Layer-2 Interoperability in Blockchain: Towards a more Secure and Performant Interoperability Verification DAO

Siddharth Srinivasan
Dept. of Computer Science
University of Colorado, Boulder
sisr9857@colorado.edu

***Abstract*—L2 based projects serve as the key to offloading transaction processing away from the main network, thereby increasing scalability for blockchains. Recent interoperability solutions provide a seamless mechanism to transfer funds across L2s in a faster, efficient, and inexpensive manner. While these solutions offer mechanisms to ensure truthfulness of transactions, they do not offer guarantees of transaction legitimacy and non-repudiation. To address these concerns while being efficient and performant, we propose a peer-to-peer verification system in the form of a Decentralized Autonomous Organization (DAO) which acts as an intermediary between cross L2 transfers.**

I. INTRODUCTION

As the blockchain ecosystem continues to expand, the popularity of these networks affects the network bandwidth, processing throughput and response times as well. The core network of Ethereum (also referred to as Layer 1) suffers from a bottleneck termed scalability trilemma [1]. Proposed by Ethereum's co-founder Vitalik Buterin [2], the trilemma states that a blockchain network can satisfy at most two of the three goals – security, scalability, and decentralization. Traditional Blockchains like Bitcoin [3] and Ethereum rely on full nodes to verify every transaction, hence satisfying decentralization and security without scalability. Sharding is one solution employed by Ethereum 2.0's Phase 1 rollout, but improves scalability only to an extent limited by Ethereum's saturation on network efficiency improvements.

Several solutions are currently leveraged to minimize the strain on the base blockchain network (also known as *Layer 1* network, or simply L1) by scaling workloads and processing transactions off-chain. These solutions are rightly called the *Layer 2* (L2) category of protocols. There exist different types of *Layer 2* scaling technologies, the popular ones being sidechains and rollups. These typically work by processing transactions off-chain, batching them up and submitting them in a single shot to the main Ethereum chain. This is advantageous because it not only offloads processing and improves throughput of the *layer 1*, but also minimizes any gas fees incurred as well. Several DeFi protocols and decentralized apps (dApps) are currently cooperating with L2s for the same reason. This growth brings about the problem of L2 interoperability, i.e. transferring funds efficiently between different L2s.

In view of this, L2 interoperability solutions have been on the rise. These aim to support transferring funds between L2s while minimizing the interaction with L1 during the transfer(s) through different methods. StarkEx is one such solution powered by STARK [4], an L2 network popularized for its use by Zero-Knowledge Rollups (zkRollups). At its core, StarkEx uses Liquidity Providers (LPs) to manage transfers via L1s to other L2s by batching only those transactions that have met its pre-conditions (hence known as conditional transactions). Polygon's Hermez [5] works on the idea of batching all those transactions that are destined to the same L2 address together and withdraw all funds from these transactions in a single shot to L1 and deposit to L2 simultaneously. Connext [6] uses the concepts of state routing that aims to bridge directly to other L2s without dependence on L1.

In all these cases, the truthfulness of these transactions are determined by the L2 is ensured only by the L2 scaling solution, typically by solving and issuing fraud proofs [7] or validity proofs [7]. These L2s however do not attempt to verify the legitimacy of the latter L2s involved during a transfer, and do not account for non-repudiation for the account residing in the L2s (i.e., the assurance that the L2 accounts cannot deny that the transaction took place). In purview of minimizing verification complexities across transfers through disparate L2 channels, we propose a layer of intervention in the form of an EVM-compatible, broker-based DAO system that verifies such interoperable transactions and provides legitimacy and non-repudiation as required.

II. PROPOSAL

Using a DAO to achieve such verification tasks can be advantageous in many ways:

- **Decentralization**: Supporting peer-to-peer coordination across stakeholders and entities enables multilateral movements and can minimize the barrier of entry into the DAO.
- **Autonomy**: The system governs itself. Same goes for the entities involved. A DAO can conveniently establish such governance through a set of rules codified in smart contracts.
- **Organization**: With the privilege of autonomy, comes the requirement of consensus for a variety of DAO-related activities, including but not limited to (a) voting on contract proposals, (b) resolution of challenge(s) and disputes while verifying transaction authenticity, and (c) incentivizing (and discrediting) contributors, verifiers and other L2 entities in lieu of their activities within the system.

One possible solution is for the DAO to expose APIs for legitimacy verification to be consumed by L2s, like the JSON-RPC mechanism exposed in L1 Ethereum [9]. This however reeks of major limitations:

- There is no means of guaranteeing non-repudiation.
- There is no guarantee that the primary $L2_1$ will execute the verification API for $L2_2$, (and vice-versa).

To reinforce such guarantees of legitimacy, we propose reusing the JSON-RPC communication to send all

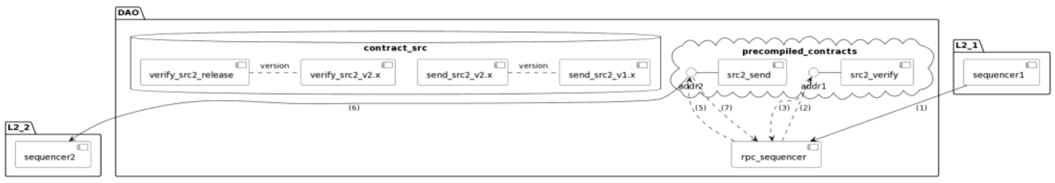

Figure 1: Components of the proposed verification DAO

transactions to the DAO using a different workflow which ensures transaction validity and integrity.

A. Architecture

Figure 1 presents the components involved in the DAO and a sample workflow involving fund transfer between two L2 solutions, $L2_1$ and $L2_2$. Summarizing the components, we have:

- **DAO Sequencer**: Queues and handles receive and dispatch of transactions to the verification contracts and ultimately the destination account at $L2_2$. It is desirable if the sequencer utility is split into a receiving sequencer for incoming requests, and a dispatch sequencer to submit the funds to $L2_2$.
- **Source container**: Stores and maintains the source code of the contracts involved through file versioning mechanisms.
- **Precompiled Container**: Maintains the recent precompiled version of the contracts at locally fixed DAO addresses.
- **Node**: Peer-to-peer component(s) that serve as the DAO backbone and run services, hold fixed contract addresses, represent stakeholders, provide the runtime for contract execution and dispute resolution.

B. Verification Workflow

Following actions are expected to occur in the sample workflow presented in Figure 1:

1. $L2_1$ submits a transaction batch to the DAO using JSON-RPC as the communication mechanism.
2. DAO Sequencer forwards the batch to a contract that performs sanity on the receiving batch, and then another contract that does the actual verification.
3. If everything goes well, the sequencer runs the contract responsible for initiating the transfer to the destination address at $L2_2$.

C. Entities

Sovereignty in the hands of a few is certainly disregarded. That said, certain entities precede over others depending on their level of involvement to ensure the stability of the system. These entities take the form of:

- **Primary Stakeholders**: Those having an active and a rather high degree of involvement fall in this category. This would include (but not limited to) protocol developers, project and system maintainers, core members and document/proposal champions.
- **Secondary Stakeholders**: Encompasses project contributors, transaction verifiers and validators in the system.
- **Layer 2 entities**: Refers to external protocols who leverage our DAO solution. These entities also receive a stake depending on their level of involvement with the system.

Such entities described above are incentivized with native tokens, proportionate to their level of involvement and contribution within the system.

III. GOVERNANCE

Entities are ultimately responsible for upholding consensus within the DAO. To achieve this, the two core mechanisms that entities must use are:

- End-to-End Verifiable Voting System(s) for supporting a L2 integrations, approving a smart contract and system re-design(s).
- Dispute resolution mechanism for challenging the demerits of a given verification contract.

We now explore each topic in depth.

A. Verifiable Voting

For every L2 provider that plans to integrate his protocol with our DAO, stakeholders are required to establish consensus on the legitimacy and integrity of the L2. One approach is to deploy a contract that performs authenticity verification and validates the overall functioning of the L2. This is exhaustive due to couple of reasons:

- The L2 operates under different primitives which could drive code complexity of the validation contract, and hard forks/revisions need to be accounted for as well.
- Having a lower number of contracts to test a wide range of L2 solutions will not only introduce program complexity, but is inefficient and could be severely error-prone as well.

Due to the reasons stipulated above, we propose that only the primary stakeholders are involved in the vetting process through means of electoral voting. A primary stakeholder can incur penalties in this process if

- He stays neutral by voting on both sides.
- His vote goes against the majority in an unbalanced proportion.
- He avoids voting in the first place.

A voting scheme similar to the rules proposed above can be be leveraged to accommodate other relevant scenarios, such as:

- Approving a verification contract for a given L2.
- Handling dispute resolutions.

- electing an improvement on the existing infrastructure, or voting on a fork towards major change(s). Secondary stakeholders can also be included for voting on few of these situations as applicable.

B. Dispute Resolution

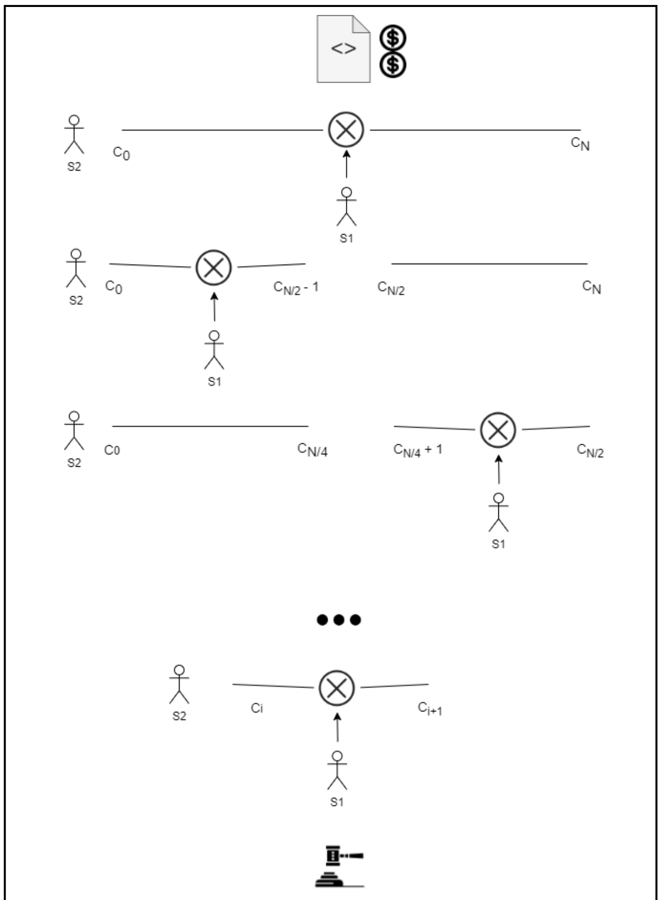

Figure 2: Source Contract Bisection

Handling challenges and debates will require a mechanism for formal resolution. Possible disputes include challenging:

- The correctness of a contract in verifying transaction legitimacy.
- The compute and bandwidth that is optimal for executing the contract, without affecting the DAO as a whole.
- The speed of execution, and gas fees incurred while executing such a contract against L2 transaction(s).

One such possibility is that we can utilize the bisection algorithm used by Arbitrum [10]. **Figure 2** provides the rough sketch of the bisection workflow. The procedure would be useful for ensuring contract correctness, and could look like the following sequence of events:

- Suppose a certain stakeholder S1 challenges the correctness of a verification contract developed by another stakeholder S2, and the contract has N instructions of execution.
- Both S1 and S2 withdraw native tokens as a deposit (B1 and B2). The stakeholder who wins the challenge in the end gets back his share plus half the other's share of the deposit. The remaining deposit is used as gas fee on supporting the bisection algorithm.
- Once the above assertion is made, the challenger must challenge one of the assertions before this activity undergoes recursion on the bisected code section even further.
- Care must also be taken that the challenger must be able to select steps that occur in between assertions since the challenge could encompass instructions residing in both sections of code.
- Once a challenge for a code snippet is finalized, stakeholders vote on their decision, i.e. to accede or deny the challenge.
- Depending on the outcome, the challenger is either provided with incentives from the deposit or penalized in the process otherwise by losing his share of the deposit.

The time complexity of this activity is $O(\log_2 N)$, excluding the time it takes for S1 to issue a new challenge to the respective code stub.

What if the contract was challenged for being inefficient, slow, and/or draining the system compute and resources? We have two options to solve this situation:

- We perform the bisection as discussed above on the code snippet that is inefficient, and trigger a round of voting to decide consensus.
- We execute the contract with the default runtime. As we execute the contract, we capture the amount of CPU, memory used, and the amount of network bandwidth consumed. Based on these metrics obtained, the challenger reasons out to the system in case one of the readings is off-limits. We can run a pre-compiled contract that analyzes the metrics and decides if it's within the range of a suitable working environment. If this is not the case, the challenger is incentivized, and the contract is no longer used.

IV. Verification Contracts

A core functionality of the DAO is to pre-compile the verification contract(s) for a given L2 and maintain them, due to the following benefits:

- **Separation of Concerns:** It provides loose coupling between the contract's logic and the data that it depends on.
- **Performance:** Contracts in native runtime languages such as C++ boast a performance improvement in contrast to their solidity equivalent. Parallelism and multi-threading support can be leveraged for improved efficiency of processing.

Instead of residing at a fixed address on the Ethereum chain, we propose that the object file corresponding to the contract be cached within our DAO for off-chain execution instead, hence maintaining a pre-compiled contract. This executes the contract using the default node runtime instead of using the less-performant EVM.

In addition to this, we also propose to cache the source file(s) corresponding to the pre-compiled verification contracts in such a way that changes to the contracts are recorded. This is desirable since we can simply revert to the previous version of the contract if the execution of the currently deployed version is undesirable. Since saving file copies at different modification times incurs a lot of I/O for writing and demands more storage, we can leverage a versioning file system instead. A versioning system ensures that the history of changes recorded for a file is captured in a space-efficient manner. In our case, the SEV versioning system proposed by Xunjie Li [11] is optimal for our use case, since it proposes augmenting file inodes using a copy-on-write (CoW) mechanism to track file changes. This approach boasts of space efficiency while achieving reasonable file storage and retrieval speeds as well.

We could also utilize the concepts outlined in Versionfs [12], since this retains Unix properties such as ownership, permissions etc. in a way that the DAO stakeholders can enforce policies which dictate access, control and file recovery and version retention as well.

References

[1] Mirko Bez, Giacomo Fornari and Tullio Vardanega, "The scalability challenge of ethereum: An initial quantitative analysis" IEEE SoSE vol 1, April 2019.

[2] Vitalik Buterin, The Scalability Trilemma, "Why sharding is great: demystifying the technical properties", https://vitalik.ca/general/2021/04/07/sharding.html, April 2021.

[3] Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf, Dec 2010.

[4] Eli Ben-Sasson, Iddo Bentov, "Scalable, transparent, and post-quantum secure computational integrity" J. Name Stand. Abbrev., in press.

[5] Polygon lightpaper: Ethereum's internet of blockchains. https://polygon.technology/lightpaperpolygon.pdf, Dec 2021

[6] Connext Network, https://www.connext.network/, Oct 2021

[7] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin, "Fraud Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities", Sep 2018.

[8] The Validity Contributors, "Validity: A Digital Value Transfer and Information Verification System", Jan 2019

[9] JSON-RPC 2.0 Specification, https://www.jsonrpc.org/specification, Mar 2010.

[10] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen "Arbitrum: Scalable, private smart contracts", Proceedings of the 27th USENIX Security Symposium, Aug 2018.

[11] Xunjie (Helen) Li, "SEV: a Storage-Efficient Versioning File System", Strauss T12, Mar 2013

[12] Kiran-Kumar Muniswamy-Reddy, Charles P. Wright, Andrew Himmer, and Erez Zadok, "A Versatile and User-Oriented Versioning File System", Proceedings of the Third USENIX Conference on File and Storage Technologies, Apr 2004