FVM + Tendermint = Fendermint

Concepts and Milestone 1 Demo

Akosh Farkash

Mother Of All Demo Days: April 2023

Consensus Lab

Table of Contents

Consensus Lab

Table of Contents

Consensus Lab

InterPlanetary Consensus (IPC)

- https://ipc.space
- Used to be Hierarchical Consensus (HC)
- Subnets are recursive sidechains rooted at Filecoin, where:
  - subnets can have arbitrary consensus
  - messaging with parent and child subnets is IPLD based, using checkpoints
- We need nodes with different consensus than Lotus:
  - Eudico is fork of Lotus running Mir and Trantor
  - Fendermint is using off-the-shelf Tendermint with FVM

Consensus Lab

Table of Contents

Consensus Lab

What is Tendermint?

- a Proof-of-Stake BFT consensus protocol
- the generic blockchain application platform Tendermint Core
- now living under Comet BFT
- very convenient for application developers:
  - instant block finality
  - forward-only with no rollbacks and reorgs
  - does not restrict the implementation in any way
- scales to ca. 100 validators with a default 2 sec. block time
- Rust libraries:
  - tendermint-rs
  - tower-abci
- Tendermint Core != Cosmos SDK

Consensus Lab

ABCI (Application BlockChain Interface)

Calls to the Application from Tendermint during the lifecycle of a block:

- `begin_block`: called with the current block header; run `cron`
- `deliver_tx`: called for each transaction; return receipt
- `end_block`: return changes to the power table
- `commit`: flush all changes to the database; return new state hash

Other methods: `init_chain`, `check_tx`, `query`, `list_snapshots`, ...

Consensus Lab

ABCI (Application BlockChain Interface)

Calls to the Application from Tendermint during the lifecycle of a block:

- `begin_block`: called with the current block header; run `cron`
- `deliver_tx`: called for each transaction; return receipt
- `end_block`: return changes to the power table
- `commit`: flush all changes to the database; return new state hash

Other methods: `init_chain`, `check_tx`, `query`, `list_snapshots`, ...

State Management

The Application must maintain uncommitted state in memory until the `commit` arrives, and a projected one for `check_tx`. Perfect fit for the machinery the FVM comes with!

Consensus Lab

ABCI++

So far the Application received blocks which were already finalized by Tendermint; it had to mark faulty transactions as such and move on. This would be a problem if we wanted to only contain CIDs of transactions in the blocks.

ABCI++ adds new methods[1]:

- `prepare_proposal`: the Application can reorder or replace transactions found in the Mempool before they are added to a block
- `process_proposal`: the Application can inspect all transactions in a block proposed by the round leader and decide whether or not to vote for the block

Consensus Lab

[1] https://github.com/tendermint/tendermint/blob/v0.37.0-rc2/spec/abci

ABCI++

So far the Application received blocks which were already finalized by Tendermint; it had to mark faulty transactions as such and move on. This would be a problem if we wanted to only contain CIDs of transactions in the blocks.

ABCI++ adds new methods[1]:

- `prepare_proposal`: the Application can reorder or replace transactions found in the Mempool before they are added to a block
- `process_proposal`: the Application can inspect all transactions in a block proposed by the round leader and decide whether or not to vote for the block

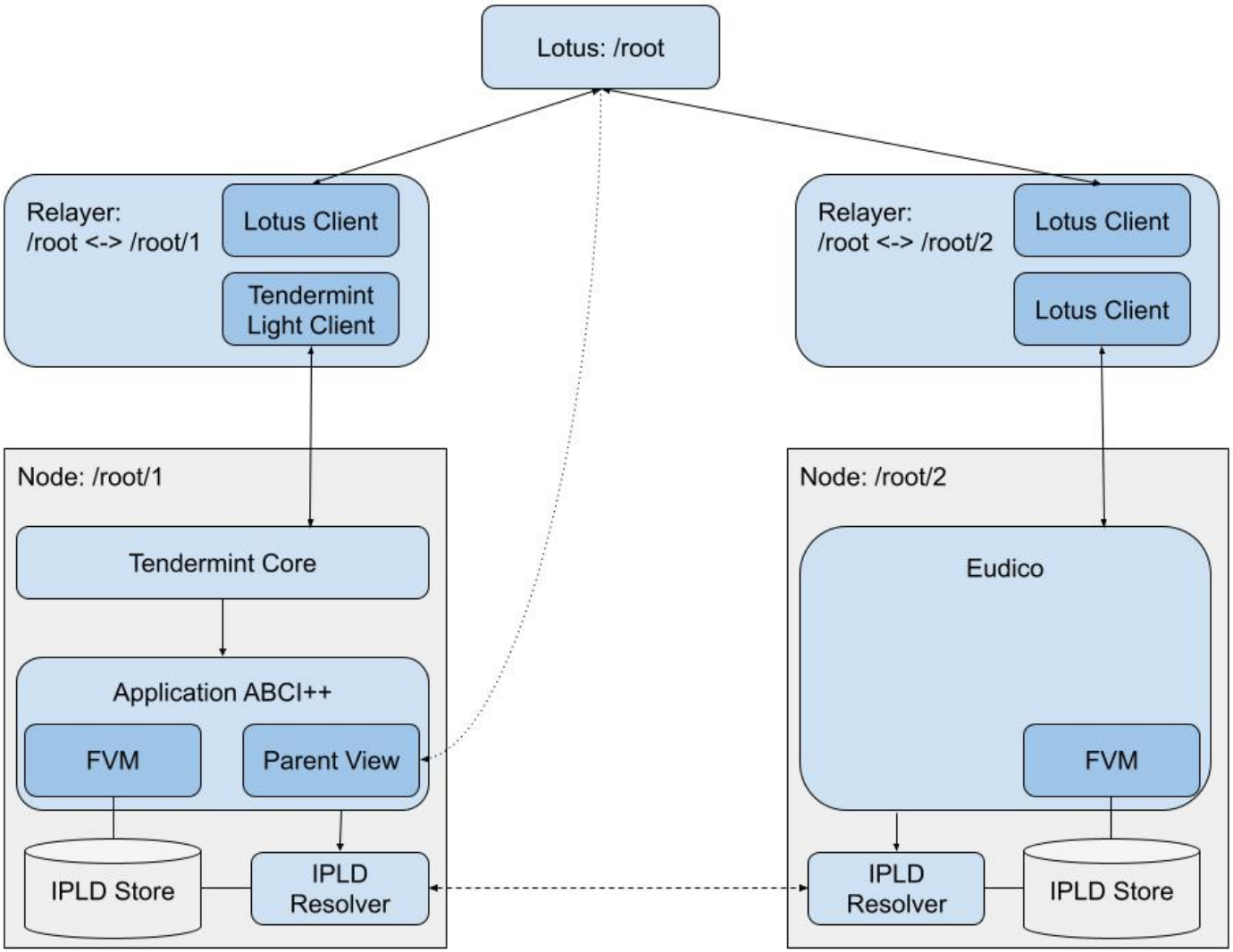
Availability Certificates

We can use this mechanism to reduce transactions to CIDs and only vote on blocks if the content is available.

[1] https://github.com/tendermint/tendermint/blob/v0.37.0-rc2/spec/abci

Table of Contents

Proposed Architecture



Consensus Lab

PoS Sidechain

Fendermint runs as a PoS Sidechain[2]:

- top-down: *direct observation* of the parent (ie. run at least a full node on all ancestor subnets)
- bottom-up: periodic checkpoints signed by the subnet validators, submitted to smart contracts running on the parent subnet

Consensus Lab

[2] P. Gaži, A. Kiayias and D. Zindros, "Proof-of-Stake Sidechains," 2019

PoS Sidechain

Fendermint runs as a PoS Sidechain[2]:

- top-down: *direct observation* of the parent (ie. run at least a full node on all ancestor subnets)
- bottom-up: periodic checkpoints signed by the subnet validators, submitted to smart contracts running on the parent subnet

Observability

Validators need to agree on:

- top-down: the finality of a message on the parent subnet
- bottom-up: the availability of a checkpoint from a child subnet (where the checkpoint only contains a CID)

Consensus Lab

[2] P. Gaži, A. Kiayias and D. Zindros, "Proof-of-Stake Sidechains," 2019

Data Availability

- Checkpoints contain an unknown number bottom-up messages
- sending a checkpoint as a self-contained message wouldn't scale
- we could send a Merkle commitment and relay constituent messages one by one, or we can rely on IPLD resolution and send a root CID.
- Before parent validators commit to executing a checkpoint from a child, they must know the majority of them have its contents; otherwise a single validator could not decide if it's a data availability attack.

Take ideas from NC-Max[3]:

- divide the block space in two:
  1. transactions for *resolution* (asynchronous)
  2. transactions for *execution* (synchronous)
- remove transaction propagation from the critical path

Use ABCI++ `process_proposal` to reject unavailable execs.

Consensus Lab

[3] R. Zhang et al, "NC-Max: Breaking the Security-Performance Tradeoff in Nakamoto Consensus", 2020

IPLD Resolver

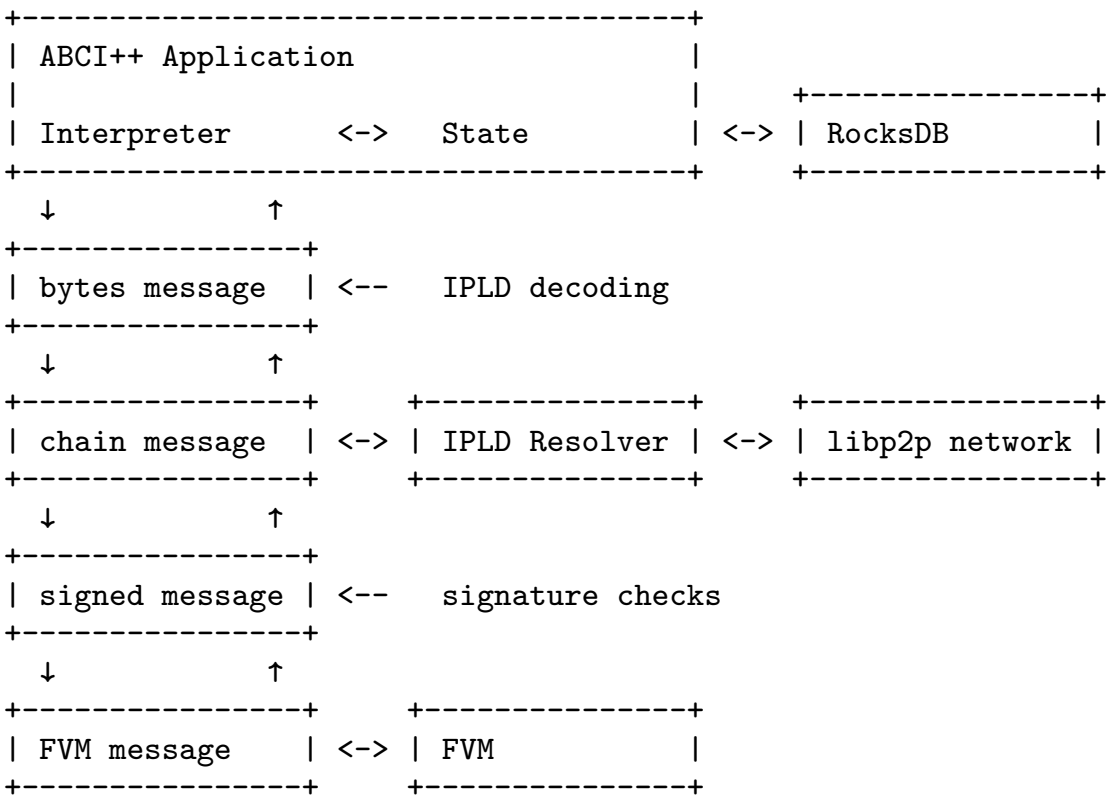
A P2P library[4] we can use to advertise subnet membership and resolve arbitrary content; see the docs for an overview. Much of it was based on Forest. Uses the following libp2p protocols:

- Kademlia for peer discovery
- Identify for exchanging listening addresses
- Gossipsub for
  - advertising subnet membership
  - to propagate votes about CIDs, which can be used to agree on availability (bottom-up) or finality (top-down)
  - to publish content to the parent subnet, pre-empting resolution requests (requires subscription)
- Bitswap for recursively resolving CIDs from a subnet, and putting the data into the `Blockstore`

Consensus Lab

[4] https://github.com/consensus-shipyard/ipc-agent/tree/main/ipld/resolver

Application Architecture

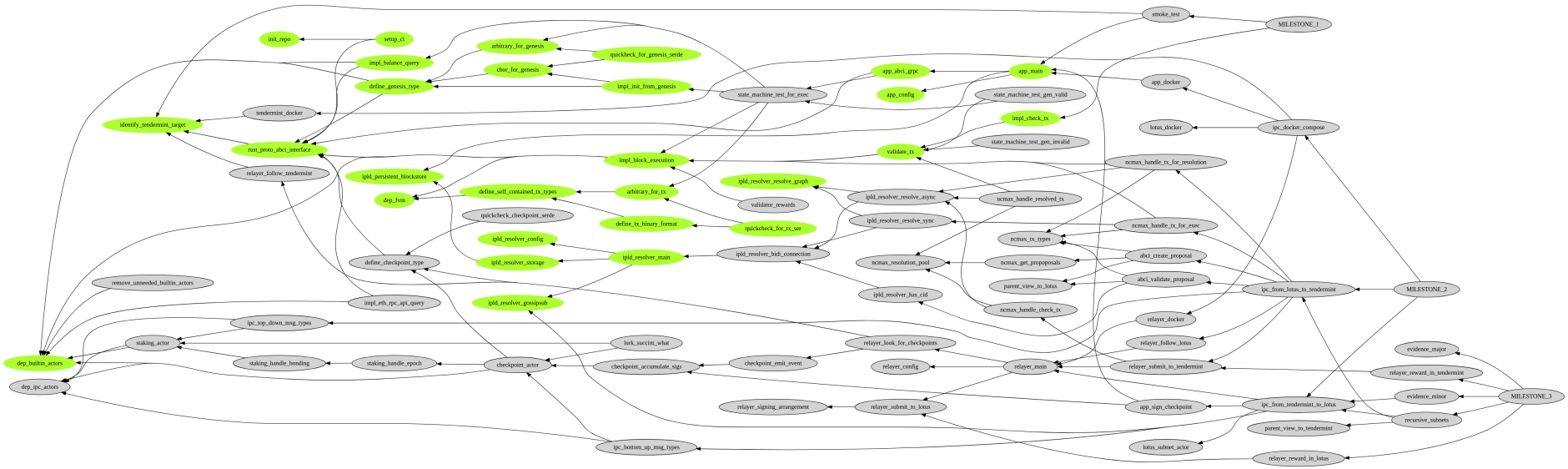


Interpreter Stack

Type of message changes as it passes through short circuiting interpreters.

Milestones

- M1: **Standalone Tendermint with FVM and IPLD**
- M2: Fendermint plugged into IPC under Lotus rootnet
- M3: Recursive subnets, incentives, Ethereum JSON-RPC for FEVM



Consensus Lab

Table of Contents

Consensus Lab

Demo

Current state of affairs:

- CLI commands to:
  - generate keys
  - construct a genesis JSON file with initial accounts and validators [5]
  - run the Application process
- RPC client library and CLI commands to:
  - transfer tokens
  - send IPLD encoded messages to FVM actors
  - create and invoke FEVM contracts with ABI encoded args
  - query the state of an actor
  - get IPLD content by CID

See the docs[6] for examples of using the CLI and simplecoin.rs for statically typed interaction with a FEVM contract through JSON-RPC.

Consensus Lab

[5] built-in actors: `system`, `init`, `eam`, `evm`, `account`, `multisig`, `cron`

[6] https://github.com/consensus-shipyard/fendermint

Questions

Thank you!

https://github.com/consensus-shipyard/fendermint