Deep dive into Interledger: Understanding the Interledger ecosystem – Part 2 –

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 series of documents is to guide, through several hands-on activities, community members who want to engage at different levels. The series of documents consolidate 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

1 What this document covers

In *Part 2*, we are going to discuss two other components of the Interledger protocol suite and of the infrastructure suite, namely the *STREAM protocol* and, respectively, the *Connectors*. 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

The protocols suite comprises multiple protocols, of which the most important are BTP, ILP, Streaming Transport for the Realtime Exchange of Assets and Messages (STREAM) and SPSP. Below we are going to discuss the STREAM protocol.

3.1.1 The Streaming Transport for the Realtime Exchange of Assets and Messages

The **Streaming Transport for the Realtime Exchange of Assets and Messages** (STREAM) is a Transport Protocol working with ILPV4. Application level protocols like SPSP make use of the STREAM protocol to send money. STREAM splits payments into packets, sends them over ILP, and reassembles them automatically. It can be used to stream micro-payments or larger discrete payments and messages. It is a successor of the Pre-Shared Key V2 (PSK2) Transport Protocol and is inspired by the QUIC Internet Transport Protocol.

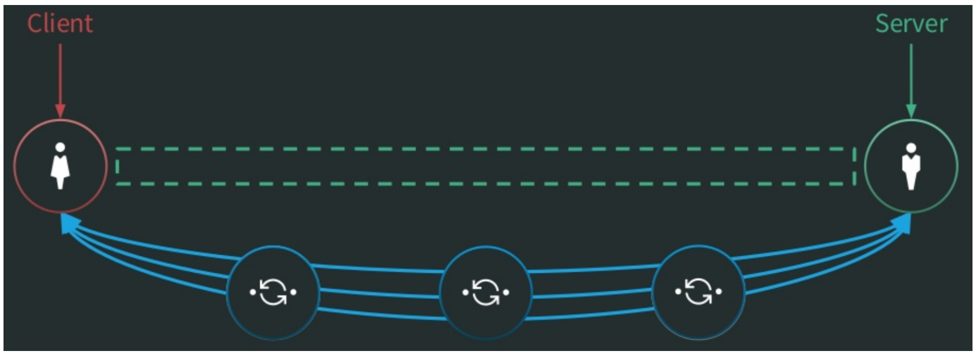


Fig. 1: STREAM is a logical, bidirectional channel over ILP. [1]

The specification of the STREAM protocol is available on the online repository of Interledger[1]. As illustrated with green in Figure 1, a STREAM connection establishes a two-ways, virtual channel of data and money between the payer and payee. Fields of STREAM packets are encoded with the Octect Encoding Rules and STREAM packets are encrypted using AES-256. Then, we will explain in the *Part 3* of this document series when a STREAM packet is sent in one of the following ILP Packet Type: *ILP Prepare* (type 12 ILP packet), *Fulfill* (type 13

[1] https://github.com/interledger/rfcs/blob/master/0029-stream/0029-stream.md, accessed June 2019

ILP packet), or *Reject* packets (type 14 ILP packet). The logical connection is used to send authenticated ILP packets between the "client" and "server" (the blue connections in Figure 1). Either the payer or the payee can be the server or the client. STREAM provides also authentication, flow control (i.e. ensure one party doesn't send more than the other can process), and congestion control (i.e. avoid flooding the network over its processing power).

STREAM servers are waiting for clients to connect over ILP. The servers connect to a specific plugin on the local machine and wait for the ILP packets. Usually, *ilp-plugin* is used to connect to Moneyd. The server generates a unique *ILP address* and *shared secret*, which will be used to encrypt data and generate fulfillments for ILP packets in relation to a specific client. The *request* for the address and secret, and the *response*, are not handled by STREAM, but for example by SPSP. After a client has the ILP address and secret (obtained with SPSP for example), it can connect to the STREAM server by using these credentials [2, 3].

Example 1. We now provide a more advanced explanation regarding the same situation presented in *Part 1, Example 1.* We will refer to Figure 2, and extend the explanation from *Part 1, Example 1.*

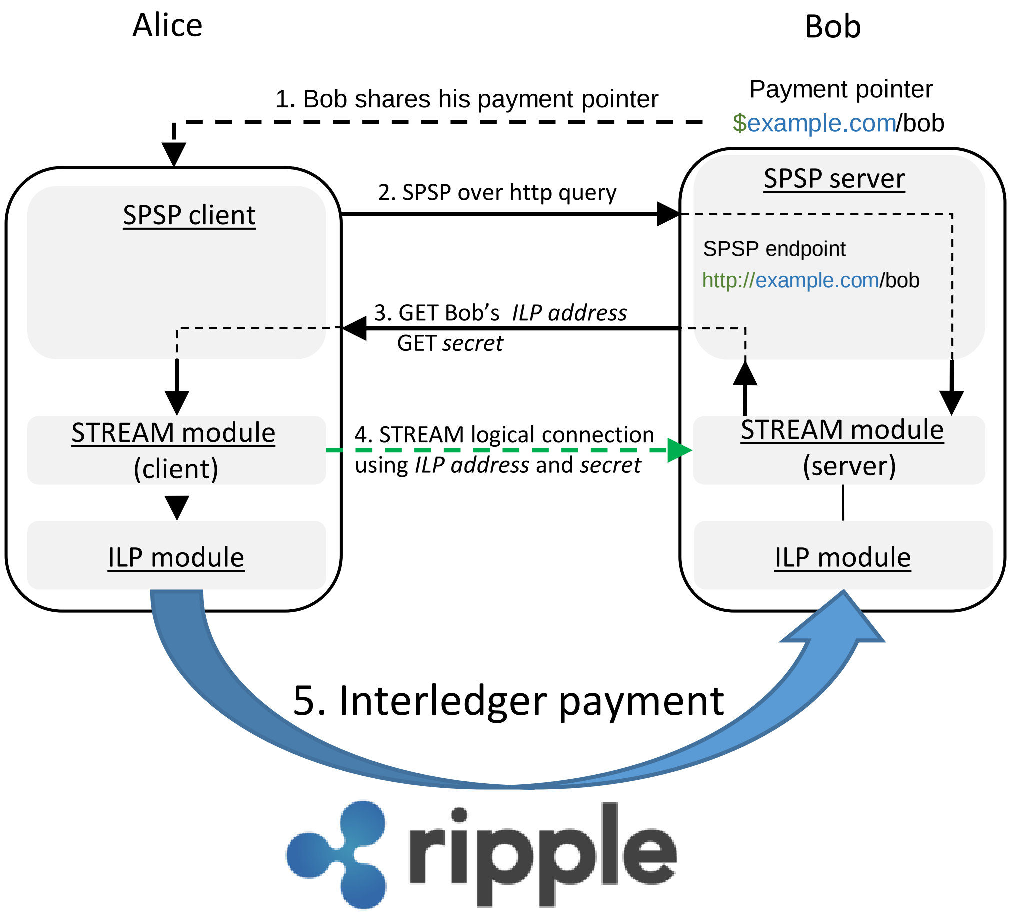


Fig. 2: Example 1: STREAM payment.

- Alice's SPSP client:
  - resolves the *payment pointer* *"$example.com/bob"* to https://example.com/bob

  - connects over HTTP to Bob's SPSP server at address https://example.com/bob (2)
  - queries the SPSP server for Bob's *ILP address* and a *unique secret* (2). The SPSP server forwards the request to the STREAM server module and fetches the answer

- Bob's SPSP server sends Bob's *ILP address* and the *secret* to Alice's SPSP client (3)
- Alice's SPSP client passes the credentials to the STREAM client module which initiates a logical STREAM connection over ILP, using the ILP modules (4)
- Bob receives his payment over the Interledger (5).

Further details on the STREAM protocol are illustrated in the finite state machine diagram of the STREAM protocol, presented in Figure 3. The black arrows point the normal flow rather than the red arrows point the error flow. Notes are stating key elements of a packet or of the connection at a specific moment in the interaction. Given the non-synchronicity in the communication, there is no notion of time. The diagram shows the different states of a stream session from its initialization to its termination. There is two boxes representing the two possible roles for a connector, which is either client (endpoint initiating a STREAM connection) or server (endpoint accepting incoming STREAM connections). Based on the textual specification, the diagram shows the messages required to trigger a transition and consequently move from one state to the other.

4 The connectors

Connectors are transaction 'intermediaries' lying in-between the payer and the payee, connecting them and facilitating the transaction. They are the 'market makers' or 'liquidity providers', and their role is especially evident when the sender's and receiver's wallets hold different currencies, as depicted in Figure 4.

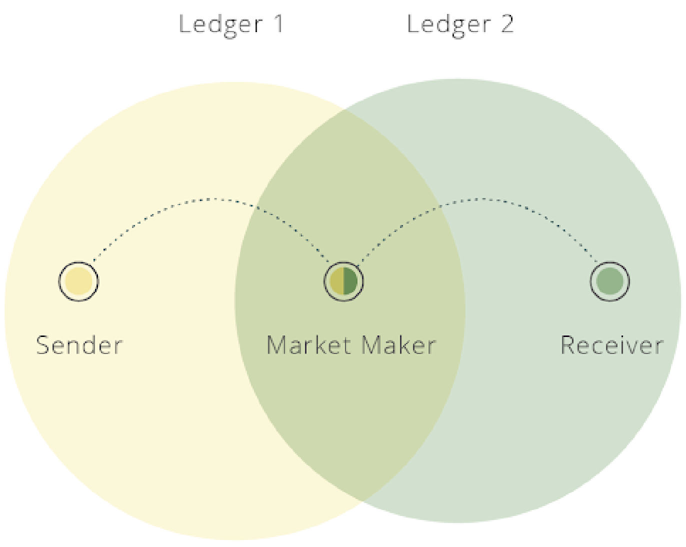
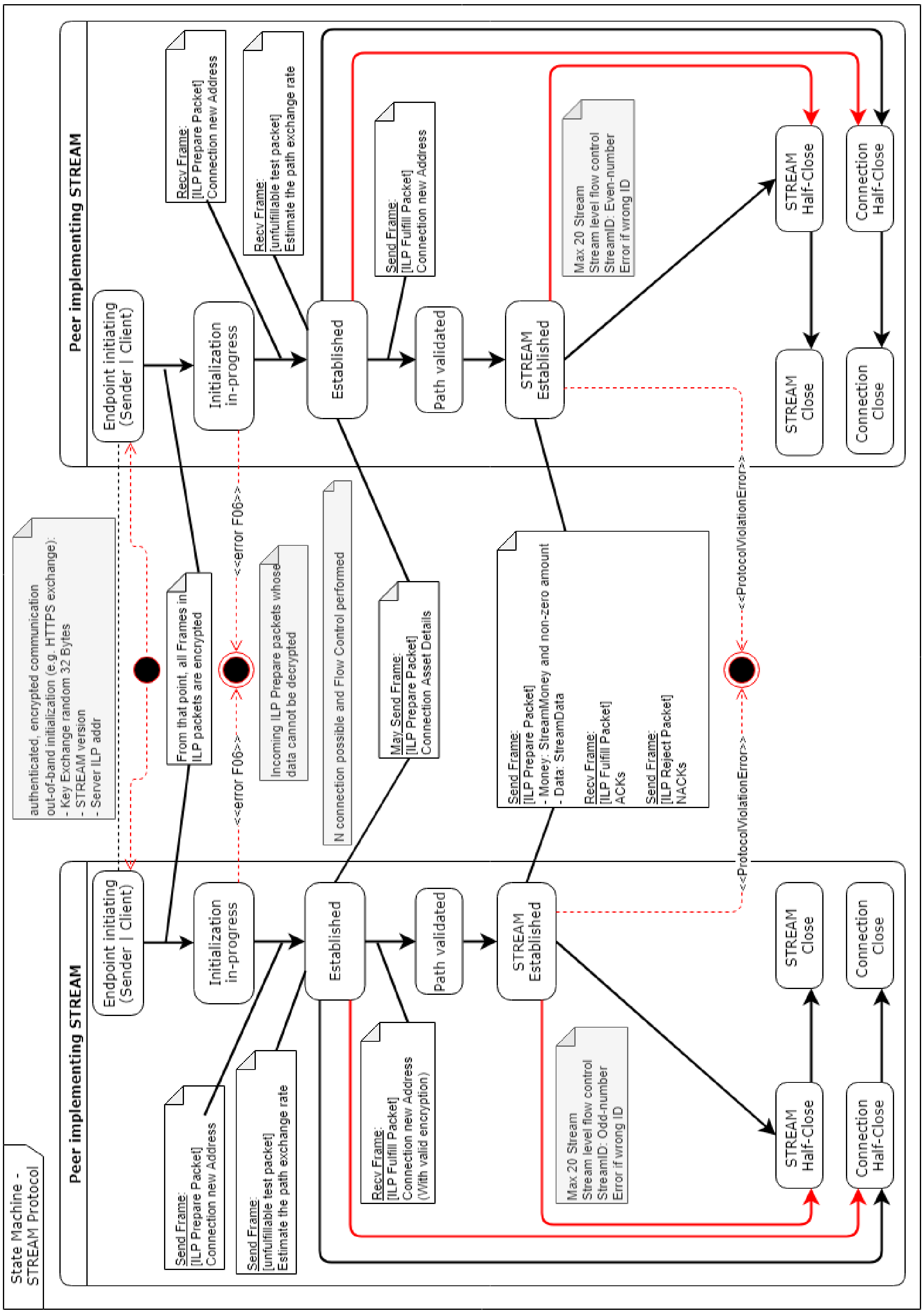


Fig. 4: A connector (money maker) holding two wallets on two different networks.

A connector would take the sender's money in the sender's currency and pay the receiver with the receiver's currency while charging a small fee for the service. In order to be able to



State Machine - STREAM Protocol
Peer implementing STREAM
Endpoint initiating (Sender | Client)
Initialization in-progress
Established
Path validated
STREAM Established
STREAM Half-Close
STREAM Close
Connection Half-Close
Connection Close
Recv Frame: [ILP Prepare Packet] Connection new Address
Recv Frame: [unfulfillable test packet] Estimate the path exchange rate
Send Frame: [ILP Fulfill Packet] Connection new Address
Max 20 Stream Stream level flow control StreamID: Even-number Error if wrong ID
authenticated, encrypted communication out-of-band initialization (e.g. HTTPS exchange): - Key Exchange random 32 Bytes - STREAM version - Server ILP addr
From that point, all Frames in ILP packets are encrypted
<<error F06>>
Incoming ILP Prepare packets whose data cannot be decrypted
N connection possible and Flow Control performed
May Send Frame: [ILP Prepare Packet] Connection Asset Details
Send Frame: [ILP Prepare Packet] - Money: StreamMoney and non-zero amount - Data: StreamData
Recv Frame: [ILP Fulfill Packet] ACKs
Send Frame: [ILP Reject Packet] NACKs
<<ProtocolViolationError>>
Send Frame: [ILP Prepare Packet] Connection new Address
Send Frame: [unfulfillable test packet] Estimate the path exchange rate
Recv Frame: [ILP Fulfill Packet] Connection new Address (With valid encryption)
Max 20 Stream Stream level flow control StreamID: Odd-number Error if wrong ID

Fig. 3: STREAM protocol: FSM diagram.

do this, a connector owns two wallets, one on each currency involved in the transaction. *'A connector is a host holding a balance on two or more ledgers. Connectors trade a debit against their balance on one ledger for a credit against their balance on another as a means of facilitating the payment between the two ledgers.'* [4]

While providing their services, the connectors act as an internet service provider would. In Interledger, the entities running the connectors are known as *Interledger Service Providers* or *ILSPs*. For example, the ILP reference connector written in js[2] can be run by an ILSP. An ILSP can run one or more connectors. Moneyd is a stripped version of a connector which is not sending or receiving routes, and is used as a *"home router"*, by end-users or customers in order to dial-up and connect to the ILSPs running a connector. As such, Moneyd[3] or Switch API[4] are examples of 'customer' apps connecting to their preferred ILSP and sending requests.

'Connectors implement the Interledger protocol to forward payments between ledgers and relay errors back along the path. Connectors implement (or include a module that implements) the ledger protocol of the ledgers on which they hold accounts. Connectors also implement the Connector to Connector Protocol (CCP) to coordinate routing and other Interledger control information.'[5]

Currently, the connectors rely on plugins to *settle* the transactions (new architectures are currently being considered or implemented, e.g. the Rafiki connector).

Making the analogy to a real-life example, swiping a credit card at a cashier's desk is considered a *payment*. The *settlement* occurs when the money is debited from the card holder's bank and credited to the merchant's bank. When you swipe the credit card and introduce your PIN you create and sign an irrevocable obligation for payment. On an Interledger paychan, this signed obligation for payment is known as a *"claim"*. A *redeemed claim* would translate to a bank transaction which has been "cleared" or "went through" (the money completely left the payer's bank account, and are visible and available in the payee's account; or analogously, the money completely left the sender's wallet/ledger and have shown up on the receiver's wallet, ledger and currency).

Concerning the plugins, they are installed on the same machine with the connector and configured according to purpose. This architecture is illustrated in Figure 5.

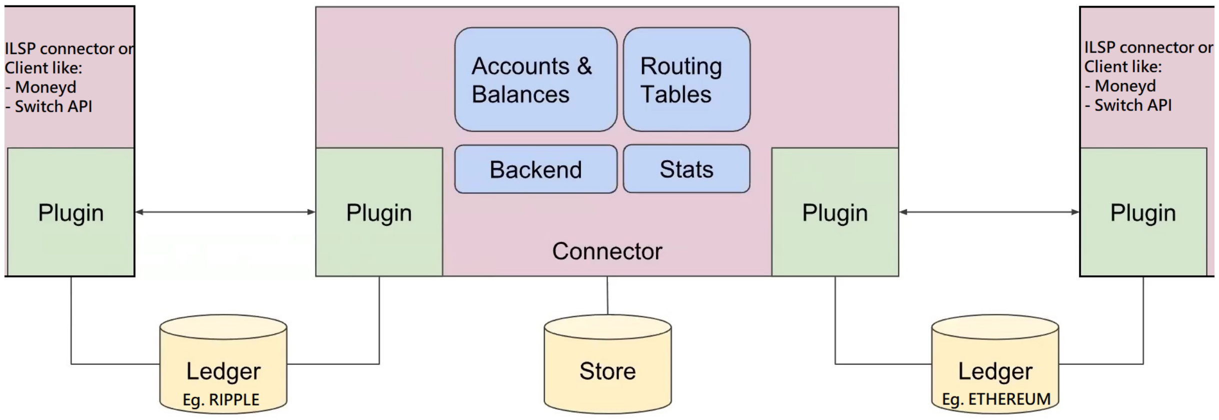


Fig. 5: Architecture overview. [6]

[2]https://github.com/interledgerjs/ilp-connector, accessed June 2019
[3]https://github.com/interledgerjs/moneyd, accessed June 2019
[4]https://github.com/Kava-Labs/ilp-sdk/blob/master/README.md, accessed June 2019

Some plugin examples would be:

- **ilp-plugin-xrp-paychan**[5] creates a direct peer relation with other connectors. It is an Unconditional Payment Channel plugin, where one has to trust his peer for the in-flight XRP amounts.
- **ilp-plugin-xrp-asym server**[6] enables the ILSP server to accept new client connections and creates an internal ILP account for each of them. It is the plugin appropriate for provider - customer relationships. This service will be exposed publicly and 'customers' will connect to it.
- **ilp-plugin-xrp-asym-client**[7] will be used by a 'customer' to connect to his provider's plugin, i.e the *ilp-plugin-xrp-asym-server* above.
- **moneyd's uplink-xrp plugin**[8] makes use of *ilp-plugin-xrp-asym-client.*
- **ilp-plugin-mini-accounts**[9] can be used to connect Moneyd-GUI to Moneyd or to the reference ILSP connector, for example.
- Kava Labs has been involved in the development of **@kava-labs/ilp-plugin-xrp-paychan**[10] and **ilp-plugin-ethereum**[11]. Both can be used in conjunction with Switch API. Ilp-plugin-ethereum settles Interledger payments with ether and is powered by Machinomy smart contracts for unidirectional payment channel.

Another way to illustrate a payment chain forwarding the transactions through the *connectors* with the use of SPSP, the ILP protocol, some of the plugins above and the validating servers forming the XRP ledger (to be discussed in *Part 3*) is provided in Figure 6. Some other ledger examples could be the ETH or Lightning server networks. It is worth being noted that PSK was upgraded to STREAM. As such, the sender can pay in XRP and the receiver can get his money in BTC.

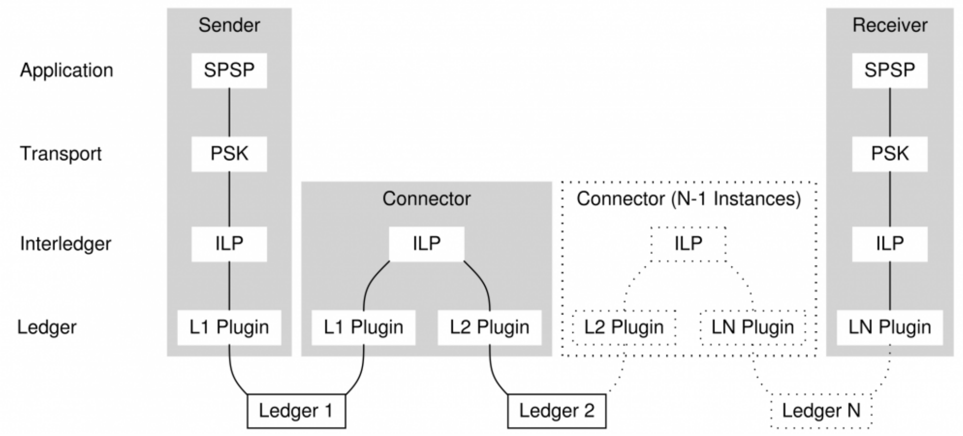


Fig. 6: The protocol stack in the payment chain.

Below we reproduce a nice explanation on payment channels that we found worth adding: *'In order to avoid having to go through the consensus process for each and every transaction,*

[5] https://github.com/interledgerjs/ilp-plugin-xrp-paychan, accessed June 2019
[6] https://github.com/interledgerjs/ilp-plugin-xrp-asym-server, accessed June 2019
[7] https://github.com/interledgerjs/ilp-plugin-xrp-asym-client/, accessed June 2019
[8] https://github.com/interledgerjs/moneyd-uplink-xrp, accessed June 2019
[9] https://github.com/interledgerjs/ilp-plugin-mini-accounts, accessed June 2019
[10] https://github.com/Kava-Labs/ilp-plugin-xrp-paychan, accessed June 2019
[11] https://github.com/interledgerjs/ilp-plugin-ethereum, accessed June 2019

only the summary of several transactions is validated on the blockchain. The intermediate transactions are conducted outside of the Ripple Ledger, off-chain. Ripple Labs has said that with Payment Channels (introduced in summer 2017), several thousand transactions per second can be processed. This number is approaching the transaction capacity of the VISA network. Because of the increased efficiency, Payment Channels are a feasible micro-payment alternative.' [7]

Also, payment channels are worth being used with expensive or slow ledgers. The two parties' transactions are being performed on the paychan under the limits established. Sometimes they offset each other. When the conditions for settlement are met, the settlement can occur. This lowers the cost and time involved by the overall process.

In regards to installation and set-up of connectors, a new guide[12,13] from Strata Labs has just been released, so you can try the bundle they propose if you want to hit the ground running.

The tutorial[14] provided by Adrian Hope-Bailie is a very good and thorough step-by-step guide. Inspired from his guide on installing the reference connector, a faster and easier minimal set-up and a few tips are provided below. This procedure will miss some features in the original guide. For the advanced set-up, the original post can be followed. For convenience, the Step # has been kept the same as in the original tutorial.

- Step 5: install node

```
    - curl -o-
      https://raw.githubusercontent.com/creationix/nvm/v0.34.0/install.sh | bash
    - Restart terminal
    - nvm install v10.15.3
```

- Step 7: install "redis"[15]

- Step 8: install pm2

- Step 10: get and fund an XRP Ledger address. Alternative to *'ripple-wallet-cli'*, it is also possible to generate a wallet directly, using the *'wallet_propose'* method:
  'user@saintmalo: /rippled/ccabuild$ ./rippled –conf /home/user/rippled/cfg/rippled-example.cfg wallet_propose'.

- Step 12: pick an ILP Address. The format should be *'g.somethingunique'*. For an independent private network, we also used the 'production' settings and 'g' as address prefix. Any of the others (private, local, ..) did not seem to work right.

- Step 13: create your config file using *'pm2 init'*. A file named *'ecosystem.config.js'* will be created, possibly in the folder *'home/user'*. Check it, update as needed, and move it to *'/home/user/ilp-connector/'*.

- Step 14: start it with:
  'cd /home/user/ilp-connector: $ pm2 start ecosystem.config.js'
  Use *'pm2 stop ecosystem.config.js'* to stop it, *'pm2 restart ecosystem.config.js –update-env'* for restart, and *'pm2 logs connector'* to see the logs. The log files are located in '/home/user/.pm2/logs/connector-out.log'.

[12]https://www.stratalabs.io/mainnet, accessed June 2019
[13]https://github.com/d1no007/easy-connector-bundle, accessed June 2019
[14]https://medium.com/interledger-blog/running-your-own-ilp-connector-c296a6dcf39a, accessed June 2019
[15]https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-redis-on-ubuntu-18-04, accessed June 2019

How to setup a connector:
First of all the following example is deployed in Ubuntu bionic (kernel 4.15 but not important).

sudo apt-cache madison npm - at the time of writing 3.5.2.
sudo apt-cache madison nodejs - at the time of writing version 8.10.
sudo apt-get install nodejs npm build-essential
npm config get prefix This will return the path for packages installed with -g. In our case, we will install the packages locally in the folder *jsilp*.
npm install memdown. We will use memdown for this example but for production, you should use another type of databases.
pm2 restart launch.config.js

The connector will be much easier to admin and understand using an interface, at this moment Moneyd-GUI. Installed on the same machine as the connector, it will provide UI access in a browser at http://127.0.0.1/7770. For the graphical interface, Moneyd-GUI loads some resources from online.

Below we provide an example configuration for the *Connector* trading only in XRP, which we named ILSP1. For our use-case we used ws, but in a real scenario wss is used.

```
'use strict'
const path = require('path')

const address = 'rMqUT7uGs6Sz1m9vFr7o85XJ3WDAvgzWmj' // <YOUR RIPPLE ADDRESS>
const secret = 'shjZQ2E3mYzxHf1VzYBJCQHqLvt7Y'  // <YOUR RIPPLE SECRET>

const peer1 = {
    relation: 'peer',              // establish a 'peer' relationship.
    plugin: 'ilp-plugin-xrp-paychan', //peer with another ILSP connector over XRP
    assetCode: 'XRP',
    assetScale: 9,   //"Interledger amounts are integers, but most currencies are
        typically represented as fractional units, e.g. cents. This property defines
        how many Interledger units make up one regular units. For dollars, this would
        usually be set to 9, so that Interledger amounts are expressed in nanodollars."
    balance: {
        maximum: '1000000000',
        settleThreshold: '-5000000000',
        settleTo: '0'
    },
    options: {
        listener: { //If you want your peer to connect to you as a ws client (which
            doesn't change the nature of the liquidity relationship) set the
            `listener` argument in the constructor.
        port: 10666, //this ws server listens for ws clients on port 10666
        secret: '2afe5e6cece84ed0027f9a2463edfa6358901bd6f1c9f3e1b0e43c13ff1ae2eb' //
            this is the token that your peer must authenticate with.
        },
        //server: 'btp+ws://:its_a_secret@192.168.1.146:10666', //this connector would
            be a ws client connecting to its peer ws server at port 10666
        //It should be possible to use it without credentials like this, also: server:
            'btp+ws://:@192.168.1.146:10666'
        // You may specify both the server and client options; in that case it is not
            deterministic which peer will end up as the ws client.
        rippledServer: 'ws://192.168.1.98:51233',         //the server that you submit
            XRP transactions to //MAINNET - wss://s2.ripple.com
        peerAddress: 'rLR52VSZG3wqSrkcpfkSnaKnYoYyPoJJgy', //<PEER RIPPLE ADDRESS>
        address,
        secret
     }
}

const ilspServer = {                                  //MoneyD XRP clients
    relation: 'child',                                //Moneyd apps will be 'children'
    plugin: 'ilp-plugin-xrp-asym-server',             //plugin that exposes the ILSP server
        to downstream clients
    assetCode: 'XRP',
    assetScale: 6,
    options: {
        port: 7443,                                   //port on which to listen to client
            apps
        xrpServer: 'ws://192.168.1.98:51233',         //MAINNET - wss://s2.ripple.com
        address,
        secret
    }
}

const SwitchAPIServer = {
      relation: 'child',                              //Switch API connects as a 'child'
      plugin: '@kava-labs/ilp-plugin-xrp-paychan',
      assetCode: 'XRP',
      assetScale: 6,
      options: {
          role: 'server',
          port: 7444,                                 //Switch API will connect on this port
          xrpSecret: 'shjZQ2E3mYzxHf1VzYBJCQHqLvt7Y', //this connector's secret
          xrpServer: 'ws://192.168.1.98:51233',
          // Very asymmetric... you fund a channel for $0.50 in XRP, we'll open one to
              you for $10!
          outgoingChannelAmount: '32658000',          // ~= 10$ in XRP drops
          minIncomingChannelAmount: '1632900',        // ~= 0.5$ in XRP drops
          // Use plugin maxPacketAmount (and not connector middleware) so F08s occur
              before T04s
          maxPacketAmount: '653200'                   // ~= 0.2$ in XRP drops
      }
}

const moneydGui = {                         //MoneyD GUI for this connector
  relation: 'child',
  plugin: 'ilp-plugin-mini-accounts',
  assetCode: 'XRP',
  assetScale: 6,
  options: {
    port: 7768                              //MoneyD GUI will connect on this port
  }
}

const connectorApp = {
    name: 'connector',
    env: {
        DEBUG: 'ilp*,connector*',
        CONNECTOR_ENV: 'production',
        CONNECTOR_ADMIN_API: true,
        CONNECTOR_ADMIN_API_PORT: 7769,         //this should not conflict with
            moneydGUI, set here on 7768
        CONNECTOR_ILP_ADDRESS: 'g.conn1',       //<YOUR ILP ADDRESS>
        CONNECTOR_BACKEND: 'one-to-one',
        CONNECTOR_SPREAD: '0',
        CONNECTOR_STORE: 'memdown',             //comment this if using the store below
        //CONNECTOR_STORE: 'ilp-store-redis',   //if using a store
        //CONNECTOR_STORE_CONFIG: JSON.stringify({
        //    prefix: 'connector',
        //    port: 6379
        //}),
        CONNECTOR_ACCOUNTS: JSON.stringify({
            conn2: peer1,                       //arbitrary names easy to remember
            ilsp_clients: ilspServer,
            moneyd_GUI: moneydGui,
            switchXRP: SwitchAPIServer
        })
    },
    script: path.resolve(__dirname, 'src/index.js')
}

module.exports = { apps: [ connectorApp ] }
```

Further, we reproduce an example configuration for the ILSP2 Connector which is peered with the ILSP1 Connector. This connector has two wallets, one in XRP and one in ETH, so it is able to provide cross payments between XRP and ETH. This use case fits the architecture illustrated in Figure 5. It is also equipped with Moneyd-GUI for easier administration through a Chrome browser (recommended), and can as well perform SPSP payments given SPSP is installed. SPSP was discussed in more detail in *Part 1, Section 4.0.1*.

```
'use strict'
const path = require('path')

const address = 'rLR52VSZG3wqSrkcpfkSnaKnYoYyPoJJgy' // <YOUR RIPPLE ADDRESS>
const secret = 'ssrnzXKsJKWDh9cFpmZSLWHN3D5HM'   // <YOUR RIPPLE SECRET>

//to get the gas price
const { convert, usd, gwei } = require('@kava-labs/crypto-rate-utils')
const axios = require('axios')

const getGasPrice = async () => {
  const { data } = await axios.get(
    'https://ethgasstation.info/json/ethgasAPI.json'
  )

  return convert(gwei(data.fast / 10), wei())
}
//
const peer1 = {
   relation: 'peer',
   plugin: 'ilp-plugin-xrp-paychan', //peer with other connector/node over XRP
   assetCode: 'XRP',
   assetScale: 9,
   balance: {
       maximum: '1000000000',
       settleThreshold: '-5000000000',
       settleTo: '0'
   },
   options: {
       //listener: {     //this connector would be a server listening on port 10666
       //port: 10666,
       //secret: '2afe5e6cece84ed0027f9a2463edfa6358901bd6f1c9f3e1b0e43c13ff1ae2ea'
           // this is the token that your peer must authenticate with.
       //},
       server:
           'btp+ws://yourcustomsequence:2afe5e6cece84ed0027f9a2463edfa6358901bd6f1c9f3e1b0e43c13ff1a
           //this connector is a ws client connecting to its ws server at port 10666
       rippledServer: 'ws://192.168.1.98:51233',   //PORT?   //wss://s2.ripple.com //
           ?Specify the server that you submit XRP transactions to?
       peerAddress: 'rMqUT7uGs6Sz1m9vFr7o85XJ3WDAvgzWmj', //<PEER RIPPLE ADDRESS>
       address,
       secret
    }
}

const peerETH = {
      relation: 'child',
      plugin: 'ilp-plugin-ethereum',
      assetCode: 'ETH',
      assetScale: 9,
      options: {
         role: 'server',
         port: 7442,
         ethereumPrivateKey:
             '0x43c50a578883922df30a33eb74418fb568c0081c40256e4675df02dcc28b6ef6',
             //this connector's ETH address; different from machinomy contract address
         ethereumProvider: 'kovan', //goes to ETH plugin as identifier
         getGasPrice: getGasPrice, //'20000000000',
         outgoingChannelAmount: '71440000',   //10 usd
         minIncomingChannelAmount: '3570000', // 0.5usd
         // In plugin (and not connector middleware) so F08s occur before T04s
         maxPacketAmount: '1430000' // 0.2USD
      }
}

const ilspServer = {
   relation: 'child',
   plugin: 'ilp-plugin-xrp-asym-server', // ILSP server for downstream clients
   assetCode: 'XRP',
   assetScale: 6,
   options: {
       port: 7443,                       //port on which to listen to client apps
       xrpServer: 'ws://192.168.1.98:51233', //MAINNET wss://s2.ripple.com
       address,
       secret
   }
}

const moneydGui = {
  relation: 'child',
  plugin: 'ilp-plugin-mini-accounts',
  assetCode: 'XRP',
  assetScale: 6,
  options: {
   port: 7768
  }
}

const connectorApp = {
   name: 'connector',
   env: {
       DEBUG: 'ilp*,connector*',
       CONNECTOR_ENV: 'production',
       CONNECTOR_ADMIN_API: true,
       CONNECTOR_ADMIN_API_PORT: 7769,
       CONNECTOR_ILP_ADDRESS: 'g.conn2', //<YOUR ILP ADDRESS>
       CONNECTOR_BACKEND: 'one-to-one',
       CONNECTOR_SPREAD: '0',
       CONNECTOR_STORE: 'memdown',
       //CONNECTOR_STORE: 'ilp-store-redis',
       //CONNECTOR_STORE_CONFIG: JSON.stringify({
       //    prefix: 'connector',
       //    port: 6379
       //}),
       CONNECTOR_ACCOUNTS: JSON.stringify({
           conn1: peer1,
           ilsp_clients: ilspServer,
           moneyd_GUI: moneydGui,
           peer_ETH: peerETH
       })
   },
   script: path.resolve(__dirname, 'src/index.js')
}

module.exports = { apps: [ connectorApp ] }
```

This connector supports SPSP client-server. This is explained in *Part 1, Section 4.0.1.*

Below, we provide a more advanced practical example involving two ledgers (XRP and Ethereum), a connector trading on the two ledgers, two customers - one with an XRP wallet and one with an Ethereum wallet, and Machinomy, which will be further detailed next, in *Part 3*.

Example 2. XRP-ETH ILP payment using Moneyd, SPSP and a connector

We will discuss the configuration presented in Figure 7. It is comprised of:

- The *XRP ledger*, or the XRP network, made up of servers running the "Rippled" software.

Mainly, the ledger holds the account balances for all users and validates the transactions performed in-between users.

- The *ETH ledger*, with a similar function.
- *Alice*, holding an account on the XRP ledger, operating Machine A, and running a user-level ILP XRP app, in this case Moneyd-XRP and SPSP.
- Bob, holding an account on the ETH ledger, operating Machine B, and running a user-level ILP Ethereum app, in this case Moneyd-ETH and SPSP.
- A *Connector*, having 2 accounts - one on each ledger. The connector will act as a facilitator - an intermediary between the two users. It will accept XRP from Alice and will forward the corresponding value, denominated in ETH, applying its exchange rate, to Bob.

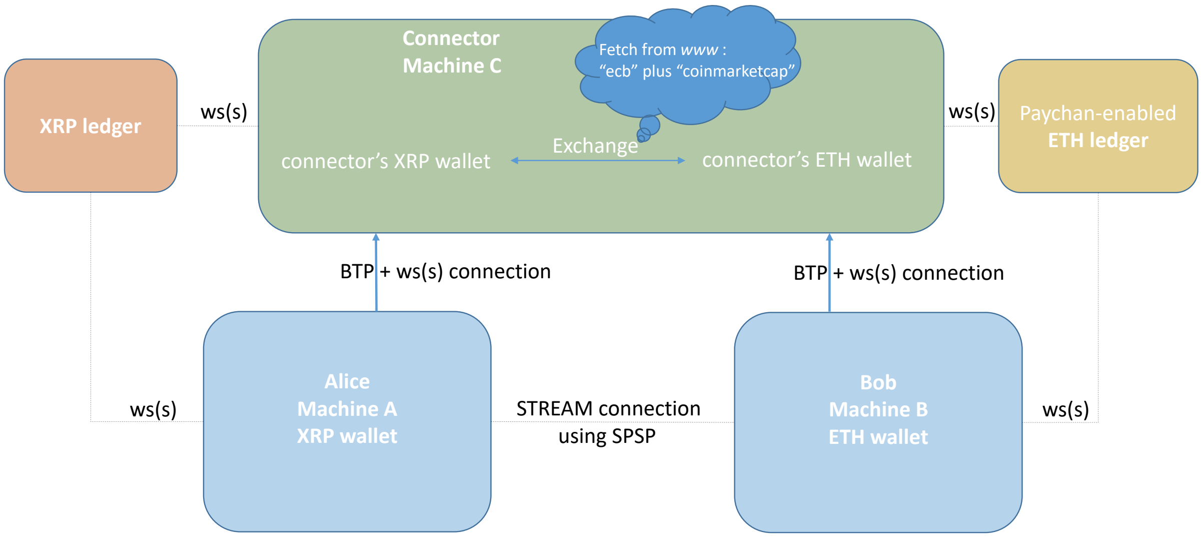


Fig. 7: Example 2: Interledger payment.

A more advanced representation of the same setup is provided in Figure 8 and explained below. In order to be able to settle the payments in ETH, *Machinomy* smart contract has to be deployed on the *ETH ledger*.

- *Alice* negotiates and opens a paychan denominated in XRP with the *connector*
- *Bob* negotiates and opens a paychan denominated in ETH with the *connector*
- Alice and Bob's machines comprise the following:
  - Moneyd-XRP (Alice) or ETH (Bob), comprising of:
    - Moneyd-core
    - XRP/ETH plugin, providing the settlement means
    - XRP/ETH uplink, providing the uplink to the connector
  - Moneyd-GUI, providing a visual admin interface
  - SPSP modules:
    - SPSP server: listens for connections from SPSP clients and receives payments

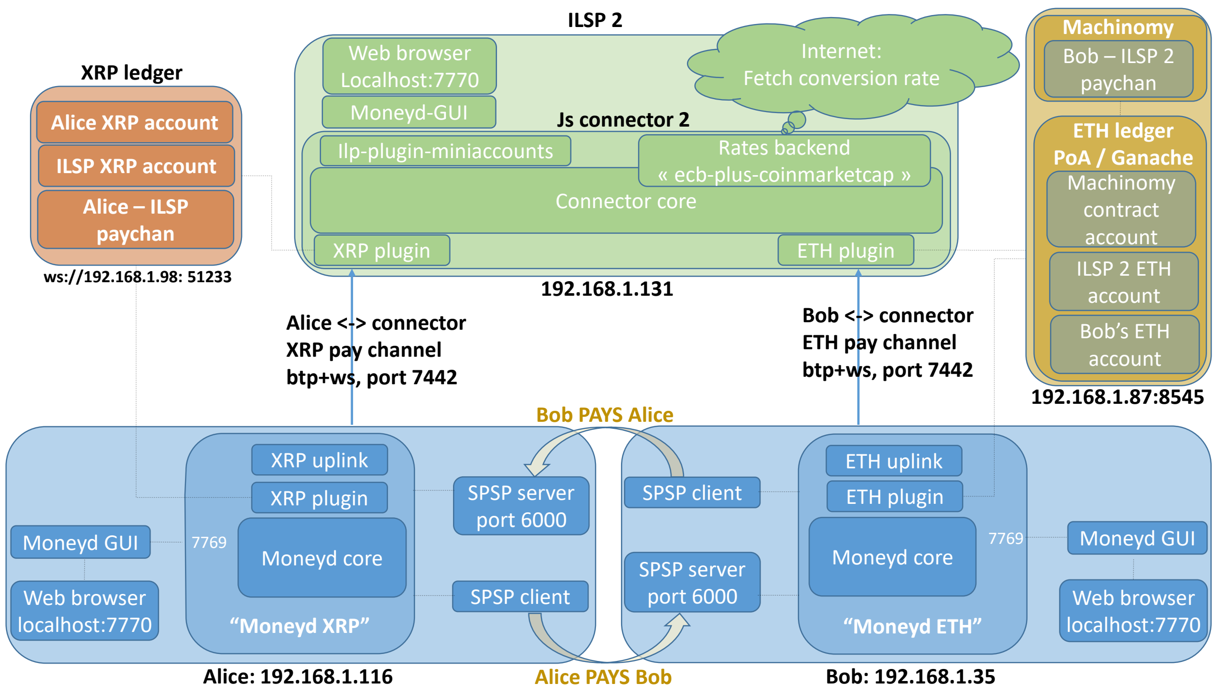


Fig. 8: Example 2: Interledger payment, advanced.

  * SPSP client: connects to SPSP servers and sends payments

- The connector, comprising of:
  - Connector core
  - Different plugins:
    * XRP plugin
    * ETH plugin
    * Possibly, "ilp-plugin-mini-accounts" - to make use of Moneyd-GUI as a visual admin interface
    * Possibly other plugins
  - The rates backend, which fetches the exchange rates from the internet. We will be using "ecb-plus-coinmarketcap". Other possibilities are: ecb, ecb-plus-xrp, , one-to-one. "One-to-one" applies an exchange rate of 1 to everything and is used by connectors operating in a single currency environment.

Into perspective, the situation can be represented as in Figure 9, where:

- *Sender* and *Receiver* are Alice and Bob
- *Node B, Node C* are Moneyd
- *Node A* is the connector
- *Application* is the SPSP client
- Ledger A and B are the XRP and ETH ledgers

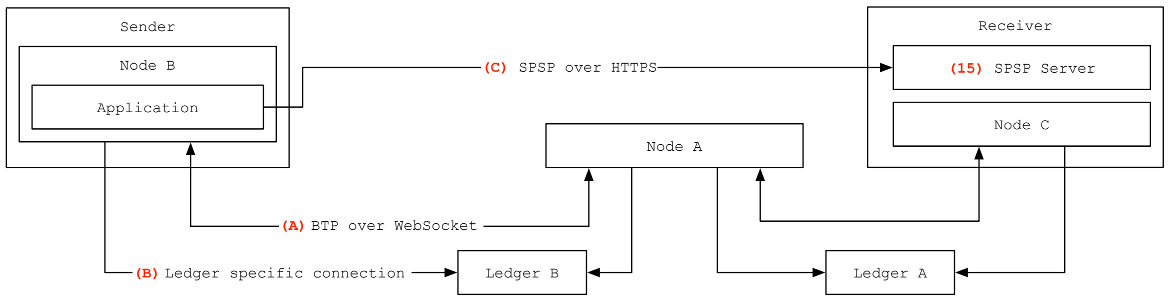


Fig. 9: Perspective: connections. [8]

- Moneyd connects to the connector over *BTP (A)* and also has a *ledger specific connection (B)* for settlement.

The protocol interactions are the same as in Example 1, and also Alice's machine is the same as in Example 1. The main differences are that on the ETH ledger, for settlement, Machinomy smart contract must be deployed, and that the paychan between Bob and the ILSP (Connector) is recorded there instead of the XRP ledger. As such, on the XRP ledger we find:

- Alice's XRP account
- ILSP (the connector) XRP account
- Alice - ILSP paychan,

while on the ETH side:

- Bob's ETH account
- ILSP (connector) ETH account
- Bob - ILSP paychan
- The Machinomy smart contract account, deployed in order to help manage the paychans and settlements on Ethereum.

An advanced diagram of connections and protocols interactions is provided by Ripple in [9]. The explanations are extensive and beyond the scope of this paper, but they can be retrieved by the interested readers by following the link to the reference.

For orientation, we provide as example a Moneyd-ETH configuration file:

```
  {
"version": 1,
"uplinks": {
  "eth": {
    "relation": "parent",
    "plugin": "/home/user/node_modules/ilp-plugin-ethereum/index.js",
    "assetCode": "ETH",
    "assetScale": 9,
    "sendRoutes": false,
    "receiveRoutes": false,
      "options": {
        "role": "client",
        "ethereumPrivateKey":
            "0x72f3b5a36a6719492913f6480b8b5036bf5cc5f312152351886c8e216fc63288",
        "ethereumProvider": "kovan",
        "outgoingChannelAmount": "50000000",
        "balance": {
          "maximum": "1000000",
          "settleTo": "0",
          "settleThreshold": "300000"
        },
        "server":
            "btp+ws://ASDG:294a4788a4b0a7a048332c7d2390e6ce06bcd63e59585493f50e8738650
        a948a@192.168.1.131:7442"
      }
    }
  }
}
```

Other connectors, functional but still in development, are the *Rafiki*[16] connector and the *Rust*[17] connector. A basis for a Java connector is also in the works, as *Quilt*[18].

Rafiki is a modular connector which is meant to improve on the ”reference” js connector in regards to practical aspects like cloud deployment, more manageable and ’hot’ changes of configuration, etc [10].

The Rust connector is meant to be a faster connector for high traffic. One important update is the new concept of ”Settlement engine” and the elimination of the plugins. Official information on the Settlement Engines’ architecture can be found in the new RFC on the Interledger website[19]. Also because of the modular architecture, at least in the case of Rust and at the present moment, in order to run a connector, the user needs to separately start 3 processes: the connector itself, the settlement engine, and in some cases the Redis database. Separate configuration need to be provided[20].

Next time we are going to discuss the *ILP protocol* which is the core of the Interledger, and about the *Ledgers*, a primary infrastructure component. We are also going to see some more advanced theoretical and practical examples.

Acknowledgements

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

[16]https://github.com/interledgerjs/rafiki, accessed June 2019
[17]https://github.com/emschwartz/interledger-rs/tree/master, accessed June 2019
[18]https://www.hyperledger.org/projects/quilt, accessed July 2019
[19]https://interledger.org/rfcs/0038-settlement-engines/, accessed January 2020
[20]https://github.com/interledger-rs/interledger-rs/tree/master/examples, accessed January 2020

Acronyms

API Abstract Programming Interface. 7, 8, 19

BTP Bilateral Transfer Protocol. 3, 17

FSM Finite State Machine. 2, 6

GUI Graphical User Interface. 8, 10, 12, 15, 16

ILP Interledger Protocol. 3–5, 7–9, 14, 15

ILSP Interledger Service Provider. 7, 8, 12, 17

SPSP Simple Payment Setup Protocol. 3–5, 8, 12, 14–16

STREAM Streaming Transport for the Realtime Exchange of Assets and Messages. 3

Glossary

Moneyd An ILP provider, allowing all applications on an end-user computer to use funds on the live ILP network. 4, 7, 8, 14–17

Switch API A SDK for cross-chain trading between BTC, ETH, DAI and XRP with Interledger Streaming. 7, 8

XRP Ripple's digital payment asset which is used for Interledger payments. 8, 9, 12, 14–17

References

[1] Evan Schwartz. *Protocol Stack Deep Dive - Boston Interledger Meetup*, [Online] Accessed: June 6, 2019. https://www.slideshare.net/Interledger/interledger-protocol-stack-deep-dive-boston-interledger-meetup.

[2] Evan Schwartz. *STREAMing Money and Data Over ILP*, [Online] Accessed: June 11, 2019. https://medium.com/interledger-blog/streaming-money-and-data-over-ilp-fabd76fc991e.

[3] Ripple. *STREAM: A Multiplexed Money and Data Transport for ILP*, [Online] Accessed: June 11, 2019. https://interledger.org/rfcs/0029-stream/.

[4] S. Thomas, E. Schwartz, and A. Hope-Bailie. *The Interledger Protocol*, July. 2016. [Online] Accessed: April 10, 2019. https://tools.ietf.org/html/draft-thomas-interledger-00.

[5] Ripple. *Interledger Protocol (ILP)*, [Online] Accessed: April 10, 2019. https://interledger.org/rfcs/0003-interledger-protocol/.

[6] Adrian Hope-Bailie. *Interledger Community Group Call*, Nov. 2018. [Online] Accessed: April 10, 2019. https://zoom.us/recording/play/rCj_0BZfNzUAHmg1lR52z0mGql55XICJ1CMJMl1sqDRNZmP3xFpyI52rgSOG6C8s?continueMode=true.

[7] HowToToken Team. *How Is Ripple Different From All Other Cryptocurrencies? An Ultimate Guide*, [Online] Accessed: April 10, 2019. https://howtotoken.com/explained/ripple-different-cryptocurrencies-ultimate-guide/#ripple-payment-channels.

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

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

[10] Adrian Hope-Bailie. *Introducing Rafiki*, [Online] Accessed: June 24, 2019. https://medium.com/interledger-blog/introducing-rafiki-e3de4710d3de.