The Ripple XRP Ledger

From theory to practice

Lucian Trestioreanu

Dr. Wazen Shbair

Dr. Cyril Renaud Cassagnes

Prof. Habil. Radu State

ripple | University Blockchain Research Initiative

Luxembourg National Research Fund

Supported by Ripple UBRI and Luxembourg National Research Fund through grant PRIDE15/10621687/SPsquared.

CONTENT

The XRP Ledger (XRPL) ecosystem ..... 55 minutes

- Overview
- The Interledger protocol: Core concepts, protocols, software suite
- The XRP Ledger
  - XRPL consensus.
  - XRPL core messaging
  - Network topology, performance
  - “HOOKS” Smart Contracts
  - NFT support
  - PayString
  - Possible improvements/research directions

Demo session ..... 55 minutes

- Deploying an XRP node on testnet / mainnet
- Deploying a private testbed
- Deploying HOOKS smart contracts .

WHO THIS TUTORIAL IS FOR

GENERAL AUDIENCE

“Keep it simple in the middle and complicated at the edges.”

OVERVIEW - XRP Ledger (XRPL)

Open-source, permissionless and decentralized blockchain technology.

Appreciated for its transaction throughput, speed and the low fees:

- Can settle transactions in 3-5 seconds.
- 1,500 Tx/s , and can scale to same throughput as Visa*.
  (ETH 15Tx/s, BTC 3-6 Tx/s)
- $0.0000774 - Network Transaction Fee (on April 21, 2022).

Eco-friendly: negligible energy consumption.

SOURCE: https://ripple.com/xrp/

OVERVIEW - XRP Ledger (XRPL)

Seeks to streamline fiat & crypto cross-border payments.

Many fiat payments still function on old rails involving combinations of low speed, large fees, low transparency, poor/expensive inter-connectivity, with some of these, e.g. interconnectivity, also concerning cryptocurrencies.

Aim is to make universal, global value transfers as easy and fast as sending information. With the advent of Central Bank Digital Currencies (CBDC) which will also need interconnectivity, the argument becomes even stronger.

“XRP: standard digital asset for payments”

Enhances the world-wide payments infrastructure and services by providing XRP tokens to ensure quick liquidity and acting as a global settlement network.

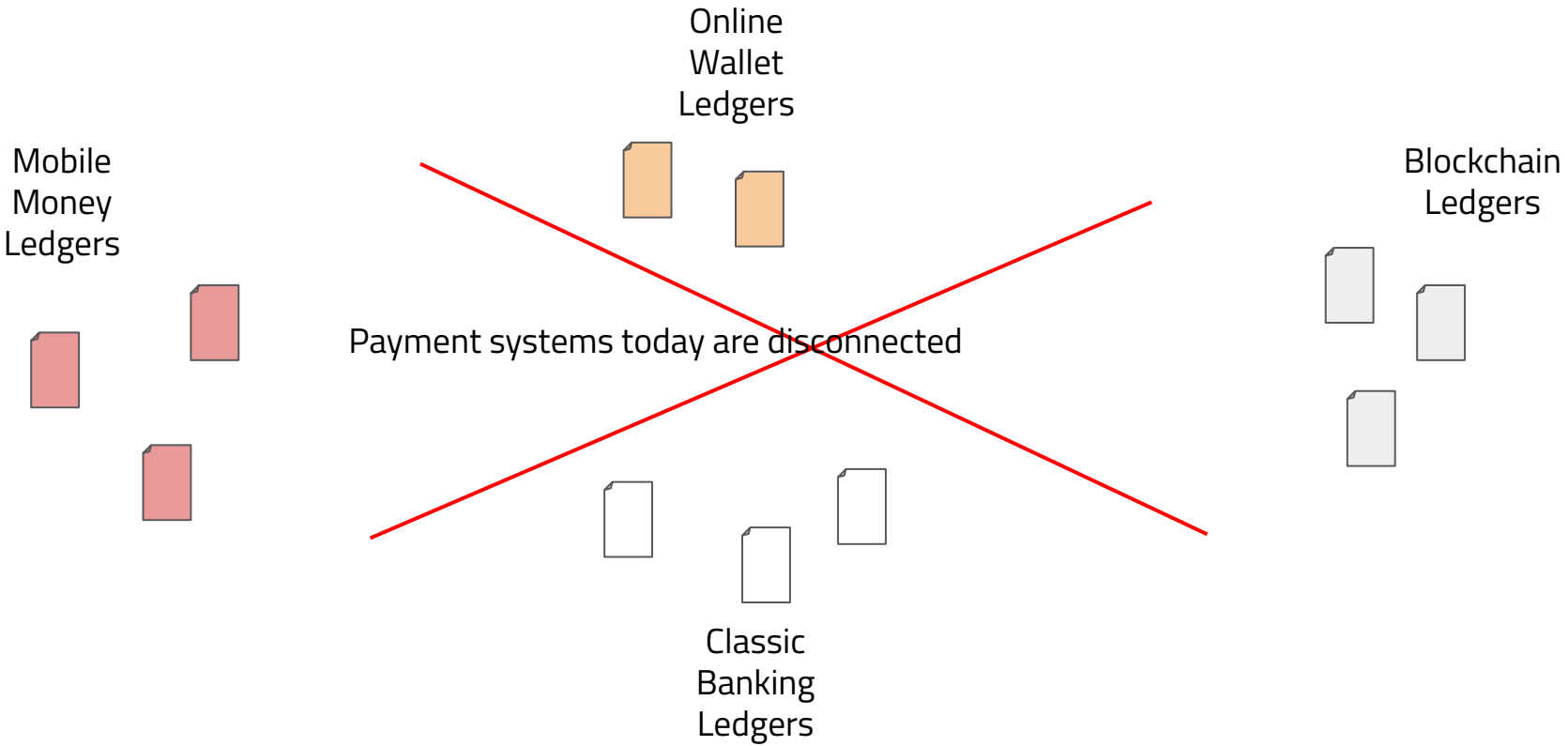
XRP can act as a “bridge” asset that businesses and financial institutions can use to bridge a transfer between two different currencies.

OVERVIEW - XRP Ledger (XRPL)

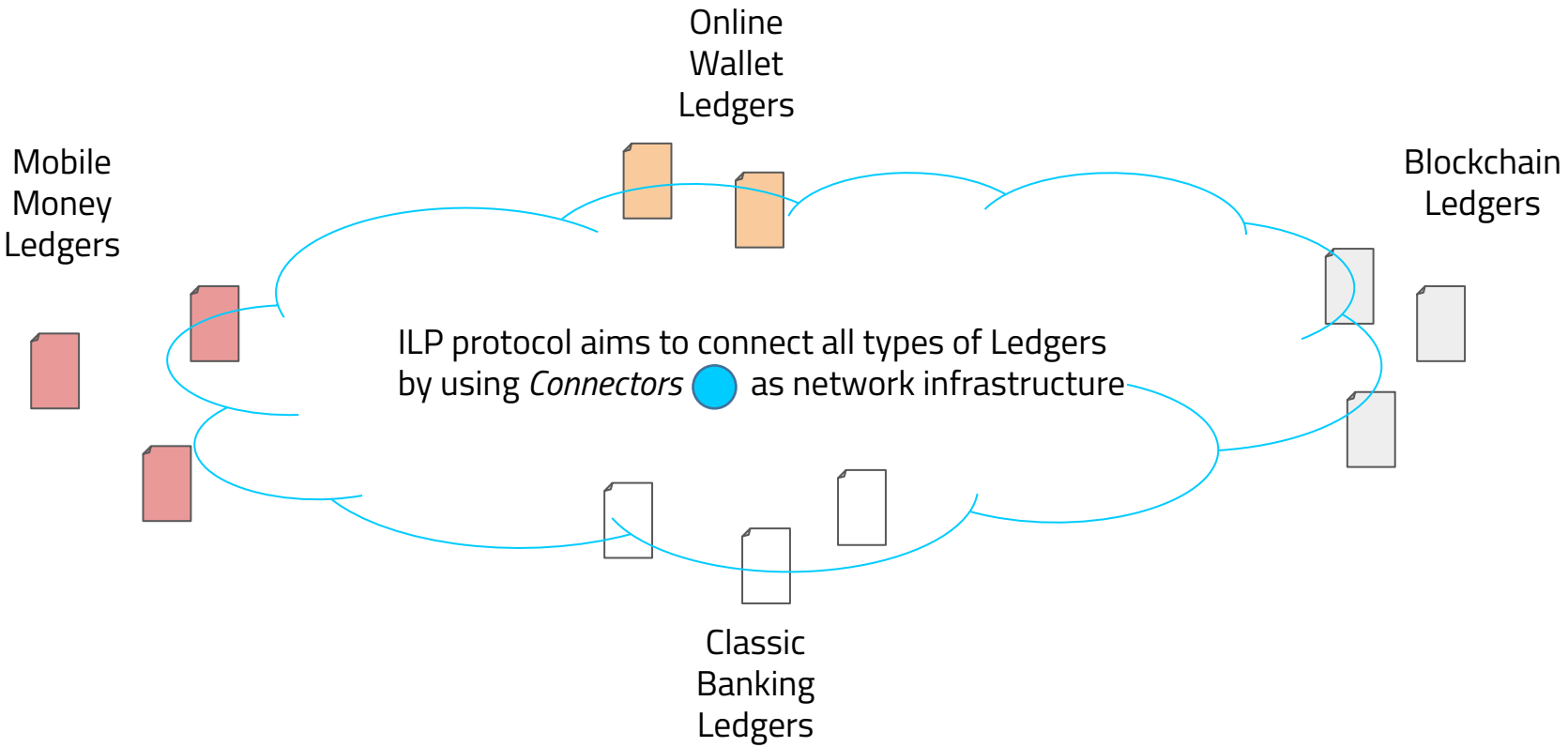
Solutions for faster, cheaper, more transparent and secure cross-border and inter-ledger payments explored by academic, private and governmental institutions: Interledger (Ripple), Stella (EU/Japan), Jasper (Canada), Ubin (Singapore), seek to leverage DLT technology like the XRP ledger, Quorum (JP Morgan), Open Chain (Ant Financial), Corda, Stellar, Ethereum, Hyperledger Fabric to achieve such goals.

Ripple's technology including the XRP ledger is laser-focused on the above mentioned problematic, with the aim to enable frictionless transfers between virtually any value holding systems (ledgers).

Interledger | Inter-Ledger



Interledger | Inter-Ledger

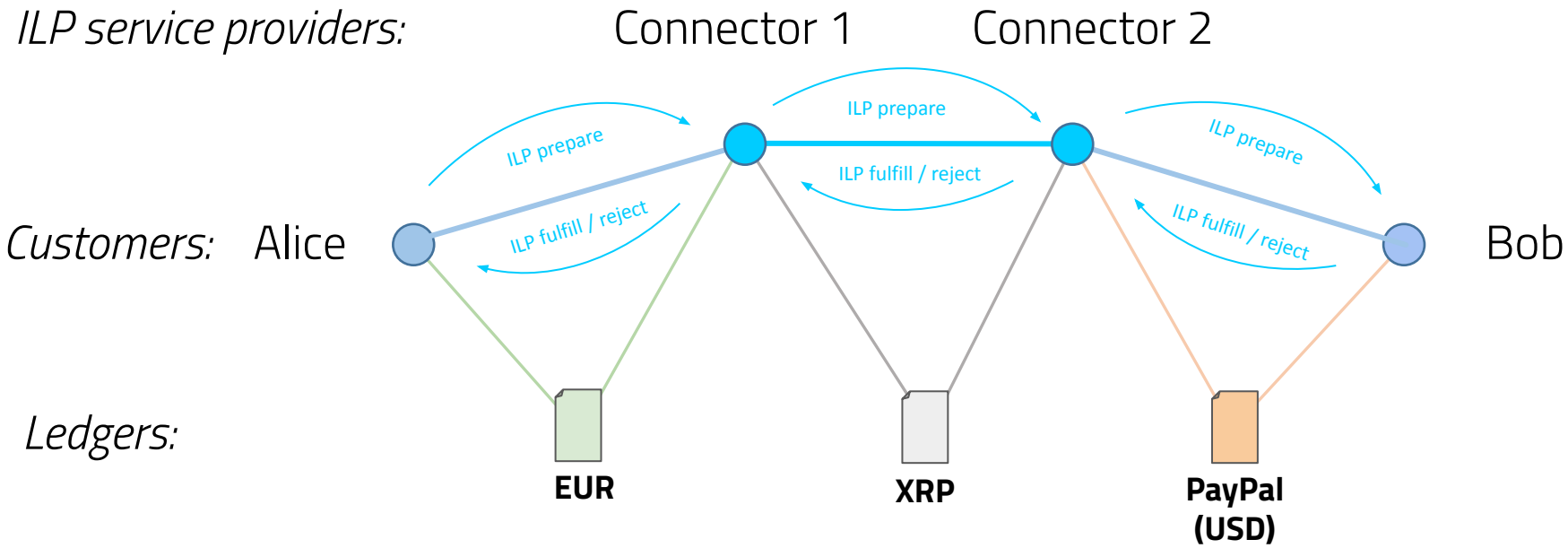


Interledger

- Is an Open protocol
- Lower transaction fees
- Gives user immediate feedback
- Works on any ledger (interoperability)
- Is scalable (infrastructure - wip)
- Aims to fasten bank transfers

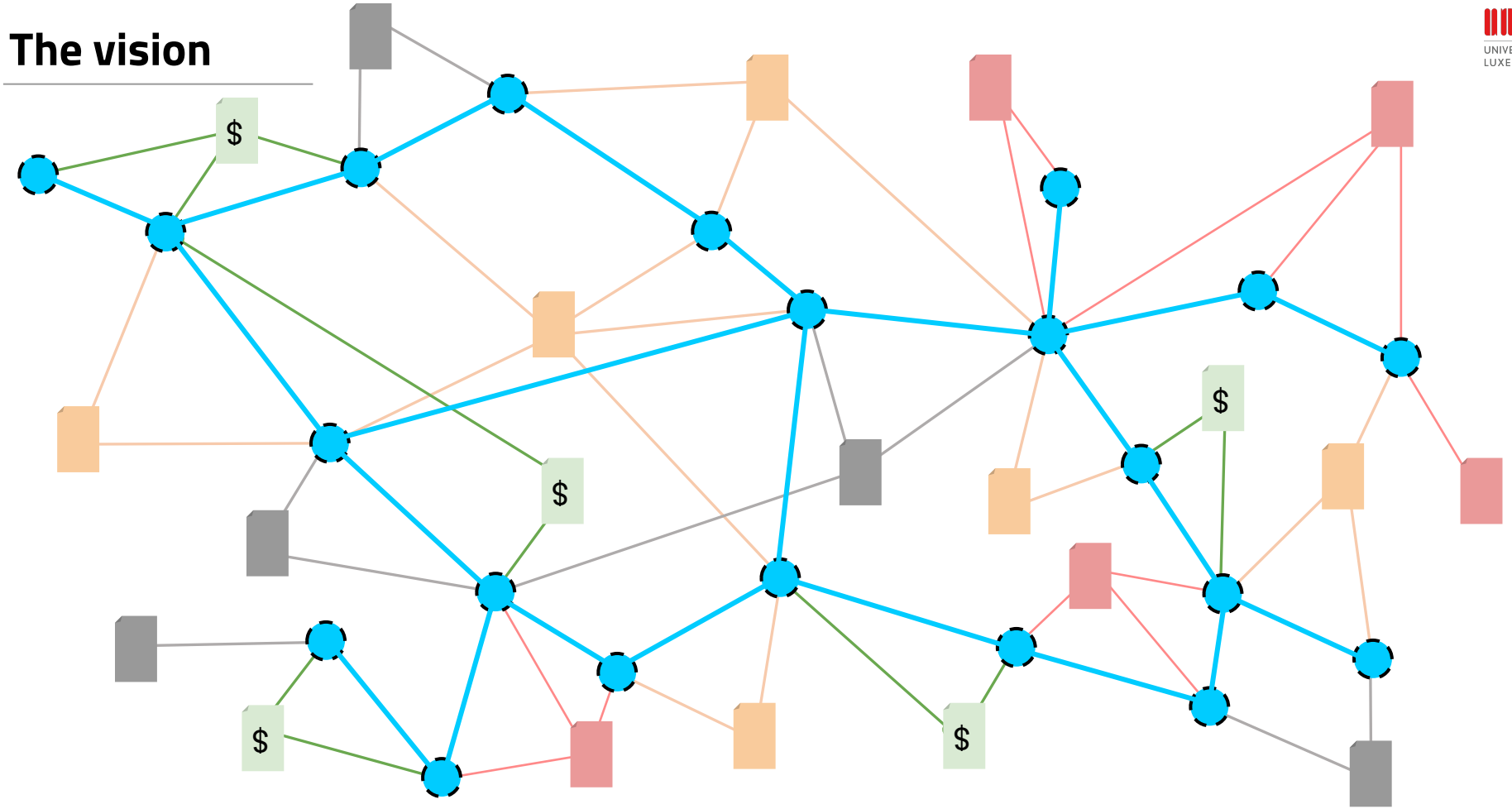
Based on the community group at W3C

Interledger



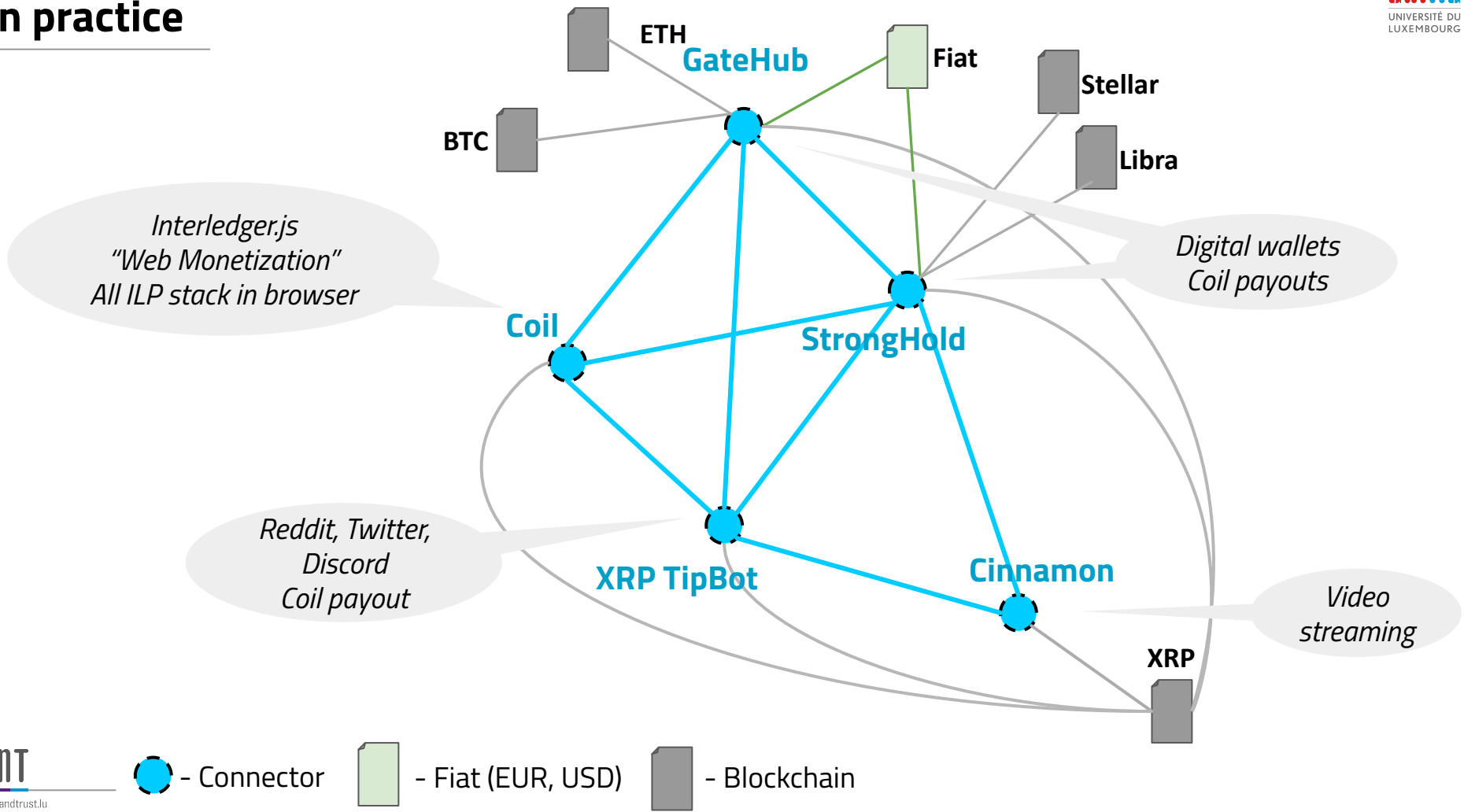
Once connected, two *Connectors* have a ledger between them

The vision



- Connector
$ - Classic Banks
- Blockchain
- Mobile Money (e.g. Google Pay)
- Online Wallets (e.g. PayPal)

In practice


ETH
GateHub
Fiat
Stellar
BTC
Libra
Interledger.js
"Web Monetization"
All ILP stack in browser
Digital wallets
Coil payouts
Coil
StrongHold
Reddit, Twitter,
Discord
Coil payout
XRP TipBot
Cinnamon
Video
streaming
XRP
- Connector
- Fiat (EUR, USD)
- Blockchain

Interledger

INFRASTRUCTURE

Ledger: Ripple Ledger - RippleNet

Connectors: reference connector in JS, production
Rafiki (different architecture)
Rust (performance)
Java

End user apps: Moneyd (stripped connector)
Switch (trading)
SPSP app

PROTOCOLS

SPSP
STREAM
ILP
BTP

CCP
ILDCP

Addressing schemes

SPSP:	*payment pointer:*	$example.com/bob"	Easy use by humans.
	endpoint:	"https://example.com/bob"	Exchange pay details.
STREAM, ILP:	*ILP address:*	"g.acme.bob"	Used for routing.
LEDGER:	*XRP address:*	"rMqUT7uGs6Sz1m9vFr7o85XJ3WDAvgzWmj"	

Interledger packets

Streaming micropayments: $/sec
Average ILP packet amount = $0.00001

ILP prepare packet:

- *amount:* UInt 64
- *expiration*: timestamp
- *execution condition:* UInt256
- *destination:* ILP address
- *data* (info for recipient): octet string

ILP fulfill packet:

- *execution condition fulfillment:* UInt256
- *data* (info for sender): octet string

ILP reject packet:

- *code:* IA5String
- *triggered by*: ILP address
- *message:* UTF8String (for user)
- *data:* octet string (machine readable error)

Interledger

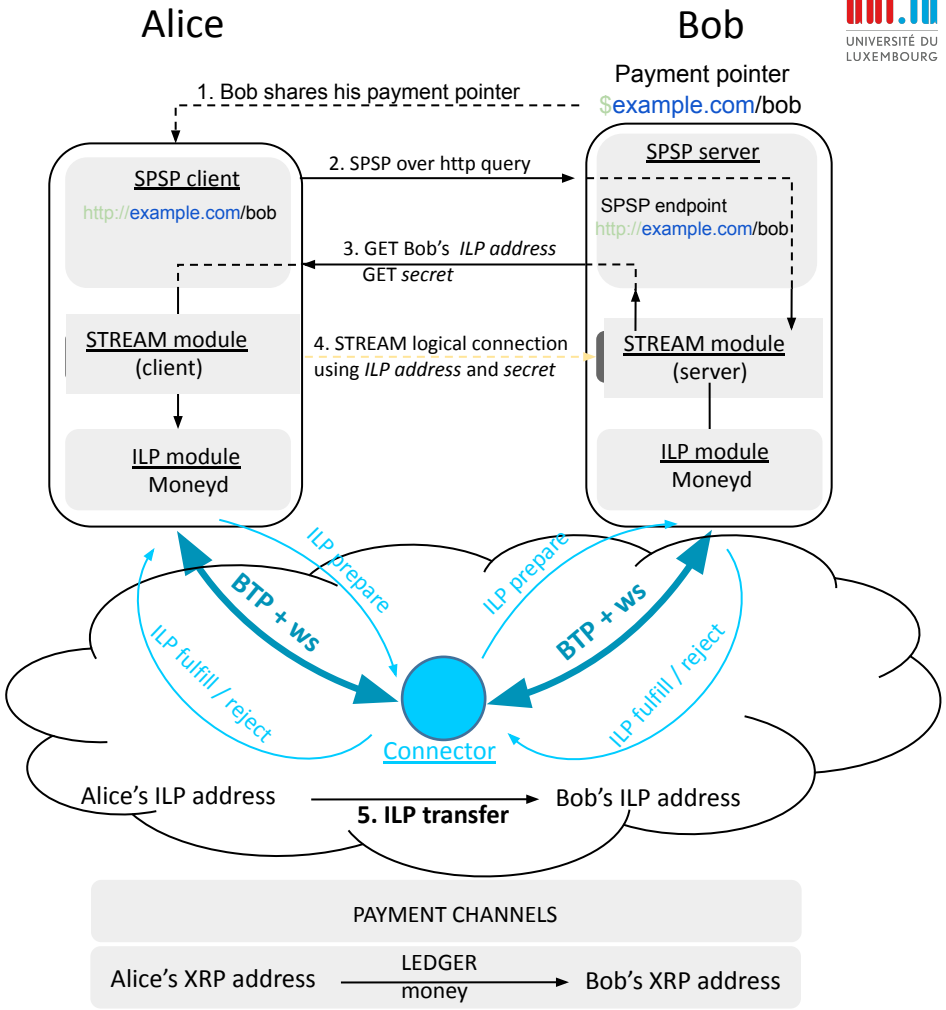
L5. Application: SPSP

L4. Transport: STREAM

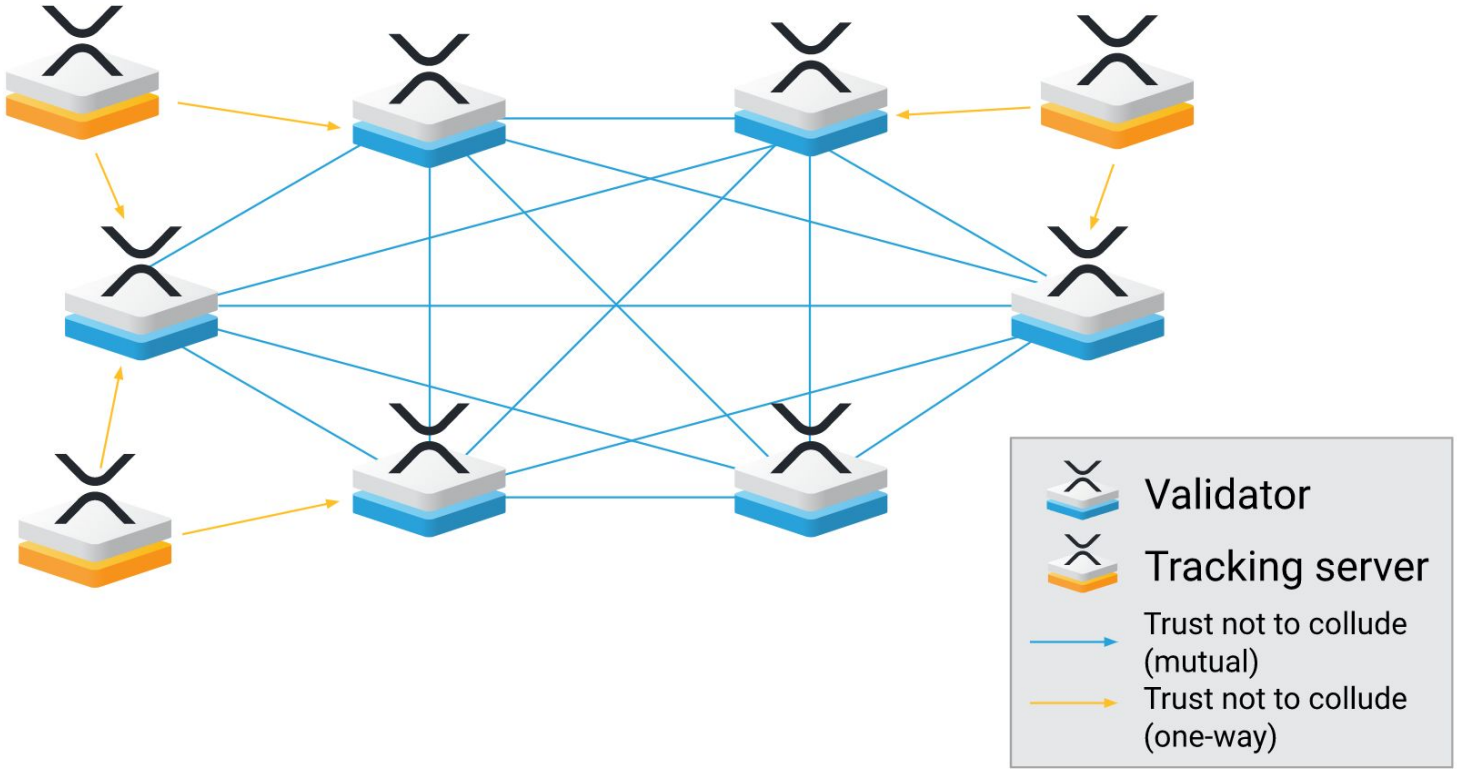
L3. Interledger: ILP

L2. Network: BTP / state channels

L1. Ledgers: classic banks, blockchain



XRPL node types



SOURCE: https://xrpl.org/consensus.html

XRPL node types

Nodes that receive, relay and process transactions (Tx) may be either tracking servers or validators.

Tracking servers: distribute transactions from clients
respond to queries about the ledger

Validating servers: same functions as tracking servers, plus
work to advance the ledger history

For resource loading reasons, it is **recommended** though that validators work as much as possible only to advance the ledger history.

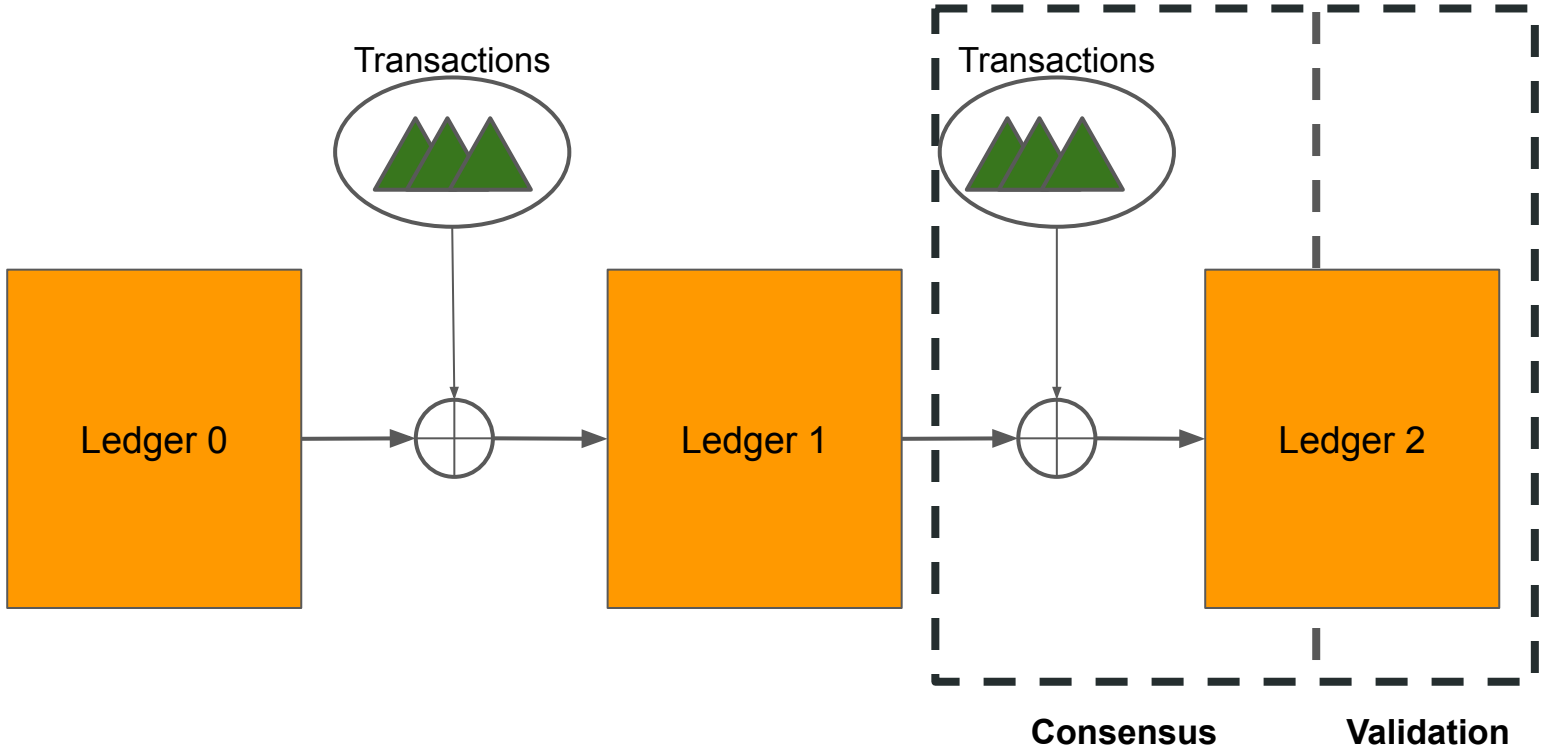
SOURCE: https://github.com/ripple/rippled/blob/develop/docs/consensus.md

XRPL Consensus

Block Chain -> “Ledger Chain”

Consensus: participants agree on the transactions to apply to a prior ledger **AND** time the ledger was closed

Validation: participants agree on what ledger was generated, based on the ledgers generated by chosen peers



SOURCE: https://github.com/ripple/rippled/blob/develop/docs/consensus.md

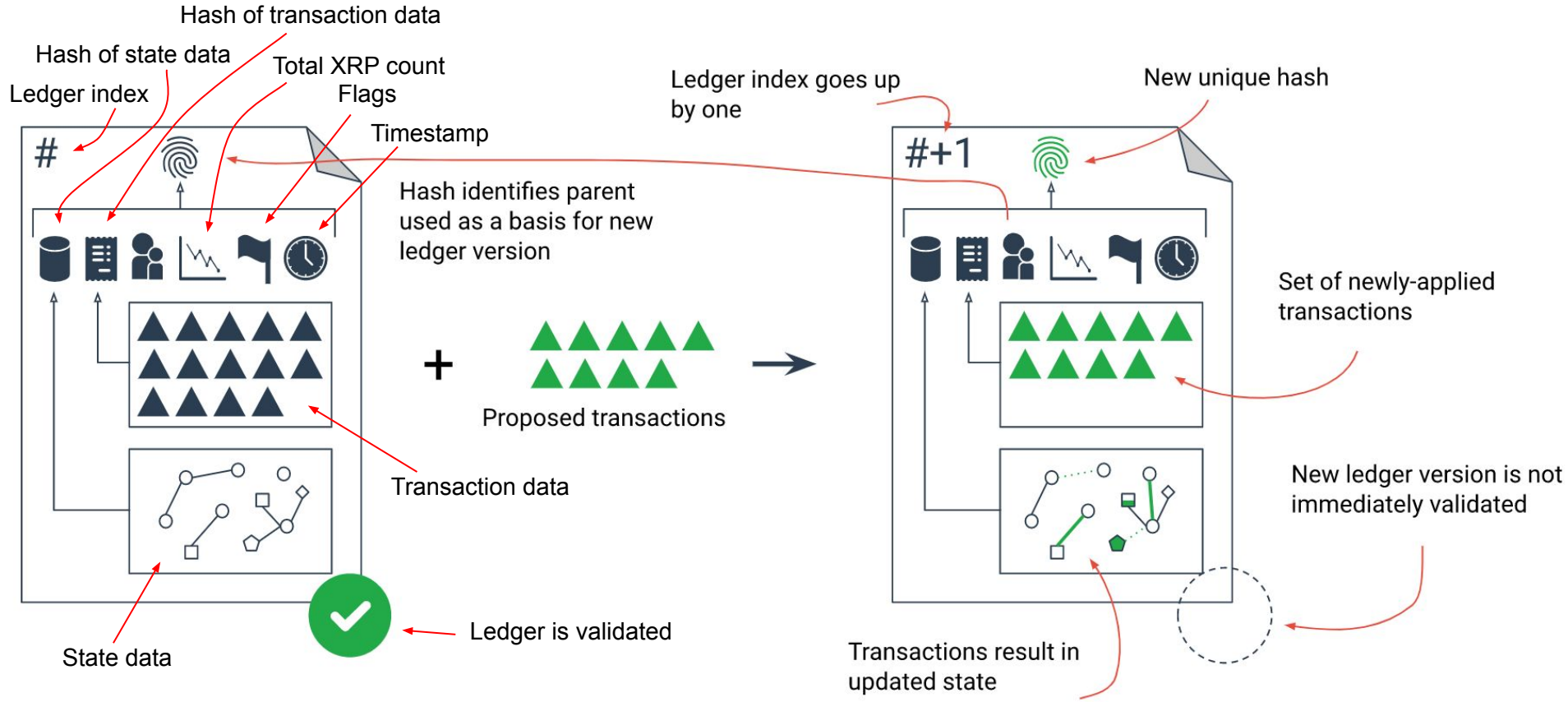
XRPL Consensus

During consensus

The "*previous*" *or* "*last-closed ledger*" : is the most recent ledger seen by consensus
is the basis upon which it will build the next ledger

Transaction set : is a set of transactions under consideration by consensus
It has an unique ID
the goal of consensus is to reach agreement on this set

XRPL Consensus



SOURCE: https://xrpl.org/consensus.html

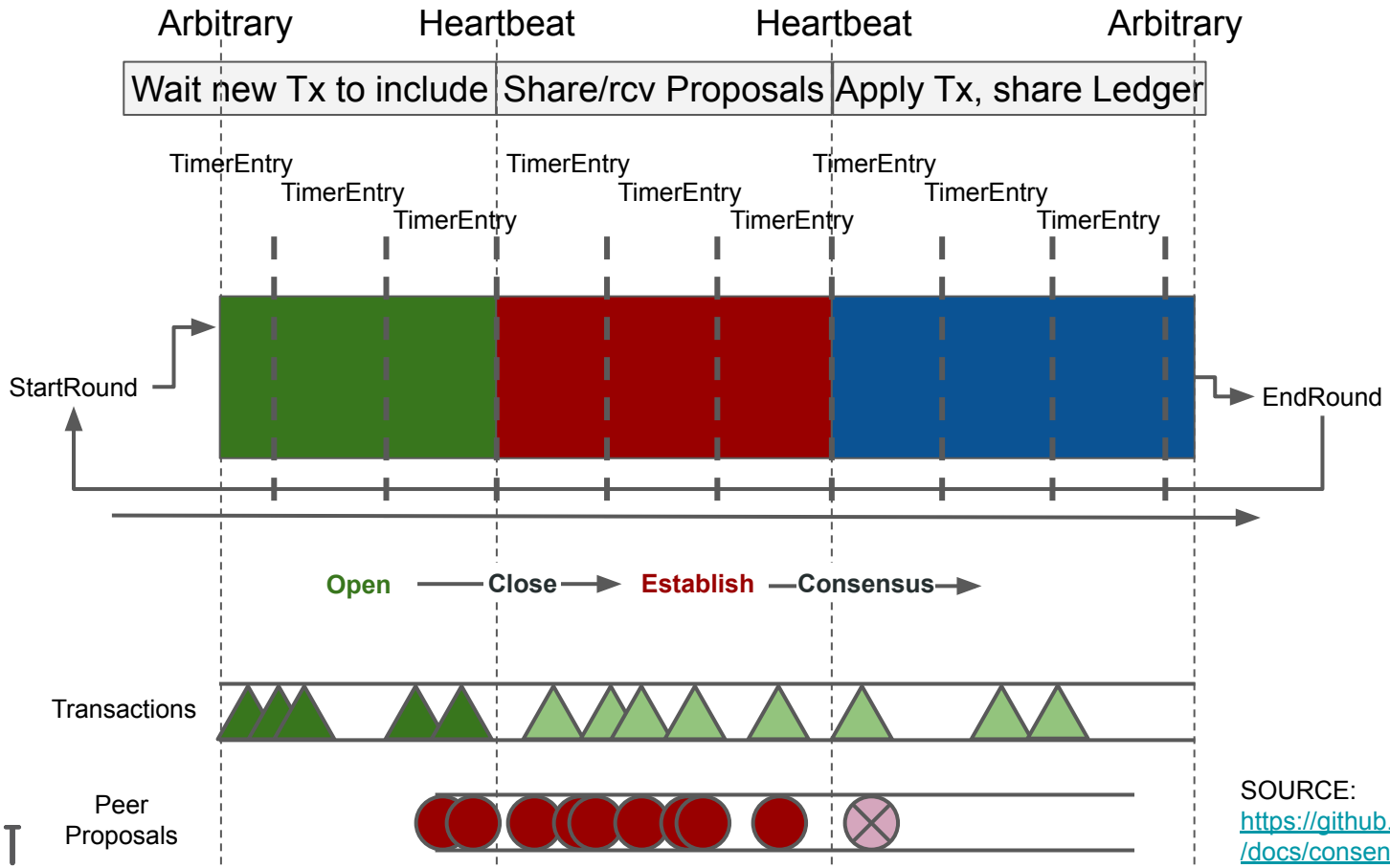
XRPL Consensus

- *Position* is the current belief of the next ledger's transaction set **AND** close time. Position can refer to the node's own position or the position of a peer.

- *Proposal:*
  - Initial proposal - starting position taken by node before it considers any peer positions.
  - If node updates its position in response to its peers, it will issue an updated proposal.
  - ID of proposing node, ID of position taken, ID of prior ledger the proposal is for, seq # of proposal

- *Dispute:* Tx not part of a node's position or not in a peer's position. During consensus, the node will add or remove disputed Tx from its position based on that tx's support amongst peers

XRPL Consensus



Phases
- Open
- Establish
- Accept

Actions
- Start round
- Close
- Consensus
- End Round

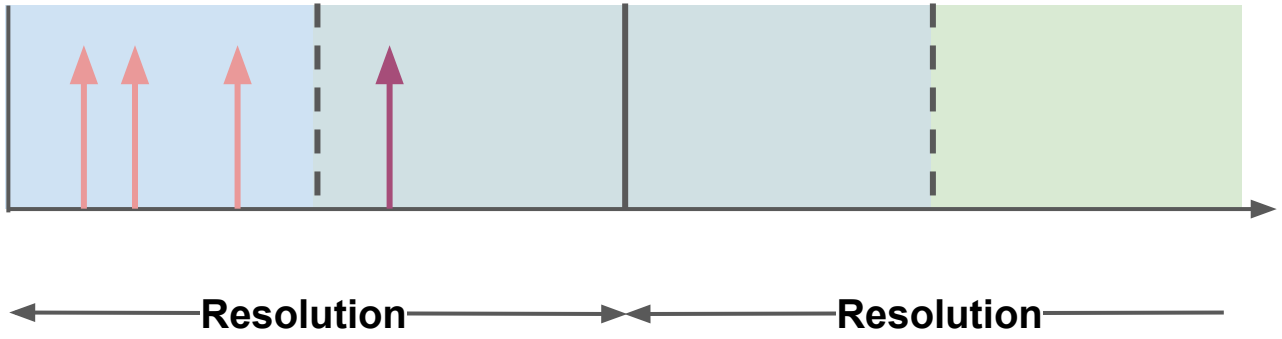
TimerEntry = `LEDGER_MIN_CLOSE`

SOURCE:
https://github.com/ripple/rippled/blob/develop/docs/consensus.md

XRPL Consensus Timings

Each node calculates its own close time when it closes the open ledger.
This exact close time is rounded to the nearest multiple of the current effective close time resolution.

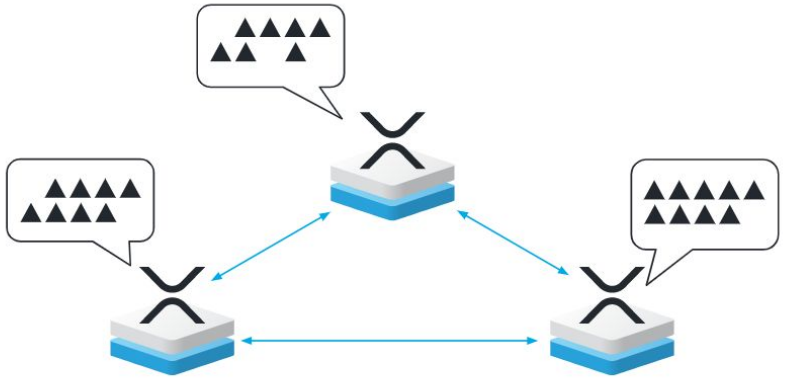
Effective close time is part of the node's position and is shared with peers in its proposals.



SOURCE:
https://github.com/ripple/rippled/blob/develop/docs/consensus.md

XRPL Consensus Modes

Validators each propose a set of transactions to be included in the next ledger version.

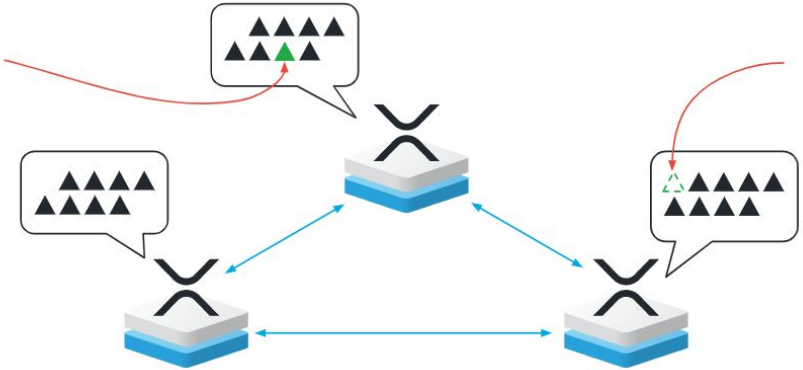


Round 1

Validators add transactions to their proposals if most other validators they trust proposed those transactions

Validators remove transactions if most other validators they trust didn't propose them.

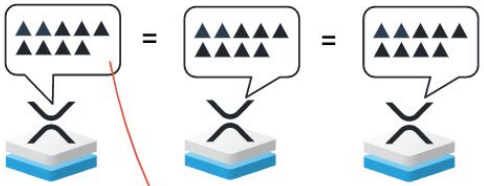
(The removed transactions are usually proposed again for inclusion in the next ledger version.)



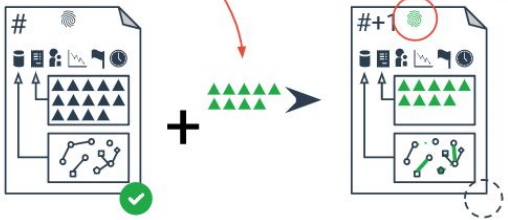
Round 2

SOURCE: https://xrpl.org/consensus.html

XRPL Consensus Modes



Validation begins when validators agree on a transaction set.



Each validator and tracking server calculates the next ledger version by applying the agreed-upon transactions to the previous ledger version.

Validators announce the identifying hash of their resulting ledger to the network.

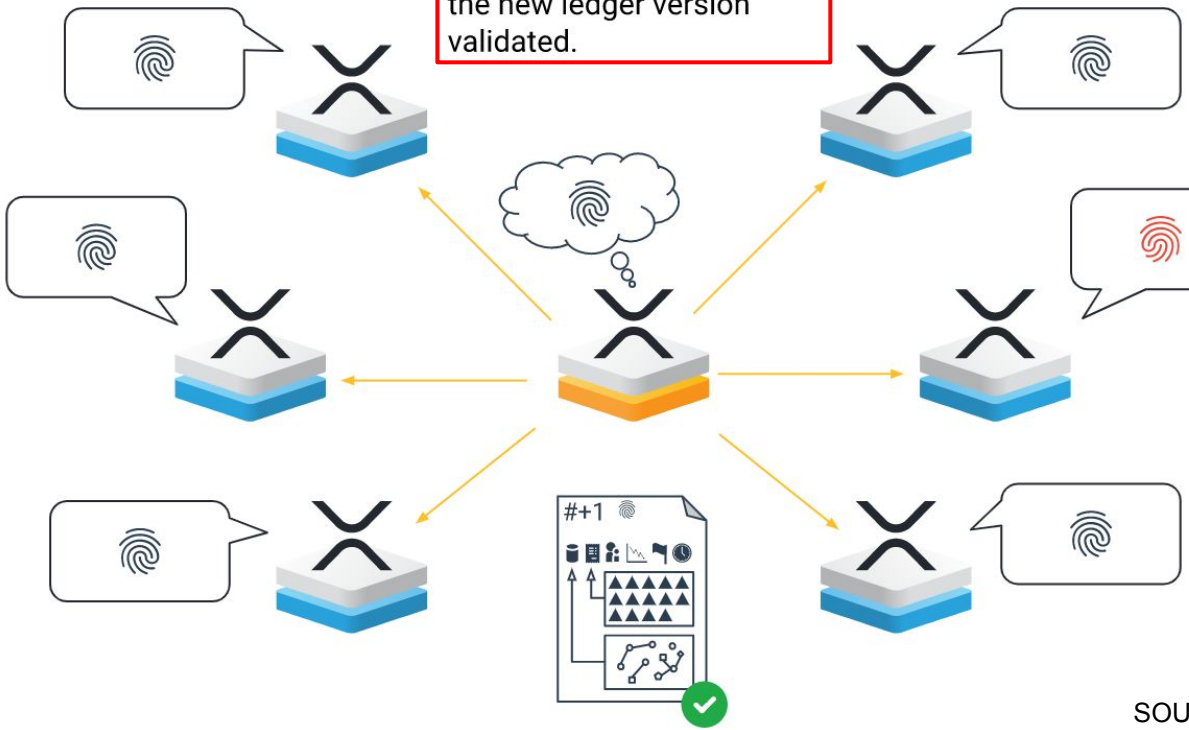
Tracking servers keep their results to themselves.

SOURCE: https://xrpl.org/consensus.html

XRPL Consensus Modes

Quorum

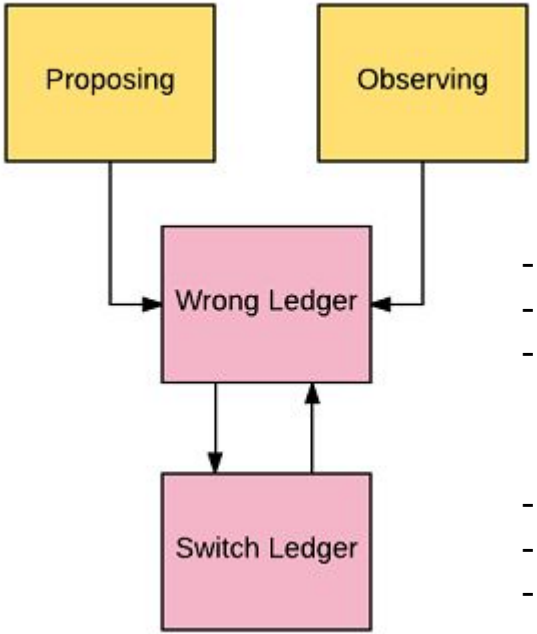
If ≥80% of a server's trusted validators got the same hash, the server declares the new ledger version validated.



If a server got the wrong hash, it must re-calculate or download the correct ledger.

SOURCE: https://xrpl.org/consensus.html

XRPL Consensus Modes

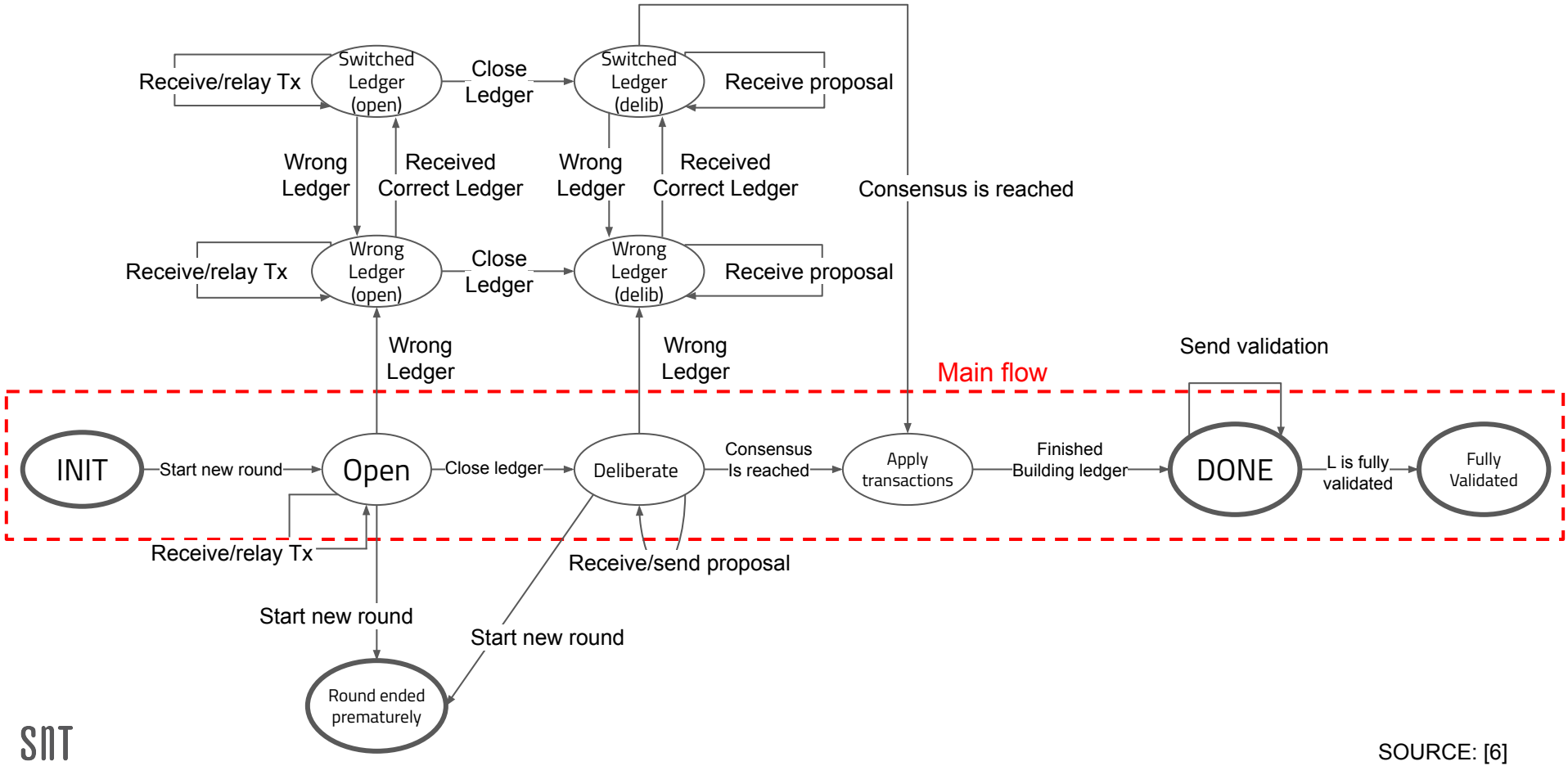


- Bow out of Consensus
- Work this round in Observing mode
- Trigger acquire correct ledger from Network

- This round also in Observing mode
- Switch to Proposing from next round IF all OK
- If NOT OK go back to Wrong Ledger

SOURCE:
https://github.com/ripple/rippled/blob/develop/docs/consensus.md

XRPL Consensus



SOURCE: [6]

XRPL Consensus

Ledger CLOSE

Case 1) Typical behavior: there are transactions in the open ledger AND more than LEDGER_MIN_CLOSE has elapsed.
Case 2) NO open transactions AND a suitably longer idle interval has elapsed.
Increases opportunity to get some tx into the next ledger and avoids doing useless work closing an empty ledger.
Case 3) > half the number of prior round peers already closed or finished present round.
This indicates the node is falling behind and needs to catch up.

ESTABLISH (the active period of consensus)

The node exchanges proposals with peers in an attempt to reach agreement on:
the consensus transactions AND
effective close time

Node is looking for a Tx set supported by a super-majority of peers. Node works towards this set by adding/removing disputed Tx from its position based on an increasing threshold for inclusion.

XRPL Consensus

CONSENSUS declared when all below are true:

- LEDGER_MIN_CONSENSUS time has elapsed in the establish phase
- At least 75% of prior round proposers have proposed OR
  This establish phase is LEDGER_MIN_CONSENSUS longer than the last round's establish phase
- minimumConsensusPercentage of ourself and our peers share the same position

ACCEPT

- Starting from LCL, place the agreed Tx set in canonical order so that all nodes proceed identically
- Process each Tx according to instructions, in order. Update ledger state data
- Update ledger header with appropriate metadata (ledger index, hash of LCL (the "parent"), ledger approx. close time, hashes of ledger's contents)
- Calculate identifying hash of the new ledger

VALIDATION - Validators share their results as signed messages containing the hash of calculated ledger. These messages, called validations, allow servers to check if they got same results and declare a ledger final.

- Calculate the resulting ledger version from an agreed-upon transaction set.
- Compare results and declare the ledger validated IF enough trusted validators agree (quorum)

XRPL Messaging

For robustness, flooding is being used as main dissemination protocol.

Main types of data being flooded:

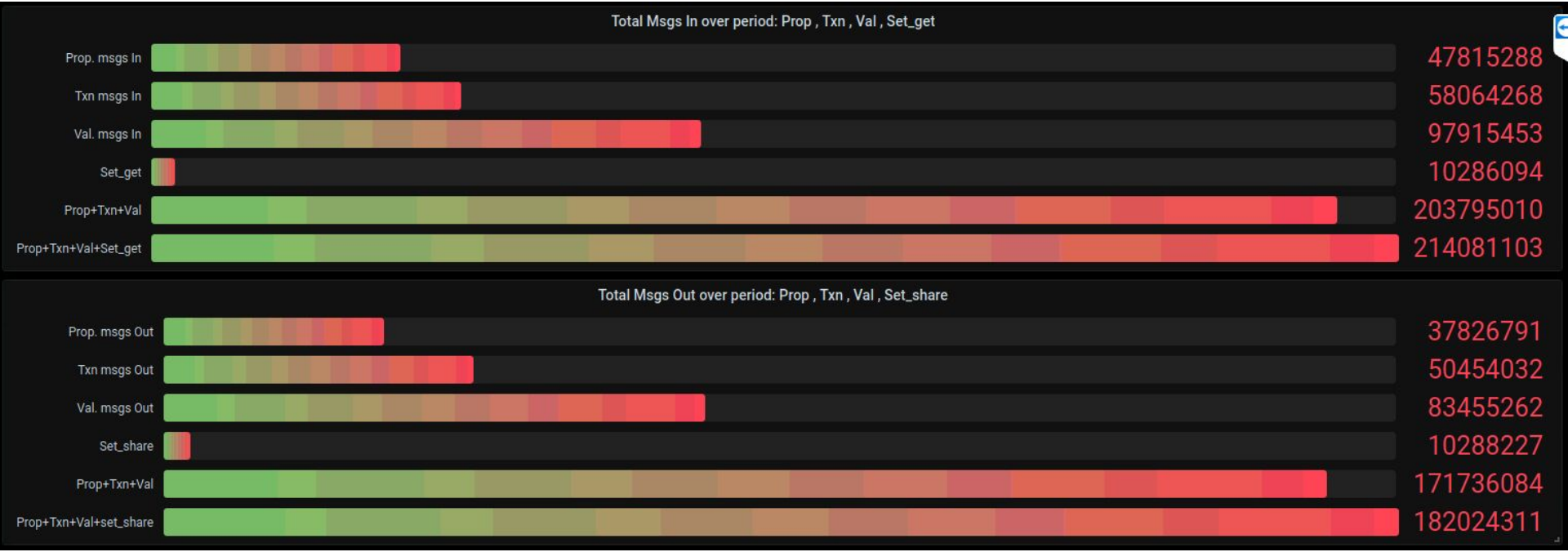
- Transactions (Tx)
- Proposals
- Validations

As the network grows, the overhead incurred by flooding of these messages needs to be addressed:

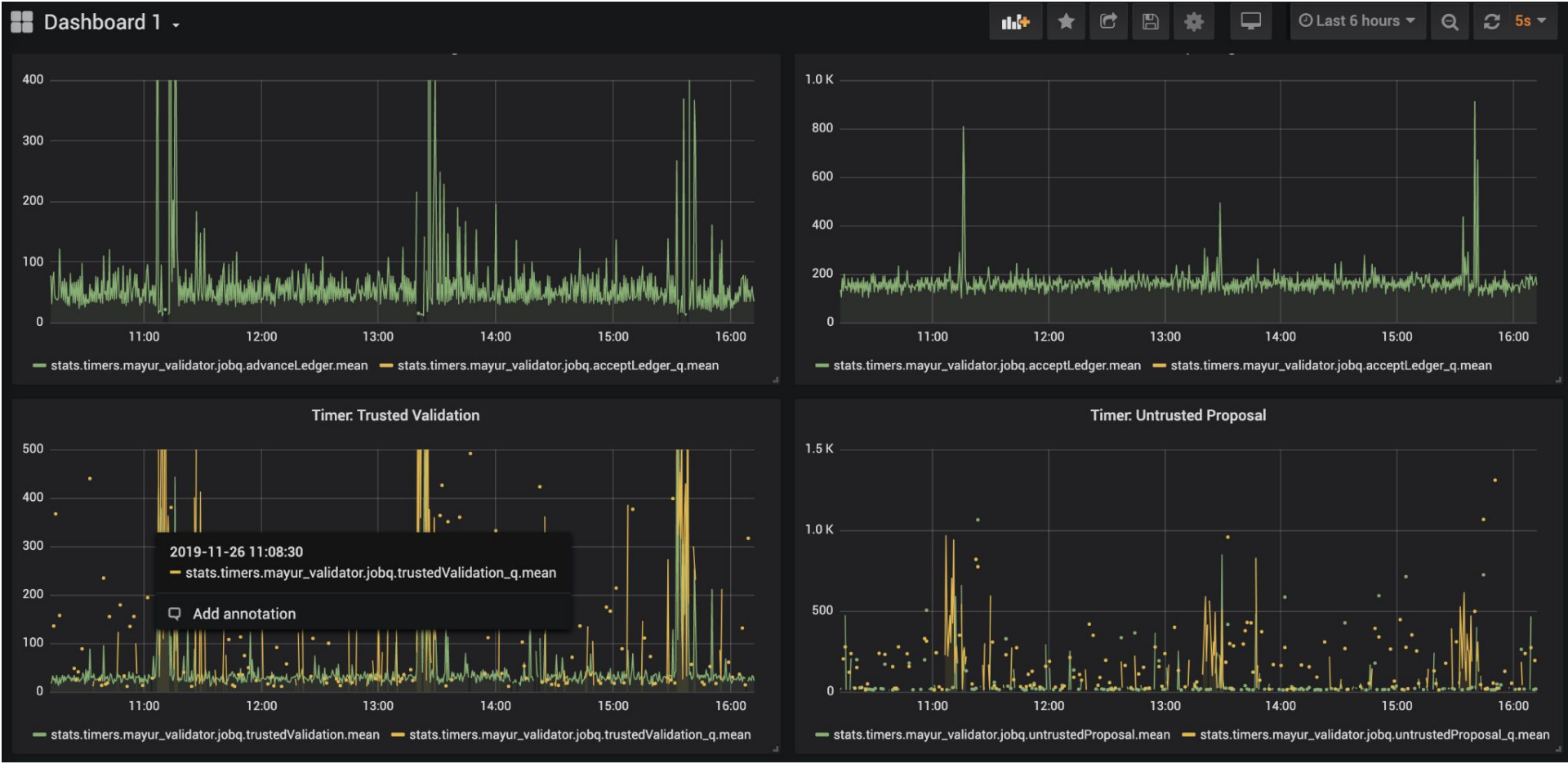
- Impact on network BW and CPU time.
- Needs increasingly higher hw spec and network connection.
- Can result in decrease of overall network performance.

XRPL Messaging

To mitigate flooding overhead, different dissemination improvement approaches could be evaluated.



XRPL Messaging - Rippled Monitor (Grafana) https://github.com/ripple/rippledmon



XRPL Network Topology (2021)

Fetched large topo (158 validators + stock servers) = 892 nodes

- Avg distance = 2.37678. (avg shortest path: sum of all shortest paths between vertex couples / total # of vertex couples)
- Diameter = 5. (greatest dist between any pair of vertices; graph "compactness")
- Giant biconnected component of 867, + 25 smaller ones.

Analysis of the big biconnected comp (867 nodes)

Nodes # 867; Edges # 9172; Mean_degree: 21.158; Max_degree: 296

- mean_dist 2.3405,
- median_dist 2.37,
- diameter 4,
- radius 3

Rank	Degree	ID
1	296	41
2	294	36
3	277	65
4	264	3
5	248	55
..	.	.

HOOKS - Smart Contract proposal for the XRP Ledger

- Small, efficient pieces of code - webassembly binary uploadable to the XRPL.
- 'Layer one' custom code to influence the behaviour and flow of Tx's:
  Allow logic to be executed before and/or after Tx's.
- Deliberately not Turing-Complete. (Undesirable at layer 1 because it is impossible to determine when the program would end. Without predictable max execution durations the ledger might never advance.)
- Currently live on a public testnet. https://hooks-testnet.xrpl-labs.com/
- HOOKS can store small, simple data objects.

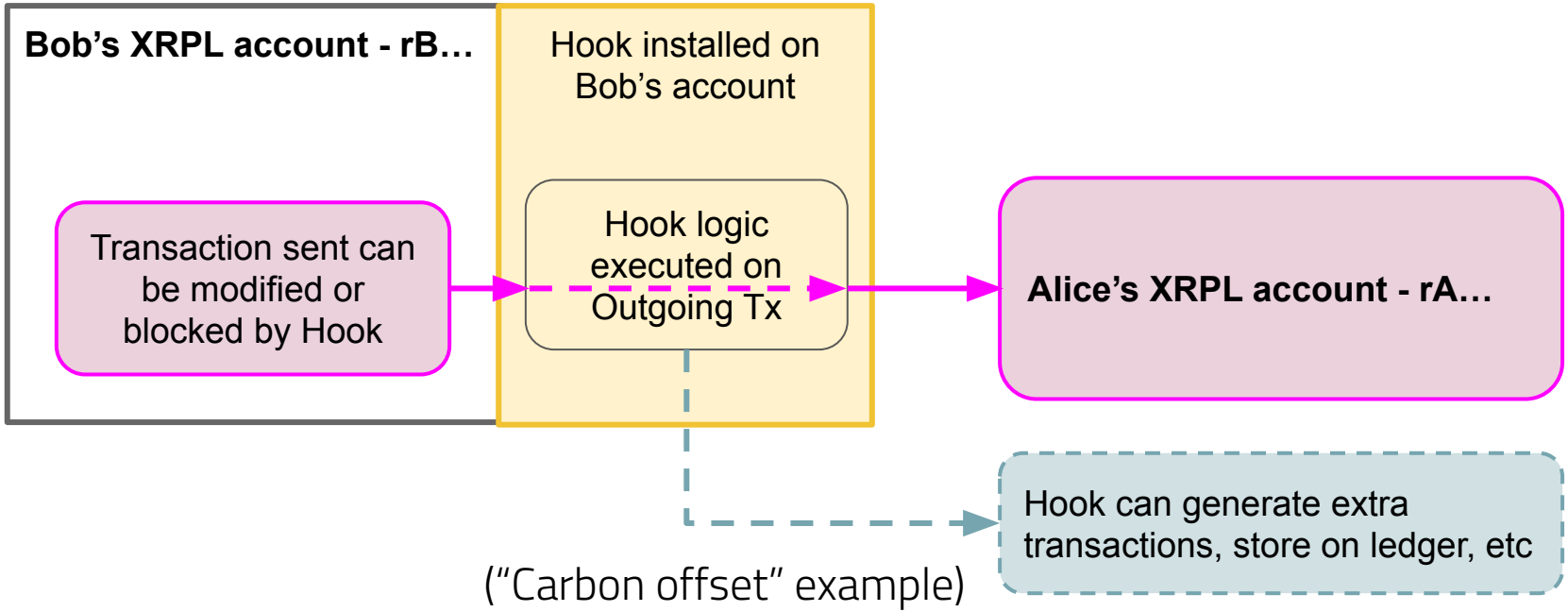
Examples:

- *"reject payments < 10 XRP"*
- *"for all outgoing payments, send 10% to my savings account"*
- *"for incoming payments transactions, check if the sending account is in a **list** maintained by another hook, and if present: reject the transaction"*

HOOKS - Smart Contract proposal for the XRP Ledger

Hook triggered on outgoing transaction

Hook installed on sending account

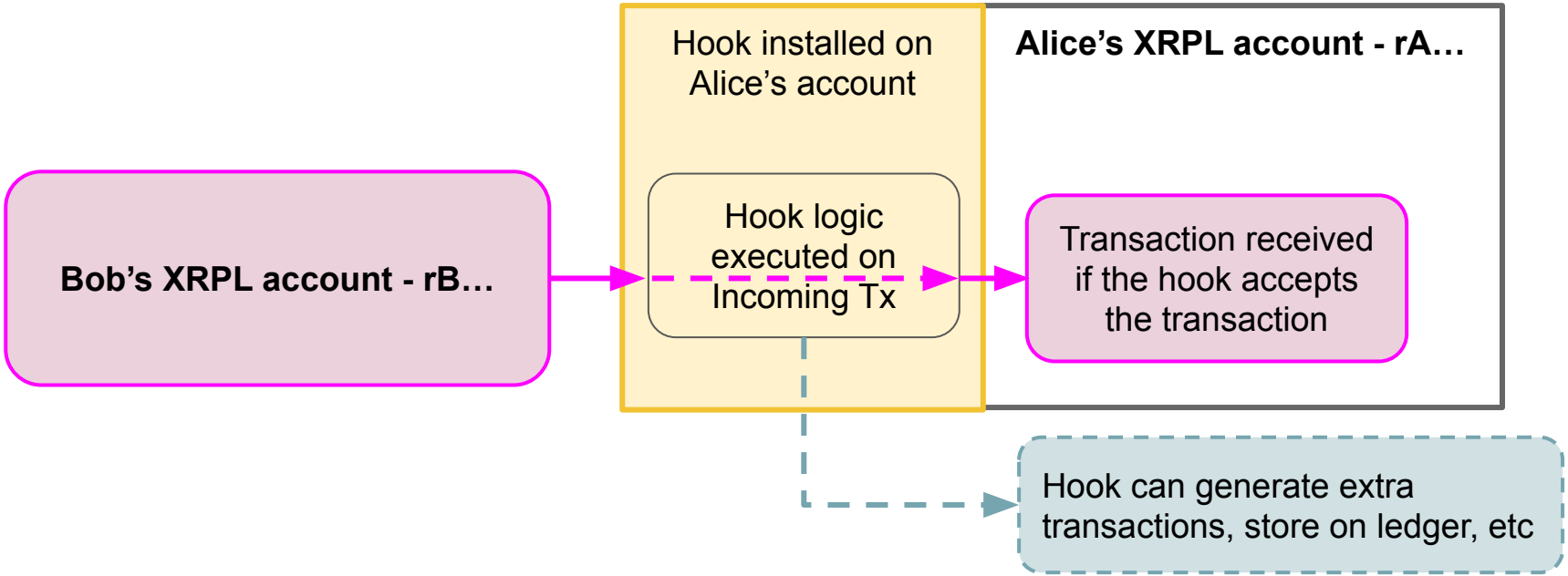


SOURCE: https://xrpl-hooks.readme.io/

HOOKS - Smart Contract proposal for the XRP Ledger

Hook triggered on incoming transaction

Hook installed on **receiving** account



SOURCE: https://xrpl-hooks.readme.io/

Runtime Loop Guarding

"Guards" in the Hook source code allow XRPL to monitor and calculate worst case resource cost, and reject Hooks if they are too inefficient.

```
for (uint32_t i = 0; i < destination_tag; ++i)
{
    // some expensive code here
}
```

0 to 4.29 billion times

```
for (uint32_t i = 0; GUARD(10), i < destination_tag; ++i)
{
    // some expensive code
}
```

```
while(1)
{
    GUARD(10);
    if (i >= destination_tag)
        break;
}
```

“Code” to “XRPL”:
"I Promise I will not hit this guard more than 10 times.
If I do then you may terminate and rollback this hook."

SOURCE: https://dev.to/wietse/hooked-2-hooks-security-smart-contracts-on-the-xrp-ledger-83e

HOOKS anatomy

Emitted Transactions. The Originating Transaction does exactly what the contents of the Transaction say it should. If the Hook needs to make an additional change to the ledger such as sending a payment, it creates and then emits a brand new transaction.

- In the callback (“cbak”) we define the emitted tx of a hook

```
/** This hook just accepts any transaction coming through it **/
#include "../hookapi.h"

// this hook has no emitted tx and therefore no callbacks
int64_t cbak(int64_t reserved) {
   return 0;
}
int64_t hook(int64_t reserved ) {
   TRACESTR("Accept.c: Called.");
   accept (0,0,0);
   _g(1,1);   // every hook needs to import guard function and use it at least once
   // unreachable
   return 0;
}
```

Hook install

Hook program code is installed onto an XRPL account using the SetHook transaction.

```
let sethook_transaction =
{
        Account: "r4GDFMLGJUKMjNhhycgt2d5LXCdXzCYPoc",
        TransactionType: "SetHook",
        CreateCode: fs.readFileSync('firewall.wasm').toString('hex').toUpperCase(),
        HookOn: "0000000000000000"
  }
```

HookOn

Each bit in the unsigned 64-bit integer indicates whether the Hook should execute on a particular transaction type.

SOURCE: https://dev.to/wietse/hooked-2-hooks-security-smart-contracts-on-the-xrp-ledger-83e

NFT support

"Fungible" : can replace or be replaced by another identical item; mutually interchangeable.

"Non-Fungible" : cannot be replaced (1919 US Mail Stamp upside-down).

Fungible

Non-fungible

$1.5 million

XRPL NFTS encode ownership of unique physical, non-physical, or purely digital goods, such as works of art or in-game items.

- XLS-20 standard preliminary NFT implementation that can be used in test networks.
- Not yet available as an amendment on XRP Ledger.
- Should be supported in next XRPL v1.9.0 https://xrpl.org/blog/2022/rippled-1.9.0.html

SOURCE: *https://xrpl.org/non-fungible-tokens.html*

NFT support

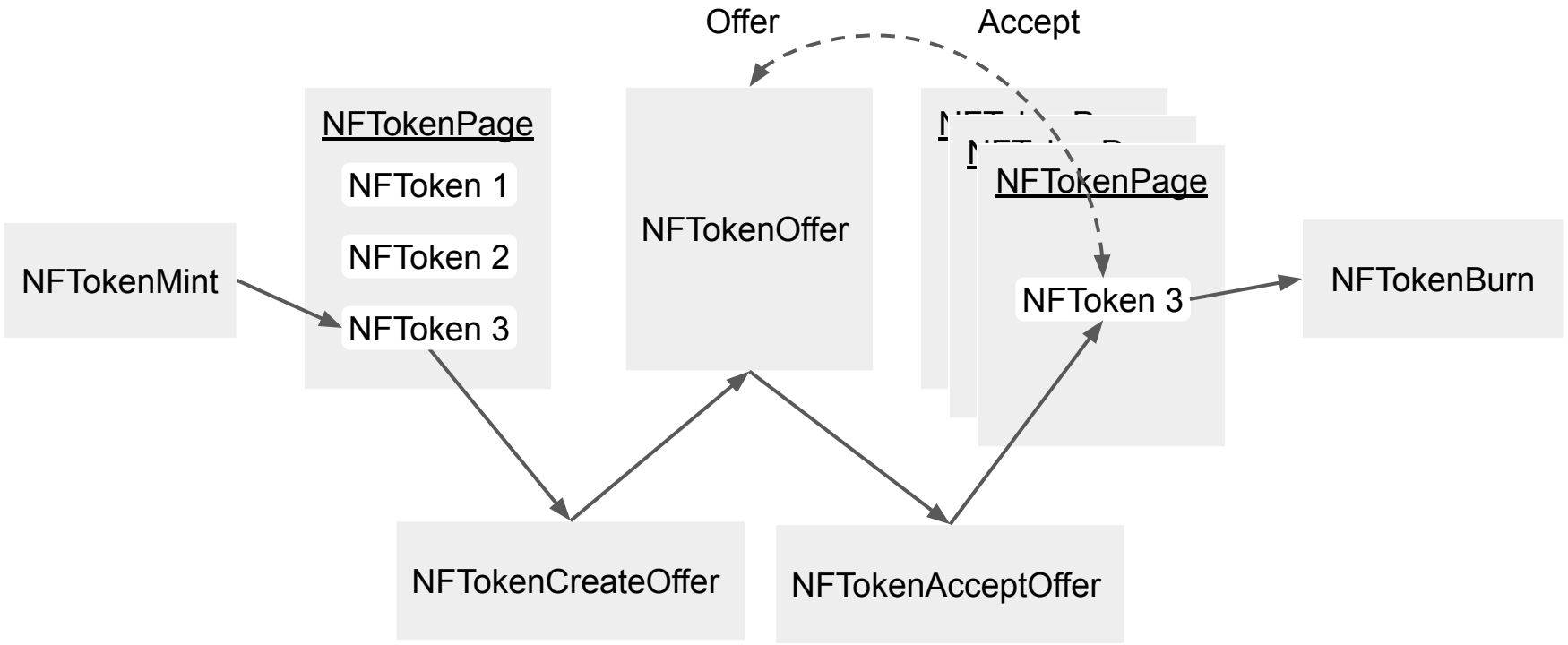
- NFToken, a native XRPL NFT type, can enumerate, purchase, sell, and hold such tokens.
- NFToken is a unique, indivisible unit that is not used for payments.
- NFTokenPage object contains a set of NFToken objects owned by the same account.
- **Create** a new NFToken using the *NFTokenMint* transaction.

  NFToken lives on the NFTokenPage of the issuing account.
- NFTokenOffer object describes an **offer to buy or sell** a single NFToken.
- **Destroy** an NFToken using the *NFTokenBurn* transaction.

More info: https://xrpl.org/label-nfts.html

SOURCE: *https://xrpl.org/non-fungible-tokens.html*

NFT lifecycle



SOURCE: https://xrpl.org/non-fungible-tokens.html

PayString - The Universal Payment Identifier

Complicated, hard-to-remember payment addresses lead to poor UX: confusion, errors, loss of funds. A major barrier to adoption of blockchain and other payments technology.

PayString:

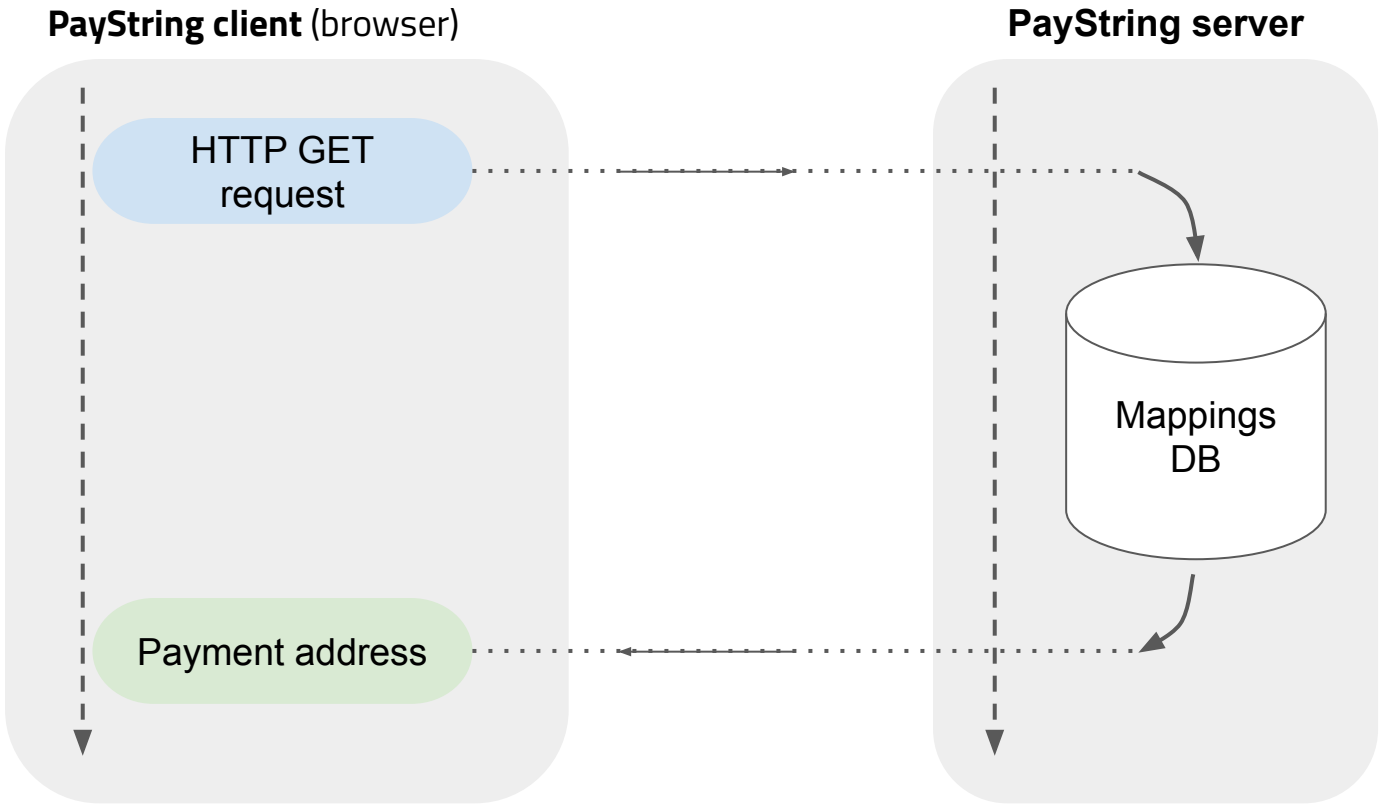
- standard for human-readable payment addresses that can be resolved securely and privately to any payment rail [5] (provide mapping).
- simple request/response application-layer protocol built on top of HTTP and DNS

PayString URI scheme:

- Email style identifier separating user from host with a '$'.
- Resolves to a URL using HTTPS.

PayString URI: *user$host*
PayString URI: *alice$example.net* PayString URL: https://example.net/alice
PayString URI: *john.doe$example.net*
PayString URI: *jane-doe$example.net*

PayString - The Universal Payment Identifier



Problem: anybody can ask anyone's payment details to the PayString server.
>>> Need for access control, authentication, authorization.

Possible improvements | research directions

- Consensus behavior in presence of multiple UNLs
- PayString Privacy
- Scalability issues due to messaging overhead

QUESTIONS

Demo Session

- Deploying an XRP node on testnet ..... 15 min

  Create two XRP accounts, get XRP from XRP faucet and transfer some funds

- Deploying an XRP node on mainnet ..... 10 min

- Deploying a private testbed consisting of 3 nodes ..... 20 min

- Deploying HOOKS smart contracts ..... 10 min

XRPL on Ripple TestNet

1. Deploy an XRP node on Ubuntu
2. Create two XRP accounts
3. Get XRP from XRP faucet
4. Transfer some funds between the two accounts
5. Verify account balances

XRPL on Ripple TestNet

1. Deploy an XRP node
    a. Build from source (will also need to build “Boost”)
    https://github.com/ripple/rippled/blob/develop/Builds/linux/README.md
    (we’ll build from source the XRPL node that we will deploy on MainNet)

    b. Building from packages
    https://xrpl.org/install-rippled-on-ubuntu.html

```
sudo apt -y update
sudo apt -y install apt-transport-https ca-certificates wget gnupg
wget -q -O - "https://repos.ripple.com/repos/api/gpg/key/public" | \
  sudo apt-key add -
apt-key finger
echo "deb https://repos.ripple.com/repos/rippled-deb focal stable" | \
    sudo tee -a /etc/apt/sources.list.d/ripple.list
sudo apt -y update
sudo apt -y install rippled
systemctl status rippled.service
```

XRPL on Ripple TestNet

- Check that rippled is running:

```
sedan@ICBC-testnet:~$ systemctl status rippled.service
● rippled.service - Ripple Daemon
     Loaded: loaded (/lib/systemd/system/rippled.service; enabled; vendor preset: enabled)
     Active: active (running) since Sun 2022-04-24 13:15:26 CEST; 25s ago
   Main PID: 67373 (rippled)
```

If not running: `sudo systemctl start rippled.service`
If we want to enable it to work as a service: `sudo systemctl enable rippled.service`

- Wait a few minutes for the node to sync to the network.
- Checking if the node is running correctly:
  `/opt/ripple/bin/rippled server_info`
  `/opt/ripple/bin/rippled server_state`
  - Look for server state:
    - should be “proposing” if it’s a validator
    - should be “full” if it’s a tracker node
  - Check if it has the latest sequence of ledgers, and that this is not fragmented
- Also one can check the *“Genesys account”* balance.
  The Genesys account is hardcoded, always the same address, even on a private XRPL.
  `/opt/ripple/bin/rippled account_info` **`rHb9CJAWyB4rj91VRWn96DkukG4bwdtyTh`**

XRPL on Ripple TestNet

- Check peers connection:
  `/opt/ripple/bin/rippled peers`
- The logs are in `/var/log/ripped.debug.log` and can be seen with:
  `tail -f /var/log/rippled/debug.log`
- Change log level realtime:
  `sudo /opt/ripple/bin/rippled --conf /opt/ripple/etc/rippled.cfg log_level trace`
  Change log level in `rippled.conf` file:

```
# Turn down default logging to save disk space in the long run.
# Valid values here are trace, debug, info, warning, error, and fatal
[rpc_startup]
{ "command": "log_level", "severity": "warning" }
```

By default, the node will start as a tracker node and will connect to MainNet.
We want to connect to TestNet, so we'll stop it:
`sedan@ICBC-testnet:~$ sudo systemctl stop rippled.service`

Then we tweak the config files for TestNet:
`/opt/ripple/etc/rippled.cfg`

```
# To use the XRP test network
# (see https://xrpl.org/connect-your-rippled-to-the-xrp-test-net.html),
# use the following [ips] section:
# [ips]
# r.altnet.rippletest.net 51235
```

XRPL on Ripple TestNet

/opt/ripple/etc/validators.txt

```
[validator_list_sites]
https://vl.ripple.com
https://vl.xrplf.org

[validator_list_keys]
#vl.ripple.com
ED2677ABFFD1B33AC6FBC3062B71F1E8397C1505E1C42C64D11AD1B28FF73F4734
# vl.xrplf.org
ED45D1840EE724BE327ABE9146503D5848EFD5F38B6D5FEDE71E80ACCE5E6E738B

# To use the test network (see https://xrpl.org/connect-your-rippled-to-the-xrp-test-net.html),
# use the following configuration instead:
#
# [validator_list_sites]
# https://vl.altnet.rippletest.net
#
# [validator_list_keys]
# ED264807102805220DA0F312E71FC2C69E1552C9C5790F6C25E3729DEB573D5860
```

MainNet

TestNet

- **EXAMPLE** configs: https://github.com/treisto/ICBC22/tree/main/testNetXRPL
- We start the node again, this time from the command line (we see the log in terminal):

```
sudo /opt/ripple/bin/rippled -conf /opt/ripple/etc/rippled.conf
```

- After 15 minutes we check that the node is synchronized to the network:

```
sudo /opt/ripple/bin/rippled -conf /opt/ripple/etc/rippled.conf server_info
```

XRPL on Ripple TestNet

```
sedan@ICBC-testnet:~$ sudo /opt/ripple/bin/rippled --conf /opt/ripple/etc/rippled.cfg server_info
Loading: "/opt/ripple/etc/rippled.cfg"
2022-Apr-24 12:32:25.130754515 UTC HTTPClient:NFO Connecting to 127.0.0.1:5005

{
   "result" : {
      "info" : {
         "build_version" : "1.8.5",
         "complete_ledgers" : "27204049-27204330",
         "fetch_pack" : 1331,
         "hostid" : "ICBC-testnet",
         "peers" : 3,
         "pubkey_node" : "n9Mha2VHuL9cNQvCCHuoSrtJz8LFPZejCQ8J7e7wHi5nZukpP2Gh",
         "pubkey_validator" : "none",
         "server_state" : "full",
         "validation_quorum" : 5,
         "validator_list" : {
            "count" : 1,
            "expiration" : "2022-May-24 00:00:00.000000000 UTC",
            "status" : "active"
         }
```

XRPL on Ripple TestNet

```
sedan@ICBC-testnet:~$ sudo /opt/ripple/bin/rippled --conf /opt/ripple/etc/rippled.cfg server_info
Loading: "/opt/ripple/etc/rippled.cfg"
2022-Apr-24 12:32:25.130754515 UTC HTTPClient:NFO Connecting to 127.0.0.1:5005

{
   "result" : {
      "info" : {
         "build_version" : "1.8.5",
         "complete_ledgers" : "27204049-27204330",
         "fetch_pack" : 1331,
         "hostid" : "ICBC-testnet",
         "peers" : 3,
         "pubkey_node" : "n9Mha2VHuL9cNQvCCHuoSrtJz8LFPZejCQ8J7e7wHi5nZukpP2Gh",
         "pubkey_validator" : "none",
         "server_state" : "full",
         "validation_quorum" : 5,
         "validator_list" : {
            "count" : 1,
            "expiration" : "2022-May-24 00:00:00.000000000 UTC",
            "status" : "active"
         }
```

XRPL on Ripple TestNet

We generate two TestNet accounts, each holding 1000 XRP from the XRP faucet: https://xrpl.org/xrp-testnet-faucet.html

Address	rrpJ22B6cgGzFw9ZaF72bM63T8siVH1Tbe	rJoDXhe8jodS7Mtdgytk56pDC92EURH5TA
Secret	snVW7jLiUzjfPRuJQMYKtJohGazVV	sawDcRubxnWjy5VVbsMpEPWLyNx15
Warning: The above are test accounts with fake money. Do not disclose your MainNet account's Secret!		
Balance	1,000 XRP	1,000 XRP
Sequence Number	27204483	27204567

We check these accounts' balances from our TestNet node:

```
/opt/ripple/bin/rippled --conf /opt/ripple/etc/rippled.cfg account_info rrpJ22B6cgGzFw9ZaF72bM63T8siVH1Tbe
/opt/ripple/bin/rippled --conf /opt/ripple/etc/rippled.cfg account_info rJoDXhe8jodS7Mtdgytk56pDC92EURH5TA
```

```
sedan@ICBC-testnet:~$ sudo /opt/ripple/bin/rippled --conf /opt/ripple/etc/rippled.cfg account_info rrpJ22B6cgGzFw9ZaF72bM63T8siVH1Tbe
Loading: "/opt/ripple/etc/rippled.cfg"
2022-Apr-24 12:51:35.664297280 UTC HTTPClient:NFO Connecting to 127.0.0.1:5005

{
   "result" : {
      "account_data" : {
         "Account" : "rrpJ22B6cgGzFw9ZaF72bM63T8siVH1Tbe",
         "Balance" : "1000000000",
         "Flags" : 0,
         "LedgerEntryType" : "AccountRoot",
         "OwnerCount" : 0,
         "PreviousTxnID" : "56EF947E6D7DB1D81A8133B39D303A4C651F201A311F0268A0CAB997A0D45F19",
         "PreviousTxnLgrSeq" : 27204483,
         "Sequence" : 27204483,
         "index" : "4F77F9A06AD6EF44826AC2B9FB1BD02742C20F870D560AAA081FE8295DD77115"
      },
```

XRPL on Ripple TestNet

To transfer funds we need to install an extra piece of code, “ripple-lib”.
For this we also need to install nvm.

```
sudo apt-get install curl
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash
```
https://github.com/nvm-sh/nvm

```
restart terminal
nvm ls-remote (to check latest stable version)
nvm install v16.14.2
node -v
cd /home/user/
git clone https://github.com/treisto/ICBC22.git
cd ICBC22
npm i ripple-lib
cd fund_accounts
node fund_accounts.js (must update accounts’ credentials with your own; the script can connect to local or remote node)
```

Check accounts’ balances again:

```
/opt/ripple/bin/rippled --conf /opt/ripple/etc/rippled.cfg account_info rrpJ22B6cgGzFw9ZaF72bM63T8siVH1Tbe
/opt/ripple/bin/rippled --conf /opt/ripple/etc/rippled.cfg account_info rJoDXhe8jodS7Mtdgytk56pDC92EURH5TA
```

```
sedan@ICBC-testnet:~/ICBC22/fund_accounts$ /opt/ripple/bin/rippled --conf /opt/ripple/etc/rippled.cfg account_info rrpJ22B6cgGzFw9ZaF72bM63T8siVH1Tbe
Loading: "/opt/ripple/etc/rippled.cfg"
2022-Apr-24 14:07:39.799329038 UTC HTTPClient:NFO Connecting to 127.0.0.1:5005

{
   "result" : {
      "account_data" : {
         "Account" : "rrpJ22B6cgGzFw9ZaF72bM63T8siVH1Tbe",
         "Balance" : "989999988",
```

XRPL on Ripple TestNet

```
sedan@ICBC-testnet:~/ICBC22/fund_accounts$ node transfer-funds.js
Connected...
A4A78FA3815EDD397529525AE6B6C4B490B6B251F19A8E6A18AE6E0814D95D7C
{
  "resultCode": "tesSUCCESS",
  "resultMessage": "The transaction was applied. Only final in a validated ledger.",
  "engine_result": "tesSUCCESS",
  "engine_result_code": 0,
  "engine_result_message": "The transaction was applied. Only final in a validated ledger.",
  "tx_blob": "120000228000000024019F1B83201B019F21D96140000000009896806840000000000000C7321033F8A4A8CCDDB0E763CC296FBA4
EEDC71A621E6A1063EAF783C8C2AFD562D85417447304502210 0F093BA32A64E10C2B26495F0B0516CCD8DD06A44297C05F92C843AF6CDD772EC0220
48945EF50241CF8E0147A0D10CD38BF112B63ED192AD2AD9EF3B8616FFD547378114003F170C37D5CF901A333CEADD1358411843DDA98314C337B8EB
3E64D2928F573D862F2A1D485B18EA22",
  "tx_json": {
    "Account": "rrpJ22B6cgGzFw9ZaF72bM63T8siVH1Tbe",
    "Amount": "10000000",
    "Destination": "rJoDXhe8jodS7Mtdgytk56pDC92EURH5TA",
    "Fee": "12",
    "Flags": 2147483648,
    "LastLedgerSequence": 27206105,
    "Sequence": 27204483,
    "SigningPubKey": "033F8A4A8CCDDB0E763CC296FBA4EEDC71A621E6A1063EAF783C8C2AFD562D8541",
    "TransactionType": "Payment",
    "TxnSignature": "3045022100F093BA32A64E10C2B26495F0B0516CCD8DD06A44297C05F92C843AF6CDD772EC022048945EF50241CF8E0147A
0D10CD38BF112B63ED192AD2AD9EF3B8616FFD54737",
    "hash": "A4A78FA3815EDD397529525AE6B6C4B490B6B251F19A8E6A18AE6E0814D95D7C"
  }
}
```

XRPL on Ripple TestNet

Method 2:

```
cd /home/user/ICBC22
cd fund_accounts
node get-accounts-info.js (update account credentials with your own)
```

```
sedan@ICBC-testnet:~/ICBC22/fund_accounts$ node get-accounts-info.js
Getting account info for Genesis address:  rHb9CJAWyB4rj91VRWn96DkukG4bwdtyTh
{
  sequence: 40,
  xrpBalance: '45857.597577',
  ownerCount: 0,
  previousAffectingTransactionID: '3AA035300A9C12262A58D1E94AA6199A2041304ED6903B183E7A60B1E3BE438F',
  previousAffectingTransactionLedgerVersion: 27120425
}
getAccountInfo done
################################################################################
Getting account info for Account1:  rrpJ22B6cgGzFw9ZaF72bM63T8siVH1Tbe
{
  sequence: 27204484,
  xrpBalance: '989.999988',
  ownerCount: 0,
  previousAffectingTransactionID: 'A4A78FA3815EDD397529525AE6B6C4B490B6B251F19A8E6A18AE6E0814D95D7C',
  previousAffectingTransactionLedgerVersion: 27206102
}
getAccountInfo done
################################################################################
Getting account info for Account2:  rJoDXhe8jodS7Mtdgytk56pDC92EURH5TA
{
  sequence: 27204567,
  xrpBalance: '1010',
  ownerCount: 0,
  previousAffectingTransactionID: 'A4A78FA3815EDD397529525AE6B6C4B490B6B251F19A8E6A18AE6E0814D95D7C',
  previousAffectingTransactionLedgerVersion: 27206102
}
getAccountInfo done
################################################################################
done and disconnected.
```

Other XRP Ledger methods

Other rippled methods (selection):

Public

account_objects - Get all ledger objects owned by an account.
account_tx - Get info about an account's transactions.
ledger - Get info about a ledger version.
submit - Send a transaction to the network.
sign - Cryptographically sign a transaction.

Admin

log_level - Get or modify log verbosity.
wallet_propose - Generate keys for a new account.
validation_create - Generate formatted for rippled node key pair.
stop - Shut down the rippled server.
peers - Get information about the peer servers connected.
consensus_info - Get information about the state of consensus as it happens.
validator_info - Get information about the server's validator settings, if configured as a validator.
validators - Get information about the current validators.
print - Get information about internal subsystems.

More at https://xrpl.org/rippled-api.html

Deploy an XRPL node on Ripple MainNet

Build from source https://github.com/ripple/rippled/blob/develop/Builds/linux/README.md

```
apt-get update
apt-get install -y gcc g++ wget git cmake pkg-config libprotoc-dev protobuf-compiler libprotobuf-dev
libssl-dev
Download and build Boost:
wget https://boostorg.jfrog.io/artifactory/main/release/1.70.0/source/boost_1_70_0.tar.gz
tar -xzf boost_1_70_0.tar.gz
cd boost_1_70_0
./bootstrap.sh
./b2 headers
./b2 -j<Num Parallel> (how many threads to use during compilation)
cd ~
git clone https://github.com/ripple/rippled.git
cd rippled
git checkout master
export BOOST_ROOT=~/boost_1_70_0
mkdir my_build
cd my_build
cmake -DCMAKE_BUILD_TYPE=Release -DBOOST_ROOT=~/user/boost_1_70_0 -DCMAKE_INSTALL_PREFIX=/opt/local ..
cmake --build . --target install -- -j <parallel jobs>
```

!!! Currently last master branch version 1.9.0 does not work. Need to download and manually unpack previous version 1.8.5 from git tag.

Default installation will run the rippled node in tracker mode.

Deploy an XRPL node on Ripple MainNet

To deploy as **Validator**:

Generate 1 validator key and token pair:

```
sedan@ICBC-testnet:/opt/ripple/bin$ ./validator-keys create_keys
Validator keys stored in /home/sedan/.ripple/validator-keys.json

This file should be stored securely and not shared.

sedan@ICBC-testnet:/opt/ripple/bin$ ./validator-keys create_token
Update rippled.cfg file with these values and restart rippled:

# validator public key: nHUUT2rn1TLZYWStJuPDSJNZnkfpektpy2SxvUeSkegDwJ1xWMmS

[validator_token]
eyJtYW5pZmVzdCI6IkpBQUFBQUZ4SWUxdlR4N08zTTBlaENidVQxOHBBZmtFZG1qRHlRVXBi
ajlsTnJtdHhERXRWSE1oQTloY01PS1A3cVFQREsybUJldnQ2ZzY1ckdsTHovTzBmYWpjWGVW
K0p6ckRka2N3UlFJaEFQUXQ5UnVCd1pQSXV3aHIyTmhYZ2svMFY0VlpON2Y5ekwrSzJ4OFR4
aksxQWlBUlN3REErMExLOFJ2ZVNXbnRCVlBsdE15Q0ZSUElJKzJaMVY3N2pwZ3MrSEFTUUt6
d3pKZ0lINVhGZ3BjVzdLaEZXdEUrUEc0L1lPWi9GU1FJblUzWFBBL3JqNk1hV3RQU2NyVUpZ
dXFXKzZjaUx5NUZFYXJtK0NZL2tJNDhFUGgxRXc0PSIsInZhbGlkYXRpb25fc2VjcmV0X2tl
eSI6IjlBRTVCNDA1MDkzNjU4RUE4OEUxMERCREM2ODcwQzhEQ0M1QzUxRTZCQjEyNEZCMzA0
MTVDNkQ2NUMxN0Y0MDAifQ==
```

Update *rippled.cfg* as instructed, then restart the node.

As the network grew, it became harder to maintain a node in sync on mainNet with commodity hardware.

https://xrpl.org/system-requirements.html

Deploy an XRPL node on Ripple MainNet

To improve chances to sync a validator node on MainNet (and keep it synced):

- Hardware

  RAM 64Gb, [node_size]="huge"

  High-grade SSD with low-latency random reads and high throughput
  - Over 10,000 reads/sec (heavily-used public server clusters)
  - Over 7,000 writes/sec (dedicated performance testing)

    https://xrpl.org/capacity-planning.html

- rocksDb (I/O latency better than nuDb) - validators keep a short ledger history
- compile in release mode (debug mode tends to run slower)
- specifically set-up to connect to more than 10 peers (15 -20)
- set log_level to "warning" (less writing to SSD)

Deploy an XRPL node on Ripple MainNet

```
sedan@ICBC-mainnet:/opt/local/bin$ sudo ./rippled --conf /opt/local/etc/rippled.cfg server_info
Loading: "/opt/local/etc/rippled.cfg"
2022-Apr-27 11:30:27.600879915 UTC HTTPClient:NFO Connecting to 127.0.0.1:5005

{
   "result" : {
      "info" : {
         "build_version" : "1.8.5",
         "complete_ledgers" : "71269417-71269698",
         "hostid" : "ICBC-mainnet",
         "initial_sync_duration_us" : "30458789",
         "io_latency_ms" : 1,
         "jq_trans_overflow" : "0",
         "last_close" : {
            "converge_time_s" : 3.002,
            "proposers" : 33
         "load_factor" : 1,
         "node_size" : "huge",
         "peer_disconnects" : "11",
         "peer_disconnects_resources" : "0",
         "peers" : 20,
         "pubkey_node" : "n9KQtSo7NaSUwdrbe9dTFCZcA4bhsPLKLLwv8cg3tPwXKTZrYGNC",
         "pubkey_validator" : "nHUGmPPcdvc3QUhzah3xAvDPoS3f91uSiHALaYweBQNdb8vHF5oo",
         "server_state" : "proposing",
         "server_state_duration_us" : "168462442",
```

Deploy an XRPL node on Ripple MainNet

```
        "validation_quorum" : 28,
        "validator_list" : {
           "count" : 1,
           "expiration" : "2022-Jun-01 00:00:00.000000000 UTC",
           "status" : "active"
```

Generate a wallet on MainNet (needs to be funded):

```
lucian@ICBC22-mainNet:~$ /opt/local/bin/rippled --conf /opt/local/etc/rippled.cfg wallet_propose
Loading: "/opt/local/etc/rippled.cfg"
2022-Apr-27 13:10:38.414837871 UTC HTTPClient:NFO Connecting to 127.0.0.1:5005

{
   "result" : {
      "account_id" : "rK5t5sXM1wYTH4dykgFe3fmvBnL1QDwoWb",
      "key_type" : "secp256k1",
      "master_key" : "MOVE PUB SAC SOAR GO WEEK VAN LAIR FOAL CHAD LAY END",
      "master_seed" : "ssBkjbFHemBf2b2GY4D5dNUQEPxE1",
      "master_seed_hex" : "233682B20D367543F2759170E7A8A6BF",
      "public_key" : "aB4cyd58QJW2QkCN3o3VXKTpjfw2vbH6CBcpZqFARAafoGSgvMQg",
      "public_key_hex" : "02575D8AA6F7861AAE68586656D177EFFB901FEF87FE25DDDF56D58665D1FB2F28",
      "status" : "success"
   }
}
```

- Example config files are at: https://github.com/treisto/ICBC22/tree/main/mainNetXRPL

Deploy a private XRPL testbed of 3 validators

- Install 3 nodes on 3 machines (from packages)
- Generate 3 validator keys and tokens, one for each validator.
- Update */opt/ripple/etc/validators.txt* with the 3 validator public keys generated, under the stanza [validators] Comment all other stanzas in *validators.txt*.
- Update */opt/ripple/etc/ripple.cfg* as below:
  - Under [ips_fixed] add the ips of the 3 nodes + peer_port 51235, e.g.:
    ```
    [ips_fixed]
    192.168.16.79 51235
    ```
  - Add [peer_private] and set it to 1
  - Set [ssl_verify] to 0
  - Add and set [node_size] according to your specs (medium/huge), avoid “large” as it’s “buggy”
  - Add [validator_token] stanza, and in each validator node, add (only) its own validator token
  - To have all features enabled, add the below in *rippled.cfg* under [features] stanza:
    PayChan, Escrow, CryptoConditions, fix1528, DepositPreauth, FeeEscalation, fix1373, MultiSign, TickSize, fix1623, fix1515, TrustSetAuth, fix1513, fix1512, fix1571, Flow, fix1201, fix1523, fix1543, SortedDirectories, EnforceInvariants, fix1368, DepositAuth, fix1578
- Start the 3 nodes one after each other with:
  ```
  sudo /opt/ripple/bin/rippled --conf /opt/ripple/etc/rippled.cfg
  ```
- Example config files are at: https://github.com/treisto/ICBC22/tree/main/privateXRPL

Deploy a private XRPL testbed of 3 validators

- *Server_info* reveals that the ledger sequence is just starting, the node is “proposing”, quorum=3, peers=2

```
sedan@riprel:~$ sudo /opt/ripple/bin/rippled --conf /opt/ripple/etc/rippled.cfg server_info
Loading: "/opt/ripple/etc/rippled.cfg"
2022-Apr-24 17:33:21.015051758 UTC HTTPClient:NFO Connecting to 127.0.0.1:5005

{
   "result" : {
      "info" : {
         "build_version" : "1.8.5",
         "complete_ledgers" : "14-57",
         "peers" : 2,
         "pubkey_node" : "n9L9EmEAiGKAVGdSed5oZMwWgLhgEWfJTpPbncCfVyXvZfqSYDTi",
         "pubkey_validator" : "nHUtkW3XmJsdHqhTtqrQeDDTMQDVMgjchhUKweLwXH5tyX7ZSdj7",
         "server_state" : "proposing",
         "validation_quorum" : 3,
         "validator_list" : {
            "count" : 1,
            "expiration" : "never",
            "status" : "active"
```

- Checking the Genesys account we notice that we have “all the money”:

```
sedan@riprel:~$ sudo /opt/ripple/bin/rippled --conf /opt/ripple/etc/rippled.cfg account_info rHb9CJAWyB4rj91VRWn96DkukG4bwdtyTh
Loading: "/opt/ripple/etc/rippled.cfg"
2022-Apr-24 17:34:31.261148788 UTC HTTPClient:NFO Connecting to 127.0.0.1:5005

{
   "result" : {
      "account_data" : {
         "Account" : "rHb9CJAWyB4rj91VRWn96DkukG4bwdtyTh",
         "Balance" : "100000000000000000",
```

Deploy and run Hooks smart contracts on XRPL test network

- Hooks are written in C; files ending in .c are hooks.
- File ending in .h are headers, in examples used by all hooks.
- Hooks compile to *.wasm* - hook binaries ready to be installed onto an XRPL account using the *SetHook* Tx.
- Install hooks and interact with XRPL: through *ripple-lib,* by means of *.js* files runnable with *node.js.*
- Build a hook by running *make* (in examples, from any hook's directory).

  An example *makefile* in each directory shows how to build a hook.
- Example hooks are located in directories: liteacc, firewall, carbon, accept, notary, peggy.
  To install a hook run `node <hook name>.js`
- Additional *.js* files can be run to interact with the Hook; new interactions can be written.

More details:

https://github.com/XRPL-Labs/xrpld-hooks/tree/hooks-ssvm/hook-api-examples

https://dev.to/wietse/hooked-1-smart-contracts-on-the-xrp-ledger-5eb6

https://dev.to/wietse/hooked-2-hooks-security-smart-contracts-on-the-xrp-ledger-83e

Deploy and run Hooks smart contracts on XRPL test network

- A modified XRPL “hooks-enabled” node is provided by Ripple in a Docker container here https://github.com/XRPL-Labs/xrpld-hooks

  Deploy container:

```
sudo docker run -d -i -t -p 6006:6006 --name xrpld-hooks xrpllabsofficial/xrpld-hooks-testnet
```

  - It will connect to a Hooks-enabled XRPL TestNet.
  - The Hooks feature is not yet deployed on the MainNet.
  - Inside the Docker:
    1. A rippled instance which will connect to the Hooks Public Testnet.
    2. A compiler toolchain for building hooks (**wasmcc**, wasm2wat,...)
    3. Example hooks and support scripts to install them on your account.

- After downloading and running the Docker container, Get Hooks TestNet XRP from

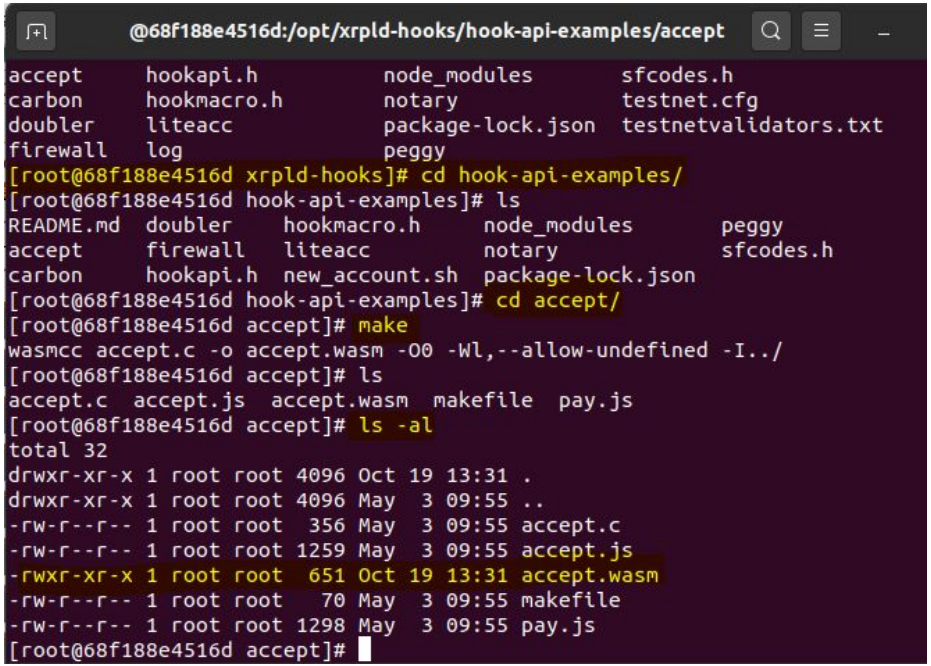
https://hooks-testnet.xrpl-labs.com/

Address_1
rHYdscAGcGZTbi2zk1krdmfPc5Rp8vDxT1
Secret1
snMEdrFZcRviyyPX5L6AwCyQCH3ms
Address_2
rGj9BL2CjBbrTFEomMNUPrTagFuK2NvZvE
Secret2
shubCNw5SNitNkS5Gb7rej37G98P2

Deploy and run Hooks smart contracts on XRPL test network

EXAMPLES: HOW TO BUILD A HOOK

- The XRPL kook server docker container must be running
- Inside the container go to *hook-api-examples* folder
- Enter the folder of one example hook and issue *"make"* command
- A file *"hook_name.wasm"* will be created after compilation



Deploy and run Hooks smart contracts on XRPL test network

Examples: “Accept” hook

Sender ACCOUNT_1 sends funds to receiver ACCOUNT_2, which deploys a HOOK to accept funds.
This is the simplest hook which just accepts any transaction coming through it.

- Tail log of the two Hook-XRPL accounts to see what happens. Use grep to filter the log:

```
docker exec -it xrpld-hooks tail -f log | grep -a rHYdscAGcGZTbi2zk1krdmfPc5Rp8vDxT1
docker exec -it xrpld-hooks tail -f log | grep -a rGj9BL2CjBbrTFEomMNUPrTagFuK2NvZvE
```

- To **accept** to receive on ACCOUNT_2 funds sent from ACCOUNT_1 (deploy the **accept** hook), run:

```
Usage: “node accept <receive account family seed>"
node accept shubCNw5SNitNkS5Gb7rej37G98P2
```

- Run *“node pay.js”* to send a payment to receiver’s ACCOUNT_2 from sender ACCOUNT_1

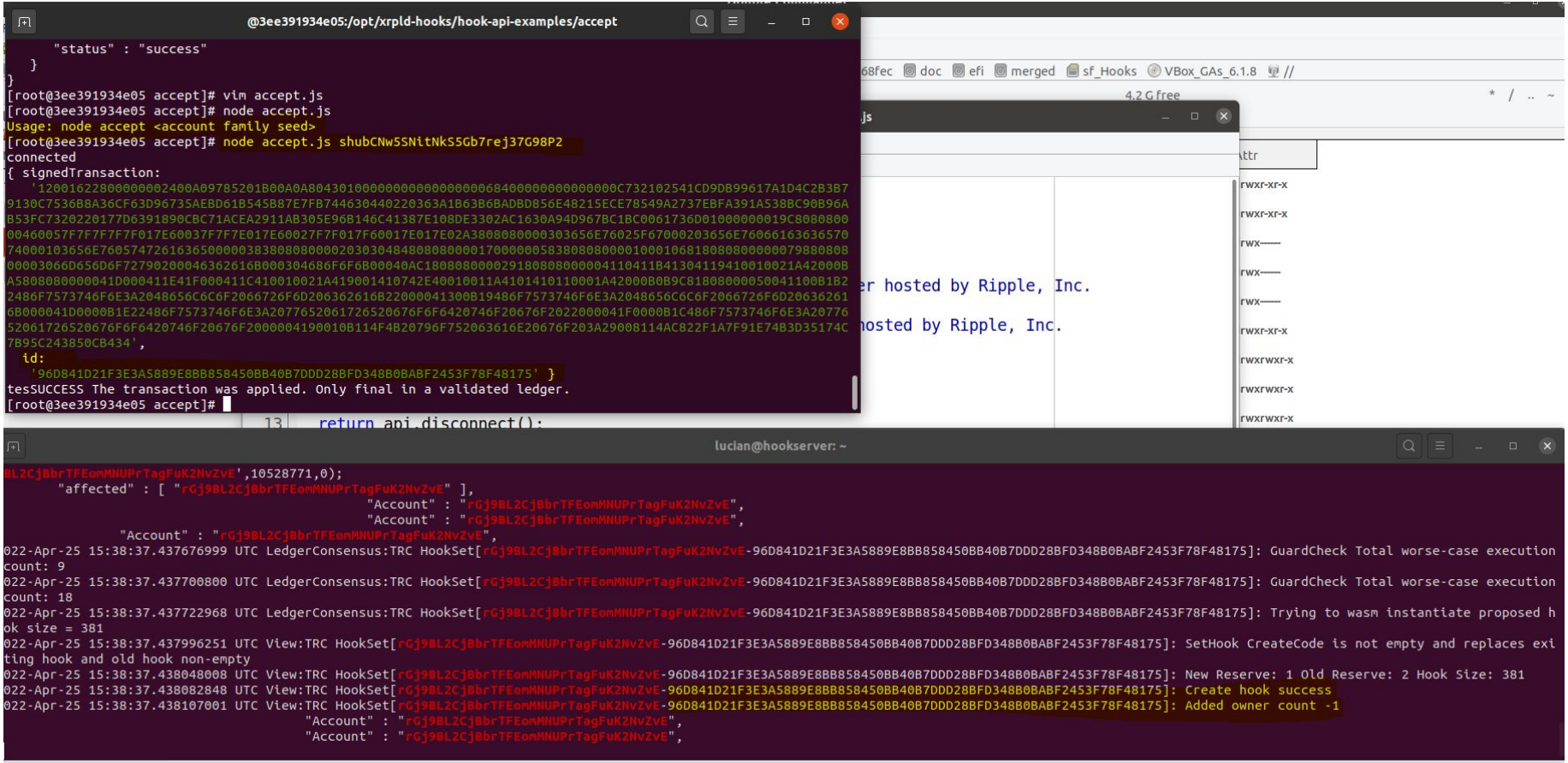
```
Usage: “node pay <source family seed> <amount xrp> <destination account>"
node pay snMEdrFZcRviyyPX5L6AwCyQCH3ms 1 rGj9BL2CjBbrTFEomMNUPrTagFuK2NvZvE
```

- Check accounts balance after transactions

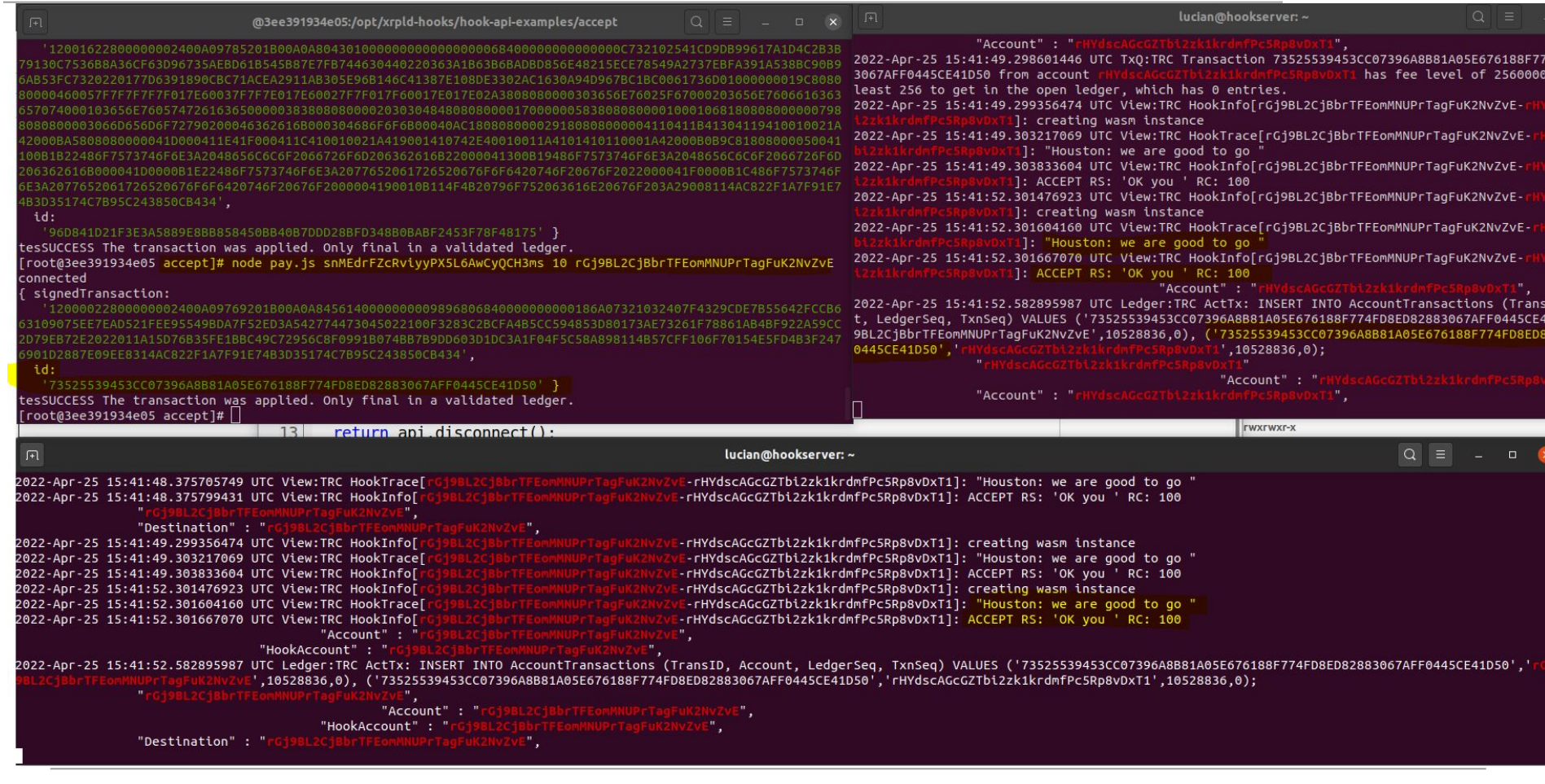
```
- /opt/xrpld-hooks/rippled account_info rHYdscAGcGZTbi2zk1krdmfPc5Rp8vDxT1
  ANSWER: "Balance" : "9998900000"
- /opt/xrpld-hooks/rippled account_info rGj9BL2CjBbrTFEomMNUPrTagFuK2NvZvE
  ANSWER: "Balance" : "10000992958"
```

We can also deploy a modified “Houston” accept hook which prints some text, to see better.

Deploy and run Hooks smart contracts on XRPL test network

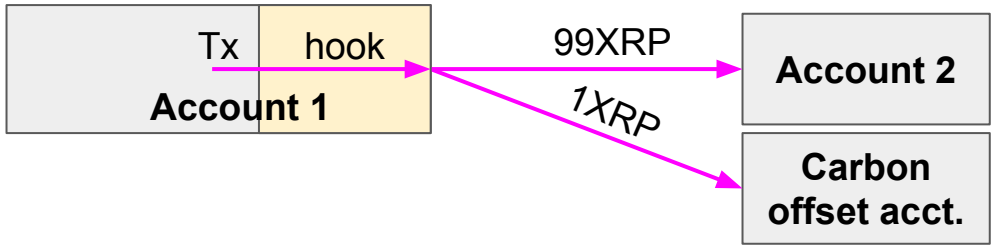


Deploy and run Hooks smart contracts on XRPL test network



Deploy and run Hooks smart contracts on XRPL test network

Examples: "Carbon offset" Outgoing Tx from sender account triggers an additional Tx from same account for 1% of original Tx value. This is sent to a carbon offset account controlled by an NGO which use it to plant trees.



The Carbon offset receive account is already set in the *carbon.c* hook. Can be changed.

```
rfCarbonVNTuXckX6x2qTMFmFSnm6dEWGX
```

- Tail the log of the three Hook-XRPL accounts to see what happens. Use grep to filter the log:

```
sudo docker exec -it xrpld-hooks tail -f log | grep -a rHYdscAGcGZTbi2zk1krdmfPc5Rp8vDxT1
sudo docker exec -it xrpld-hooks tail -f log | grep -a rGj9BL2CjBbrTFEomMNUPrTagFuK2NvZvE
sudo docker exec -it xrpld-hooks tail -f log | grep -a rfCarbonVNTuXckX6x2qTMFmFSnm6dEWGX
```

- Install hook on sender account:

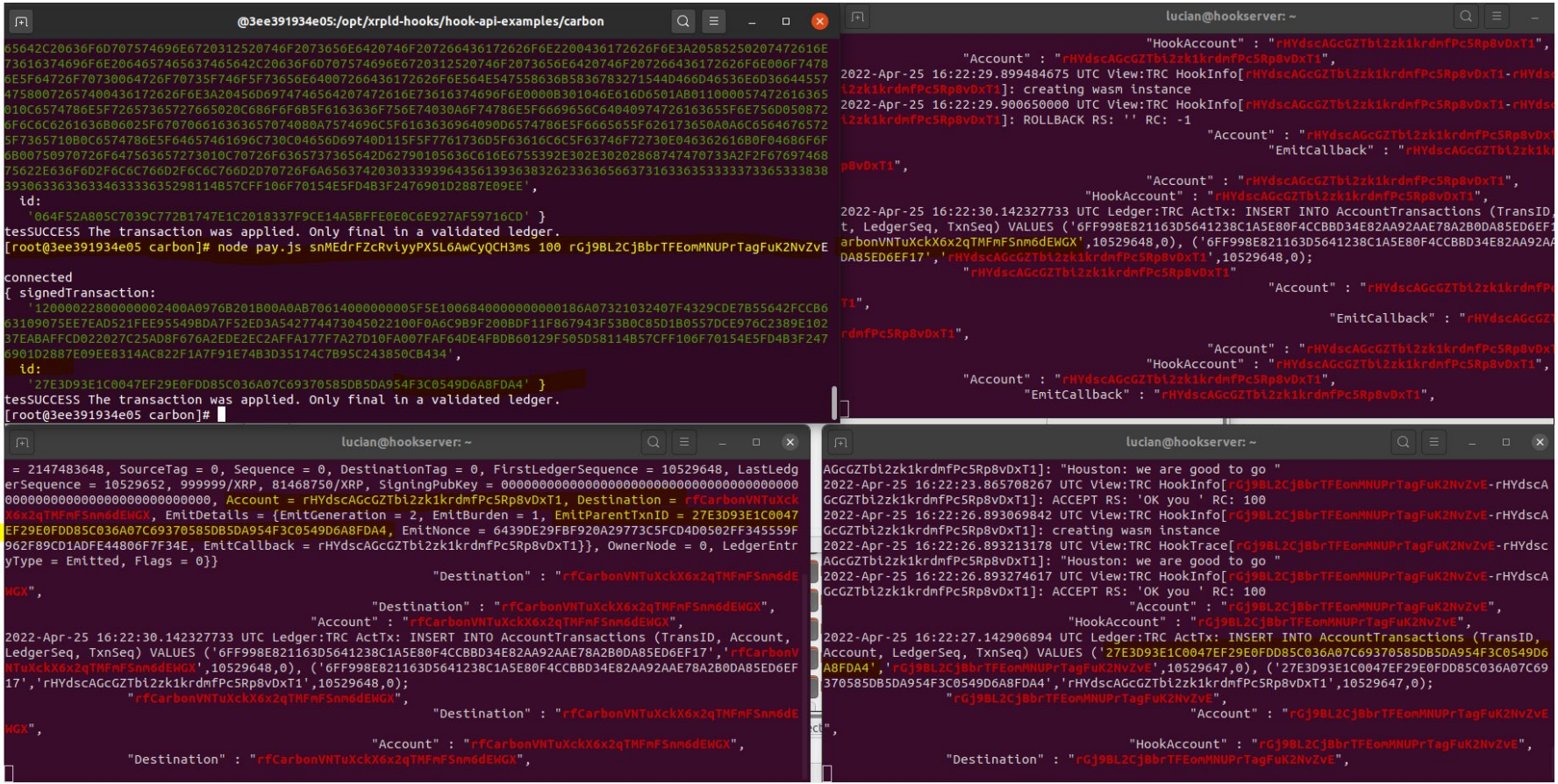
```
node carbon.js <sender account private key> snMEdrFZcRviyyPX5L6AwCyQCH3ms
```

- Send money from sender Account1 to receiver Account2, using:

```
node pay <sender private key> amount <receiver address>
node pay snMEdrFZcRviyyPX5L6AwCyQCH3ms 10 rGj9BL2CjBbrTFEomMNUPrTagFuK2NvZvE
```

The amount will be sent to receiver, with 1% sent to the Carbon offset account.

Deploy and run Hooks smart contracts on XRPL test network



REFERENCES

[1] https://xrpl-hooks.readme.io/

[2] *"Consensus Crash Testing: Exploring Ripple's Decentralization Degree in Adversarial Environments",* K. Christodoulou, E. Iosif, A. Inglezakis, Marinos Themistocleous https://www.mdpi.com/1999-5903/12/3/53/pdf

[3] https://interledger.org/interledger_**whitepaper**.pdf

[4] https://github.com/XRPL-Labs/xrpld-hooks

[5] https://paystring.org/**whitepaper**.pdf

[6] *"Log inference on the Ripple Protocol: testing the system with an empirical approach"* M. Roelvink, M. Olsthoorn, A. Panichela; Delft University of Technology, 2020-06-22

[7] https://dev.to/wietse/hooked-1-smart-contracts-on-the-xrp-ledger-5eb6

[8] https://dev.to/wietse/hooked-2-hooks-security-smart-contracts-on-the-xrp-ledger-83e

[9] https://github.com/treisto/ICBC22 - **examples**

[10] https://github.com/ripple/rippled/blob/develop/docs/consensus.md

[11] https://xrpl.org/non-fungible-tokens.html

[12] https://ripple.com/files/**ripple_consensus_whitepaper.**pdf

[13] https://xrpl.org/consensus.html

[14] "*Security Analysis of Ripple Consensus",* I. Amores-Sesar, C. Cachin, J. Mićić, https://arxiv.org/abs/2011.14816

QUESTIONS

XRPL Messaging - Performance XRPL_flood vs XRPL_squelch