Introduction to the Tezos Blockchain

Victor ALLOMBERT, Mathias Bourgoin, Julien Tesson

Nomadic Labs - Paris, France

15 July 2019

Table of Contents

1. Introduction to Blockchains
2. The Tezos Blockchain
3. Interacting with Tezos
4. Writing Decentralized applications
5. Exercises

Table of Contents

1 Introduction to Blockchains
   The building blocks

2 The Tezos Blockchain

3 Interacting with Tezos

4 Writing Decentralized applications

5 Exercises

What is a blockchain ?

At an abstract level, a blockchain is:

- An immutable database
- Operating in a decentralized network

Key concepts

Relies on:

- Public Key Cryptography / Digital signature / Cryptographic Hash Functions
- A probabilistic solution to the 'Byzantine generals problem' for consensus among all nodes
- A p2p / gossip network for low level communication

Often called `crypto-ledgers`:

- Electronic book
- Recording transactions
- Users identity and book immutability cryptographically ensured

Validate and append transactions to the ledger.

Generic algorithm:

1. Send/receive/broadcast new transactions to all “participants” of the network (**nodes**)
2. Aggregate transaction into **blocks**
3. The next block is broadcasted by one (or several) nodes
4. Nodes express their acceptance of a block by including its hash in the next block they create

Difficulties

- Sybil attacks
- Byzantine attacks
- Liveness
  - What if the next block producer is offline?
- Network delays
  - must be accounted for
  - confuse honest nodes
- Forks
  - block producer could be malicious
  - different chains could fork
- T-consistency
  - Transaction in the last T blocks could be reverted

Case study: Bitcoin

- 2008 Satoshi's "Bitcoin: A Peer-to-Peer Electronic Cash System"
- Open-source implementation released in 2009

Bitcoin is:

- **Decentralized** electronic cash
- **Protocol**
- **P2P network**

Proof of Work (*PoW*)

- Resource is computing power (hw + energy)
- Solve a cryptographic puzzle to produce (mine) a block
  - Difficulty is adapted every ~2 weeks, aims at 10 min interval
  - delays between blocks is mandatory
- Hard to find, easy to check
- Sybil attacks are difficult/expansive
- Liveness is easy
  - pick the longest chain

Distributed applications: Smart contracts

- Blockchain as a **decentralized platform**, popularized by **Ethereum**
- User code in the blockchain (vending machine analogy)
  - user stores code in block
  - other users can call this code
- Contract has state, can perform blockchain operations
  - interacts with outside services (oracle)
  - can perform access control
- Many applications: financial contracts, new currencies, voting, games, crowd-funding

Table of Contents

1 Introduction to Blockchains

2 The Tezos Blockchain
- Tezos specifics
- Proof-of-Stake Consensus
- On-chain governance

3 Interacting with Tezos

4 Writing Decentralized applications

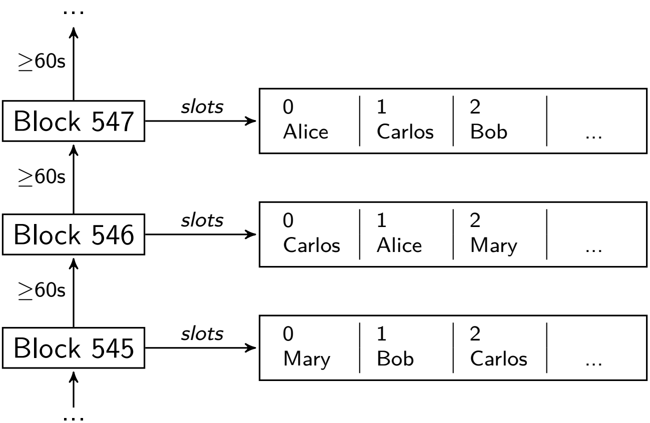
5 Exercises

Tezos in a nutshell

- Protocol upgrade/Voting process
- Liquid Proof of Stake
- Michelson smart contract language
- Formally verified cryptographic primitives

Current Tezos protocol consensus

To push new block at a certain level, *n* validators (*bakers*) are randomly selected using a priority list



Specificities

- A baker must have a minimum of 8.000_{tz} (a *roll*) to get slots
- Slot attribution is proportional to the number of rolls
- If a participant does not wish to bake, it is possible to *delegate* its stake

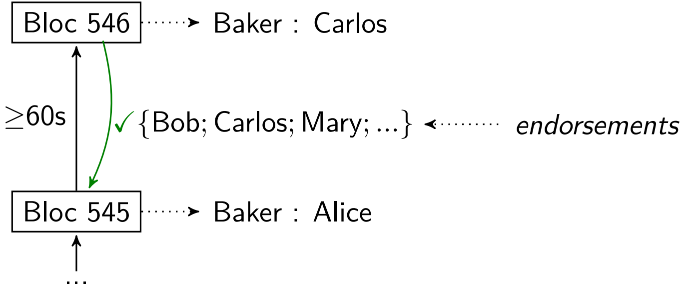
Endorsements

- In order to reach a finality faster, participants are incentivised to endorse blocks
- The highest block resulting score is considered head of the chain where the score is :

$$\texttt{score}(\mathcal{B}_{n+1}) = \texttt{score}(\mathcal{B}_n) + 1 + \texttt{nb\_endorsements}$$

with $\mathcal{B}_n$ a block at level n and `nb_endorsements`, the number of endorsements for $\mathcal{B}_n$ included in $\mathcal{B}_{n+1}$.

Economic incentive & Rewarding (1/2)



Rewards:

- Baking a block: 16_{tz}
- Endorsing a block: $2_{tz} \times 32$ (depending on the slot)

Economic incentive & Rewarding (2/2)

When a baker emits a new block or endorsement, a deposit bond is frozen for ~2 weeks (256_{tz} /64_{tz})

Double-baking

- A baker injects two **different** blocks for a same level

Double-endorsing

- A baker endorses two **different** blocks for a same level

If a baker is caught cheating, the deposit and all pending rewards are forfeited.

Self-amendment

We define self-amendment as the process to upgrade the protocol over time through on-chain voting:

- Reduce Forks and fraction/friction in the community
- Voting allows to amend the mechanism that governs the blockchain

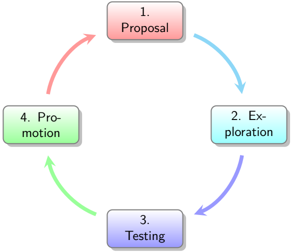
Exemples of protocol amendments:

- Switch to a different consensus,
- Extend the smart-contract language,
- Modify the rewarding system,
- Anonymous transactions, ...

Tezos current voting process

The voting process is split in 4 periods of ~3 weeks each:

1. Participants submit a new protocol proposal
2. A first voting happens for every submitted proposal
3. A side test chain spawns with the elected protocol
4. A final vote occurs to act the upgrade



If the final vote is successful, every participant will automatically switch to the new protocol.

Athens, the first on-chain governance

Nomadic Labs' first protocols proposals for Tezos:

- Athens A:
  - Small tuning and improvements
  - Increase smart-contracts gas limit
  - Reduce the size of rolls from $10K_{tz}$ to $8K_{tz}$

- Athens B:
  - Small tuning and improvements
  - Increase smart-contracts gas limit

Resulting in the vote of "Athens A" with 71%. Supported by 170 bakers representing 25855 rolls (49% of the network).

Table of Contents

1 Introduction to Blockchains

2 The Tezos Blockchain

3 Interacting with Tezos
- Setting up a Tezos node
- Using basic client (wallet) commands
- Using RPCs

4 Writing Decentralized applications

5 Exercises

Setting up a Tezos node

Generate an identity to join the network

```
$ tezos-node identity generate
```

Generating a new **identity**... (level: 24.00)
Stored the new **identity** (idrzT7PV27ieo3TS1St7Kte1dBW5AC) into
↪ '~/.**tezos**−**node**/**identity**.json'.

Setting up a Tezos node

Run the Tezos node

```
$ tezos-node run --rpc-addr 127.0.0.1
```

```
Jul 1 10:25:31 — node.main: Starting the Tezos node...
Jul 1 10:25:31 — node.main: No local peer discovery.
Jul 1 10:25:31 — node.main: Peer's global id: idrzT7PV27ieo3TS1St7Kte1dBW5AC
Jul 1 10:25:31 — main: shell—node initialization: bootstrapping
Jul 1 10:25:31 — main: shell—node initialization: p2p_maintain_started
Jul 1 10:25:31 — validator.block: Worker started
Jul 1 10:25:31 — validation_process.sequential: Intialized
Jul 1 10:25:31 — node.validator: activate chain NetXkaRXbyeogSM
Jul 1 10:25:31 — validator.chain_1: Worker started for NetXkaRXbyeog
Jul 1 10:25:31 — p2p.maintenance: Too few connections (0)
Jul 1 10:25:31 — prevalidator.NetXkaRXbyeog.PtBMwNZT94N7_1: Worker started for NetXkaRXbyeog.
  ↪    PtBMwNZT94N7
Jul 1 10:25:31 — node.main: Starting a RPC server listening on ::ffff:127.0.0.1:8732.
Jul 1 10:25:31 — node.main: The Tezos node is now running!
...
```

Setting up a Tezos node

Import a storage snapshot

```
$ tezos-node snapshot import last.full
```

...
Jul 1 10:28:38 – node.snapshots: Sucessful import (**from** file ./snapshot.full)

Using basic client commands

Checking chain's head state

```
$ tezos-client bootstrapped
```

```
Current head: BLVd2PEveT62 (timestamp: 2019−07−01T08:57:53Z, validation
↪      : 2019−07−01T08:58:04Z)
Bootstrapped.
```

Using basic client commands

Generate new keys

```
$ tezos-client gen keys bob
```

Using basic client commands

Test networks faucet: https://faucet.tzalpha.net/

Activating account using the faucet

```
$ tezos-client activate account alice with tz1\_xx\_.json
```

Using basic client commands

Check account balance

```
$ tezos-client get balance for alice
```

Using basic client commands

Transfer tokens

$ **tezos−client transfer** 1 **from** alice **to** bob −−*fee* 0.05

Using basic client commands

Wait for inclusion

$ **tezos−client wait for** <operation hash> **to be included**

List RPcs

```
$ tezos-client rpc list
```

Transfer command unboxing

$ **tezos−client** −l **transfer** 1 **from** alice **to** bob −−*fee* 0.05

1. request operation counter
2. check signature
3. check boostrapped status
4. get constants
5. get current head hash
6. get chain identifier
7. simulate operation
8. adjust fees and add signature
9. inject operation

Table of Contents

1 Introduction to Blockchains

2 The Tezos Blockchain

3 Interacting with Tezos

4 Writing Decentralized applications
- Smart contracts
- Michelson

5 Exercises

Smart contracts ?

- User code in the blockchain (vending machine analogy)
  - user stores code in block
  - other users can call this code
- Contract has state, can perform blockchain operations
  - Supplied with data by outside services (oracle)
  - can perform access control
  - can represent token

Smart contracts ?

Resources consumption

Contract storage (disk space) and execution (CPU usage) is replicated on every node

Needs incentive to be conservative

- Storage usage implies burn fees
- Contract execution is limited by a *gas* limit

What is Michelson ?

- A strongly typed, interpreted, language
- Stack language; a stack, no heap, no variable, no I/O
- The contracts code is a pure function transforming a stack
- Avoids ambiguous and implicit behaviour
- Stacked values are immutable
- Minimizes runtime failures

Execution cost: gas

Estimate the execution time of:

- Serialization
- Type checking
- Instruction cost
- Storage read/write access

A Michelson contract is made of ...

- Input : parameter and storage
- Output : storage and operation list
- Atomic execution and storage
- Allows storage upgrades
- A contract does not returns any value

Michelson code structure

```
storage int;
parameter unit;
code {
  instruction;
  instruction;
  ...
  instruction;
};
```

Data types

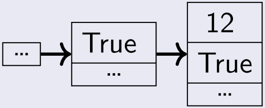
- Standard: Booleans, integers, naturals, strings, ...
- Non-recursive Algebraic datatypes: pair, or
- High level: option, list, set, map, big_map,..
- Domain-specific: timestamp, mutez, contract, address, operation, key, key_hash, signature

Stack manipulation: PUSH

PUSH 'a x: Adds an element x of type 'a on the stack

Evaluation:

```
PUSH bool True;
PUSH nat 12;
```



Type:

:: 'A $\rightarrow$'a : 'A

iff x :: 'a

Semantics:

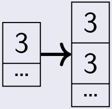
> PUSH 'a x / S => x : S

Stack manipulation: DUP

DUP: Duplicated the element on top of the stack

Evaluation:

DUP;



Type:

`:: 'a : 'A →'a : 'a : 'A`

Semantics:

`> DUP / x : S => x : x : S`

Stack manipulation: IF

IF `{bt} {bf}`: Conditional branch

Type:

```
:: bool:'A →'B
iff bt :: [ 'A →'B ]
    bf :: [ 'A →'B ]
```

Semantics:

```
> IF bt bf / True : S => bt / S
> IF bt bf / False : S => bf / S
```

High-level primitives

string	CONCAT, SLICE, SIZE	
pair	PAIR	UNPAIR, CAR, CDR
or	LEFT 'a, RIGHT 'a,	IF_LEFT { tbranch } { fbranch }
option	SOME, NONE 'a,	IF_NONE { tbranch } { fbranch }
list	NIL 'a, CONS,	IF_CONS { tbranch } { fbranch }, ...
set	EMPTY_SET 'type,	ITER { body }
	MEM, UPDATE	
map	EMPTY_MAP 'key 'val	ITER { body }, MAP { body }
	MEM, UPDATE, GET	

big_map (with lazy resolution)

Chain related primitives

- mutez: int64 based amount, can overflow (failure)
- Forge operation: CREATE_CONTRACT, CREATE_ACCOUNT, SET_DELEGATE, ...
- Permissions: SOURCE, SENDER, SELF
- BALANCE: contract amount
- AMOUNT: transaction amount

Higher level languages

- PascaLIGO et CamLIGO
- SmartPy (Python syntax)
- Morley (Haskell Library)

- Albert: Intermediate language as compilation target for Higher-level languages

Formal verification of smart contract

Mi-cho-coq

- Coq proof assistant
- Michelson semantics (as an executable interpreter)
- Weakest precondition calculus

Ultimate goal: extract the formalised interpreter and use it in the node.

Other tools

On-going works:

- SMT-based model checker Cubicle
- Certified compilation from Albert to Michelson
- Abstract interpretation tools

Table of Contents

1. Introduction to Blockchains
2. The Tezos Blockchain
3. Interacting with Tezos
4. Writing Decentralized applications
5. Exercises

Voting contract

```
storage (map string int) ;
parameter string ;
  code { AMOUNT ; PUSH mutez 5000000 ;
    # FAIL if amount send is less than 5tez
    COMPARE ; GT ; IF { FAIL } {} ;
    # Dups the map
    DUP ; DIP { CDR ; DUP } ;
    CAR ; DUP ;
    # stack: ::string::string::map::map
    # gets the value associated to string in the map
    DIP { GET ; ASSERT_SOME ;
      # Checks if input parameter is one of the choices
      PUSH int 1 ; ADD ; SOME } ;
    # stack: ::string::option int::map::
    # updates the map
    UPDATE ;
    # returns the pair (storage, operations)
    NIL operation ; PAIR }
```

Voting contract

Improvements:

1. Max voting count < 10
2. Output the winner when the vote is finished
3. Allow to add new choice if amount > 50 tz
4. Redistribute fees to winner/voting winners