Deep dive into Interledger: Understanding the Interledger ecosystem – Part 4 –

Lucian Trestioreanu, Cyril Cassagnes, and Radu State

Ripple UBRI @ Interdisciplinary Centre for Security, Reliability and Trust,
University of Luxembourg
29, Avenue JF Kennedy, 1855 Luxembourg, Luxembourg

Abstract. At the technical level, the goal of Interledger is to provide an architecture and a minimal set of protocols to enable interoperability for any value transfer system. The Interledger protocol is literally a protocol for interledger payments. To understand how is it possible to achieve this goal, several aspects of the technology require a deeper analysis. For this reason, in our journey to become knowledgeable and active contributor we decided to create our own test-bed on our premises. By doing so, we noticed that some aspects are well documented but we found that others might need more attention and clarification. Despite a large community effort, the task to keep information on a fast evolving software ecosystem is tedious and not always the priority for such a project. Therefore, the purpose of this document is to guide, through several hands-on activities, community members who want to engage at different levels. The document consolidates all the relevant information from generating a simple payment to ultimately create a test-bed with the Interledger protocol suite between Ripple and other distributed ledger technology.

Contents

List of Figures

List of Tables

1 What this document covers

In *Part 4*, we are going to conclude our series by discussing one last major component of the Interledger protocol suite, namely the *Bilateral Transfer Protocol (BTP)* which is a link protocol and a carrier for ILP, and a trading app named Switch. Finally, we are going to see the architecture of an entire, functional, private Ripple network which is currently the basis of our test-bed. We are also going to discuss some examples along the way. For easier orientation, we kept the general chapter structure unmodified.

2 Who this document is for

No prerequisites regarding the Interledger ecosystem are expected from the reader. However, developers, computer science students or people used to deal with computer programming challenges should be able to reproduce our setup without struggle.

3 The Interledger ecosystem

3.1 The Interledger protocol suite

3.1.1 The Bilateral Transfer Protocol

The Bilateral Transfer Protocol (BTP) emerged as a necessity, due to a combination of ILP goals (fast and cheap transactions) and the realities of some ledgers (expensive and/or slow settlements). With BTP, two parties can send funds directly to each other, up to a maximum amount they are willing to trust before settlement. BTP is used between connectors (Moneyd included) for transferring ILP packets and messages necessary to exchange payments, settlement, configuration and routing information.

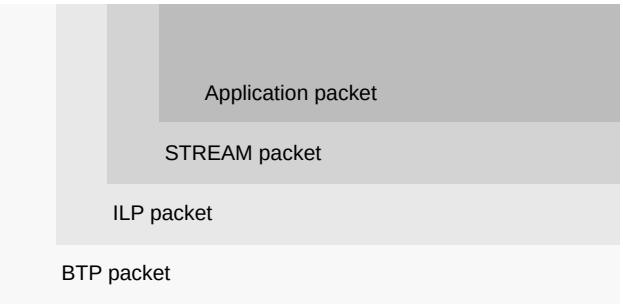

Fig. 1: Packet data structure. [1, 2]

As shown in Figure 1, BTP is a "carrier" for ILP packets and as such, for other protocols like STREAM for example. BTP establishes the "link" between connectors, on top of which the ILP packets are being sent. When setting-up the connector plugins, one also generally sets-up a BTP connection. The data is sent over web socket connections. One of the peers acts as a server while the other is connected as a client. It implements a Bilateral Ledger, where the two peers keep track of their (yet) un-settled accounts and balances. The Bilateral Ledger, a micro-ledger kept by the two peers in-between them, is not to be confused with the Underlying Ledger - the main ledger where all accounts and transactions are stored, e.g. the Ripple ledger. With regards

to Figure 5, it is to be noted that ILP can still work without BTP [3].

Example 1. We can now complete our diagrams presented in *Part 1 - Example 1*, *Part 2 - Example 1* and *Part 3 - Example 1* with the BTP protocol, which is illustrated in Figure 2.

In order to connect to Interledger, each Alice and Bob's ILP modules establish a BTP connection over wss with the parent connector. As long as they are connected to Interledger, this connection will be live. The ILP packets will travel over BTP. While opening the BTP connection, both of them also negotiate a unique paychan with their direct peer, the connector.

It is to be noted that while we made this choice for clarity, the order we presented SPSP, STREAM, ILP and BTP is not necessarily the real temporal order of events. This is illustrated in greater depth in *Part 1 - Example 2*, *Part 2 - Example 2* and *Part 3 - Example 2*.

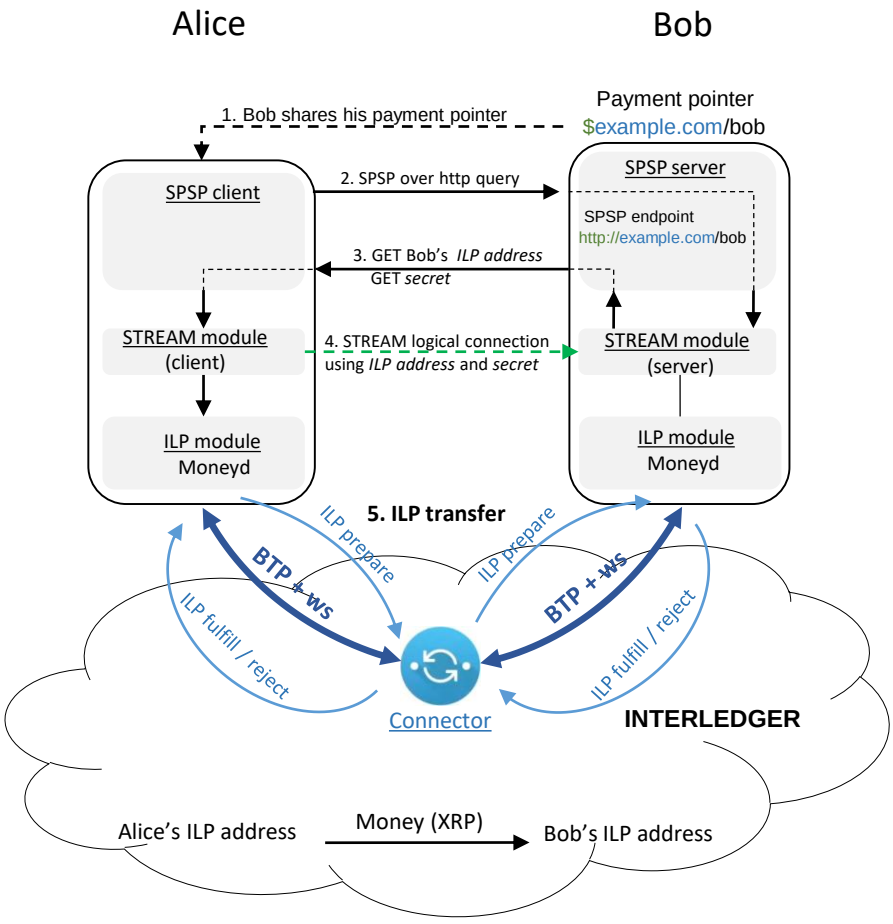

Fig. 2: Example 1: BTP. [1, 2]

The finite state machine of the BTP protocol is presented in Figure 3.

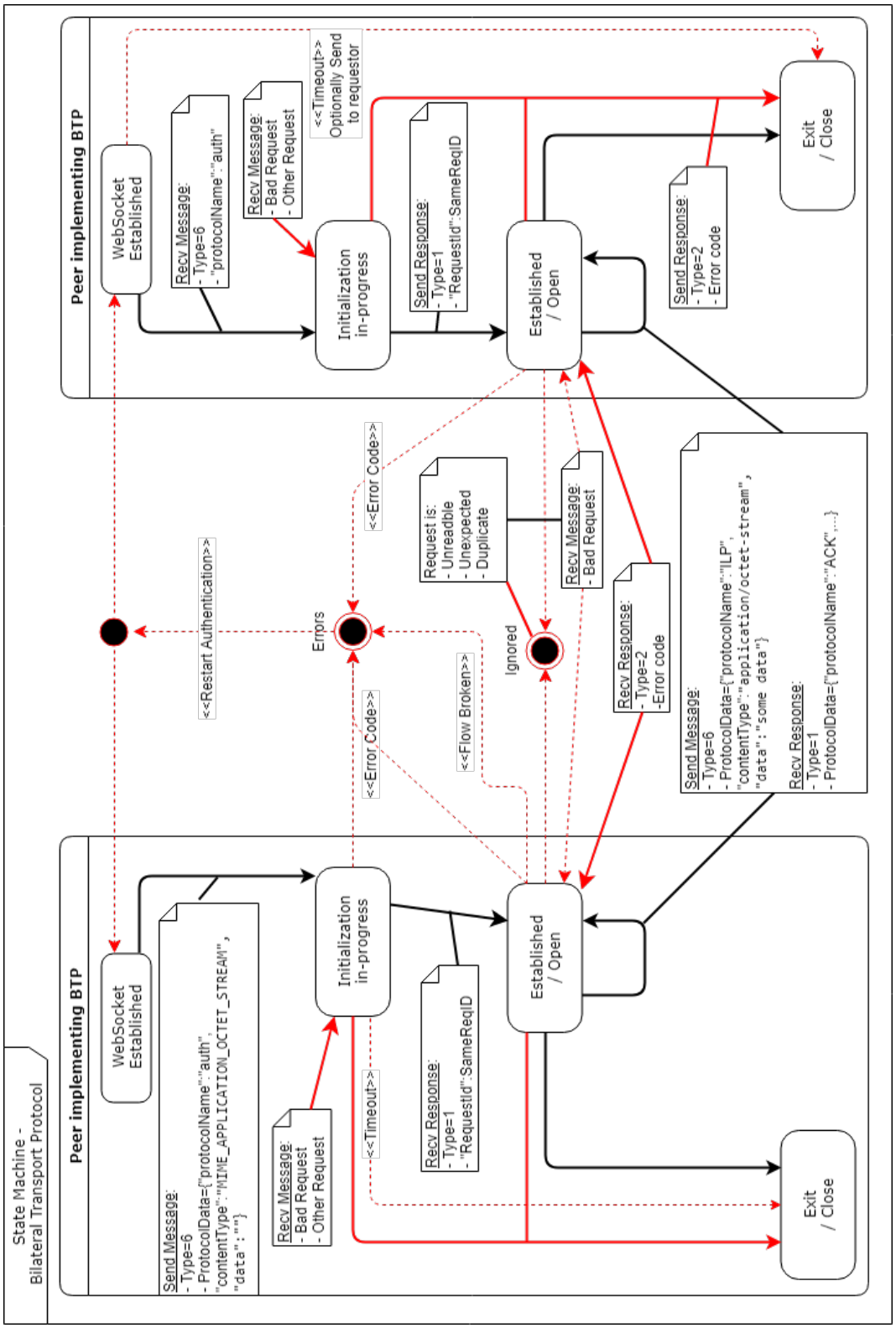

Fig. 3: BTP: the finite state machine diagram.

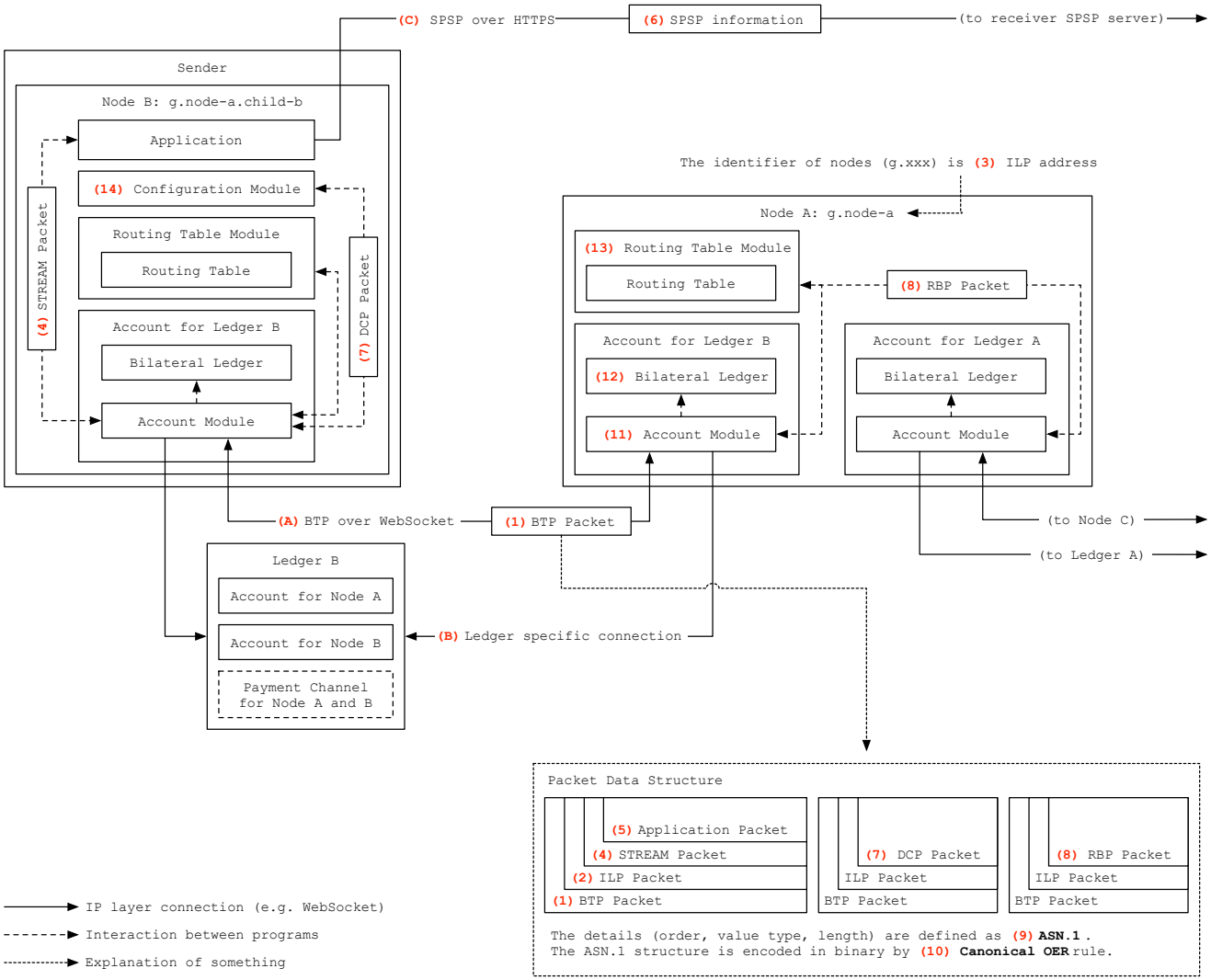

Fig. 4: Protocols and details. Advanced diagram. [1, 2]

The relationship between protocols, and especially the STREAM protocol, can be best understood by referring to Figure 4 and further, by reading the thorough explanations provided by [1, 2].

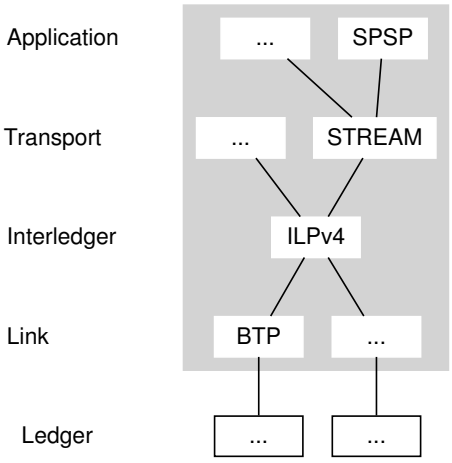

Fig. 5: The protocol suite. [4]

By concluding BTP, we have now seen all the major protocols of Interledger, namely BTP, ILP, STREAM and SPSP, a protocol suite also depicted in Figure 5.

Other protocols examples are the **Interledger Dynamic Configuration Protocol** (ILDCP), or the **Route Broadcasting Protocol** (RBP). DCP is built over ILP and used to exchange node information such as ILP address, while RBP is used to transfer routing information. Both use the data field in the ILP packets [1].

4 Customer apps for money transfer

4.0.1 @Kava-Labs: Switch API

Switch API[1] has been built mostly for cryptocurrencies trading like from XRP to ETH or Lightning. This means that the accounts involved in the currency swap belong to the same user. It 'streams money', meaning that for example, a 20 units transfer would be split into small chunks and each of these chunks would be separately sent on the paychan until the whole amount is sent [5, 6].

Switch API handles multiple uplinks, with dedicated plugins for each currency - XRP, ETH and Lightning. We investigated XRP and ETH. For communicating with the XRP connector we set up a dedicated XRP plugin[2], while for the Ethereum uplink we use a dedicated Ethereum plugin[3].

To handle the ETH settlement, Machinomy contracts have to be deployed on the ETH network, as explained in *Part 3 - Section 4.2*. We have tested a stream payment between XRP and ETH using Ganache[4] as ETH provider.

Some particular aspects of running Switch API - as of May 2019:

- The modules "ethers" and "ilp-plugin-ethereum" must be updated to the last version on the connector machine and Switch API machine.
- When setting up Switch API the credentials must be lowercase.
- After each run, it creates a config file in */home/user/.switch/config*. If run with the same credentials (for tests), this file must be manually deleted or would output the warning "can not create duplicate uplink".
- When using a private ETH network, Ganache included, the network ID and the Machinomy contract address should be set. We have used the Kovan network ID, 42, which we have set in Ganache, while in the following files, we have changed the address to the Machinomy contract address deployed on the ETH network (Ganache):
  - *'/home/user/node_modules/ilp-plugin-ethereum/build/utils/channel.js'* on the **reference node.js connector handling the ETH uplink** - the ILSP 2 connector in Figure 6, AND
  - in the same file **on the machine running the Switch API** app

[1] https://github.com/Kava-Labs/ilp-sdk/blob/master/README.md, accessed June 2019
[2] https://github.com/Kava-Labs/ilp-plugin-xrp-paychan, accessed June 2019
[3] https://github.com/interledgerjs/ilp-plugin-ethereum, accessed June 2019
[4] https://trufflframework.com/ganache, accessed June 2019

```
42: {
       unidirectional: {
           abi: Unidirectional_testnet_json_1.default,
           address: '0xa711d0a8b93faacd0f0f1897c11a1d7286d29720'
       }
   }
```

The Machinomy contract address is the "Unidirectional contract" address deployed by Machinomy.

- Settings regarding settlement, **on the machine running Switch API**, when running Switch API on private XRP and ETH networks:
  - File: *'switch-api/build/settlement/machinomy.js'*:

```
       remoteConnectors: {
           local: {
               'Kava Labs': (token) =>
                   `btp+ws://:${token}@192.168.1.131:7442` // Reference ETH
                   connector IP:port. ILSP 2 in Figure 26.
           },
           testnet: {
               'Kava Labs': (token) =>
                   `btp+ws://:${token}@192.168.1.131:7442` // Reference ETH
                   connector IP:port. ILSP 2 in Figure 26.
            },
           mainnet: {
                 'Kava Labs': (token) =>
                     `btp+ws://:${token}@192.168.1.131:7442`
                }
       }
```

  - File: *'switch-api/build/settlement/xrp-paychan.js'*:

```
const getXrpServerWebsocketUri = (ledgerEnv) => ledgerEnv === 'mainnet'
     ? 'ws://192.168.1.98:51233' // XRP validator IP
     : 'ws://192.168.1.98:51233'; // XRP validator IP
  . . . . . .
  remoteConnectors: {          //XRP parent connector
                  local: {
                         'Kava Labs': (token) =>
                             `btp+ws://:${token}@192.168.1.146:7444` //
                             XRP referenceConnector - ILSP 1 in Figure 26.
                  },
                  testnet: {
                         'Kava Labs': (token) =>
                             `btp+ws://:${token}@192.168.1.146:7444` //
                             XRP referenceConnector - ILSP 1 in Figure 26.
                  },
                  mainnet: {
```

```
                    'Kava Labs': (token) =>
                        `btp+ws://:${token}@192.168.1.146:7444` //
                        XRP referenceConnector - ILSP 1 in Figure 26.
                }
            }[ledgerEnv],
```

- Additional settings **on the machine running Switch API**. The "Ethers" module provides support for setting up different providers [5].
  - file: /home/user/node_modules/ethers/utils/networks.js:

```
    kovan: {
          chainId: 42,
          name: 'kovan',
          _defaultProvider:
              etcDefaultProvider('http://192.168.1.87:8545') // set ETH
              provider IP and port (Ganache)
    }
```

  - in file /home/user/node_modules/ethers/ethers.js, set network to kovan:

```
function getDefaultProvider(network) {
    console.log('ETHERS.js get default provider (network): network:',
        network);
    if (network == null) {
         network = 'kovan'; //set kovan
    }
```

  - in file /home/user/node_modules/ilp-plugin-ethereum/build/index.js, set provider to kovan:

```
class EthereumPlugin extends eventemitter2_1.EventEmitter2 {
    constructor({ role = 'client', ethereumPrivateKey, ethereumProvider =
        'kovan', getGasPrice, outgoingChanne
```

- Additional setting **on the machine running the connector providing the ETH link**:
  - in file /home/user/node_modules/ethers/utils/networks.js:

```
kovan: {
       chainId: 42,
       name: 'kovan', _defaultProvider:
           etcDefaultProvider('http://192.168.1.87:8545') //ETH provider
           IP:port (Ganache)
       }
```

- Example script which can be used for streaming XRP-ETH using Switch API [5]:

```
const { connect } = require('@kava-labs/switch-api')
```

[5] https://docs.ethers.io/ethers.js/html/api-providers.html, accessed June 2019

```
const BigNumber = require('bignumber.js')

async function run() {
  // Connect the API
  console.log('*** example-js ***: adding API')
  const api = await connect()

   //Add new uplink with an account
     console.log('**** example-js ****: addING uplink machinomy')
       const ethUplink = await api.add({
       settlerType: 'machinomy',
       privateKey:
           '6da09c0a78255932210aaf5b9f61046a00e9e3ab389c7357e388c4b35682342e'
       }) //switch Api wallet ETH
     console.log('*** example-js ***: addED uplink eth')

  // Add new uplink with an XRP testnet credential
     console.log('*** example-js ***: addING uplink XRP')
       const xrpUplink = await api.add({
       settlerType: 'xrp-paychan',
       secret: 'sasa3hrRUndoxAMoXEc3MMyZHNL3W' //switch API wallet XRP
       })
     console.log('*** example-js ***: addED uplink XRP')

  // Display the amount in client custody, in real-time
  xrpUplink.balance$.subscribe(amount => {
    console.log('XRP Interledger balance:', amount.toString())
  })
  ethUplink.balance$.subscribe(amount => {
    console.log('ETH Interledger balance:', amount.toString())
  })

  // Deposit 20 XRP into a payment channel
  console.log('EXAMPLE.js: start depositing 20XRP')
  await api.deposit({
    uplink: xrpUplink,
    amount: new BigNumber(20)
  })
  console.log('EXAMPLE.js: depositED 20xrp')


  // Deposit 0.05 ETH into a payment channel
  console.log('EXAMPLE.js: start depositing 0.05ETH')
  await api.deposit({
    uplink: ethUplink,
    amount: new BigNumber(0.05)
  })
  console.log('EXAMPLE.js: depositED 0.05ETH')

  // Stream 10 XRP to ETH, prefunding only $0.05 at a time
  // If the connector cheats or the exchange rate is too low, your funds are
      safe!
  await api.streamMoney({
    amount: new BigNumber(10),
```

```
    source: xrpUplink,
    dest: ethUplink
  })

  await api.disconnect()
}

run().catch(err => console.error(err))
```

This file can be placed in Switch API home directory and run with:
DEBUG= node –inspect ./file-name.js*

On top of this, Kava Labs has built an app for swapping the BTC, ETH and XRP cryptocurrencies just in a matter of seconds[6].

5 A private Ripple network example

We are concluding our series by providing an example of a private Ripple network, which is currently our testbed. The diagram of the network is illustrated in Figure 6.

It can be noticed that depending on the currency they operate, all *customer apps* and *connectors* are connected to at least on ledger in order to exchange ledger-related data. The connectors are peered over BTP using WS and a dedicated xrp plugin, which is different from the plugins used with MoneyD, Moneyd GUI, Switch API, etc. Switch API's uplinks can connect to the same connector or to different connectors, as it is the case here.

Apps that need real-time information like currency rates also need an internet connection: the connector trading ETH and Switch API.

Alice (XRP wallet), Bob (ETH wallet) and Charlie (XRP wallet), operating Moneyd instances and SPSP, can send each other value in the following combinations:

- Alice XRP <-> Charlie XRP
- Alice XRP -> Bob ETH and Bob ETH -> Alice XRP
- Charlie XRP -> Bob Eth and Bob ETH -> Charlie XRP

[6] https://github.com/Kava-Labs/switch, accessed June 2019

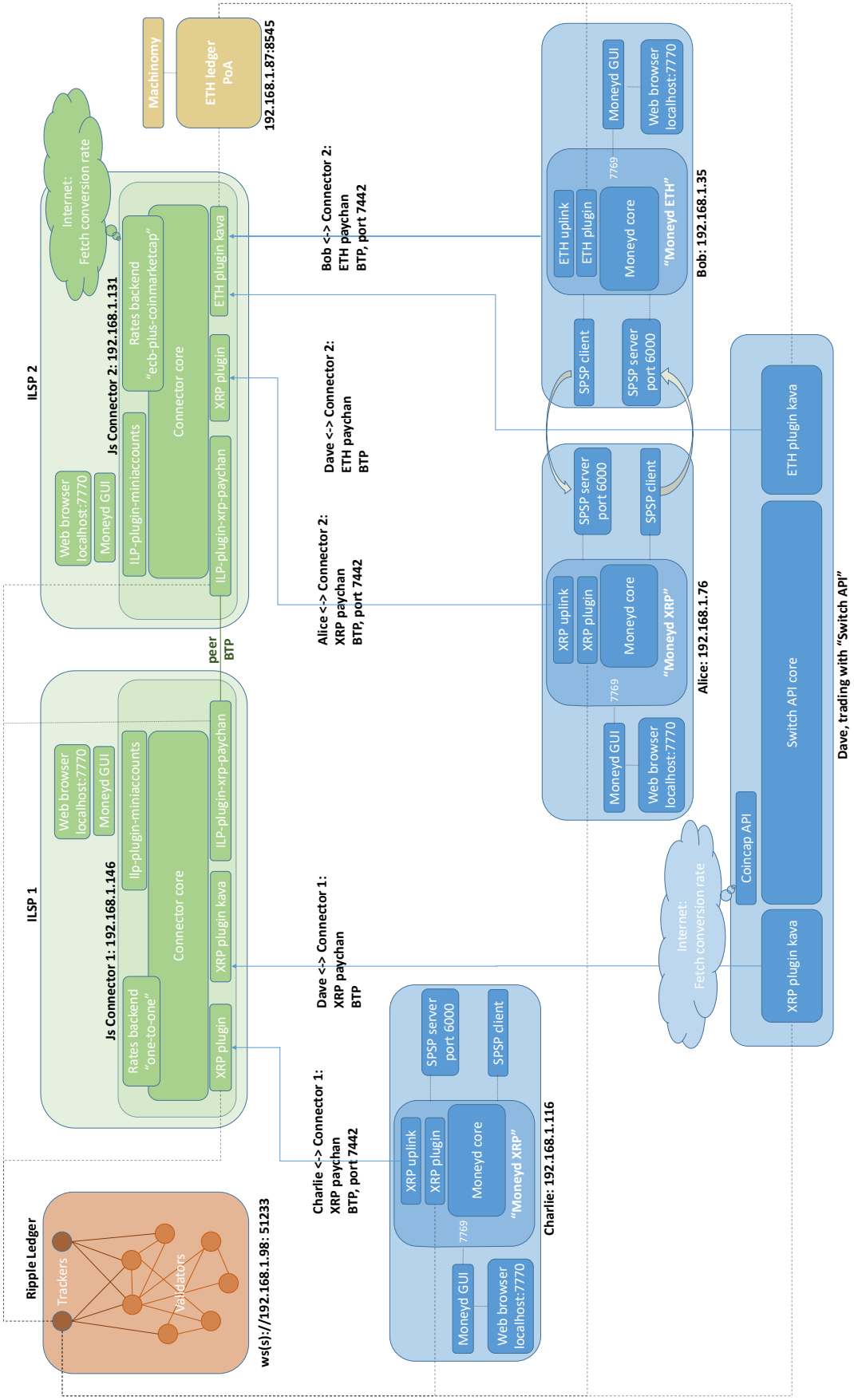

Fig. 6: Ecosystem overview. The machines involved are time-synchronized using time servers. The ETH gas price and the currency rates are fetched from online. To keep the diagram readable we didn't illustrate all plugin connections to the ledgers; each plugin provides for connection to the appropriate ledger using wss or ws.

6 Evaluation and discussion

In this paper, we have provided the details on how to set-up a private ILP network comprising of two ledgers - XRP and ETH, ILP service providers (connectors), and customer apps (Moneyd, SPSP, Switch API). The payments can be streamed from one ledger to the other with the Switch API app, making use of Machinomy smart contracts deployed on the ETH ledger. Time (the time on the machines must be synchronised), conversion rates and gas price are fetched from the internet.

In our opinion, at the present moment, as one moves from the core - the Rippled servers making-up the ledger, to the periphery - the customer apps, the support and availability of apps decreases. The most information to be found concerns the Rippled servers, while in regards to customer apps we have tried so far, at present only Moneyd-XRP seems fairly supported. Some of the plugins are undergoing changes (e.g. ETH plugin), and with the advent of connectors like Rafiki, they may be, at least partially, replaced with new approaches like the "settlement engine". Intuitively this is the way the ecosystem should be built and we are confident the future will bring many improvements.

7 Conclusions and future work

Sometimes abstract concepts are explained separately from the actual implementation, making it difficult to make the connections. This work fills a hole in the documentation regarding a lack of a comprehensive high level view of the ecosystem and how the different pieces are joined together.

We are currently studying the all-new @Coil/Rafiki which is still in beta and will soon provide the results.

Acknowledgements

This work was done in the framework of the Ripple UBRI initiative.

Acronyms

API Abstract Programming Interface. 7–9, 11, 13, 14

BTP Bilateral Transfer Protocol. 3, 4, 7

ILP Interledger Protocol. 3, 4, 7, 13

ILSP Interledger Service Provider. 7

SPSP Simple Payment Setup Protocol. 4, 7, 13

Glossary

Moneyd An ILP provider, allowing all applications on an end-user computer to use funds on the live ILP network. 3, 13

Switch API A SDK for cross-chain trading between BTC, ETH, DAI and XRP with Interledger Streaming. 7–9, 11, 13

XRP Ripple's digital payment asset which is used for Interledger payments. 7–9, 11, 13

References

[1] Ripple. *Relationship between Protocols*, [Online] Accessed: June 14, 2019. https://interledger.org/rfcs/0033-relationship-between-protocols/.

[2] Ripple. *Interledger Architecture*, [Online] Accessed: June 6, 2019. https://interledger.org/rfcs/0001-interledger-architecture/#protocol-layers.

[3] Ripple. *The Bilateral Transfer Protocol*, [Online] Accessed: June 11, 2019. https://interledger.org/rfcs/0023-bilateral-transfer-protocol/draft-2.html.

[4] Ripple. *Install Rippled*, [Online] Accessed: June 6, 2019. https://developers.ripple.com/install-rippled.html.

[5] Kincaid O'Neil Kava Labs. *Lightning fast, non-custodial trades - in 20 lines of code*, [Online] Accessed: June 13, 2019. https://medium.com/kava-labs/fast-non-custodial-trading-using-layer-2-ddeb2283f71b.

[6] Kevin Davis Kava Labs. *Kava Development Update #3*, [Online] Accessed: June 13, 2019. https://medium.com/kava-labs/kava-development-update-3-69e20f88b4c9.