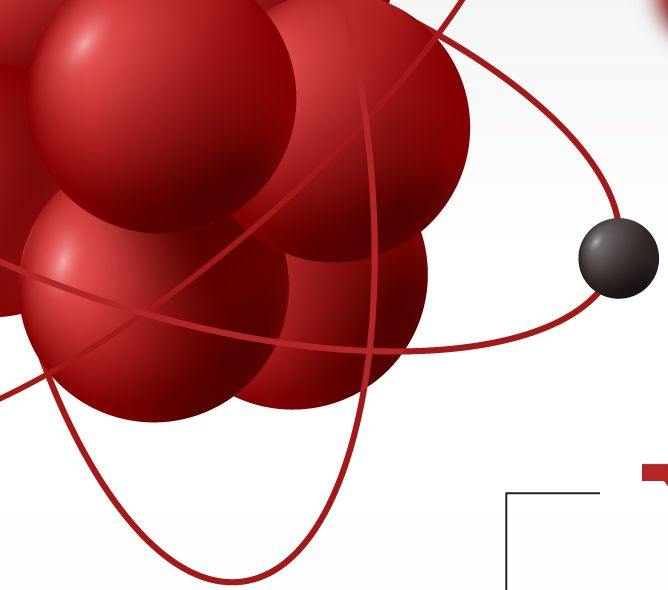
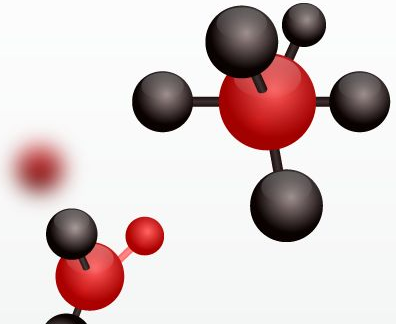


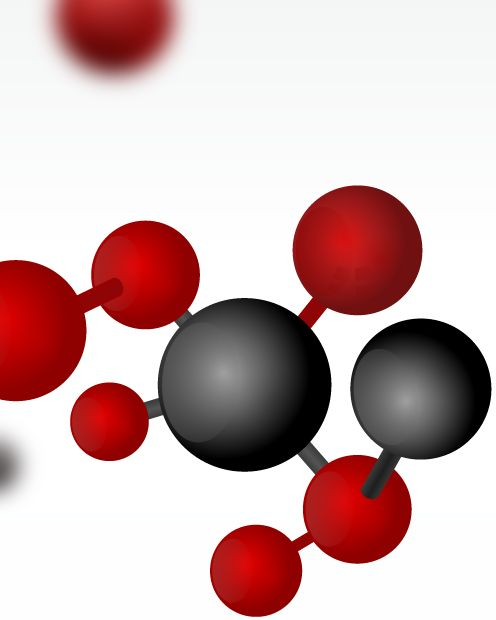
√RADIX DLT

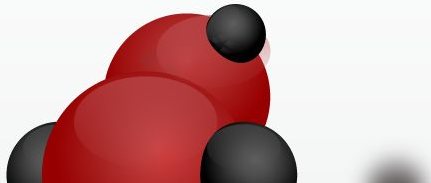
Introduction and experimental analysis



Matteo Berti 2019

TABLE OF CONTENTS





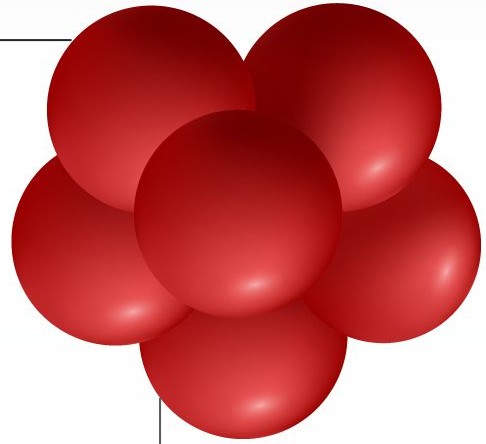
01

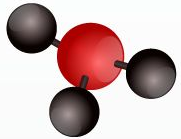
TEMPO

Radix Consensus Algorithm

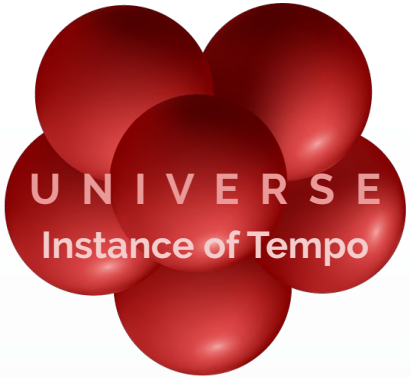
INTRODUCTION

In 2016 **Daniel Hughes** invented **Tempo**: a novel distributed ledger **architecture** and **consensus algorithm**. This algorithm is **designed to scale**.

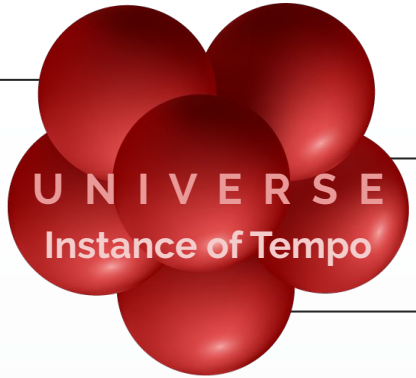




RADIX TEMPO



RADIX TEMPO



PAYLOAD ATOM

A communication, an email, a message.

TRANSFER ATOM

Transfer the ownership of an item, such as currency.

EXOTIC ATOM

Variants created for specific application purposes.

LEDGER ARCHITECTURE

NODE
Local ledger instance

STORE ALL the global ledger

STORE PART of the global ledger

LEDGER ARCHITECTURE

NODE
Local ledger instance

STORE ALL the global ledger

STORE PART of the global ledger

A **subset** of the global ledger is known as a **SHARD**

Nodes can store any **shard**, this enables **IoT devices** to actively participate in a Universe.

LEDGER ARCHITECTURE

To compute which **Shard** an **Atom** belongs:

```
ShardID = HASH(atomDestinationAddress) % ShardSpace
```

LEDGER ARCHITECTURE

To compute which **Shard** an **Atom** belongs:

```
ShardID = HASH(atomDestinationAddress) % ShardSpace
```

Atoms with **multiple destinations** will be present in **multiple shards**.

└── Increases **redundancy** and **availability** of Atoms.

LEDGER ARCHITECTURE

To compute which **Shard** an **Atom** belongs:

```
ShardID = HASH(atomDestinationAddress) % ShardSpace
```

Atoms with **multiple destinations** will be present in **multiple shards**.

- Increases **redundancy** and **availability** of Atoms.

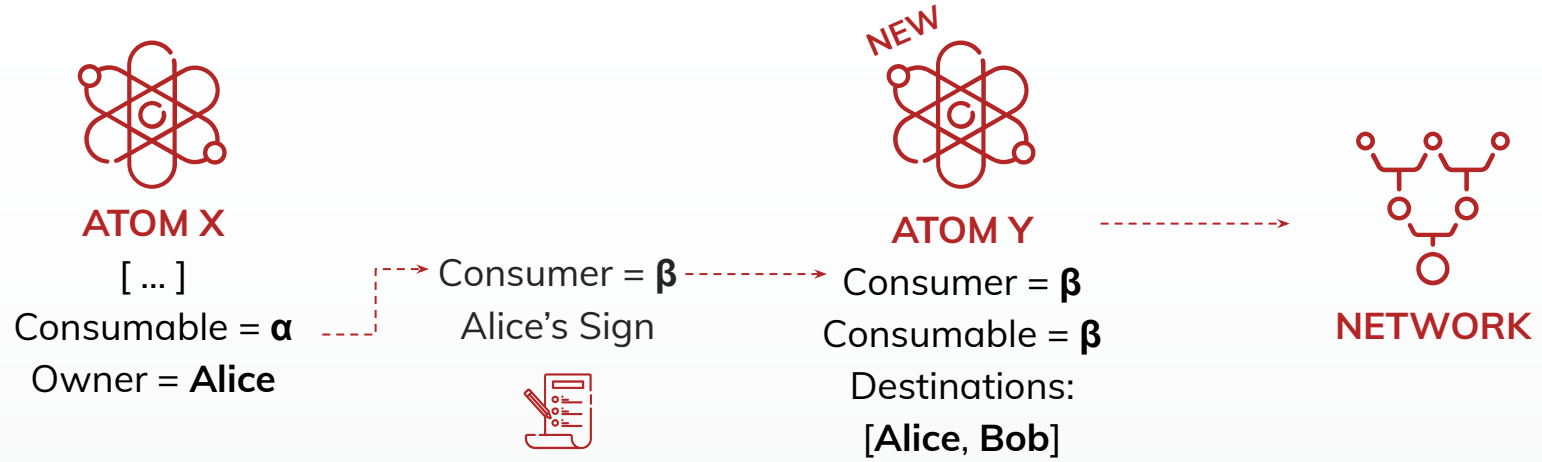
Indeed an Atom that performs an **inter-shard transfer** is present in both the **previous owner**'s and **new owner**'s **shards**.

- Eliminates the need for a **global state**.

TRANSFERS

An **owned item** is represented by a **CONSUMABLE**.

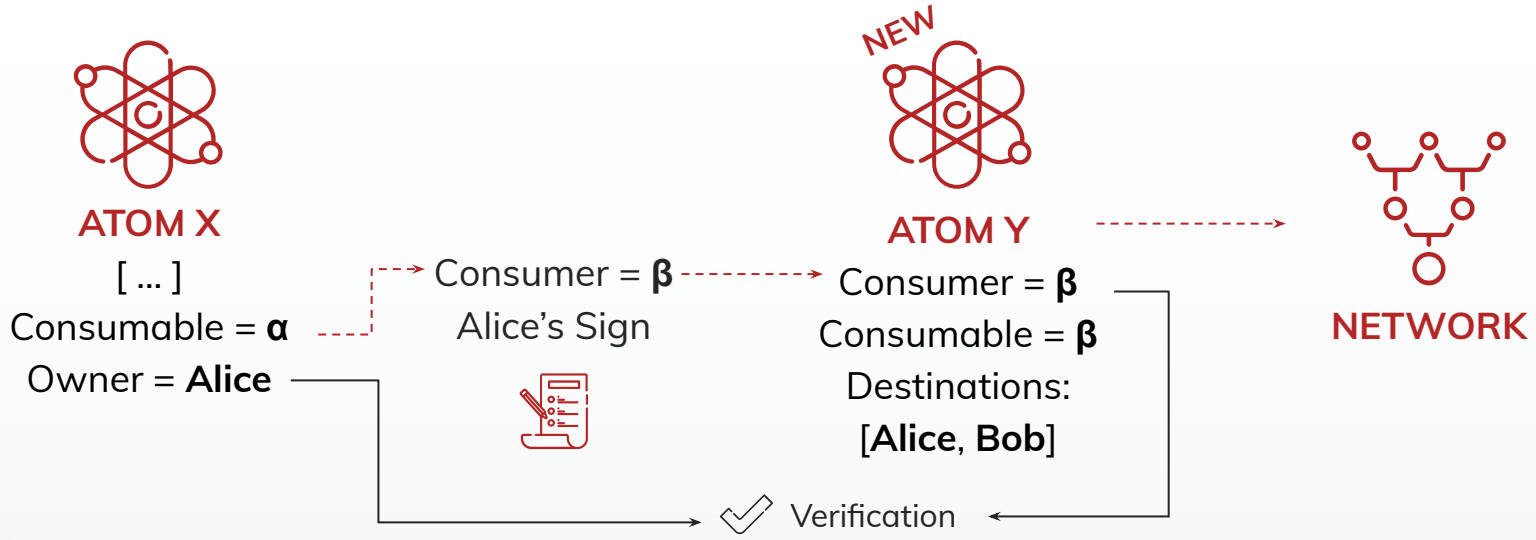
OWNERSHIP TRANSFER



TRANSFERS

An **owned item** is represented by a **CONSUMABLE**.

OWNERSHIP TRANSFER



EVENT AVAILABILITY

Atoms are **routed to** the **nodes** that contain the **associated shards** through a **Gossip** protocol.

ATOM Y

[...]

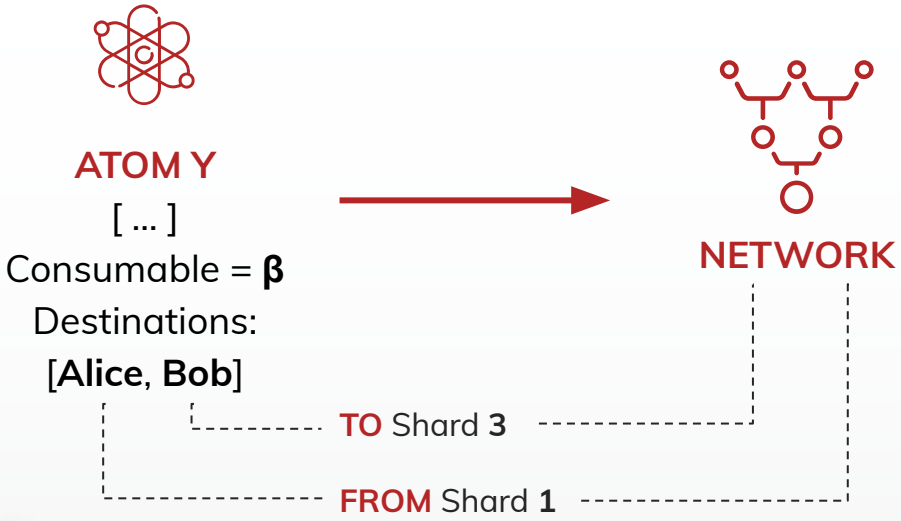
Consumable = **β**

Destinations:

[**Alice**, **Bob**]

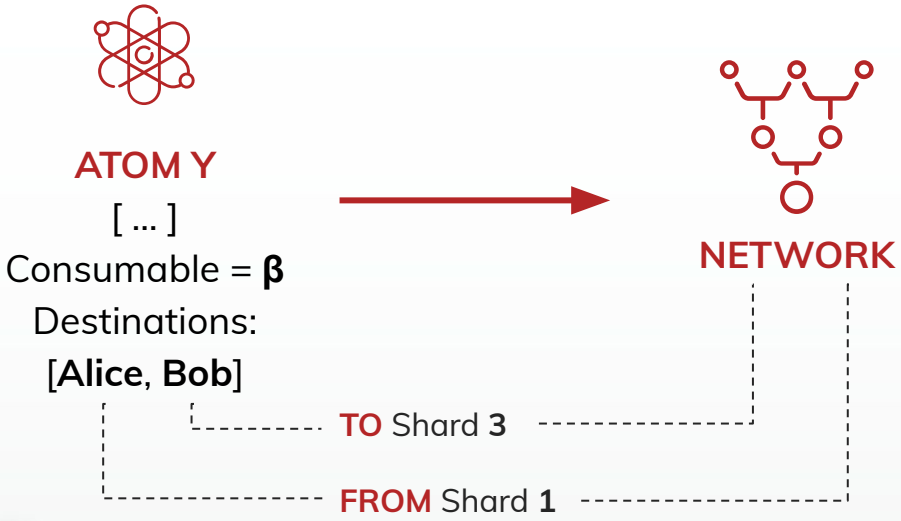
EVENT AVAILABILITY

Atoms are **routed to** the **nodes** that contain the **associated shards** through a **Gossip** protocol.



EVENT AVAILABILITY

Atoms are **routed to** the **nodes** that contain the **associated shards** through a **Gossip** protocol.

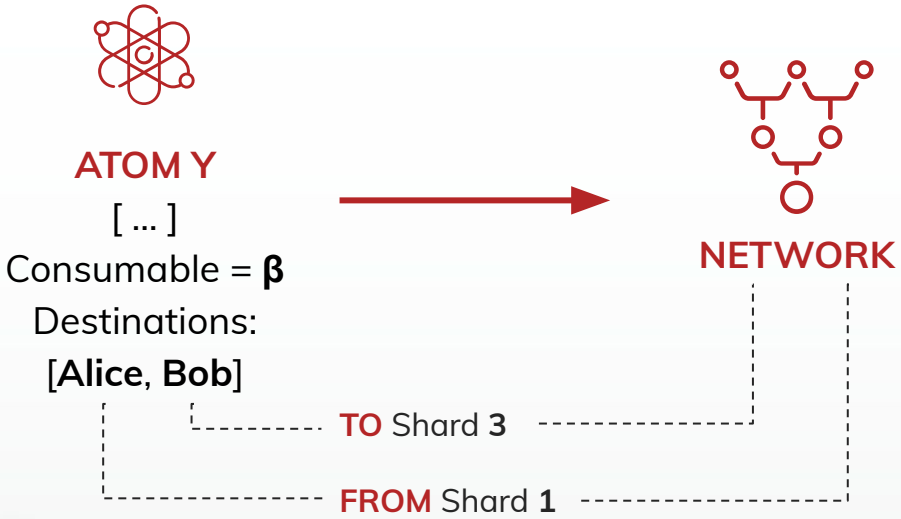


Nodes storing **Shard 1** and **Shard 3** need to be aware of the event of:

- Alice's **spend**
- Bob's **receipt**
- **State** of Item (α) consumed

EVENT AVAILABILITY

Atoms are **routed to** the **nodes** that contain the **associated shards** through a **Gossip** protocol.



Nodes storing **Shard 1** and **Shard 3** need to be aware of the event of:

- Alice's **spend**
- Bob's **receipt**
- **State** of Item (α) consumed

POST THE EVENT

The **responsibility** of the item's state has transferred from node storing **Shard 1** to those storing **Shard 3**.

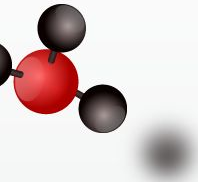
LOGICAL CLOCKS

All **nodes** have a **local logical clock**: an **ever-increasing integer** value representing the **number of new events** witnessed by that node.

TEMPORAL PROOF PROVISIONING

Temporal Proof is a solution to the **double spending** problem.

This **proof** is **carried with** the **Atom** along the network.

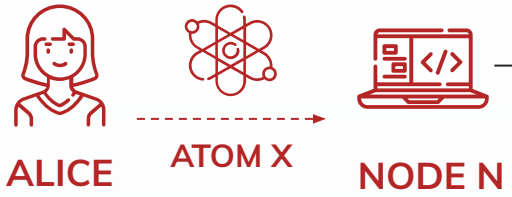


TEMPORAL PROOF PROVISIONING

Temporal Proof is a solution to the **double spending** problem.

This **proof** is **carried with** the **Atom** along the network.

TEMPORAL PROOF



If N owns a copy of **SHARD 1** **checks** that the item hasn't been **already spent** by Alice.

If any provable **discrepancy** is found the proof **fails**.

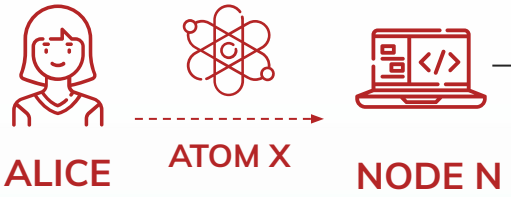
Otherwise, the node will **forward** the request to all **neighbors** storing either **Shard 1** or **3**.

TEMPORAL PROOF PROVISIONING

Temporal Proof is a solution to the **double spending** problem.

This **proof** is **carried with** the **Atom** along the network.

TEMPORAL PROOF



If N owns a copy of **SHARD 1** **checks** that the item hasn't been **already spent** by Alice.

If any provable **discrepancy** is found the proof **fails**.

Otherwise, the node will **forward** the request to all **neighbors** storing either **Shard 1** or **3**.

N **forwards** to **P** a **New** **Temporal Proof**:
Space-time coordinate: `(l, e, o, n)`

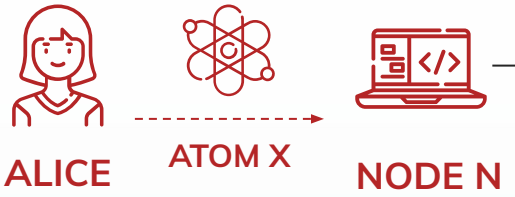
NODE P

TEMPORAL PROOF PROVISIONING

Temporal Proof is a solution to the **double spending** problem.

This **proof** is **carried with** the **Atom** along the network.

TEMPORAL PROOF



If N owns a copy of **SHARD 1** **checks** that the item hasn't been **already spent** by Alice.

If any provable **discrepancy** is found the proof **fails**.

Otherwise, the node will **forward** the request to all **neighbors** storing either **Shard 1** or **3**.

`(l, e, o, n)`

l: logical clock value for the event

e: event hash HASH(Atom)

o: node N id

n: node P id

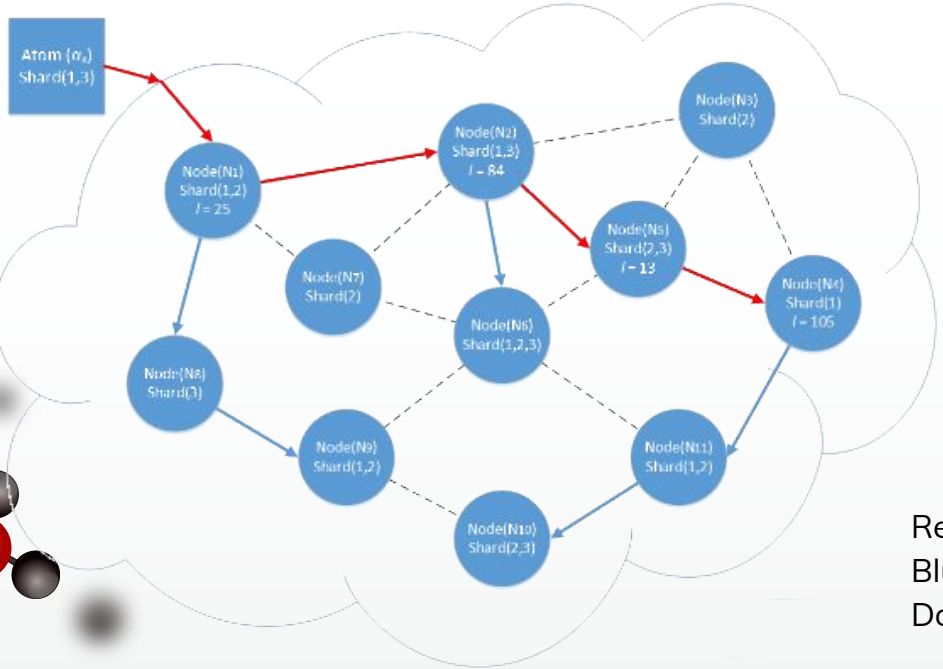
N forwards to **P** a **New Temporal Proof**: Space-time coordinate: `(l, e, o, n)`

NODE P

TEMPORAL PROOF PROVISIONING

PROVISIONING

Node P validates Atom X, appends `(1, e, o, n)` and forward it to Shard 1 or 3 neighbours.



Logical Clock	Event	Observer	Next Observer
25	Hash $((Atom(\alpha_X))$	Node $(N1)$	Node $(N2)$
84	Hash $((Atom(\alpha_X))$	Node $(N2)$	Node $(N5)$
13	Hash $((Atom(\alpha_X))$	Node $(N5)$	Node $(N4)$
105	Hash $((Atom(\alpha_X))$	Node $(N4)$	–

Red arrow: **PROVISIONING**
Blue arrow: **GOSSIP**
Dotted line: **CONNECTION**

PROVISIONING EFFICIENCY

Provisioning **length** TOO SHORT

Reduces the efficiency of **resolving conflicts**.

Provisioning **length** TOO LONG

Increase the **bandwidth load** and **time** taken.

PROVISIONING EFFICIENCY

Provisioning **length** TOO SHORT

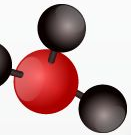
Reduces the efficiency of **resolving conflicts**.

Provisioning **length** TOO LONG

Increase the **bandwidth load** and **time** taken.

Sufficient provisioning **length**:

`log(n) * 3` or `max(3, sqrt(n))`



PROVISIONING EFFICIENCY

Provisioning **length** TOO SHORT

Reduces the efficiency of **resolving conflicts**.

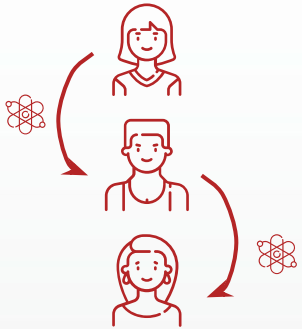
Provisioning **length** TOO LONG

Increase the **bandwidth load** and **time** taken.

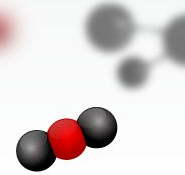
Sufficient provisioning **length**:
`log(n) * 3` or `max(3, sqrt(n))`

OPTIMIZATION

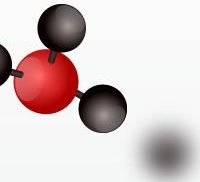
If **Alice** sends **Item** to **Bob**, and **Bob** then sends **Item** to **Carol**, the nodes involved in Alice → Bob **Temporal Proof** take also part in Bob → Carol transfer.



VECTOR CLOCKS



Vector clocks are used to determine the **partial order** of two **conflicting Atoms** (e.g. double spending).



VECTOR CLOCKS

Vector clocks are used to determine the **partial order** of two **conflicting Atoms** (e.g. double spending).

When **Atom X** and **Atom Y** conflict there are many scenarios:

1. The pair of **vector clocks** contains a **common node**:

VC(ATOM **X**)		VC(ATOM **Y**)	
A	5	B	10
D	12	G	7
F	34	P	47
P	17	L	24

VECTOR CLOCKS

Vector clocks are used to determine the **partial order** of two **conflicting Atoms** (e.g. double spending).

When **Atom X** and **Atom Y** conflict there are many scenarios:

1. The pair of **vector clocks** contains a **common node**:

VC(ATOM **X**)		VC(ATOM **Y**)	
A	5	B	10
D	12	G	7
F	34	P	47
P	17	L	24

VC(ATOM X) ≤ VC(ATOM Y)

VECTOR CLOCKS

Vector clocks are used to determine the **partial order** of two **conflicting Atoms** (e.g. double spending).

When **Atom X** and **Atom Y** conflict there are many scenarios:

2. The pair of **vector clocks** does not contain a **common node**:
    a. It can be used an **intermediate node**.

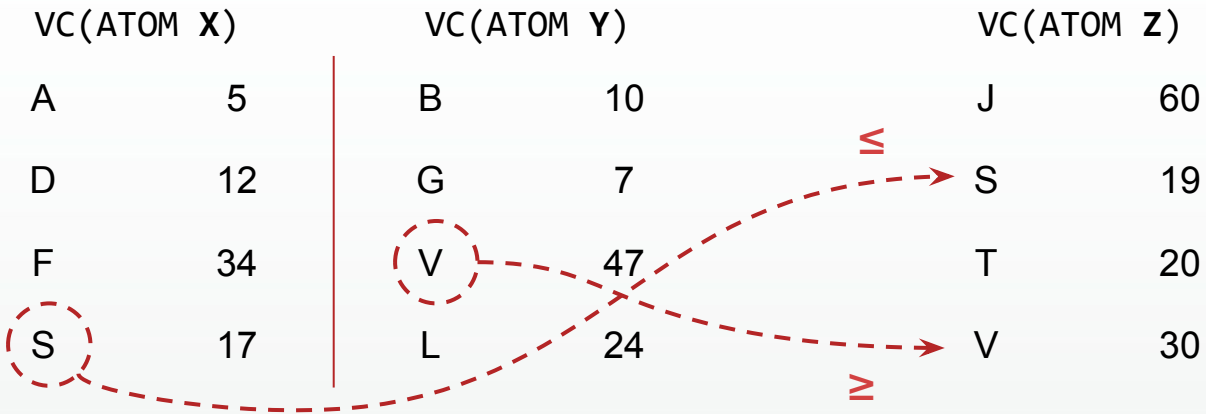
VC(ATOM **X**)		VC(ATOM **Y**)		VC(ATOM **Z**)	
A	5	B	10	J	60
D	12	G	7	S	19
F	34	V	47	T	20
S	17	L	24	V	30

VECTOR CLOCKS

Vector clocks are used to determine the **partial order** of two **conflicting Atoms** (e.g. double spending).

When **Atom X** and **Atom Y** conflict there are many scenarios:

2. The pair of **vector clocks** does not contain a **common node**:
   a. It can be used an **intermediate node**.



Indeed:
VC(ATOM X) ≤ VC(ATOM Y)

(Intermediate Atoms could be more than one)

VECTOR CLOCKS

Vector clocks are used to determine the **partial order** of two **conflicting Atoms** (e.g. double spending).

When **Atom X** and **Atom Y** conflict there are many scenarios:

2. The pair of **vector clocks** does not contain a **common node**:
    b. If an **intermediate node cannot be found**, then:

Commitment Order Determination

VECTOR CLOCKS

Vector clocks are used to determine the **partial order** of two **conflicting Atoms** (e.g. double spending).

When **Atom X** and **Atom Y** conflict there are many scenarios:

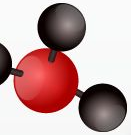
2. The pair of **vector clocks** does not contain a **common node**:
    b. If an **intermediate node cannot be found**, then:

Commitment Order Determination

For light nodes such as **IoT devices**, commitments are the **only way** to **determine order**.

COMMITMENTS

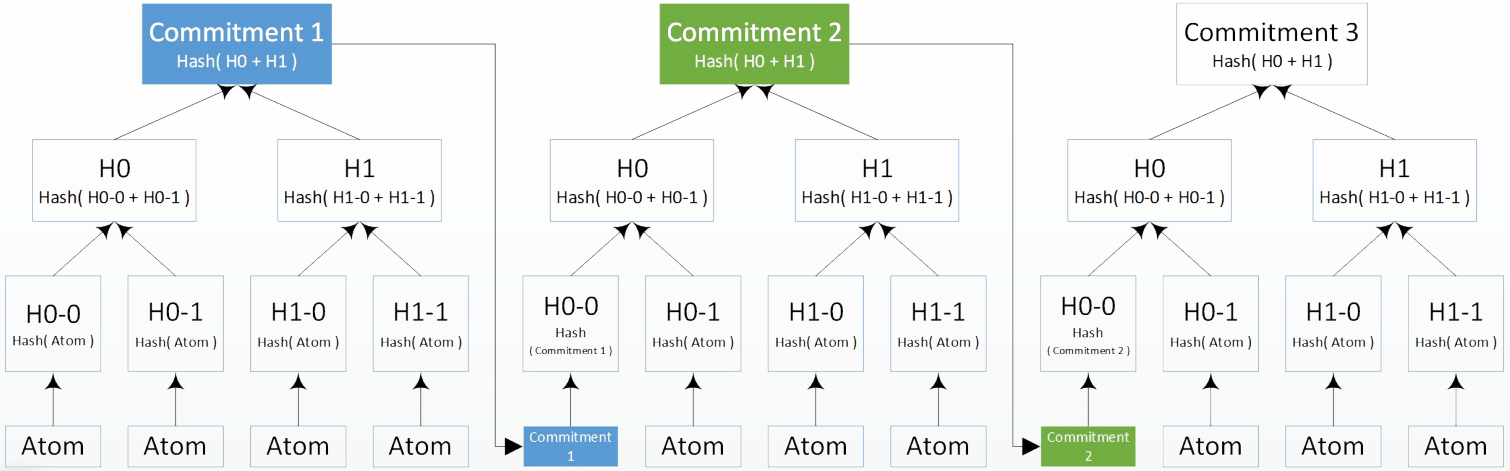
Nodes declare to the network a **periodic commitment** of **all events** they have seen.



COMMITMENTS

Nodes declare to the network a **periodic commitment** of **all events** they have seen.

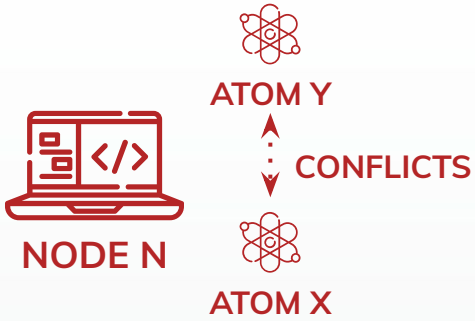
COMMITMENT: a **Merkle Hash** constructed from the **events a node has witnessed since** submitting a **previous commitment**.



COMMITMENTS

If the value of **1** for Commitment 1 was **100** and the value of **1** for Commitment 2 was **200**, then Commitment 1 should contain **100 items**.

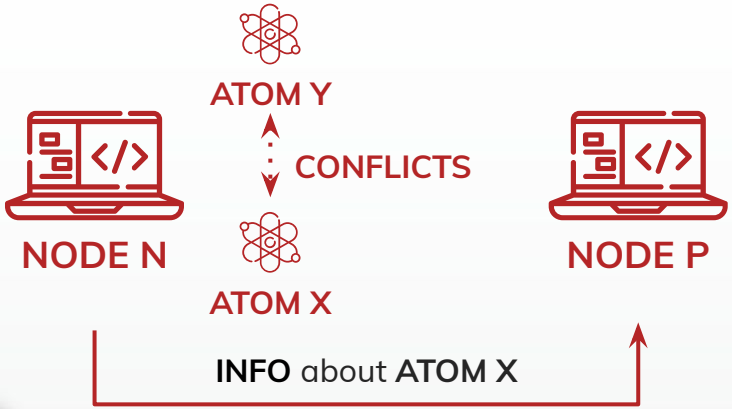
If a requesting node is not returned 100 hashes when verifying, **tampering of the logical clock** may have occurred.



COMMITMENTS

If the value of **1** for Commitment 1 was **100** and the value of **1** for Commitment 2 was **200**, then Commitment 1 should contain **100 items**.

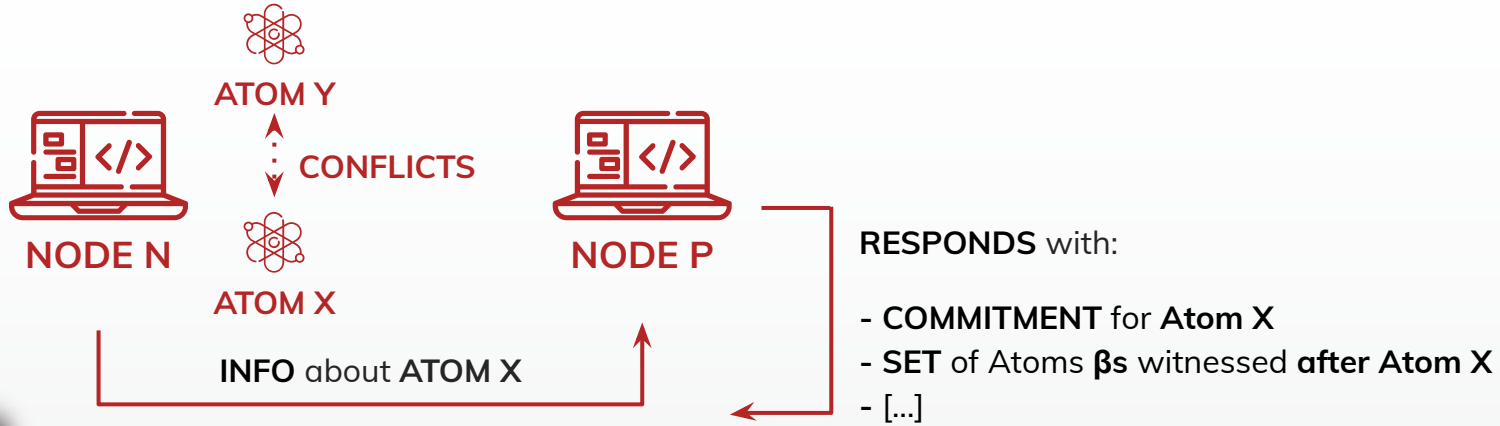
If a requesting node is not returned 100 hashes when verifying, **tampering of the logical clock** may have occurred.



COMMITMENTS

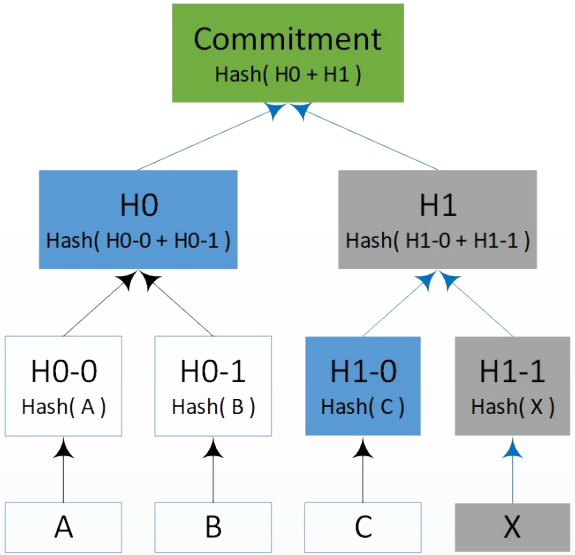
If the value of **1** for Commitment 1 was **100** and the value of **1** for Commitment 2 was **200**, then Commitment 1 should contain **100 items**.

If a requesting node is not returned 100 hashes when verifying, **tampering of the logical clock** may have occurred.



COMMITMENTS

COMMITMENT VALIDATION



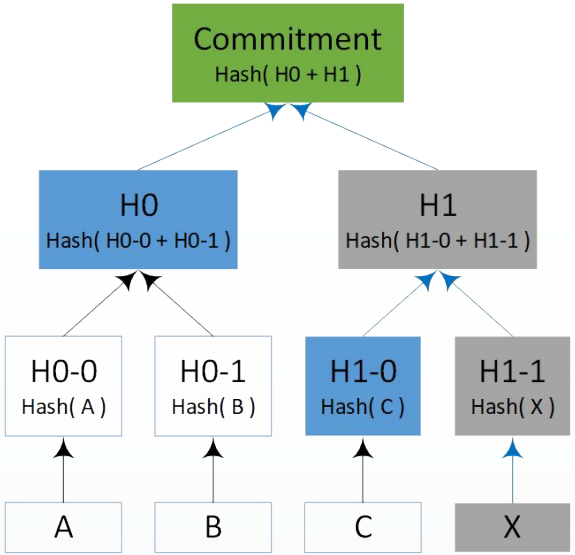
Node N queries **NODE Q** which **delivered** **Atom Y**:
- **COMMITMENT** and for **Atom Y**
- any of the **Atoms βs**
- [...]

This allows **NODE N** to **verify**

	NODE P LC	NODE Q LC
ATOM **X**	45	-
ATOM **Y**	-	465
ATOM s_1	46	-
ATOM s_2	47	441
ATOM s_3	458	-

COMMITMENTS

COMMITMENT VALIDATION

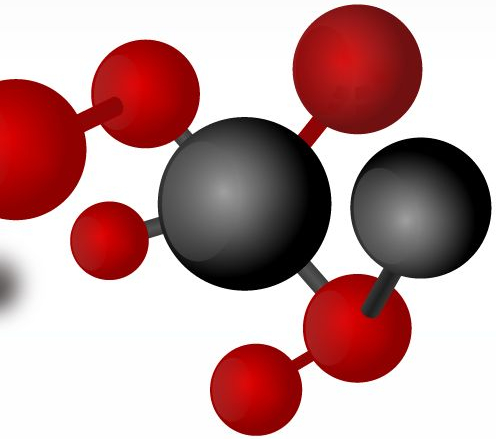


Node N queries **NODE Q** which **delivered** **Atom Y**:
- **COMMITMENT** and for **Atom Y**
- any of the **Atoms βs**
- [...]

This allows **NODE N** to **verify**

	NODE P LC	NODE Q LC
ATOM **X**	45	-
ATOM **Y**	-	465
ATOM s_1	46	-
ATOM s_2	47	441
ATOM s_3	458	-

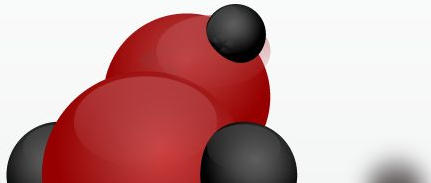
Therefore **ATOM X** happened **before** **ATOM Y**



02

TESTING ALPHANET

Performance analysis on IoT network simulation



ALPHANET

ALPHANET is the **α-testing network** of Radix DLT.

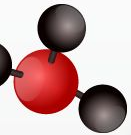
- 6 Nodes
- 1 User

GOAL

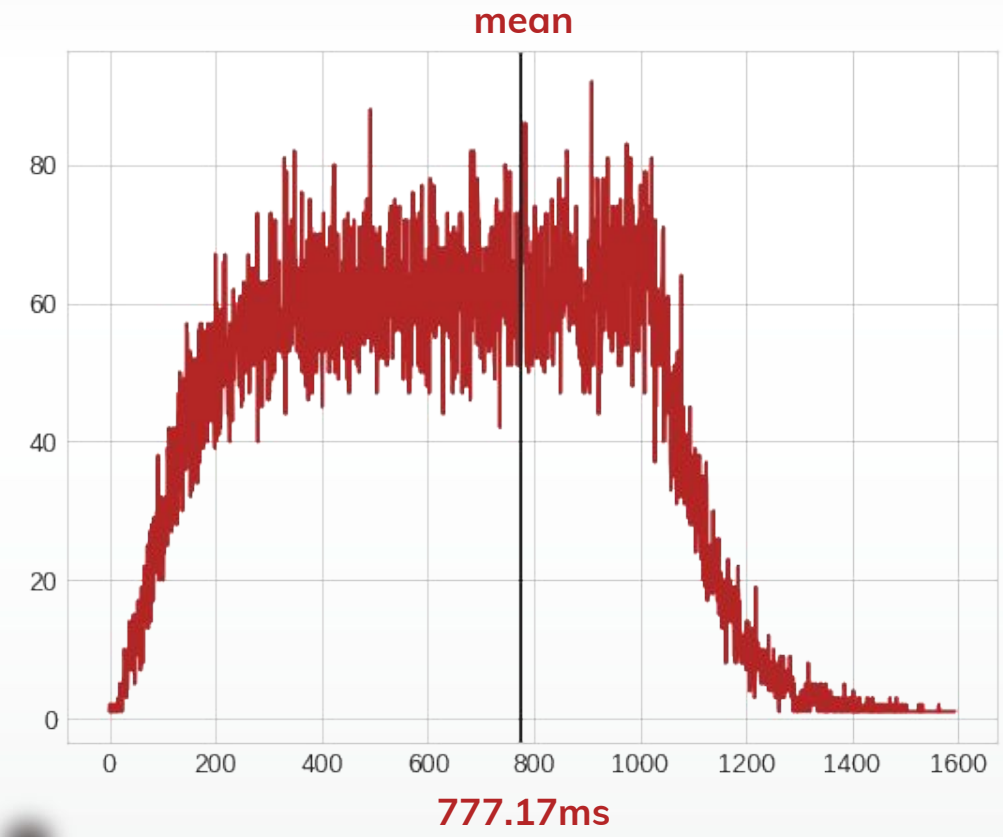
Testing the **error rate** and the (mean) **time** required to write and Atom on the ledger.

DESIGN

- Node.js server simulating **10** different **autobuses** writing data on the DLT at certain points in time.
- Run **6 parallel simulations** for **12 times** over a dedicated server (12 hours) ≃ 120 autobuses.



RESULTS

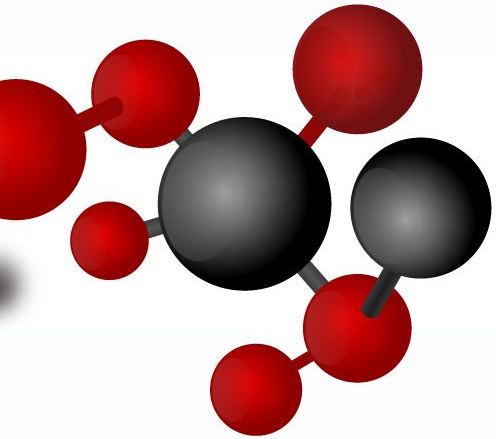


Error rate

2.73%

Mean Confidence Interval

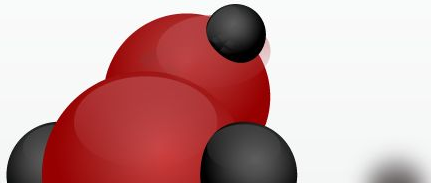
774.68ms ← 777.17ms →
779.65ms



03

TESTING BETANET

Testing tokens on local betanet emulation



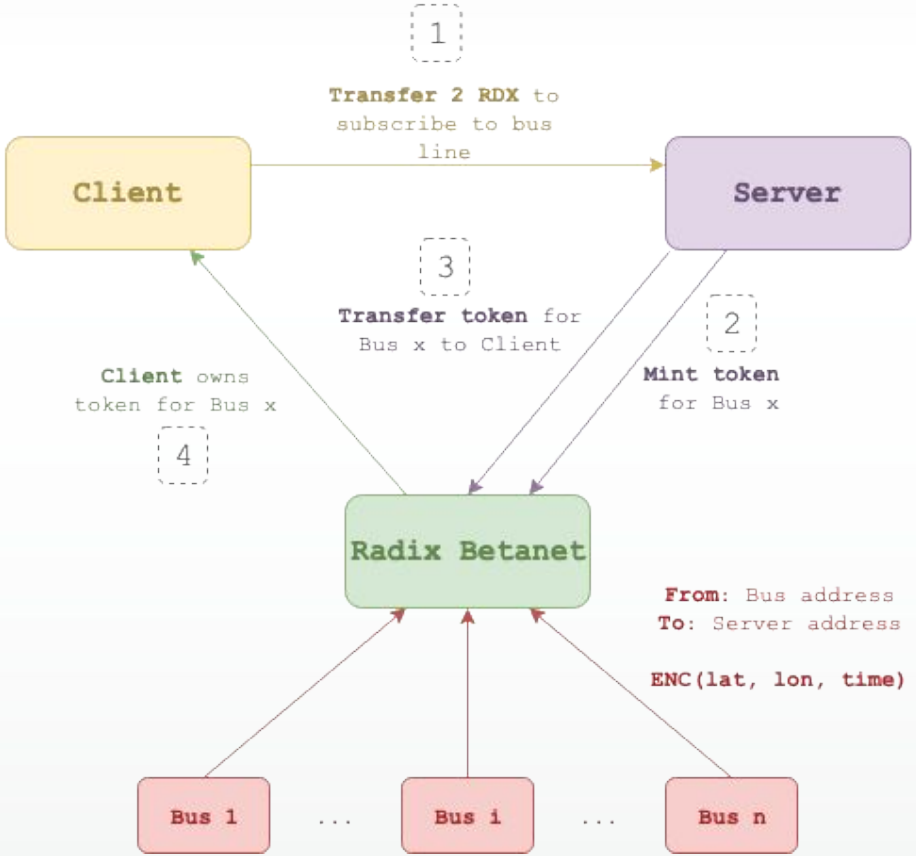
BETANET

BETANET emulated on **local computer** because online betanet will be deployed in December.

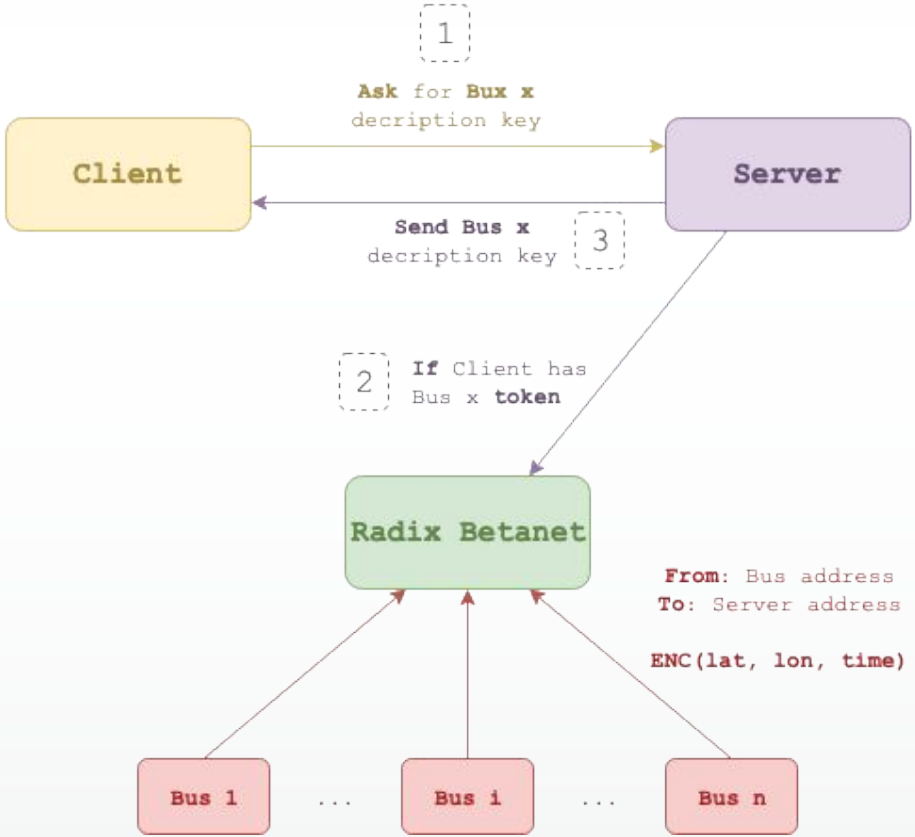
GOAL

- Getting RDX tokens from Faucet account.
- Mint new custom tokens.
- Transfer standard and custom tokens between two accounts.
- Send message and payload atoms between two accounts.
- Security checks (balance, specific tokens in wallet, sufficient funds, ...).
- Local storing of Radix identities.
- Symmetric key cryptography.

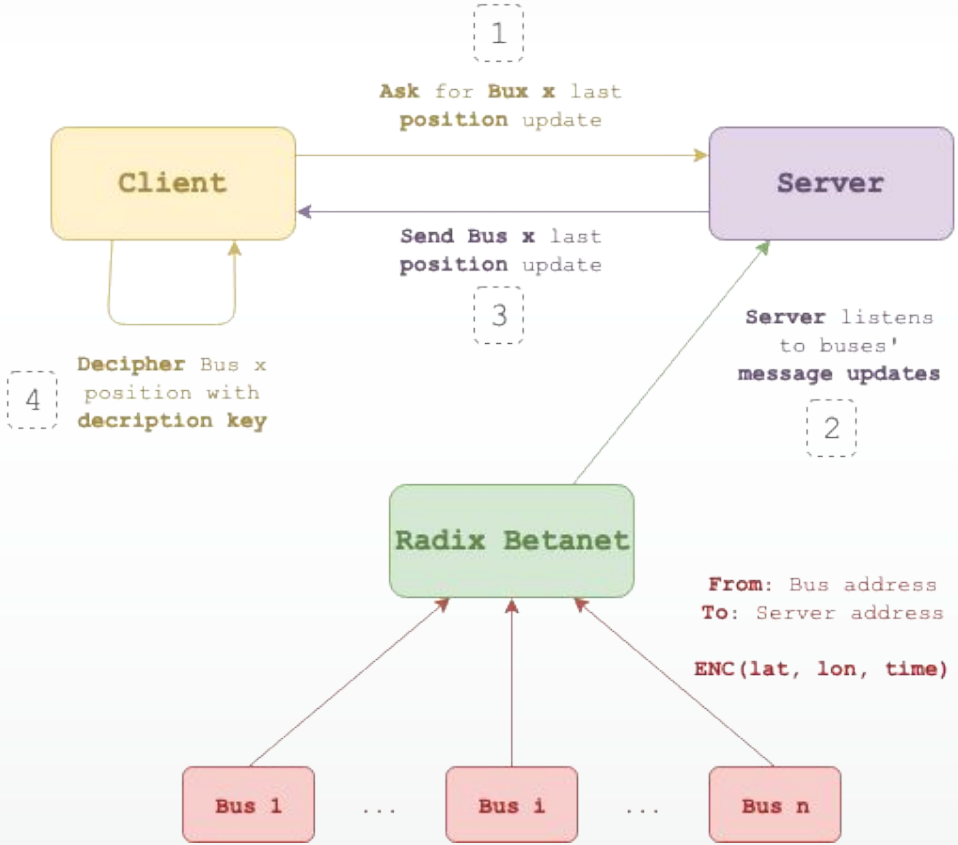
GET BUS TOKEN



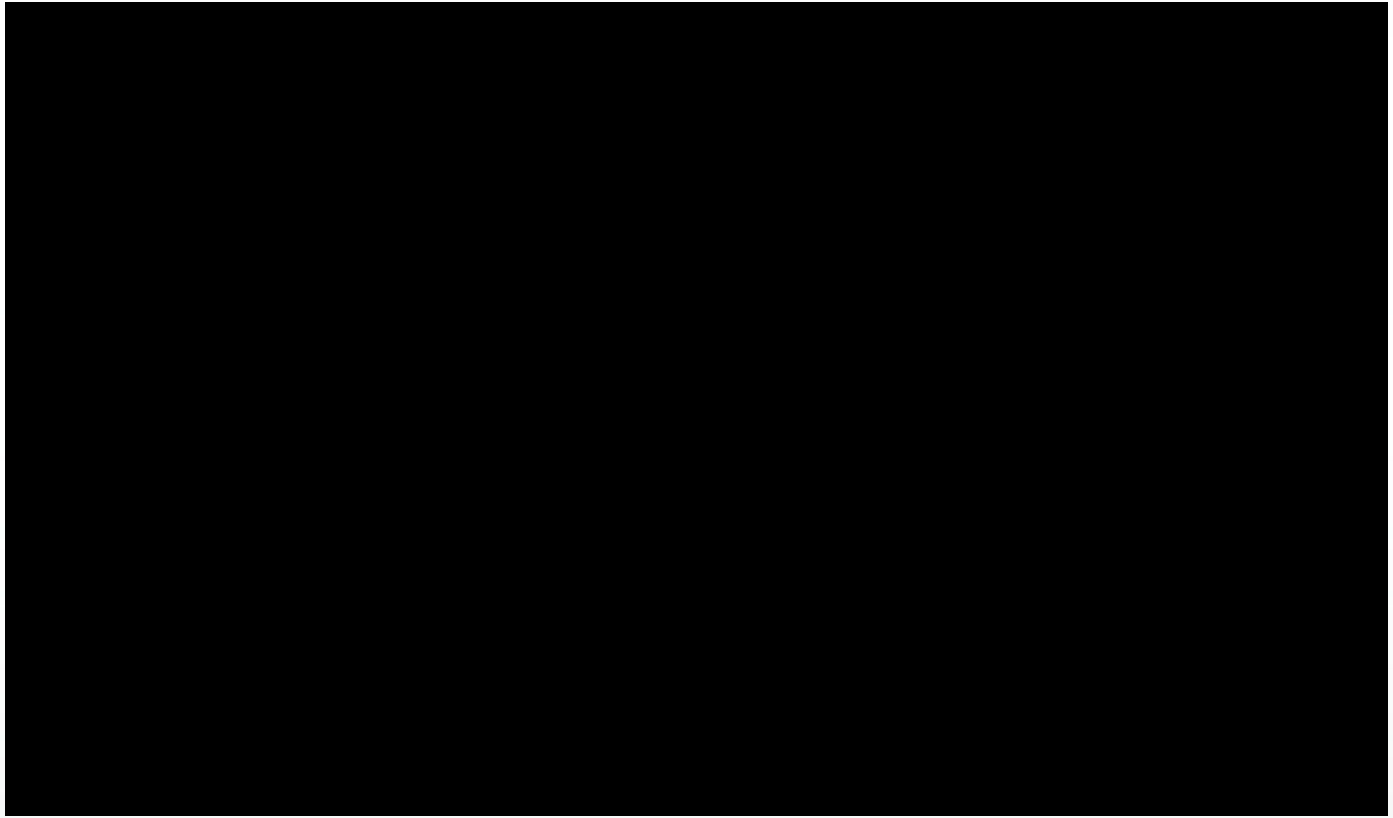
GET DECRYPTION KEY



GET BUS LINE POSITION



EXECUTION



RESEARCH RESOURCES

All material: `github.com/methk > RadixDLT-IoTSimulation`

- Dan Hughes, Radix DLT: Tempo Whitepaper - 2017
- S.Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System - 2008
- V. Buterin, Ethereum Whitepaper - 2014
- L. Lamport, Time, Clocks, and the Ordering of Events in a Distributed System - 1978
- C.J.Fidge, Timestamps in Message-Passing Systems that preserve the Partial Ordering - 1988
- R.C. Merkle, Merkle Tree - 1979