OmniLedger

Master Semester Project

Pablo Lorenceau
Supervisor: Linus Gasser
Responsible: Prof. Dr. Bryan Ford
DEDIS, EPFL

June 2018

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Table of Contents

Overview

Background

Implementation

Overview

OmniLedger is a highly scalable blockchain which uses

- a non-probabilistic consensus protocol,
- an identity-blockchain to decouple identity establishment from transaction processing,
- sharding to improve throughput,
- an atomic commit protocol for cross-shard transactions
- and assigns nodes to shards in a scure manner.

It scales to performances comparable to VISA.
Our Goal: Implement it!

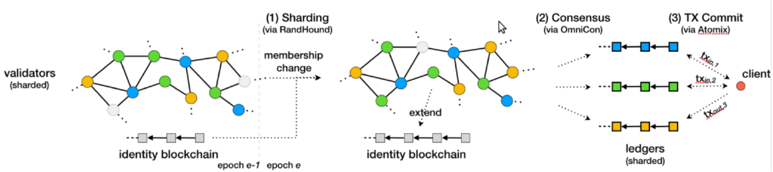


Figure : OmniLedger: Colors indicate shards. (*Kokoris-Kogias et al., IEEE S&P 18*).

Overview of the Implementation

OmniLedger is a system built on to of the skipchain which

- stores state in a Merkle-tree like data structure (collection),
- allows clients to modify state by sending transactions,
- and has different callback functions (contracts) per type of state, called when processing requests.

Clients can request a cryptographic proof about the state of the collection.

Table of Contents

Overview

Background

Implementation

Skipchain

Skipchains are blockchains which

- have more than one backward link,
- have forward links pointing to future block
- and allow clients to traverse the chain efficiently.

Skipchains are also useful for offline verification.

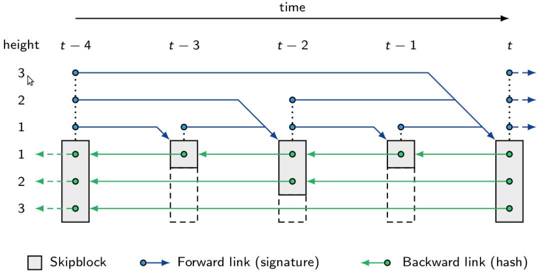


Figure : The skipchain: Backward and forward links span multiple hops, allowing clients to efficiently traverse the chain (*Nikitin et al., USENIX Security 17*).

ByzCoinX

Omniledger use ByzCoinX for consensus:

- No forks occur.
- The root node (leader) proposes a block to the other nodes (validators).
- A block is accepted $\iff \frac{2}{3}$ of the validators sign it.

Note:ByzCoinX allows for only f byzantine nodes out of $3f + 1$ total nodes.

More: Next presentation.

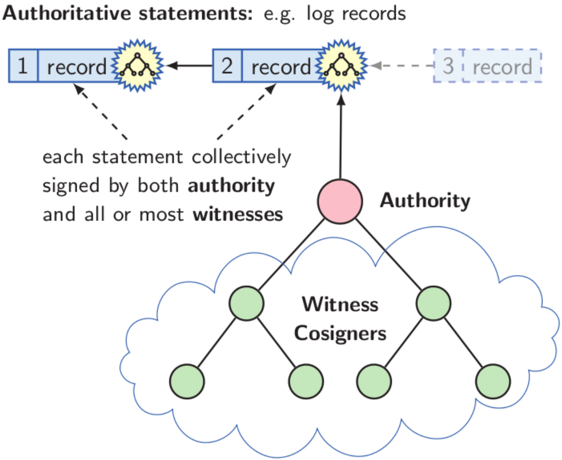

Authoritative statements: e.g. log records
1 record
2 record
3 record
each statement collectively signed by both authority and all or most witnesses
Authority
Witness Cosigners

Figure : ByzCoinX: The root proposes a block to the rest of the nodes which have to collectively sign it. (*Kokoris-Kogias et al., USENIX Security 16*).

Table of Contents

Overview

Background

Implementation

Structures

- Collections
- Darcs
- Transactions
- Contracts

Collections

Collections are

- based on Merkle trees,
- operate as a key-value store
- and can issue proofs about their state, verifiable by any client knowing the Merkle-root of the collection.

More: Next presentation.

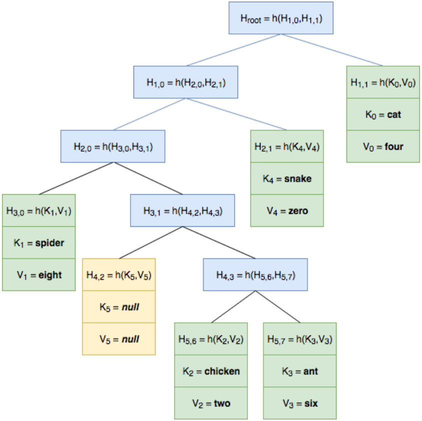


Figure : Collections: The keys are ordered in a deterministc fashion which allows to prove the absence or presence of a given key-value pair. (*github.com/dedis/student_18_omniledger/tree/master/omniledger/collection*).

Darcs

Darcs

- map actions to signature requirements,
- are stored in the collection itself
- and can be evolved by a user with the corresponding permission.

Example:
$update : Jeff \wedge (Linus \vee Kelong)$

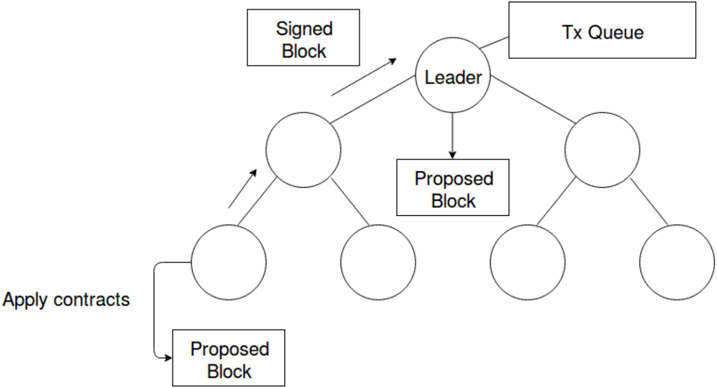


Figure : The structure of a our implementation.

Transactions

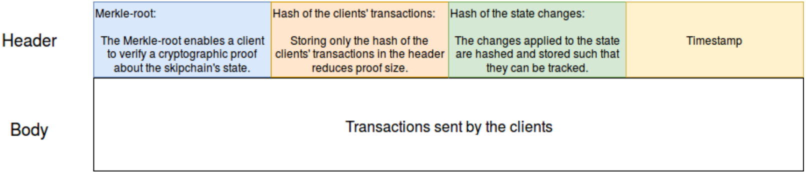
Transactions contain a list of instructions. An instruction:

- can be one of Spawn, Invoke, delete,
- contains a key for the collection,
- a key via the authenticating Darc
- and can effect multiple state changes.

Contracts

Contracts

- are called when a transaction is checked for validity: At block creation time by the leader and when voting with the validators.
- are stored in the collection itself
- and can be evolved by a user with the corresponding permission.



Header

Merkle-root:

The Merkle-root enables a client to verify a cryptographic proof about the skipchain's state.

Hash of the clients' transactions:

Storing only the hash of the clients' transactions in the header reduces proof size.

Hash of the state changes:

The changes applied to the state are hashed and stored such that they can be tracked.

Timestamp

Body

Transactions sent by the clients

Figure : The structure of a skipblock.

My Contributions

- Initial skeleton
- Apply transactions tentatively
- Dummy contracts
- Sort transactions

Conclusion and Future Work

Omniledger is a flexible system, but has some drawbacks (for now)

- Leader is assumed to be correct.
- No dynamic deployment of contracts.
- Only leader queues transactions for now.

Which are then applied to the collection.