Introduction

delta is a permissionless network of networks, such that each member network – called a *domain* – is securely and efficiently interoperable with any other one. Each end user has a *vault* (see *Account* section) associated with each domain. A vault can only spend through its associated domain, but it can receive assets from any other domain.

The network connecting the domains is operated by validators and will be referred to as the *base layer*. Validators disseminate and process transaction data from the domains, validate SNARKs [1] corresponding to this data, and maintain the security of domains and users in the face of malicious actors. Even a malicious domain cannot harm other domains or their users. Any two users from any two domains can transact and interact with each other in arbitrary ways without relying on third party custodians or other intermediaries.

The delta state model is based on CRDTs [3], enabling a highly lightweight and robust base layer protocol.

Execution

End users interface directly with domains. However, if there is domain censorship or downtime, they have the power to exit and move their funds to a different domain via the base layer. This is referred to as a *forced migration*. Consequently, a domain can be operated by a single centralized entity, a permissionless validator set, or anything in between without impacting the property rights of the end user.

The execution model is determined by sets of *global* and *local* laws. The global laws include the inability to spend through a different domain than one's own and asset-specific rules such as black-/whitelisting, minting authorities etc. These apply to every domain, and exist to preserve the security of domains and assets. Local laws are declared by domains upon their deployment, guaranteeing to their users that they will adhere to e.g. only execute a specified set of programs, or that they will run e.g. the EVM or SVM. The set of local laws can be empty, lowering the computational requirements of the domain.

The operator of the domain orders and executes user transactions at a self-defined frequency, setting the transaction latency of the domain. Executing the ordered list of transactions creates a *State Diff List* (SDL), a compressed standardized format for state changes. The SDL is submitted directly to the base layer. Using the transaction data and possibly the execution trace, the operator computes two SNARKs (one if no local laws apply) to show that the SDL satisfies the global and local laws of that domain.

Partial Ordering of SDLs

Due to the asymmetry between spending and receiving assets, SDLs do not need to be totally ordered relative to each other for the resulting state to be deterministic. In fact, they directly provide all the necessary data for an implicit partial ordering.

SDLs have a sequence number, so all SDLs from the same domain are totally ordered with respect to each other. Moreover, each SDL refers to the set of external SDLs it depends on; these are SDLs from other domains whose resulting states are considered inputs to the current one. Collectively, this data generates a directed acyclic graph (DAG) of SDL dependencies, which uniquely determines the state.

Consensus

Transactions submitted to the delta validators follow one of two protocols:

As seen in the previous section, SDLs do not depend on an external protocol to provide ordering. As such, they are disseminated via a consensusless fast path, using *Byzantine Reliable Broadcast* [2]. SDL proofs are also processed in this way.

Only a few other transaction types can be submitted to the base layer. These involve creating and removing a validator or domain, and forced migrations of user vaults from one domain to another. None of these transaction types depend on total ordering either, but the delta validators do run a minimal consensus algorithm which provides epochs, such that these transactions are processed only on epoch boundaries.

Settlement

User-relevant state is updated continuously as per the SDL fast-path and settle upon verification of their associated SNARK(s). Due to the lack of leader-based consensus and blocks, a domain can guarantee the settlement of its SDLs by submitting them through a validator which it runs. Non-SDL and non-SNARK transactions are processed as part of an *epoch transition function* (ETF) at the end of each epoch, i.e. after all SDLs and SNARKs submitted in that epoch. The ETF updates the validator and domain sets.

Assets and Accounts

Assets are part of the global state, contributing to both smooth interoperability and a unified user experience across domains – there is only one implementation of a given asset for all domains.

A delta account consists of a small amount of data stored in the *Account Registry* – this primarily includes the account's public key(s) (corresponding to one or multiple signature schemes, or multisig data).

Additionally, each account holds a *vault* on each domain, in which the account's assets on that domain is stored. Alternatively, users can store their assets locally and only keep the Merkle root in the public delta state, providing privacy benefits.

In practice, users can simply log in to applications (domains) directly with their delta keypair(s) and submit transactions to the domain, using assets which are contained in a vault within the app's part of state.

References

[1] Eli Ben-Sasson et al. *Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture.* 2013. URL: `https://eprint.iacr.org/2013/879`. Pre-published.

[2] Gabriel Bracha. "Asynchronous Byzantine Agreement Protocols". In: *Information and Computation* 75.2 (Nov. 1, 1987), pp. 130–143.

[3] Marc Shapiro, Nuno Preguiça, and Carlos Baquero. "Conflict-Free Replicated Data Types". In: *Stabilization, Safety, and Security of Distributed Systems.* Berlin, Heidelberg: Springer, 2011, pp. 386–400.