Fast and Furious Withdrawals from Optimistic Rollups

Abstract. Optimistic rollups are in wide use today as a scalability add-on for blockchains like Ethereum. In such systems, Ethereum is referred to as L1 (Layer 1) and the rollup provides an environment called L2, which reduces fees and latency but cannot instantly and trustlessly interact with L1. One practical issue for optimistic rollups is that trustless transfers of tokens and ETH, as well as general messaging, from L2 to L1 is not finalized on L1 until the passing of a dispute period (aka withdrawal window) which is currently 7 days in the two leading optimistic rollups: *Arbitrum* and *Optimism*. In this paper, we explore methods for side-stepping the dispute period when withdrawing ETH from L2. We modify a popular rollup, *Arbitrum*, to enable withdraws to be traded on L1 before they are finalized, and combine them with a prediction market on L1 that effectively provides insurance in the event that the withdraws do not finalize. As a result, anyone (including contracts) on L1 can safely accept withdrawn tokens while the dispute period is open despite having no knowledge of what is happening on L2. All fees are set by open market operations.

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

1 Introductory Remarks

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

We will describe the working details of optimistic rollups later in this paper but here are the main takeaways. Currently, rollups are faster and cheaper than Ethereum itself, however each L2 is essentially an isolated environment that cannot instantly and trustlessly interact with accounts and contracts that are running on either L1 or other L2s. An optimistic rollup project will typically provide a smart contract, called a validating bridge [5], that can trustlessly move ETH (and other tokens and even arbitrary messages) between L1 and its own

[1] L2 Beat, accessed Oct. 2022.

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

Owing to how optimistic rollups convince the L1 bridge contract of who the current owner of withdrawn ETH is on L2 (explained below), the bridge has to first wait a period of time called the dispute window. The current default is 7 days in *Arbitrum* and *Optimism*, however the filing of new disputes can extend the window. The bottom line is that users have to wait at least 7 days to draw down ETH from an optimistic rollup.

Contributions. In this paper, we compare several methods—atomic swaps and tradable exits—for working around this limitation. While we argue workarounds cannot be done generally, some circumstances allow it: namely, when the withdrawn token is liquid, fungible, and available on L1 and the withdrawer is willing to pay a fee to speed up the withdraw. We concentrate in the task of drawing down ETH specifically but the solution works for any fungible ERC20 (or related) token. While these techniques work easily between human participates that have off-chain knowledge, such as the valid state of the L2, it is harder to make them compatible with L1 smart contracts that have no ability to validate the state of L2. We propose a solution using tradable exits and prediction markets to enable an L1 smart contract to safely accept withdrawn tokens before the dispute period is over. Finally, we modify the current version, *Nitro*, of the most popular optimistic rollup, *Arbitrum*, made open source[2] by *Offchain Labs* to implement our solution and provide measurements. *Arbitrum* is a commercial product with academic origins [4].

2 Background

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

Outbox. Occasionally (*e.g.,* every 5 minutes), validators on L2 will produce a checkpoint of the state of all contracts and accounts in the complete L2 according to the latest transactions and will place this asserted state in a contract on L1 called the outbox. Note that anyone with a view of L1 can validate that the

[2] GitHub: Nitro

sequence of transactions recorded in the inbox produces the asserted checkpoint in the outbox, however asking Ethereum to validate this be equivalent to running the transactions on Ethereum. The key breakthrough is that the assertion will be posted with *evidence* that the checkpoint is correct.

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

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

2.1 Related Work

Arbitrum is described at *USENIX Security* [4]. Gudgeon *et al.* provide a systemization of knowledge (SoK) of Layer 2 technology (that largely predates rollups) [3], while McCorry *et al.* provide an SoK that covers roll-ups and validating bridges [5]. Some papers implement research solutions on Arbitrum for improved performance: decentralized order books [6] and secure multiparty computation [2]. Further academic work on optimistic rollups and bridges is largely missing at this time, but we anticipate it will become an important research area. Other related topics are atomic swaps and prediction markets. Too many papers propose atomic swap protocols to list here but see Zamyatin *et al.* for an SoK

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

Type	*Example*	No trusted third party	Within a L1 transaction	Within a L2 rollup	No griefing	No free option	No super dealers
Normal Exit (baseline)	Arbitrum	•					
Centralized	Coinbase		•	•	•	•	•
HTCL Swap	Cellar	•	○	•			
Conditional Transfers	StarkEx	•	•	•			
Bridge Token	Hop	○	○	•	•	•	
Tradeable Exits	This Work	•		•	•	•	
Hedged Tradeable Exits	This Work	•		•	•	•	•

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

of the area (and a new theoretical result) [8]. Decentralized prediction markets proposals predate Ethereum and include Clark *et al.* [1] and Truthcoin [7]. Early Ethereum projects Augur and Gnosis began as prediction markets.

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

3 Proposed Solution

For simplicity, we will describe a fast exit system for ETH but it is designed for any fungible token native to L1, available for exchange on L1. Consider an amount of 100 ETH. When this amount is in the user's account on L1, we use the notation 100 ETH_{L1}. When it is in the bridge on L1 and in the user's account on L2, we denote it 100 ETH_{L2}. When the ETH has been withdrawn on L2 and the withdrawal has been asserted in the L1 outbox, but the dispute window is still open, we refer to it as 100 ETH_{XX}. Other transitionary states are possible but not needed for our purposes.

3.1 Design Landscape

Centralized. Consider Alice who has 100 ETH_{L2} and wants 99.95 ETH_{L1} for it (or an amount close to that). We describe a set of solutions for Alice. If we admit a centralized trusted party, such as an exchange (*e.g., Coinbase*, *Binance*), then

a market for $\text{ETH}_{\mathsf{L2}}/\text{ETH}_{\mathsf{L1}}$ can be established with liquidity on both sides. Liquidity might come from a decentralized set of providers, but relaying the outcome of a L2 action to L1 requires a trusted entity. This entity can be distributed to an established set of trustees (called proof of authority) but is not decentralized (*i.e.,* not a large, *open* set of participants).

Atomic Swap. Assume Bob has 99.95 ETH_{L1} and is willing to swap with Alice. An atomic swap binds together (i) an L2 transaction moving 100 ETH_{L2} from Alice to Bob and (ii) a L1 transaction moving 99.95 ETH_{L1} from Bob to Alice. By binding, we mean either both execute or both fail. A hash-timelocked-contract (HTLC) is a standard way of achieving this that works between Ethereum and an EVM-based L2. Its main drawback is that it also has a time window where Alice (assuming she is the first mover in the protocol) must wait on Bob, who might abort causing Alice's ETH_{L2} to be locked up while waiting (called griefing). Bob also needs to monitor both chains so he cannot be a smart contract.

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

Bridge Token. Hop

Tradeable Exits. If Alice and Bob withdraws 100 ETH_{L2} on L2, which (like all L2 transactions) is recorded in the L1 inbox. Bob can agree results is eventually included in an RBlock

Hedged Tradeable Exits. Our solution ...

- Landscape: atomic swaps
- Landscape: Change the bridge instead of using a third party contract (reason: too late once you withdrawal)

3.2 Design Landscape

3.3 Fast Bridge

- Allow trading of exits (atomic unit) -> track most recent owner (constant time) in outbox. Authorization (only current owner can transfer) -> on execute, check for current owner.
- Self-insurance (staking a fidelity bond) (you need liquidity)
- Prediction market: someone else will insure it (better than insurance because you can exit quickly, change your mind, make money on over-/under- without necessarily agreeing with the position you are buying

3.4 Prediction Market

In our protocol so far, Alice wants to fast withdraw 100 ETH_{L2}. Bob has 100 ETH_{L1} that he will not use until after the dispute window. Bob also runs an L2 validator so he is assured that if Alice withdraws, it is valid and will eventually finalize. In this case, Alice will withdraw 100 ETH_{XX} and swap it for Bob's 100 ETH_{L1}, while paying a fee to Bob.

One remaining issue with this method is how specialized Bob is: he must have liquidity in ETH_{L1}, be an active trader who knows how to price futures, and be an L2 validator. While we can expect blockchain participants with each specialization, it is a lot to assume of a single entity. The goal of this subsection is to split Bob into two distinct participants: one that has ETH_{L1} liquidity but does not know about L2 (Carol) and one that knows about L2 but is not necessarily an active trader on L1 (David). The main impact of this change is that Carol can be an autonomous L1 smart contract. Carol might be an intermediary that exchanges ETH_{XX} and ETH_{L1}. Alternatively, recall that Alice wants ETH_{L1} quicky in order to do something on L1 with it; Carol can be that destination account or contract.

What is the risk if Carol just accepts ETH_{XX} as if it were ETH_{L1}? Carol can check, using L1 only, that Alice owns ETH_{XX} within a disputable assertion that is registered in the L1 outbox of the rollup. The risk is that the assertion is wrong and is not finalized.

3.5 Implementation

The solution is made up of four components: (1) tradeable exits, (2) a prediction market to hedge the exit, and (3) a market to trade hedged (or unhedged) ETH_{XX}. We implement (1) in *Arbitrum Nitro*. For (2), 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, slashed). For (3), one can again use an existing market but we make ETH_{XX} friendly to external markets by realizing an ERC20 interface. That said, ETH_{XX} does not compose will with automated market makers (*e.g., Uniswap*), the predominate DeFi exchange (see Section 4.6), so we also implement a basic offer contract for selling ETH_{XX}.

- Arbitrum's Nitro. Bridge: inbox, sequencer, outbox.
- Implemented a market, following is the gas cost related to the market: gasUsed for opening a market on an exit: 328,029 gasUsed for transferring the exit to the market: 86,701 (it was the first transfer so a bit more expensive that the 2nd, 3rd,.. see below) Gas cost for submitting the Bid for when the Bid is greater than ask -> trade occurs and settles in one single submitBid tx: 105,287 Gas cost for execution is: 92,148
- Modify outbox to allow tradeable exits
- Modify the Nitro codebase (arbnode and validator) to shrink the fraud proof window from 7 days to 1 minute: (1) Modify the confirmPeriod variable

in the arbnode/node.go file, (2) modify the MakeAssertionInterval variable validator/staker.go

- To make the bridge prediction market friendly: Modify the outbox: (1) added a Mapping that maps the proposed arbitrum block number (also known as assertion and node) to the pending state. (2) added a function which accepts a proposed block number and adds it to the pending assertions mapping. Modify the rollupcore: The validator acts through the RollupCore.sol contract when making an assertion by calling a createNewNode() function. We modifies this function so that every time a node is created by the validator it's also added to the outbox's pending assertions mapping (outbox.addToPendingAssertions(latestNodeCreated()))
- L1 gas costs: new function (transferSpender) : First Transfer: 1) Alice withdraws ETH from L2 2) She transfers her exit to Bob transferSpender in this case costs : gasUsed : 85,945 Second Transfer: 2) Now Bob transfer his exit to Carol transferSpender in this case costs : gasUsed : 48,810 Third Transfer: 2) Now Carol transfer his exit to Nancy transferSpender in this case costs : gasUsed : 48,798
  (difference of two mappings)
- L1 gas costs: execute the exit: 91,418
- Unit tests: say something
- What happens when an assertion fails? (pro: ticket fails with assertion, better for prediction markets (betting on assertion which is a batch of exits); ticket passes even if assertion fails, sounds better (caveat: probably won't happen)
- Challenges: SDK, where to change, unit test failed assertions (good assertion, bad assertion where withdrawal is ok, bad assertion where the withdrawal is problematic)
- Expose assertion failures/successes to external contract (submit ID for assertion, get back status: pending, finalized, or discarded). Write down the SDK call. (what happens to a failed assertions???)

4 Discussion

4.1 Pricing

Pricing FAIL$_{PM}$. Consider how much you would sell 100 FAIL$_{\mathsf{PM}}$ for. As an expected value, it is the product of the payout (P_0) 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. Let D be the "delivery" cost of FAIL$_{\mathsf{PM}}$—which in our case is the cost of production and transfer in L1 gas, which on Ethereum is a fixed cost, invariant to the number of shares. The valuation is:

$$P_0 \cdot \mathbf{Pr}[\mathsf{rblock} \text{ fails to finalize} | \mathsf{rblock} \text{ passes software verification}] + D.$$

Working Example. 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, the price is (100 ETH) $\cdot 10^{-15}$; a near-zero price for 100 $\text{FAIL}_{\mathsf{PM}}$. Thus the cost will be dominated by D. Assume it is 0.006 ETH to generate shares and 0.002 ETH to transfer shares, for a total of 0.008 ETH. This is an upperbound for 100 $\text{FAIL}_{\mathsf{PM}}$, assuming they are created as a batch. If they are from an even larger batch of shares, the price could be cheaper (*i.e.,* economies of scale).

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

In traditional finance [**?**], forward contracts (and futures, which are standardized, exchange traded forwards) are very similar to ETH_{XX} in that they price today the delivery of an asset or commodity at some future date. One key difference is that with a forward contract, the price is decided today but the actual money is exchanged for the asset at delivery time. When ETH_{XX} is sold for ETH_{L1}, both price determination and the exchange happen today, while the delivery of ETH_{L1} for ETH_{XX} happens in the future. The consequence is that we can adapt pricing equations for forwards/futures however the signs (positive/negative) of certain terms need to be inverted.

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

- *Spot price of ETH_{L1} (S_0):* the price today of what will be delivered in the future. ETH_{XX} is the future delivery of ETH_{L1}, which is by definition worth 100 ETH_{L1} today.
- *Settlement time (Δt):* the time until the exit can be traded for ETH_{L1}. In *Arbitrum*, the time depends on whether disputes happen. We simplify by assuming Δt is always 7 days (168 hours) from the assertion time. A known fact about forwards is that F_0 and S_0 converge as Δt approaches 0.
- *Storage cost (U):* most relevant for commodities, receiving delivery of a commodity at a future date relieves the buyer of paying to store it in the short-term. Securing ETH_{XX} and securing ETH_{L1} is identical in normal circumstances, so not having to take possession of ETH_{L1} for Δt time does not reduce costs for a ETH_{XX} holder.
- *Delivery cost (D):* the cost of delivery of the asset, which in our case will encompass gas costs. Exchanging ETH_{L1} for ETH_{XX} requires a transaction fee and also creates a future transaction fee to process the exit (comparable in cost to purchasing a token from an automated market maker). An ETH_{L1} seller should be compensated for these costs in the price of ETH_{XX}.

- *Exchange rate risk:* a relevant factor when the asset being delivered is different than the asset paying for the forward. In our case, we are determining the price in ETH_{L1} for future delivery of ETH_{L1}; thus there is no exchange risk at this level of the transaction. However the price of gas (in the term D) is subject to ETH/gas exchange rates. For simplicity, we assume this is built into D.
- *Interest / Yield ($-r+y$):* both ETH_{L1} and ETH_{XX} have the potential to earn interest or yield (compounding over Δt), while for other tokens, there might be an opportunity to earn new tokens simply by holding the token. Let r be the (risk free) interest (yield) rate for ETH_{L1} that cannot be earned by ETH_{XX}, while y is the opposite: yield earned from ETH_{XX} and not ETH_{L1}. Initially $y > 1$ and $r = 0$ however with ETH_{XX} becoming mainstream, it is possible $r = y$ (especially hedged ETH_{XX}).
- *Settlement risk:* the risk that ETH_{L1} will fail to be delivered for ETH_{XX} discounts the price of ETH_{XX}, but we treat this as zero when ETH_{XX} is hedged with prediction market shares.

Put together, the price of ETH_{XX} (F_0) is:

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

Working Example. Alice starts with 100 ETH_{XX}. She purchases 100 FAIL_{PM} for 0.008 ETH to hedge the ETH_{XX}. She deposits 100 ETH_{XX} and 100 FAIL_{PM} into a smart contract. How much should her account be credited for (in ETH_{L1})? Smart contracts can hold deposits 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) (and if the RBlock fails, the cost of redeeming FAIL_{PM} is the same). 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.

Next, consider a smaller amount like 0.05 ETH_{XX} (less than $100 USD at time of writing). Now the gas costs are more dominate. $F_0 = 0.04186$ ETH and FAIL_{PM} still costs 0.008 ETH. Alice ends up with 0.03386 ETH_{L1} which is only 67.7%. This demonstrates that fast exits are expensive for withdrawals of amounts in the hundreds of dollars.

4.2 Where to Solve Problem?

- Exit to a third party collateral contract, implements trading

4.3 Withdrawal Format

- Divisible exits
- ERC20 / ERC721? -> allowances or not?

4.4 Non-Substitutable Withdrawals

- Illiquid tokens
- NFTs (substitute for ETH -> support)
- Messages (oracle updates, read calls, any L2-to-L1 message...) (still offer insurance)

4.5 Assertion Failures

- does exit go away with the assertion or not? Pros/cons

4.6 Markets

- Can't use an AMM
- Can't run out
- Simple auction
- Divisible exits
- Exit pools

References

1. Clark, J., Bonneau, J., Felten, E.W., Kroll, J.A., Miller, A., Narayanan, A.: On decentralizing prediction markets and order books. In: Workshop on the Economics of Information Security, State College, Pennsylvania. vol. 188 (2014)
2. Demirag, D., Clark, J.: Absentia: Secure multiparty computation on ethereum. In: International Conference on Financial Cryptography and Data Security. pp. 381–396. Springer (2021)
3. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Off the chain transactions. IACR Cryptol. ePrint Arch. **2019**, 360 (2019)
4. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: USENIX Security Symposium. pp. 1353–1370 (2018)
5. McCorry, P., Buckland, C., Yee, B., Song, D.: Sok: Validating bridges as a scaling solution for blockchains. Cryptology ePrint Archive (2021)
6. Moosavi, M., Clark, J.: Lissy: Experimenting with on-chain order books. arXiv preprint arXiv:2101.06291 (2021)
7. Sztorc, P.: Truthcoin. peer-to-peer oracle system and prediction marketplace. (2015)
8. Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, E., Moreno-Sanchez, P., Kiayias, A., Knottenbelt, W.J.: Sok: Communication across distributed ledgers. In: International Conference on Financial Cryptography and Data Security. pp. 3–36. Springer (2021)