finance people might recognize it as akin to being given an American call option for free.

A solution is "opt-in anytime" if the user can withdraw normally and then (say upon realizing for the first time that there is a 7 day dispute window) decide to speed up their transaction. While it is not a design goal of our paper, many of these solutions are generic cross-chain transactions (including L2-to-L2 swaps). A drawback of our solution is that it is narrowly scoped to L2-to-L1 withdrawals on rollups. Therefore our solution is not intended as a complete replacement of atomic swaps or the other solutions in Table 1. It is designed to be best-in-class only for slow rollup withdraws.

Finally we estimate the costs involved for the seller of ETH_{L2}. For some protocols, the gas cost of the buyer might differ from the seller depending if its actions are symmetric or not – we comment on this but did not find it interesting enough to put in the table. The more interesting aspect is that many alternatives do require a third party to be involved (we generically call them "operators") and they must be compensated for their actions. In some alternatives, the operators might be not be inherently necessary (*e.g.,* an HTLC swap) but are used in practice (*e.g.,* Celer) to ease friction (*e.g.,* users finding other users to swap with): in this case, we are charitable and do not mark the fee. So the fees are for things fundamental to how the alternative works. We expand more within the discussion of each alternative below.

3.1.2 Alternatives

Centralized

Consider Alice who has 100 ETH_{L2} and wants 100 ETH_{L1} for it. A centralized exchange (*e.g., Coinbase*, *Binance*) can open a market for $\text{ETH}_{\mathsf{L2}}/\text{ETH}_{\mathsf{L1}}$. Alternatively, a bridge might rely on an established set of trustees to relay L2 actions to L1. This is called proof of authority; it is distributed but not decentralized (*i.e.,* not an *open* set of participants). The gas costs consists of Alice transferring her ETH_{L2} onto the exchange (withdraw to L1 is paid for by the exchange). An exchange will not be profitable if it offers this for free, therefore it captures a operator fee for the service.

Hash Time Locked Contracts (HTLCs)

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

The transaction (containing a hashlock and timeout) is slightly more complicated than a standard ETH transfer, requiring smart contract logic on both layers. The measurement based on Celer is not a pure HTLC and uses operators as well for liquidity and staking, but we omit these fees from the table because theoretically Alice and Bob could find each other and perform a pure HTLC with no added infrastructure.

Conditional Transfers

The intuition behind a conditional transfer (CT) is that L1-to-L2 messaging (or bridging) is fast even if L2-to-L1 messaging is slow. CT exploits this to build an HTLC-esque swap specifically for withdrawing from rollups (while HTLCs are designed generically for cross-chain swaps). Alice beings by registering her intent to trade 100 ETH_{L2} for 100 ETH_{L1} in a special registry contract on L1, and she locks (*e.g.,* for an hour) 100 ETH_{L2} in escrow on L2. If Bob agrees to the swap, Alice provides him (off-chain) with a signed transaction (called the conditional transfer) that transfers the escrowed 100 ETH_{L2} to Bob, conditioned on Alice having receiving 100 ETH_{L1} in the registry contract on L1. After Bob transfers the ETH_{L1} on L1, this fact can be bridged to the L2 escrow contract (with customization of the rollup's inbox) quickly (recall that L1-to-L2 messaging is fast). The L2 escrow contract will flag that the L1 transaction has paid by Bob, and Bob can broadcast his signed (by Alice) L2 transaction to recover 100 ETH_{L2} from escrow (if Bob broadcasts it before the flag is set, it simply reverts).

In terms of existing implementations, we could not adequately isolate the conditional transfer component from the rest of the bridge to measure gas costs (denoted in the table using a $\perp$ symbol) however it should be slight more expensive than an HTLC as the logic of the transaction is more complex.

Also note that Bob must be a validator on L2 to confirm that the state of the escrow and conditional transfer on L2 will result in him being paid – this is where the speedup really comes from, if he waits for L1 to finalize this, then the transfer happens after the dispute period and it is no different than a normal exit. Consequently, Bob cannot be an autonomous L1 smart contract unable to validate L2 state until it is finalized on L1 (which is the design goal of our alternative: hedged tradeable exits).

Bridge Token

A bridge token is not a novel technical innovation but it is a practical market design for supplying bridges with liquidity. Bridges between L1 and L2 can technically be implemented by anyone. It is natural for the inbox/outbox provider to provide a bridge but it is not strictly necessary.

Assume a third party creates a contract on L1 that accepts ETH_{L1} and releases a transferable claim for ETH_{L1}; it creates the same contract on L2. Assume enough of these claims come into circulation that a liquid market for them emerges on both layers. To move ETH_{L2} to ETH_{L1}, Alice starts by trading her ETH_{L2} for a claim to the same amount on L2. She then asks the L2 contract to transfer the claim which it does by burning them and firing an event. An authorized party, called a bonder, notices the event on L2, goes to the L1 contract and mints the same number of claims on L1 for ETH_{L1} and transfers them to Alice's address. Technically the L1 contract is insolvent as more claims exist than actual ETH_{L1} in the contract, but the L2 contract is oversolvent by the same amount. The contracts can be rebalanced (1) through movements in the opposite direction; (2) through a bulk withdrawal after the normal 7-day dispute period; or (3) by incentivize bonders to purposefully rebalance the contracts by burning claims on L1 and minting on L2. To prevent the bonder from maliciously minting tokens on L1 that were not burned on L2, it must post a fidelity bond of equal or greater value. (Alternatively, the bonder can be a trusted party which makes it the same in analysis as a centralized exchange). After the 7-day dispute period, the L1 contract can verify the bonder's actions are consistent with the burns on L2 and release its fidelity bond.

Note that when you collapse this functionality, it is equivalent to the bonder buying ETH_{XX} from Alice for ETH_{L1} and receiving their ETH_{L1} back 7 days later. The extra infrastructure is necessary because today native bridges do not support transferring ETH_{XX}. As in atomic swaps, the bonder can fail to act (griefing) which is worst in this case if Alice cannot "unburn" her tokens, but there is no free option because Bob is a relay and not a recipient of the tokens. The gas fee measurement is based on Hop and standard token transfers on L1 and L2. The main cost of bridge tokens is paying the bonder (called an operator in the table) who are providing a for-profit service.

3.2 Tradeable Exits

Alice wants to withdraw 100 ETH_{L2}. Unlike the other solutions, Bob takes the risk that the exit never finalized and therefore will offer less than 100 ETH_{L1} (say 99.95 ETH_{L1}) for it (this is denoted "discount" in Table 1). Assume Bob has 99.95 ETH_{L1} that will not use until after the dispute window. Bob also runs an L2 validator so he is assured that if Alice withdraws, it is valid and will eventually finalize. With a tradeable exit, the outbox allows Alice to change the recipient of her withdraw from herself to Bob. Thus Alice swaps her pending exit of 100 ETH_{L1} (which we call 100 ETH_{XX}) for Bob's 99.95 ETH_{L1} on L1 (note we discuss the actual difference in price in Section 5). Since ETH_{L1} and ETH_{XX} are both on

L1, Alice can place an ask price for her ETH_{XX} and the first trader willing to swap can do so atomicly, with no ability to grief or capitalize on a free option. After 7 days, Bob can ask the bridge to transfer the ETH_{L1} to his address, and the bridge checks the outbox to validate that Bob's address is the current owner of the exit.

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

Tradeable exits can be approximated by a third party L1 contract that does not modify the rollup. In this scenario, a L1 contract would act like a proxy for the exit. Alice would specify that she is exiting 100 ETH_{L2} to the proxy contract address (instead of to her address) and set the proxy contract to forward it to her address (if/when it comes through after 7 days). Before the dispute window closes, she can sign a transaction instructing the proxy contract to forward the exit to Bob instead of to her (while giving Bob signing authority over it). In this way, the exit becomes tradeable. After 7 days, the current owner can ask the proxy to fetch the actual transfer from the bridge and forward it to them. If the exit fails, the bridge will refuse the exit.

Given this option, why modify the bridge/outbox of the rollup? This paper is not intended as a strong endorsement of either approach – the reader can decide between the two approaches. Our intention with this research is to discuss, design, implement, and measure the actual functionality of what is needed. This will be largely the same whether it is placed inside or outside the bridge/outbox. The main advantage of modifying the bridge/outbox is that is backward compatible with existing web3 bridge interfaces and with current user behaviour – if web3 interfaces or users do a slow withdraw, our solution can "bail them out" after the fact. Placing the functionality inside the bridge/outbox is more challenging in some regards (*e.g.,* existing code is complex to understand) but also easier in other regards (*e.g.,* our code has direct access to state variables). An outside contract might require minor changes to the bridge anyways, such as creating public interfaces to state variables or other data (*e.g.,* as one example, we later discuss how a prediction market must be able to query the outbox to know if an RBlock is pending, finalized, or failed, which is not a current feature). By contrast, the main advantage of an outside contract is modularity and reducing complexity (and thus risk) within the bridge.

3.3 Hedged Tradeable Exits

One remaining issue with tradeable exits is how specialized Bob is: he must have liquidity in ETH_{L1} (or worst, every token being withdrawn from L2), be online and active, know how to price derivatives, and be a 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: Carol and David. Our goal is to allow Carol who does not (or functionally cannot) know anything about L2's current state to safely accept a tradeable exit as if it were equivalent to finalized ETH_{L1} (or L1 tokens). Carol could be a L1 contract that accepts the withdrawn tokens for a service or enables exchange. In order to make Carol agnostic of L2, we need David to be aware of L2: David is a L2 validator who understands the risks of an RBlock failing and is willing to bet against it happening. Therefore David needs to also have some liquidity to bet with however it could be ETH_{L1} or a stablecoin, while Alice and Carol can interact with all sorts of tokens that David need not heard of or even ones David would not want to hold himself.

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

Prediction markets

A decentralized prediction market is an autonomous (*e.g.,* vending machine-esque) third party contract. Since we are insuring L1 ETH_{XX}, we need to run the market on L1 (despite the fact that it would be cheaper and faster on L2). Consider a simple market structure based on [1]. A user can request that a new market is created for a given RBlock. The market checks the outbox for the RBlock and its current status (which must be pending). Once opened, any user can submit 1 ETH_{L1} (for example, the actual amount would be smaller but harder to read) and receive two "shares": one that is a bet that the RBlock will finalize, called 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 contacts are on L1, this can be done directly without oracles or governance. If the RBlock finalizes, it offers 1 ETH_{L1} for any 1 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 (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 [6]. With a prediction market, Alice is a utility trader and Bob is a dealer. However, there might exist speculators who want to participate in the market because they have forecasts about rollup technology, a given RBlock, the potential for software errors in the rollup or in the validator software, *etc.* Executives of rollup companies could receive bonuses in 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 crypo-economic arguments for the rollup. In reality, these externalities can never be prevented in any decentralized incentive-based system [3].

4 Implementation and Performance Measurements

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

- Adding the `transferSpender()` function which allows the exit owner to transfer the exit to any L1 address even though the dispute period is not passed.
- Adding the `isTransferred()` mapping which stores key-value pairs efficiently. The key of the mapping is the exit number and the value is a boolean.

[4] GitHub:Nitro, Fast-Withdrawals: `https://github.com/MadibaGroup/nitro/tree/fast-withdrawals`

- Adding the `transferredToAddress` mapping which stores key-value pairs efficiently. The key of the mapping is the exit number and the value is the current owner of the exit.
- Modifying the `executeTransactionImpl()` function. Once the dispute period is passed and the withdrawal transaction is confirmed, anyone can call the `executeTransaction()` function from the outbox (which internally calls the `executeTransactionImpl()`) and release the funds to the account that was specified by the user 7 days earlier in the L2 withdrawal request. With our modifications, this function is now enabled to release the requested funds to the current owner of the exit.

To execute the `transferSpender()` function; Alice (who has initiated a withdrawal for 100 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 any unconfirmed RBlock. If the exit has been transferred, the `msg.sender` is cross-checked against the current owner of the exit (recall exit owners are tracked in the `transferredToAddress` mapping added to the outbox). Once these tests are successfully passed, the `transferSpender()` function updates the exit owner by changing the address in the `transferredToAddress` mapping. This costs 85,945 units of 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 units of L1 gas for the second and third transfer respectively. The `gasUed` for executing the new `executeTransactionImpl()` function is 91,418 units of L1 gas.

Trading the exit through an L1 market

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

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

[5] A typescript library for client-side interactions with Arbitrum.
[6] GitHub:Nitro, Fast-Withdrawals: `https://github.com/MadibaGroup/nitro/tree/fast-withdrawals`

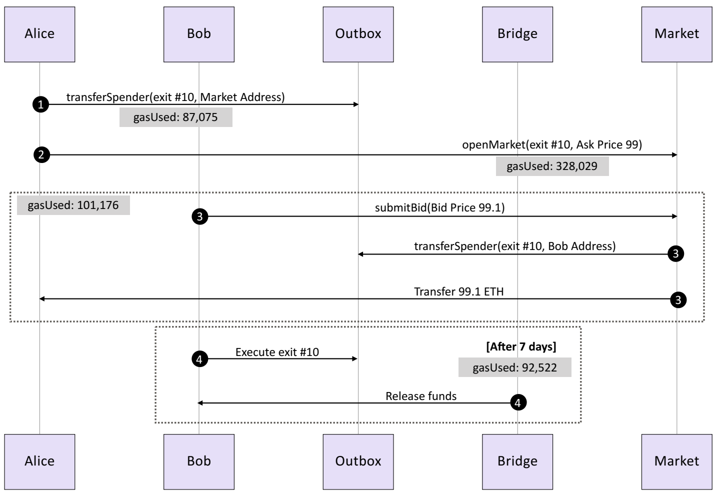

Figure 1 Overview of trading the exit through an L1 market.

4.2 Prediction Market

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

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

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

[7] GitHub:Nitro, Fast-Withdrawals: `https://github.com/MadibaGroup/nitro/tree/fast-withdrawals`

5 Pricing

Pricing ETH$_{\mathsf{XX}}$

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

In traditional finance [7], forward contracts (and futures, which are standardized, exchange traded forwards) are very similar to ETH$_{\mathsf{XX}}$ 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$_{\mathsf{XX}}$ is sold for ETH$_{\mathsf{L1}}$, both price determination and the exchange happen today, while the delivery of ETH$_{\mathsf{L1}}$ for ETH$_{\mathsf{XX}}$ 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 [7] that determine the price of a forward contract (F_0) and translate what they mean for ETH$_{\mathsf{XX}}$:

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

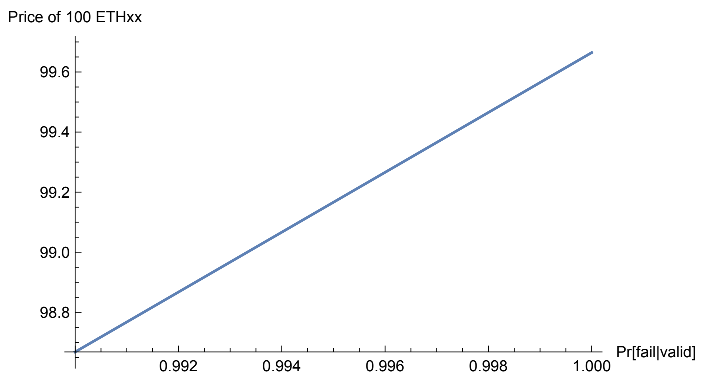

Figure 2 Price of 100 ETH_{XX} (in ETH) as the probability an RBlock actually finalizes (given the validator checks it with software validation) varies from 99% to 100%, which is denoted by R. Note that 99% is an extraordinarily low probability for this event (considering an RBlock has never failed at the time of writing). The take-away is that the price is not very sensitive to how precisely we estimate R.

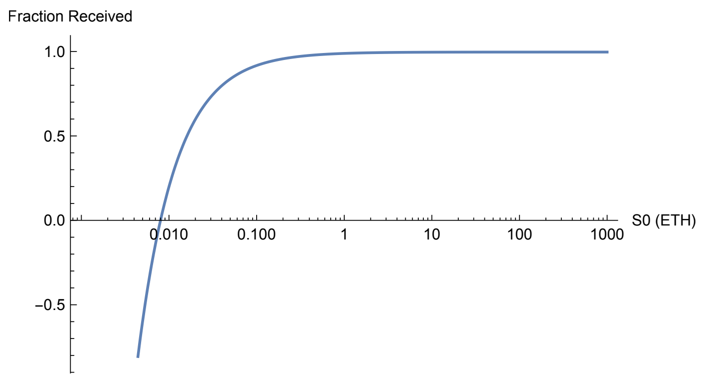

Figure 3 This chart shows the percentage of ETH recovered (F_0/S_0) as the amount withdrawn (S_0) increases (log scale), demonstrating it is only economical for withdrawing larger amounts of ETH_{L2}. At low values, the gas costs of a withdrawal dominate. At very low values, the gas costs exceed the price of ETH_{XX} causing the curve to go negative.

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.

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

Working Example

We start with R. For an RBlock to be up for consideration, it must be submitted to the outbox as a potential solution and for it to fail, a dispute must be filed with an alternative RBlock that the L1 outbox deems to be correct. In our case, the buyer of ETH_{XX} actually runs a L2 validator and thus performs software validation on the RBlock, and will not accept it if the software does not validate it. For an RBlock to fail given the software validation, it software must have an error that causes a discrepancy between it and the L1 outbox. Furthermore, at least one other validator would need to have different, correct software, and this validator would need to be paying attention to this specific RBlock and independently check it. This should be a rare event and assume $R = (1 - 10^{-15})$ for this example. Figure 2 shows a range of R values.

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. Figure 3 shows a range of withdraw amounts.

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 of price 100 FINAL_{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. This is actually a natural result: if 100 FAIL_{PM} perfectly hedges (reduces the risk to zero) the failure of 100 ETH_{XX} to finalize, then the compliment to FAIL_{PM}, FINAL_{PM}, should be priced the same as ETH_{XX}.

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 arrises 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 [11]. One could do the matchmaking on L2 and then have the buyer and seller execute on L1, but this reintroduces the griefing attacks we have tried to avoid. For now, we implement a very simple one-sided market where Alice can deposit her ETH_{XX} and an offer price, and Bob can later execute the trade against. If Alice is unsure how to price ETH_{XX}, an auction mechanism could be used instead.

6.4 Low Liquidity or Non-Fungible Tokens

For tokens that have low liquidity on L1, or in the extreme case, are unique (*e.g.,* an NFT), fast exits do not seem feasible. All the fast exit methods we examined do not actually withdraw the original tokens faster; they substitute a functionally equivalent token that is already on L1. However, we can still help out with low-liquidity withdrawals. 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 and willing to wait 7 days to take actual possession. To sell to an L2-agnostic buyer, the seller can insure the exit with enough $\text{FAIL}_{\mathsf{PM}}$ to cover the purchase price. In this case, the buyer does not get the NFT if the RBlock fails but they get their money back.

[8] "Bad Sandwich: DeFi Trader 'Poisons' Front-Running Miners for $250K Profit." *Coindesk*, Mar 2021.

7 Concluding Remarks

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

References

1 Jeremy Clark, Joseph Bonneau, Edward W Felten, Joshua A Kroll, Andrew Miller, and Arvind Narayanan. On decentralizing prediction markets and order books. In *Workshop on the Economics of Information Security (WEIS)*, volume 188, 2014.

2 Didem Demirag and Jeremy Clark. Absentia: Secure multiparty computation on ethereum. In *Workshop on Trusted Smart Contracts (WTSC)*, pages 381–396. Springer, 2021.

3 Bryan Ford and Rainer Böhme. Rationality is self-defeating in permissionless systems. Technical Report cs.CR 1910.08820, arXiv, 2019.

4 Nomic Foundation. Hardhat. `https://hardhat.org`, October 2022. (Accessed on 10/18/2022).

5 Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In *Financial Cryptography*, 2020. `doi:10.1007/978-3-030-51280-4_12`.

6 Larry Harris. *Trading and exchanges: market microstructure for practitioners*. Oxford, 2003.

7 John Hull, Sirimon Treepongkaruna, David Colwell, Richard Heaney, and David Pitt. *Fundamentals of futures and options markets*. Pearson Higher Education AU, 2013.

8 Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium*, pages 1353–1370, 2018.

9 Patrick McCorry, Chris Buckland, Bennet Yee, and Dawn Song. Sok: Validating bridges as a scaling solution for blockchains. Technical report, Cryptology ePrint Archive, 2021.

10 Sarah Meiklejohn. An evolution of models for zero-knowledge proofs. In *EUROCRYPT (invited talk)*, 2021.

11 Mahsa Moosavi and Jeremy Clark. Lissy: Experimenting with on-chain order books. In *Workshop on Trusted Smart Contracts (WTSC)*, 2021.

12 Paul Sztorc. Truthcoin. Technical report, Online, 2015.

13 Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*, 10:93039–93054, 2022.

14 Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J Knottenbelt. Sok: Communication across distributed ledgers. In *Financial Cryptography*, 2021.