Shared Sequencing: Defragmenting the L2 Rollup Ecosystem

As Ethereum has grown, its scalability worries have gone from theoretical to practical. Especially during periods of high network activity, many users have been priced out of sending transactions over the Ethereum blockchain. A solution to this problem is moving the execution of transactions off-chain through rollups. At a high level, rollups outsource a L1 blockchain's computation to a single party, which in turn is tasked to prove to the L1 blockchain that the computation was performed correctly.

There are at present two main methods by which this proof is performed. In optimistic rollups it is done through fault/fraud proofs, and in zk-rollups through validity proofs. The popularity of rollups as a scaling solution can't be understated, with a proliferation of rollups being developed, ranging from ZK to optimistic rollups and EVM to app-specific VMs. This trend is likely to continue, supported by a host of new rollup-as-a-service startups, and Ethereum rolling out its rollup-centric roadmap.

While rollups are an excellent solution for inheriting some form of economic security at scale, they introduce two key new problems:

1. By relying on a single party for transaction ordering and inclusion in a rollup, they are prone tomonopoly pricing and censorship.
2. The proliferation of multiple rollup solutions breaks composability within the Ethereum ecosystem. Liquidity will fragment between rollups, and assets and other data will have a hard time moving across domains.

Decentralizing the sequencer, which is the component of a rollup in charge of transaction ordering and inclusion, may help with (1), but does little to solve (2) on its own. The Espresso Sequencer, which is not only decentralized, but also shared between rollups, addresses both problems. In this post we focus on the benefits of shared sequencing and how it defragments the L2 ecosystem.

Bridging

Bridging—the ability of separate blockchains (or in this case rollups) to exchange messages—is essential to building a good experience for rollup users. Let's first examine how a shared sequencer benefits cross-rollup bridging through a simple example: cross-rollup token transfer. Then we'll examine more complex use cases involving cross-rollup atomic transaction bundles. We will compare scenarios in which multiple rollups are using independent sequencers vs a shared sequencer.

Review: non-custodial bridge

Let's first review how non-custodial bridging works across two independent blockchains (i.e., state machines with independent consensus protocols). We use the term chain as a shorthand notation to refer to a complete blockchain.

- chain A transaction t prepares a message for chain B. Example: "Lock ETH in a special escrow contract on chain A."
- chain B transaction t' verifies and acts on the message from chain A. Example: "Mint wrapped-ETH on chain B."

Let's begin with the chain A transaction t. There are two steps to finalization on chain A:

- Sequencing: chain A nodes must come to consensus on the ordering of t relative to other chain A transactions.
- Execution: chain A nodes must compute the effect of t on chain A's state.

The above process of sequencing and execution is repeated for transaction t' on chain B, The validity of transaction t' will depend on the result of executing transaction t successfully on chain A. In other words, the A-to-B bridge is possible only if chain B is capable of verifying light-client proofs for both sequencing and execution for chain A. For the bridge to be two-way, chain A must also run light-clients for sequencing and execution on chain B. For example, this would be necessary to

eventually return the ETH from chain B to chain A.

In general, implementing such non-custodial bridges is complex. The execution light-client is similar to a rollup (optimistic or zk), but further complicated by the time-sensitivity of bridge transactions.

Cross-rollup bridging functions similarly to cross-chain bridging, except that the two rollups may or may not share the same consensus protocol for sequencing and/or method of supporting execution light-clients (e.g., verifying zk proofs vs optimistic with fraud proof waiting periods).

Cross-rollup bridges with shared sequencing

Development savings

How would a shared sequencer affect the cost of infrastructure for cross-rollup interoperability? The rollups no longer need to implement separate sequencing light-clients for each rollup they interoperate with. Each rollup reads from the shared sequencer, which provides a global ordering over all transactions. Reduced complexity also has security benefits, reducing the attack surface and opportunity for mistakes. This is especially useful as more and more rollups join the shared sequencer over time. On the other hand, it does not eliminate the need for each rollup to support execution light-clients for bridging to other rollups.

Operational savings

In addition to the upfront costs of integrating sequencing light clients, each rollup must also pay an ongoing cost to verify every other rollup's sequencer state. This comes for free in a shared sequencer. The overall volume of transactions handled by the single shared sequencer is higher, but this can be advantageous as well: the available bandwidth in high-throughput consensus protocols is often underutilized, in which case packing together transactions from multiple domains may amortize costs.

Bridge latency

In a bridge transaction between two rollups A and B using independent consensus protocols for sequencing (e.g., locking tokens of type X on A and minting tokens of type wrapped-X on B), the operation on rollup B (e.g., minting wrapped-X) must wait for the consensus protocol of rollup A to finalize an operation on rollup A (e.g., locking X). This latency is dependent on the type of consensus protocols that are used for sequencing on each rollup. While some consensus protocols have fast best-case finalization latency, all decentralized consensus protocols have high worst-case latency. But when rollups share a sequencer bridge, transactions do not incur additional latency. Like all other transactions within a single consensus protocol, cross-rollup transactions can be chained before finalization. In the example, if the locking of X tokens on A ultimately reverts then so does the minting of wrapped-X tokens on B. As a caveat, execution light-clients can also introduce latency, particularly optimistic light-clients based on fraud proofs. Light-clients based on zero-knowledge proofs have the potential for low latency.

Security benefits

There are systemic risks arising from bridges between independent blockchains, or similarly, between rollups using independent consensus protocols for sequencing.

As an example, consider a bridge that is used to wrap tokens from rollup A to rollup B. When *y* units of a token of type X (e.g, ETH) are deposited into the bridge contract on rollup A , or "locked", *y* units of wrapped-X are then minted on rollup B. In the reverse direction, *y* units of X can be withdrawn from the bridge contract on rollup A, or "unlocked", when *y* units of wrapped-X are burned on chain B. The transaction that mints *y* units of wrapped-X on rollup B waits for the transaction locking *y* units of X to become finalized on rollup A.

If an adversary manages to compromise the finality of transactions on rollup A, reversing the transaction locking *y* units of X into the bridge after it had already been recognized as finalized by rollup B, the *y* units of wrapped X will still remain on rollup B. While this is not technically incorrect behavior for rollup B and its sequencing protocol, this has consequences for users of wrapped-X on rollup B that can be exploited by the adversary, lowering the cost and increasing the incentive to carry out this

attack:

1. The adversary can double its effective holdings of token X by depositing into the bridge, minting wrapped-X, and then stealing back its locked tokens from the bridge. Naturally this devalues the wrapped-X, but the adversary may have already dumped its holdings of wrapped-X into a DEX on rollup B before carrying out the attack.

2. The adversary may bet[^1] against the price of wrapped-X on rollup B and then carry out the same attack, reversing a large volume of deposits (by other users) on rollup B to devalue wrapped-X on rollup B. The adversary does not need to start with any units of token X on rollup A itself to profit from this attack.

The same attack would not be possible if both rollups were sharing a sequencer. The states of these two rollups are now derived from the same transaction log, which is determined by a single consensus protocol. Thus, any attack compromising this consensus protocol that reverts the locking of X tokens on rollup A by forking the finalized transaction log would also revert the subsequent minting of wrapped-X tokens on rollup B.

Of course, there are similar ways in which an adversary may profit from reverting transactions within a single consensus protocol. For example, the adversary can simply send ETH to a centralized exchange, sell it for dollars, and later reverse the transaction that sent ETH to the exchange. Assuming the centralized exchange operates over the traditional banking system, this is essentially a form of bridging between two domains. The attacker would still hold dollars in its bank account after the blockchain transaction reverts. On the other hand, there is more friction and accountability that makes this attack difficult: the exchange and banks know the identity of the attacker and can potentially cooperate to recover assets if the customer can be linked to the attack.

In summary, the seamless and non-custodial nature of DeFi has many efficiency and transparency benefits, but creates additional systemic risk when coupled with bridges between independent consensus protocols. As the L2 ecosystem becomes fragmented into many independent rollups, a shared sequencer among them can greatly mitigate these risks.

Atomic cross-rollup transactions

Cryptographically guaranteed cross-rollup atomic bundles

A bundle of transactions is called “atomic” if it comes with an “all-or-nothing” guarantee—either every transaction in the bundle executes successfully or none do. The cross-chain (or cross-rollup) bridging discussed in the previous section is a special example of cross-chain (or cross-rollup) atomicity.

Another example of an atomic cross-rollup bundle is an asset swap: Alice delivers USDC to Bob on rollup A; Bob delivers ETH to Alice on rollup B. We want either both or none of these transactions to succeed. One way to achieve this atomicity, similar to bridging, is through an escrow contract on each rollup. Alice locks her USDC inside a contract on rollup A that will release her USDC only if either (i) it sees light-client proof from rollup B that Bob locked the ETH in escrow on rollup B, or (ii) a timeout has elapsed with no action from Bob, so Alice can recover her USDC. Bob does the same on rollup B.

This generalizes to arbitrary atomic bundles across more than two rollups. For each rollup C in the bundle, someone locks the relevant piece of C’s state in an escrow contract and creates a light-client proof of this fact for other rollups. The C-portion of the bundle proceeds after the escrow contract has seen light-client proofs from the other rollups for the rest of the bundle, or it reverts after a timeout. Temporarily locking state is fundamental for atomic execution of cross-chain or cross-rollup transaction bundles.

What remains is a fairness problem relating to optionality. The party that acts first gives the other party a free option to unilaterally abort the transaction. Suppose Alice signs and publishes her rollup-A transaction t independently of Bob’s rollup-B transaction t'. In this case, Alice’s transaction might get sequenced long before Bob’s, leaving Bob with the option to abort the trade for a long duration. This option could be quite valuable during times of high price volatility!

Shared sequencers can help mitigate this fairness problem. Suppose instead that Alice and Bob could each sign the pair (t,t') of transactions, so that the shared sequencer either includes the pair as a single transaction in the log or neither. Both rollups will read from this unified log and execute their respective transactions (t by A and t' by B). While atomic inclusion in the log alone does not guarantee a joint outcome (e.g., Bob could have inserted a transaction earlier in the log that causes

t' to throw an error while t still succeeds) the result is entirely determined by the actions of all parties up until the point the pair(t,t') is finalized in a position in the log. This alleviates the fairness problem: neither party has a free long-duration option to pull out of the deal. Utilizing a cryptographic protocol for fair exchange of signatures can further enable fair atomic transactions.[^2]

Consider the following example to show why these fair atomic transactions between rollups are valuable: suppose Alice has 101 USDC and wants to buy ETH. ETH is trading for 101 USDC on rollup A (via Uniswap or similar) and 99 USDC on rollup B. If Alice can find a counterparty on rollup B, she can get a better price for ETH than if she were to swap USDC for ETH directly on chain A. Suppose Bob has 1 ETH on rollup B and is looking to swap into USDC. Bob may be willing to accept 100 USDC for 1 ETH (1 USDC more than the price on chain B), a mutually beneficial price for both parties.

Alice and Bob can engage in a cross-rollup swap where Alice sends 100 USDC to Bob on rollup A and Bob sends 1 ETH to Alice on rollup B. Crucially, if Alice and Bob participate in a fair atomic transaction, they will lock in the agreed-upon exchange rate—100 USDC for 1 ETH—and avoid any optionality. No matter what happens on rollups A and B after that time, Alice and Bob will both settle the trade at their desired rates. This shows how rollups on a decentralized, shared sequencer practically share liquidity between each other.

We have now explored one way in which the Espresso Sequencer, combined with fair exchange of signatures, can address the fairness problem for atomic cross-rollup transaction bundles. However, this does require that chains utilizing the Espresso Sequencer natively support signed cross-rollup transaction bundles as described above. (If a rollup does not support such bundles then perhaps that rollup's stakeholders might consider tweaking its execution rules to allow it.)

Cryptoeconomically assured cross-rollup atomicity

While the previously discussed atomic bundles can easily support simple use cases like CoW swaps and NFT trading, some bundles may not be so simple. For example, consider the above token swap example with a complication: Bob does not yet have ETH on rollup B—instead he must exchange USDC for ETH on rollup B before delivering the ETH to Alice. Bob wants to engage in this bundle with Alice only if he is guaranteed a USDC profit from the deal. The atomic bundle is now:

1. Bob pays at most 99 USDC for 1 ETH on chain B. Suppose this purchase is executed on a decentralized exchange such as Uniswap.
2. Bob delivers the 1 ETH to Alice on chain B.
3. Alice delivers at least 100 USDC to Bob on chain A—and Bob earns an arbitrage profit of (at least) 1 USDC.

In this example the Uniswap trade is part of the atomic bundle. Whereas in the previous example Bob locked only state that *he controls* (namely, his ETH in the rollup B escrow contract), now he must also lock *the entire state of Uniswap on rollup B*.

To what extent could the Espresso Sequencer enable this type of cross-rollup arbitrage? Or more generally, to what extent can a shared sequencer enable arbitrary conditions on cross-rollup transaction execution? Bob cannot lock Uniswap by himself—he needs help. We want Bob to be able to stipulate exactly under what conditions his bundle should execute. If the conditions are not met, the bundle will simply not be included in the chain (and he will not even pay gas!). For example, he might specify the condition, "Sequence this bundle only if step 1 would execute with price at most 99 USDC."

Because sequencer nodes aren't typically aware of the state of individual rollups they are sequencing transactions from, we can introduce a third party that is aware of all the rollup states to help out. These parties are called builders (e.g., SUAVE or sequencer nodes opting in to providing this service), and they will be tasked with creating blocks that satisfy the conditions laid out by Bob and other rollup users. This is known as proposer-builder separation, or PBS. Combined with a shared sequencer, PBS grants new interoperability superpowers. Consider the following naive example of how a user might work with a builder to perform a cross-rollup arbitrage involving Uniswap.

Bob and the block builder could sign an attestation of this stipulation. Thus, if the block builder violates this condition (so that the bundle executes but Bob paid more than 99 USDC for ETH at Uniswap on chain B) then Bob has cryptographic proof of malicious behavior from the block builder. This proof could be used to penalize the malicious block builder by, say, slashing a security deposit posted by the block builder or by paying Bob restitution from that security deposit. The block builder can also be rewarded if they fulfill their end of the bargain. While this type of interoperability does rely on a third party such as a

block builder (or an external protocol instantiating the builder), it doesn't require any in-protocol changes on the rollup's behalf, and provides powerful composability between rollups sharing a sequencer.

Without a shared sequencer, there is no easy way to enforce that all transactions in a bundle are sequenced at the same time, or atomically at all. In order for the builder to uphold its side of the deal without a shared sequencer, *the block builder must either have simultaneous control of the sequencers for multiple independent consensus protocols, or risk only including part of the bundle*. In particular, the block builder risks being slashed or forgoing their reward, even if they are correctly performing their job.

The Espresso Sequencer can enforce that a builder's block will execute as intended in its entirety, or not at all. This makes it potentially far more rewarding for block builders to focus their efforts on rollups that use the Espresso Sequencer, compared to those that use a standalone sequencer. Likewise, it may drive far more activity in general to rollups that share the sequencer compared to those that don't.

Closing thoughts

We have explored a number of different ways in which the Espresso Sequencer serves to defragment the layer 2 landscape. We expect the ability to connect liquidity and applications between rollups to greatly increase the utility of individual rollups and provide better UX for their users, both by enabling cryptographically secured atomic bundles that don't rely on third parties, and more expressive atomic bundles that rely on block builders (who may provide additional cryptoeconomic guarantees).

For more on the Espresso Sequencer, you can read here and follow Espresso Systems on Twitter.

[^1]: There are many possible ways to bet against the price of wrapped-X assuming there is a market for this (e.g., taking a conventional short position). Alternatively, if wrapped-X is trading against a token Y in an AMM on rollup B then a sudden decrease in the true value of wrapped-X can create a massive arbitrage opportunity. This primarily benefits block builders and/or proposers on rollup B who have priority in taking advantage of DEX arbitrage.

[^2]:https://crypto.stanford.edu/~dabo/pubs/abstracts/timedcommit.html, https://eprint.iacr.org/2012/288.pdf