Deep dive into Interledger: Understanding the Interledger ecosystem

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 between any value transfer systems. The Interledger protocol is a protocol for inter-blockchain payments which can also accommodate FIAT currencies. To understand how it is 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 contributors 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 up-to-date is tedious and not always the main priority for such a project. The purpose of this tutorial is to guide, through several examples and hands-on activities, community members who want to engage at different levels. The tutorial consolidates all the relevant information from generating a simple payment to ultimately creating 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

The scope of this document is to provide a walk-through the Interledger ecosystem starting with the Interledger Protocol (ILP). This ecosystem encompasses Ripple validating servers, ILP connectors, Moneyd, Switch API and more. The main goal of the document is to create a panoramic understanding of the ecosystem, how the main project systems and tools are interconnected together with practical insights into the how-tos of using the various associated systems and tools.

In our Ripple journey, we needed a comprehensive practical understanding of the general picture, but the required information was rather sparse, making it difficult to join the different bits and pieces together in order to form a complete test-bed (private network). Indeed, a positive indicator is the level of activity of the community to push the vision proposed by Ripple. Consequently, tutorials and various other resources can become outdated after a few weeks. In order to overcome this problem, this document consolidates the required documentation to setup and configure all the different parts. We will provide as much detail as possible in order to build and deploy a private Ripple network and so an Ethereum network.

The rest of this document is organized as follows. In Section 3, we review the state of the online documentation and the different community communication channels. In Sections 4, 5, 6 and 7 we remind all the important aspects of Ripple, which we subsequently evaluate in Section 8. Finally, we wrap-up the document in Section 9.

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 community

The community defines different communication channels[1] for different purposes in order to interact, get support and discuss the evolution of the protocol. The main channels are:

- The company website of Ripple[2] itself, established in 2012. The website provides all information related to business activities with RippleNet:
  - The Interledger Forum[3]
  - The Ripple Dev Blog[4]
  - The XRP Chat Forum[5]
  - The XRP Ledger Dev Portal[6] for the *rippled* servers (i.e. validators and trackers) and general concepts[7]. This is the development portal of the XRP Ledger, built on the Ripple open-source platform called *rippled*, which is the reference implementation. Thanks to their open source code-base they also have a Bug Bounty program[8].

[1] https://interledger.org/community.html, accessed June 2019
[2] https://ripple.com/, accessed June 2019
[3] https://forum.interledger.org/, accessed April 2020
[4] https://ripple.com/dev-blog/, accessed June 2019
[5] http://www.xrpchat.com/, accessed June 2019
[6] https://developers.ripple.com/, accessed June 2019
[7] https://developers.ripple.com/docs.html, accessed June 2019
[8] https://ripple.com/bug-bounty/, accessed June 2019

- The Interledger website[9]. This website aggregates pointers towards all resources[10]. Here is a sample of the resources accessible from the website:
  - The Interledger Forum[11]
  - The reference implementation[12] for ILP connector
  - The Rafiki[13] ILP connector, which generally has the same purpose as the reference implementation but a newer, different architecture
  - The Interledger[14] community calls.
- The Interledger Whitepaper publication - *A Protocol for Interledger Payments* - Stefan Thomas and Evan Schwartz – self-published online[15]
- Academic publication: *Interledger: creating a standard for payments* - Hope-Bailie, Adrian and Thomas, Stefan - Proceedings of the 25th International Conference Companion on World Wide Web, p. 281–282
- Not peer-reviewed research paper: *The Ripple protocol consensus Algorithm and Analysis of the XRP Ledger Consensus Protocol*[16]
- The Interledger Payments Community Group[17] whose scope is broader than Interledger Protocol (ILP) and aims to create an open, universal payment scheme built on Web standards.

Regarding practical experimentation with Ripple, Ethereum and Interledger, at the time of writing, the sources of documentation are sparse, disseminated over different channels like websites, forums, conference slides. Therefore, we collect and connect a relevant selection of these sources. A few tutorials on the Medium Interledger blog[18] concern setting up and starting a JavaScript connector, which is the reference implementation of the Interledger specifications and related protocols. Recently, for the same connector, Strata Labs has provided a quick-start bundle.

4 The Interledger ecosystem

RippleNet[19] aims to create a friction-less experience for sending and receiving money globally. The company targets institutions (e.g. Banks) of the financial sector. The key benefit of the solution is modernizing the traditional systems, which are many times expensive and take days to settle. However, one type of infrastructure won't fulfill all the requirements. Therefore, they strongly support the Interledger Protocol (ILP) initiative in order to realize the vision of an international friction-less payments routing system. In other words, a standard for bridging diverse financial systems.

[9] https://interledger.org/, accessed June 2019
[10] https://interledger.org/community.html, accessed June 2019
[11] https://forum.interledger.org, accessed June 2019
[12] https://github.com/interledgerjs, accessed June 2019
[13] https://github.com/interledgerjs/rafiki, accessed June 2019
[14] https://interledger.org/calls.html, accessed June 2019
[15] https://interledger.org/interledger.pdf, accessed June 2019
[16] https://arxiv.org/abs/1802.07242, accessed June 2019
[17] https://www.w3.org/community/interledger, accessed June 2019
[18] https://medium.com/interledger-blog, accessed June 2019
[19] https://ripple.com/ripplenet/, accessed July 2019

"Ripple has no direct competitors in crypto space, as it is fundamentally different from the most cryptocurrencies: it's more centralized, ***totally currency agnostic*** *and uses probabilistic voting (and not Proof of Work (PoW)) to confirm transactions."* [1]. This is possible thanks to the Interledger Protocol (ILP), and because the transaction verification process of RippleNet is not coupled to a mint process, which in the case of some others cryptocurrencies generates a direct income.

Nonetheless, besides lower cost and faster settlement than classic banking transactions, one of the most interesting aspects regarding the Interledger Protocol (ILP) is that it will seamlessly manage payments when the sender's currency is different from the receiver's currency, or when the sender's payments network is different from the receiver's payments network. For instance, a payment is issued on the fiat currency network using MasterCard, VISA, wire-transfer and the receiver receives it on an account (also known as a *Wallet*) created within a payment system using a crypto-currency. The Interledger Protocol (ILP) offers a means to bridge the crypto-currencies and fiat to enable interoperable and fast value exchange. Therefore, it is paramount to underline that Interledger Protocol (ILP) is not a blockchain, a token, nor a central service.

In ILP, money is actually not moved meaning that ILP doesn't decrease or increase of the total amount of electronic money in circulation. A connector swapping both currency has an account for each payment system it supports. Account balances are open and closed between parties involved in a particular transaction according to transaction instructions of each payment system involved. The parties are the sender, intermediaries (connectors) and the receiver. This statement is derived from the original Ripple explanation regarding connectors:

"Interledger connectors do not actually move the money, they rely on plugins for settlement. Plugins may settle by making a payment on an external payment system like (automated clearing house) ACH or they may use payments channels over a digital asset ledger like XRP Ledger or Bitcoin" [2].

In other words, when the receiver's currency is different from the sender's currency, also, no money is leaving the sender's network and no money enters the receiver's network. What happens is that at some point along the chain, some connector with accounts on both payment systems, keeps the sender's currency in one wallet (belonging to same ledger as the sender) and forwards the money towards the receiver, now denominated in the receiver's currency, from its other wallet holding that currency on the second ledger - the same ledger with the receiver's. The main difference with the classic system running today is that with ILP, the end-to-end payment becomes completely seamless thanks to the automation of many parts provided by the Interledger Protocol (ILP) Suite. For example, whereas in the classical banking systems there is no direct way for the sender to know when the recipient received the money, with ILP this information is available immediately.

In this section, we are going to quickly go through the payment infrastructure where all elements belong to one of the following three categories: distributed Ledger, user-level apps for payment, and connectors. Then, we will discuss the protocols stack and we will go through each of these components, providing details on each of them.

4.1 Main components of a unified payment infrastructure

From a high level point of view, the infrastructure comprises three main components:

Ledgers. In the context of Interledger, a *Ledger* is any accounting system that holds user accounts and balances. It can be linked to cryptocurrencies like Bitcoin, Ethereum, XRP or classic banks, PayPal and more. Here we are going to use the XRP ledger and the ETH ledger

which are distributed ledgers.

The XRP ledger comprises of a network of servers running the *rippled* software in "validator" or "tracking" mode, and the connectors can plug into it. The servers run the blockchain, record the user accounts and validate the transactions as they arise from connectors. Figure 1 illustrates the Ledger layer designed by the early adopters of the Interledger ecosystem. Each Connector is linked to at least two Ledgers (e.g. Ledger 1 = XRP Ledger and Ledger 2 = Ligthning) with dedicated plugins.

The user accounts are opened and stored on-ledger. In the case of the XRP ledger, this is how the related account info looks like at the time of opening the account:

```
{
   "result" : {
      "account_id" : "rMqUT7uGs6Sz1m9vFr7o85XJ3WDAvgzWmj",
      "key_type" : "secp256k1",
      "master_key" : "NIP SELF EDGY AQUA COME BAWD RING NEAL HINT HACK HEAT ADEN",
      "master_seed" : "shjZQ2E3mYzxHf1VzYBJCQHqLvt7Y",
      "master_seed_hex" : "925A2949624EF8CFA8A6C6A6E9211B2C",
      "public_key" : "aB4XmhndLn6C3sbsp5qK4Cy3GG9mU4KVe3wqWHatuudZX7CMhsvC",
      "public_key_hex" :
          "024B8511437A9A20E57C21A42A463DEEFE49D1DBE48ECA7FEEDE50048D02D92152",
      "status" : "success"
   }
}
```

- *"account_id"* is the public address of the account, used to identify the account and perform transactions.
- *"master_seed"* is the private key of the account. It is "the lock" of the account and should be kept safe and private.
- *"master_key"* can be used to regenerate the account details if needed.
- *"public_key"* can be used by third parties for verification.

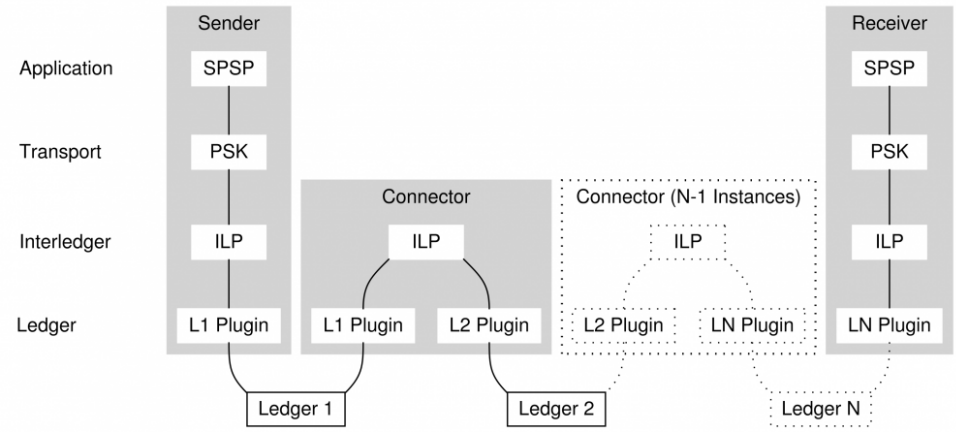

Fig. 1: Payment chain.

ILP connectors. The reference implementation of the connector specification is in JavaScript and so is the new Rafiki connector. However, other implementations are also written, in Java and Rust. Figure 1 shows that connectors are bridging all ledgers and their end-users, represented

there by the *"sender"* and *"receiver"*.

Connectors are run by different entities and offer payment inter-operability across the payment platform to the "customers" running a "customer app". The connectors are the "service providers", or "market makers", or "liquidity providers", because they provide end-users access to other payment networks, provide payment routing, exchange and liquidity. In order to do this, the Connectors make use, among others, of the Interledger Protocol (ILP), and ILP addresses.

ILP address. In order to be able to identify themselves and their users, and route the payments in a global network, the need for unique identification in the internet's IP style has arisen. As such, unique ILP addresses are being assigned to each Interledger node. In ILP a node can be a *sender*, a *connector* or a *receiver*. All nodes implement ILP.

On the production network, any ILP address starts with "g". Other prefixes can be "private" , "example" , "peer" , "self" .. We provide below some ILP address examples. These are thoroughly explained on the Interledger website [3].

g.scylla - *"scylla"* is a connector and *"g.scylla"* is its ILP address
g.acme.bob - is the address held by Bob on the *"acme"* connector
g.us-fed.ach.0.acmebank.swx0a0.acmecorp.sales.199. ipr.cdfa5e16-e759-4ba3-88f6-8b9dc83c1868.2 - is a complex, real-life address.

The connectors accept "dial-up connections" from the "customer apps", like an internet provider would provide internet services by accepting a dial-up connection from a home internet dial-up modem. As such, they function as Interledger Service Providers (ILSPs). *'An ILSP,* [depicted in Figure 27], *is a connector that accepts unsolicited incoming peering requests. It will have multiple child nodes connect to it and will assign them each an address and then route ILP packets back and forth for them onto the network. (It's modelled on the idea of an ISP that provides customers access to the Internet)'* [4]. When opening payment channels and routing payments for their customers, depending on the number of customers, a connector could bound significant amounts of money. They assume some risks and expect to make small profits in exchange for providing the service.

Customer apps are the third component of the infrastructure, and they provide end-users access to the Interledger payment system. Examples of customer apps are Moneyd or Switch API. Figure 1 shows the different layers for end-users, i.e. participants willing to generate a payment.

4.2 The Money transfer system.

The system of money transfer over ILP involves recording and manipulating money at different levels:

- *The Bilateral Balance* kept between two peers
- *Settlement* on the *payment channel (paychan)*, which involves signing claims that are recorded on the payment channel opened between the two peers. The claims concern the transactions between the two peers resulting from adjusting the Bilateral Balance above.
- *On-Ledger recorded transactions*, resulted, for example, from redeeming the previous claims submitted on the payment channel. On-Ledger transactions can also be submitted, for example, by using the Ripple API.

4.2.1 The Bilateral Balance.

Two directly connected peers hold a balance between them, for the ILP packets of value they exchange. The balance is maintained real-time and it can increase or decrease, according to the value they exchange.

The balance parameters are:

```
balance: {
    maximum: '20000000000', //maximum amount that the other party can owe
    settleThreshold: '-50000000', //the balance value that triggers an automatic
        settlement
    settleTo: '0' //balance value after settlement
}
```

Concerning Figure 2, if for example at one given moment, according to the total balance, Alice owes Bob 20 XRP, Alice's balance with Bob will be -20 XRP, while Bob's balance with Alice will be 20. Their balances will offset each other. This balance can be seen in the Moneyd-GUI (Graphical User Interface). Concerning Figure 2, this is the "ILP Balance", and it is a not-yet-settled balance.

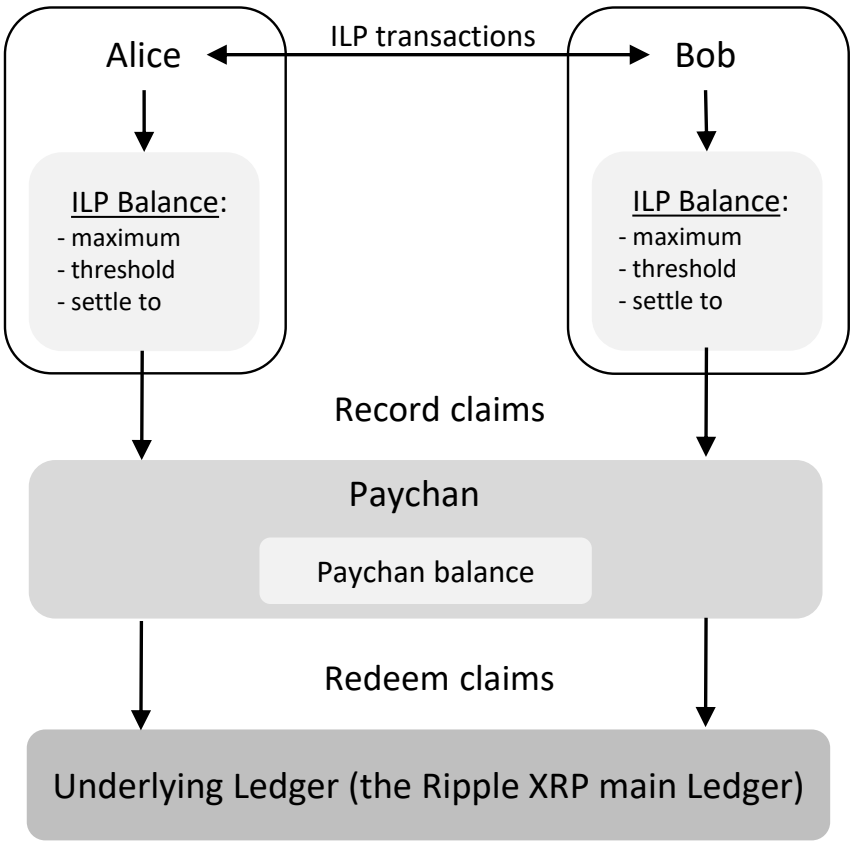

Fig. 2: The money transfer system.

Maximum Balance. Peers can set the *maximum balance* they are willing to trust (or risk) in relation to each other in the plugins configuration (for the reference js connectors and Moneyd).

Settle Threshold. The ILP peer balance can be settled manually or automatically. The connector plugins can be set to automatically settle this balance on the paychan by using the "*settleThreshold*" flag. If, for example, Alice has configured a settlement threshold of -15, this means that she will settle with Bob as soon as she owes him 15 XRP or more.

"In a correctly configured peering the additive inverse (negation) of the settlement threshold of one peer will be less than the maximum balance of the other peer." [5]

settleTo. The automatic *Settlement* process should attempt to re-establish the ILP balance to this amount. This is done through a paychan claim.

4.2.2 Payment channels.

Payment channels, or in short *"paychans"*, are an important feature of nowadays Interledger. Some distributed ledgers are defining their own payment channel concept. Therefore, it is important to keep in mind the definition of the Payment Channel agreed in the documentation of Interledger[20].

Payment channels are opened only between direct peers. *Settlement*, presented in Section 4.2.3, also occurs only between direct Interledger peers. When two peers connect over ILP, they open a payment channel. Their bilateral transactions will afterwards be carried on to the paychan. Paychans are a solution for faster, cheaper and more secure transactions, especially when the ledger involved is slow or expensive.

Below is an explained example of an Interledger paychan details [6]:

```
{
account: 'rLR52VSZG3wqSrkcpfkSnaKnYoYyPoJJgy', //the payer's XRP account address
amount: '100000', //size of the payment channel
balance: '0', //the amount the payee expects to have already received from the channel
destination: 'rMqUT7uGs6Sz1m9vFr7o85XJ3WDAvgzWmj', //the payee's XRP address
publicKey: 'ED6AF48DB11D68CDF37B22D594DC18B7C1AF2D4157A7F9A487481469A7A7C91AE2',
    //public key used for the channel. Can be the payer's master key pair. Necessary
    to verify and redeem claims.
settleDelay: 3600, //(in seconds) provides time for payee to redeem outstanding claims
sourceTag: 2100406056,
previousAffectingTransactionID:
    '8D8FF4F2AD33FAB476CFBD7256D6138419BAAC6EFDAF16ECEEFAC704752B330A',
previousAffectingTransactionLedgerVersion: 201538
}
```

When two nodes connect over ILP they negotiate the paychan details according to their business needs. The main characteristic of a paychan is its size, which is the largest claim one can sign before they need to add more money. The paychan and its corresponding details, including size, is recorded on the ledger. This is a guarantee that obligations will be eventually settled, according to the channel size, with the help of the underlying ledger. The trust invested

[20] https://github.com/interledger/rfcs/blob/master/0027-interledger-protocol-4/0027-interledger-protocol-4.md, accessed June 2019

in the ledger regarding the paychan is implicit, because the ledger was already trusted when opening the main accounts.

When transacting on a paychan, the two parties hold a Bilateral Ledger, which records the transactions performed in-between the two, and the balance. Most of the transactions are performed off-ledger, thus improving the speed and transaction costs also. Only when the peers redeem their recorded paychan claims, the specific transaction is recorded on-ledger. On the payment channel each claim is recorded individually, but they can be later redeemed individually or in bulk on the ledger [6].

4.2.3 Settlement.

Settlement is a core concept used in ILP, which is part of the larger system of money transfer over ILP. In practice, *settling* is encountered for example while setting up plugins or in relation to payment channels. The main concept, illustrated in Figure 2 [7], in practice usually involves a system of Interledger balances and paychan claims.

In relation to paychans, *settlement* involves signing a claim for the money owed. Claims do not need to be directly submitted to the ledger, but for the case of the Ripple ledger which is fast and cheap, they can [8, 6]. The process will be reflected in the paychan balance in Figure 2. Multiple claims can be signed on the paychan, and the paychan balance will update accordingly. Note that at this point, no amount or transaction has been yet recorded on-ledger (except the initial channel creation and funding), so the ledger accounts for Alice and Bob still show the same balances as before (except for cheap and fast ledgers like the Ripple ledger which makes it possible to submit claims individually if desired, to make the money available faster).

Claims can be redeemed out of the paychan and into the user ledger account in bulk or individually. The paychan can be closed or reused.

4.2.4 On-Ledger transfers.

On-ledger transfers can be initiated in different ways. The most relevant in ILP is redeeming the claims submitted on the paychan. Only at this point, the money will show up in the user wallet. Another possible way to initiate an on-ledger transaction is for example directly using the Ripple-API.

> Note: If Alice and Bob are end-users, or customers, running an ILP customer module to connect to ILSPs (connectors), the situation presented in Figure 2 will never happen, because Alice and Bob can never have a direct peering and settlement relationship. Their peering and settlement relationships are with their direct peers, respectively their parent connectors. So in this case, "Bob" should be in fact a "Connector" such that Figure 2 is correct with respect to real-life situations. As such, a real-life scenario would look in fact as shown in Figure 3: If Alice wants to send Bob 10 USD, the money will end up at Connector 1 and she will settle with Connector 1. In its turn, Connector 1 will pay 10 USD to Connector 2. Connectors 1 and 2 will settle between each other. Further, Connector 2 will forward the 10 dollars to Bob, and settle with Bob. By means of this chain, Alice has in fact sent Bob 10 USD.

4.3 The Interledger protocol suite

The Interledger architecture is often compared with the Internet architecture, as in Table 1. As a matter of fact, they adopt the same layered approach [9]. It involves multiple protocols, but the most important are BTP, ILP, STREAM and SPSP, which are presented below.

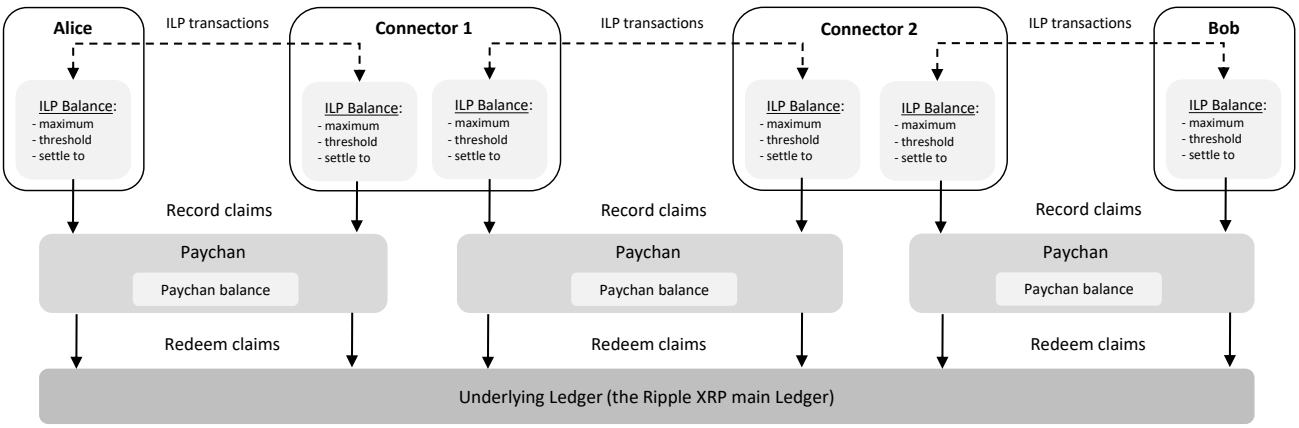

Fig. 3: The money transfer system in practice.

Table 1: A parallel between the Internet and Interledger architectures. [9]

Internet architecture		Interledger architecture	
L5 Application	HTTP SMTP NTP	L5 Application	SPSP HTTP-ILP Paytorrent
L4 Transport	TCP UDP QUIC	L4 Transport	IPR PSK STREAM
L3 Internetwork	IP	L3 Interledger	ILP
L2 Network	PPP Ethernet WiFi	L2 Link	BTP
L1 Physical	Copper Fiber Radio	L1 Ledger	Blockchains, Central Ledgers

4.3.1 The Simple Payment Setup Protocol

The Simple Payment Setup Protocol (SPSP) is a protocol for exchanging the required information in order to set-up an Interledger payment between a payee and a payer. It is the most widely used Interledger Application Layer Protocol today [10]. SPSP makes use of the STREAM protocol to generate the ILP condition and for data encoding.

Because STREAM does not specify how to exchange the required payment details, some other protocol and app have to implement this. SPSP is a protocol that uses HTTP for exchanging payment details between the *sender* and the *receiver*, such as the *ILP address* or *shared secret* [11]. In other words, SPSP is a means for exchanging the server details needed for a client to establish a STREAM connection. It is intended for use by end-user applications, such as a digital wallet with a user interface to initiate payments. The SPSP Clients and Servers make use of the STREAM module in order to further process the ILP payments. SPSP messages MUST be exchanged over HTTPS.

Payment pointers can be used as a persistent identifier on Interledger. They are a standardized identifier for accounts that are able to receive payments [12]. *Payment pointers* can also be used as a unique identifier for an invoice to be paid or for a pull payment agreement. The main characteristics of the payment pointers are:

- *Unique and Easily Recognizable*
- *Simple Transcription*: It should be easy to exchange the payment pointer details with a payee

- *Flexible*: avoid being strongly connected to a specific protocol
- *Widely Usable*: should be easy to implement and use.

The syntax is: *"$ host path-abempty"* [12].

The ***SPSP Endpoint*** is a URL used by the SPSP Client to connect to the SPSP Server, obtain information about it, and set up payments. The *payment pointer* automatically resolves to the *SPSP endpoint* by means of a simple algorithm. If *"path-abempty"* is not specified, it is replaced with *"/.well-known/pay"* [12]. Table 2 also explains how a Payment pointer is resolved to an SPSP endpoint.

Table 2: Payment pointer to Endpoint conversion. [12]

Payment pointer	SPSP endpoint
$example.com	https://example.com/.well-known/pay
$example.com/invoices/12345	https://example.com/invoices/12345
$bob.example.com	https://bob.example.com/.well-known/pay
$example.com/bob	https://example.com/bob

Technically, payments could be performed without the use of *payment pointers / SPSP endpoints*. However, for ease of use, for the practical reasons explained above, the *payment pointers* and SPSP have been introduced on top of the existing technology.

In conclusion, there are multiple forms of identification, associated to different levels of the system:

- *Ripple account address* or *Ripple wallet address*, used at *ledger level* to identify your user account holding the money on the ledger. It is analog to your bank account number
- *ILP address*, used at *ILP level* to uniquely identify your ILP node in the global network. Could be compared to an IP address.
- *SPSP endpoint*, used by the *SPSP protocol/app* to set-up an ILP payment
- *payment pointer* used by *humans* as an easy-to-handle unique payment identifier, in the same fashion as an email account address. It is also used in association with SPSP.

Example 1. Concerning the Figure 4:

- Suppose each Alice and Bob have their own accounts in XRP on the XRP Ledger.
- Alice wants to pay Bob in XRP.
- Bob already has his own *payment pointer, $example.com/bob*, which he easily shares to Alice verbally. This information is easy for Alice to remember and use. (step (1) in Figure 4)
- Supposing the machine already set-up, at her computer, Alice starts up Moneyd, and is able to pay Bob just by introducing his *payment pointer address "$example.com/bob"* and the amount paid in Moneyd-GUI, or from a terminal, by using a single line:

"ilp-spsp send –receiver $example.com/bob –amount 100"

Behind the scenes:

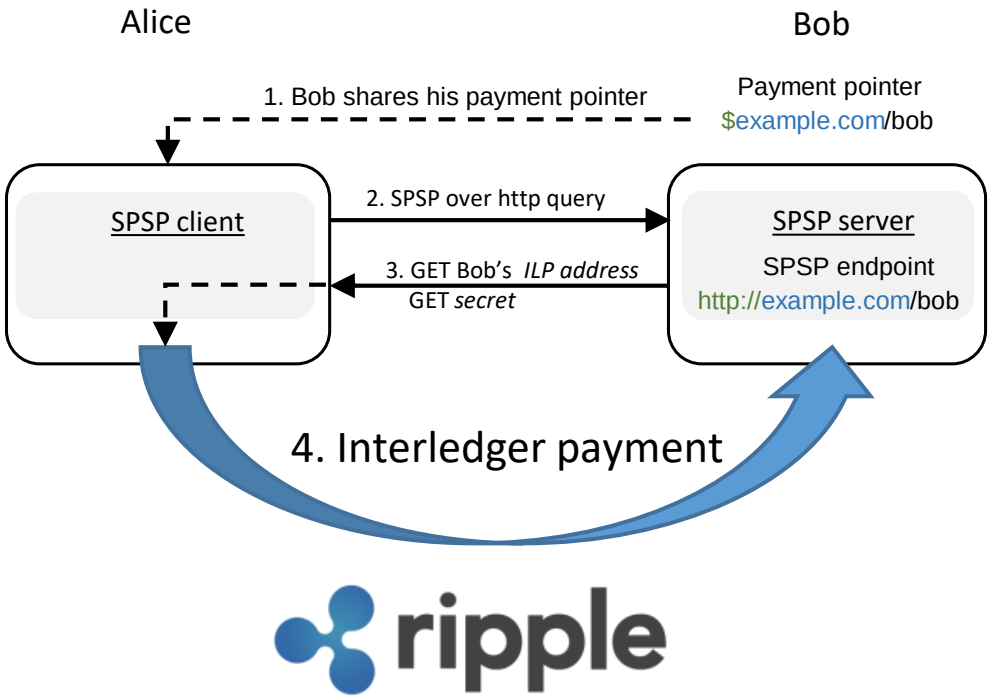

Fig. 4: Example 1: SPSP payment.

Alice's SPSP client:
- resolves the *payment pointer "$example.com/bob"* to https://example.com/bob
- connects over HTTPS to Bob's SPSP server, at the address "https://example.com/bob" (2)
- queries the SPSP server for Bob's *ILP address* and a *unique secret* (2)
Bob's SPSP server sends Bob's *ILP address* and a *secret* to Alice's SPSP client (3)
Using this information, Alice's SPSP client starts an ILP payment to Bob over Interledger (4).

The usage of public endpoints involves employing HTTP, TLS, DNS, and the Certificate Authority system for the HTTPS request that SPSP makes. *However, the alternatives leave much to be desired. Few people outside of the cryptocurrency world want cryptographic keys as identifiers and, as of today, there is no alternative for establishing an encrypted connection from a human-readable identifier that is anywhere nearly as widely supported as DNS and TLS* [10].

4.3.2 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 micropayments 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.

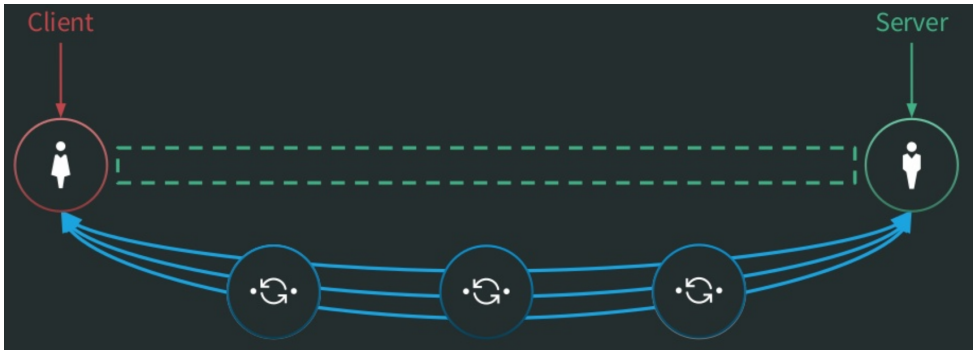

Fig. 5: STREAM is a logical, bidirectional channel over ILP. [9]

As illustrated in Figure 5 with green, a STREAM connection establishes a two-ways, virtual channel of data and money between the payer and payee. STREAM packets are encoded, encrypted, and sent as the data field in *ILP Prepare* (type 12 ILP packet), *Fulfill* (type 13 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 5). Either the payer or the payee can be the server or the client. STREAM provides authentication, encryption, flow control (ensure one party doesn't send more than the other can process), and congestion control (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 [13, 14].

Example 2. We now provide a more advanced explanation regarding the same situation presented in Example 1. We will refer to Figure 6, and extend the explanation from Example 1:

- 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 7.

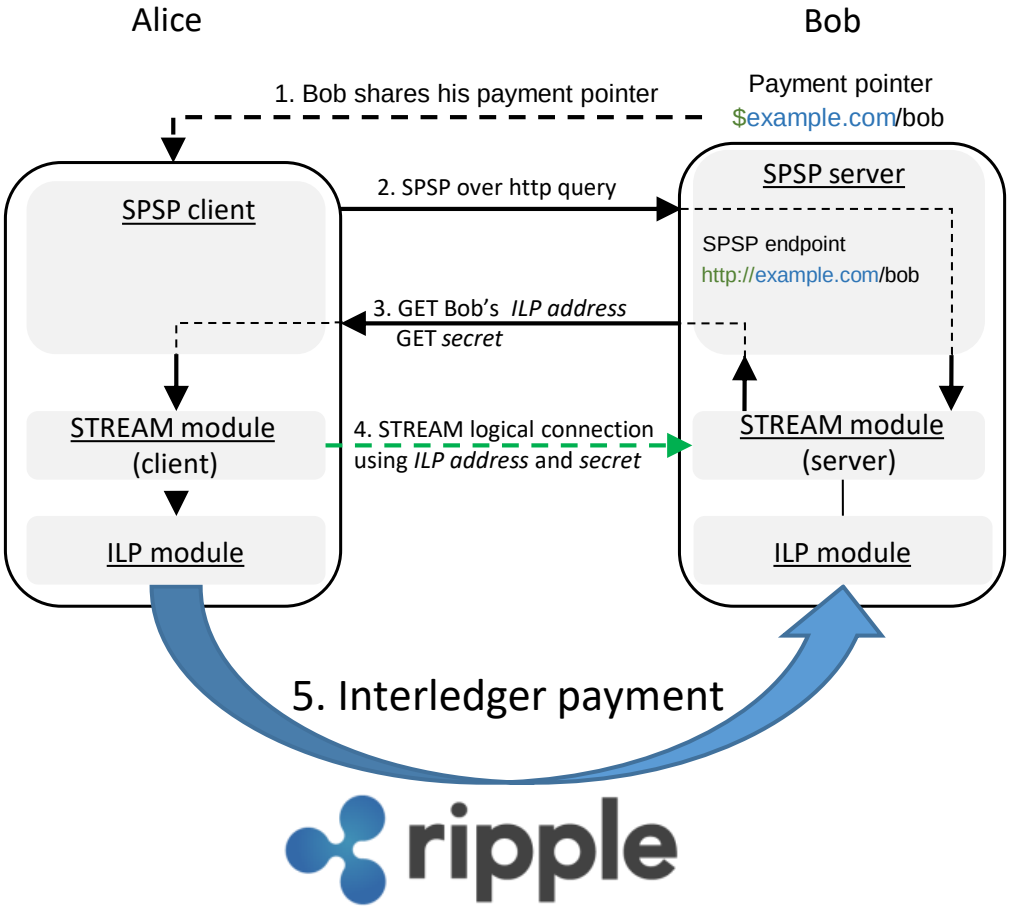

Fig. 6: Example 2: STREAM payment.

4.3.3 The Interledger Protocol

The Interledger Protocol (ILP), currently at version 4, is the main protocol facilitating the Inter Ledger money transfers. It provides a solution to route payments across disconnected ledgers while minimizing the sender and receiver's risk of losing funds. What makes it different from previous versions is that it is optimized for sending many low value packets:

"We talked about the idea of streaming payments, where if you make payments so efficient that you could pay for like a milliliter of beer or a second of video. That's the way we think about efficiency of payments."[15]

It is made for payment channels, which means faster and cheaper payments, while also accommodating any type of ledger.

ILPv4 involves *Hashed Time Lock Agreements (HTLA)* [17], and makes use of three packet types:

- *Prepare*, corresponding to request, with the following fields:
  - *destination* - ILP address,
  - *amount* - UInt64,
  - *condition* - UInt256,
  - *expiration* - timestamp,

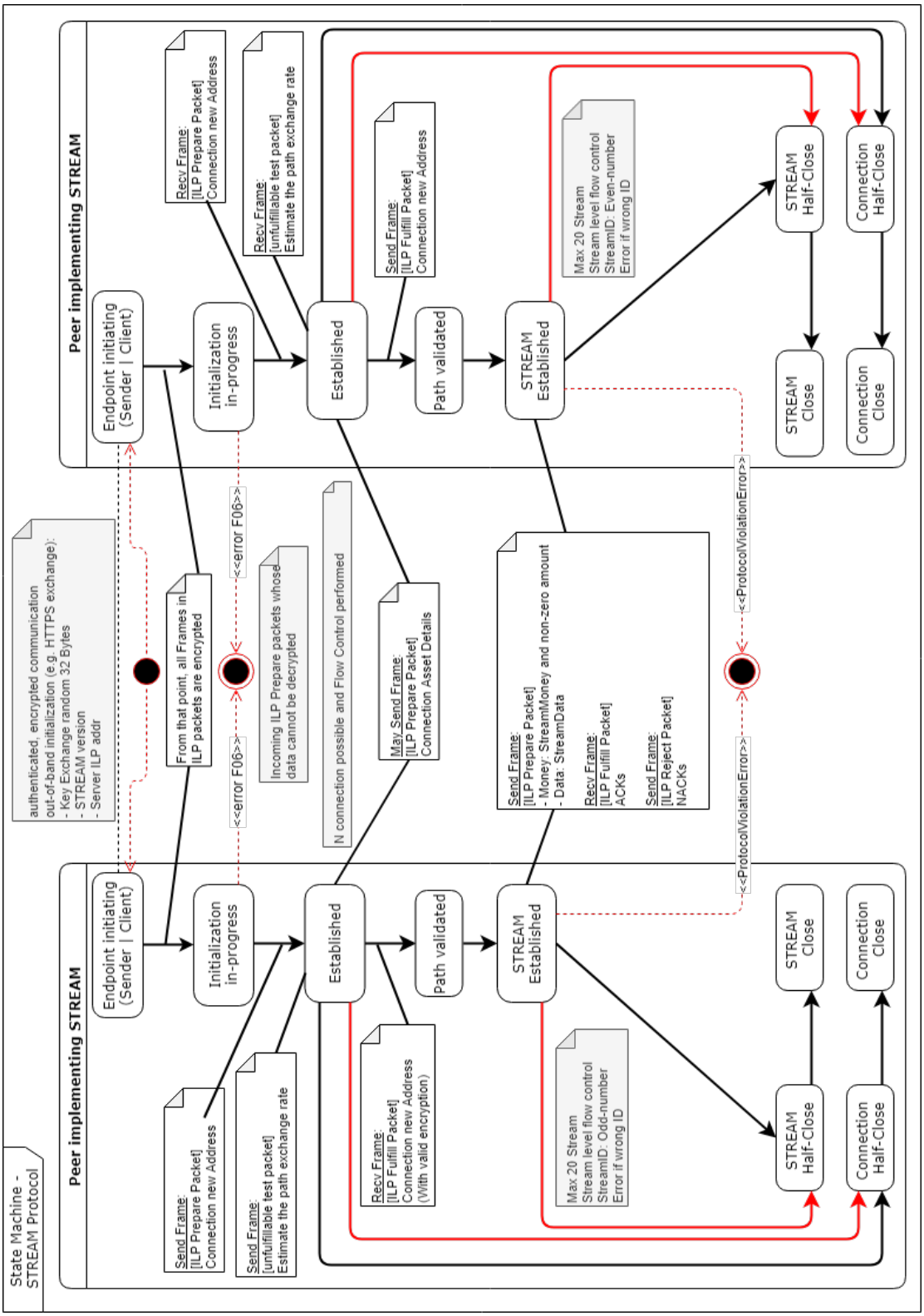

Fig. 7: STREAM protocol: FSM diagram.

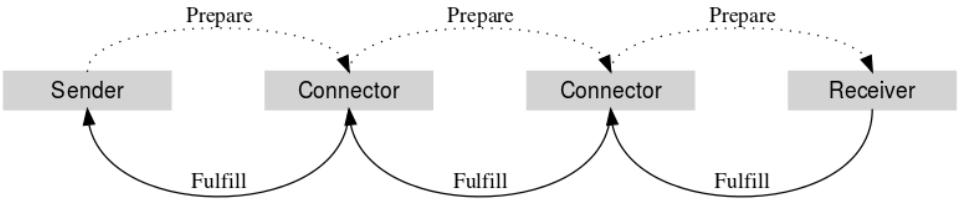

Fig. 8: ILP packet flow. [16]

– *end-to-end (sender-receiver) data* - OCTET STRING.

Example of an ilp-prepare packet:

```
{
amount= 69368000,
executionCondition= fHII9adb3JY3D5drSNSoquLTIUJJhNLMeiiADnW4li0=,
expiresAt= 2019-06-19T11:04:18.149Z,
destination= g.conn1.ilsp_clients.mduni.local.viby9ZjztwCVMtptFjaueqsdlIxWSUba
       y7Jo3BxJyGc.elrqFEKZEc8BMcZ4PDUiPEAF,
data= t6lmRiiFZecXhltYNsnyPYSgPld+Itmn+NefM5ytnFJiFDuMieyF9b2vB
       o2HPiNm34GpCBlU/HoGaCAsOQ==
}
```

- *Fulfill,* corresponding to response, and carrying the following fields:

  – *execution condition fulfillment* - UInt256,
  This is the proof that the receiver has been paid, so the fulfill packets are relayed back by the connectors from the receiver to the sender. It consists of a simple pre-image of a hash, and only the receiver can know this information.

  – *end-to-end (sender-receiver) data* - OCTET STRING.

  The components of the prepare and fulfill packets concerning HTLA are:
  - *amount, time* (expiration), and *condition* for Prepare, and
  - the *execution condition fulfillment (the hash)* for the Fulfill packet, which must be received before *expiration.* This implies that the machines involved in the process should be time-synchronized. This is not an absolute enforcement, but any time offsets will packet rejection chances.

- *Reject,* corresponding generally to error messages. They can be returned either by the receiver or the connectors in specific conditions and consist of:

  – *a standardized error code*,

  – *triggered by:* - ILP address;
  is the identifier of the participant that originally generated the error,

  – *user-readable error message* - UTF8String,

  – *machine readable error data* - OCTET STRING.

The connectors forward the *prepare* packets from the *sender* to the *receiver*, and relay back the response or the reject from the *receiver* to the *sender,* as shown in Figure 8. As such, ILP v4 uses a chaining of HTLAs to achieve an end-to-end transfer [17]. In ILP v4, HTLAs are mainly

supported over Simple Payment Channels. Simple Payment Channels are generally supported by today's major blockchains like BTC, ETH, XRP,.. [18, 19].

Concerning Figure 8 and ILP v4: even if the original ILP packet is prepared by the *Sender* and addressed for the *Receiver* (end-to-end), the transfer from the *Sender* to the *Receiver* will be in fact a chaining of transfers between the directly connected (and trusted) peers. Each pair of directly connected peers generally uses a dedicated, separate Payment Channel to settle their obligations [17, 18, 19]. Other means are possible [19], but not really used or supported [18].

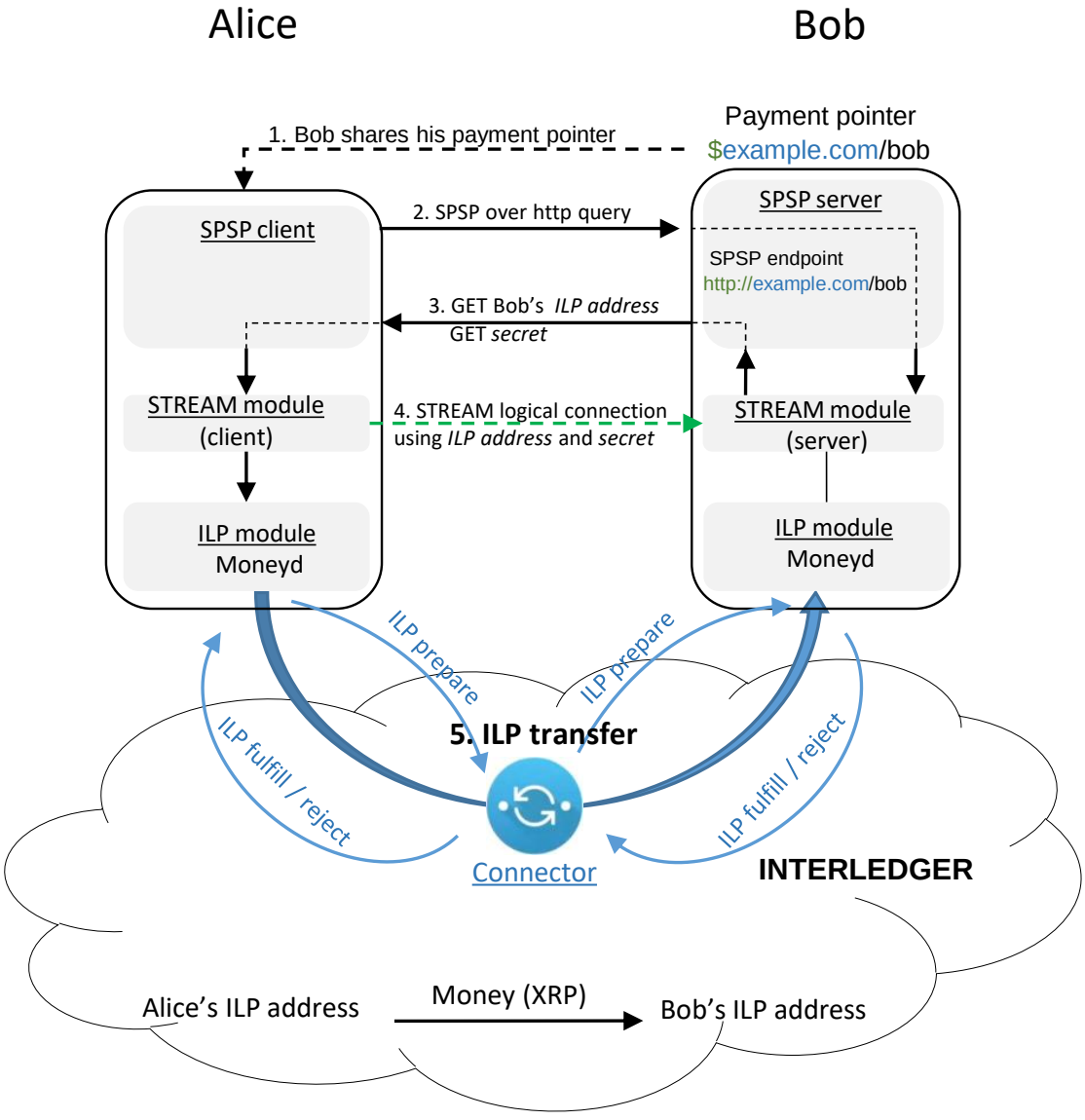

Fig. 9: Example 3: ILP.

Example 3. We will further expand on the Examples 1 and 2, using the Figure 9.

- Alice's SPSP client:

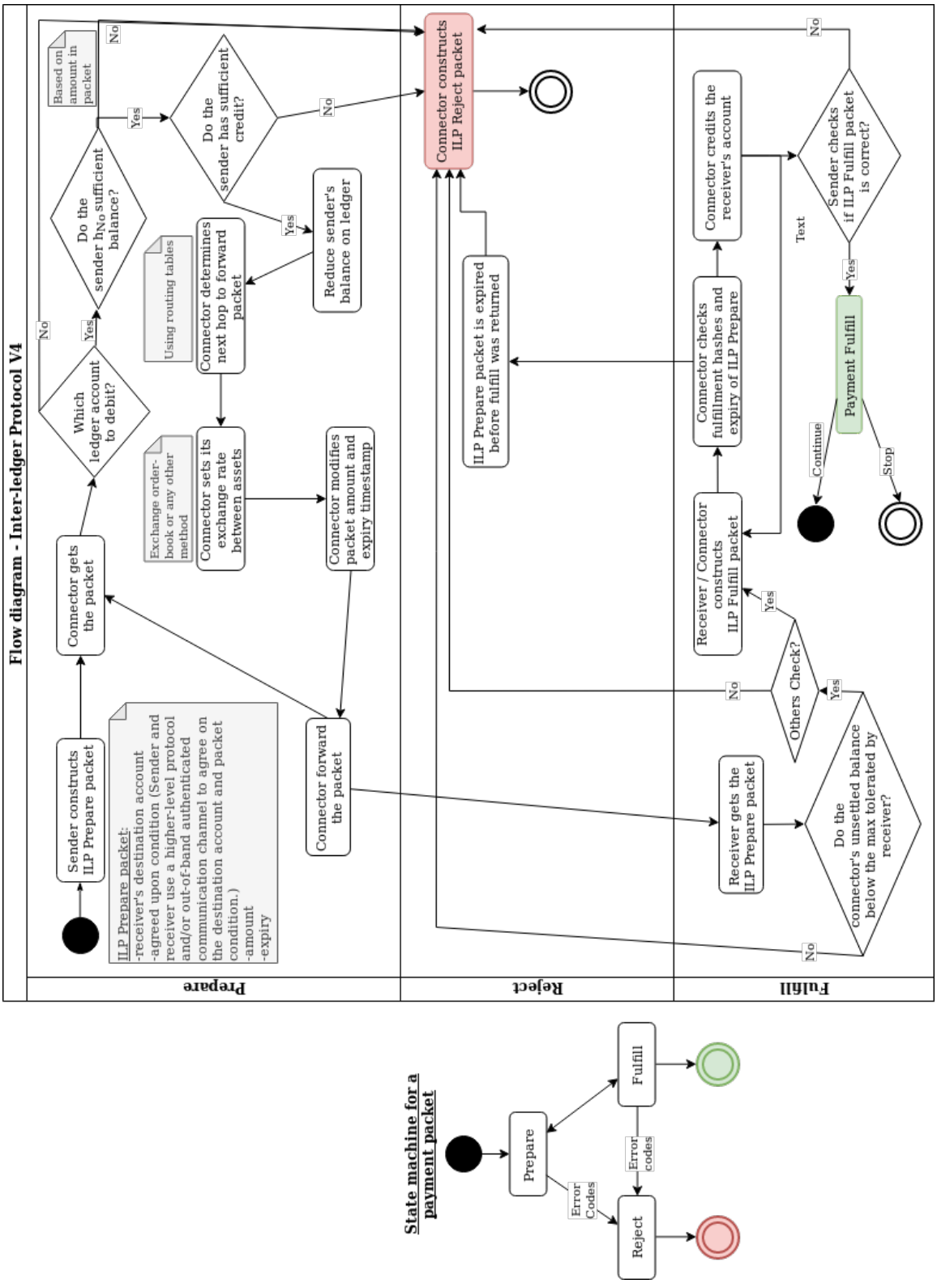

Fig. 10: ILPv4 flow diagram.

  - 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 module, in our case, Moneyd (4)
- The ILP module, Moneyd, sends the ILP packets corresponding to the STREAM virtual connection towards its upstream parent connector, which further routes them to its child, Bob's Moneyd module (5).

The STREAM module is able to break the payment into multiple packets, which would be sent over ILP using *prepare-fulfill-error* packets. The STREAM module at the *receiver's* end will finally reassemble the payment.

Wrapping-up ILP, Figure 10 presents the finite state machine diagram of an ILP packet and the protocol flow chart.

4.3.4 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.

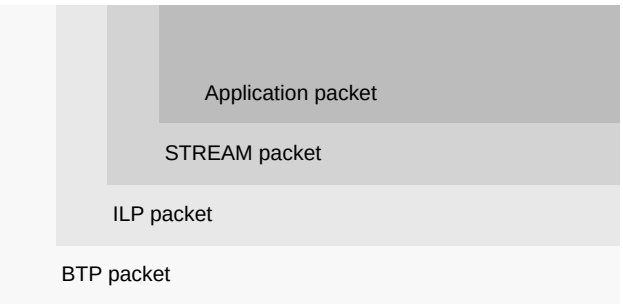

Fig. 11: Packet data structure. [20, 21]

As shown in Figure 11, 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 14, it is to be noted that ILP can still work without BTP [22].

With regards to the current state of BTP, the following post by Ewan Schwartz is worth mentioning: *"BTP is a binary request/response protocol implemented over WebSockets. It originally included message types for Prepare, Fulfill, Reject and Transfer, but BTP 2.0, which is used today, stripped out nearly everything except request/response semantics, authentication, and "sub-protocol" naming".* [10]

Example 4. We can now complete our diagrams presented in Examples 1, 2 and 3 with the BTP protocol, which is illustrated in Figure 12.

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 will be further explained in Examples 5, 6, and 7.

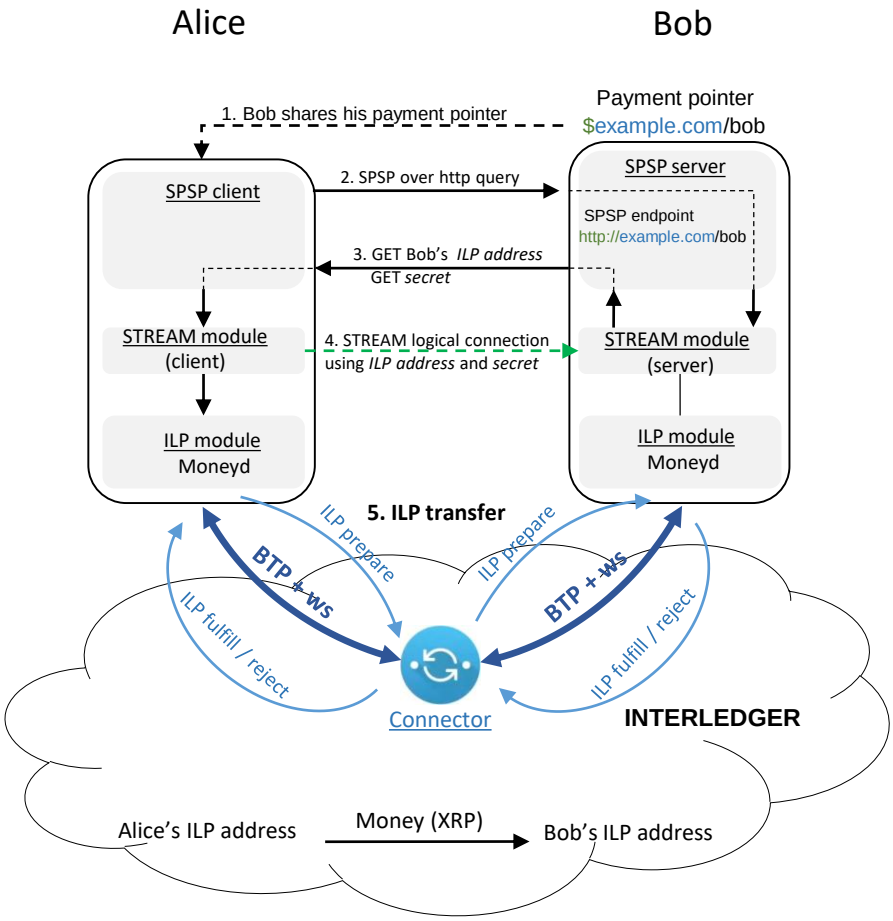

Fig. 12: Example 4: BTP. [20, 21]

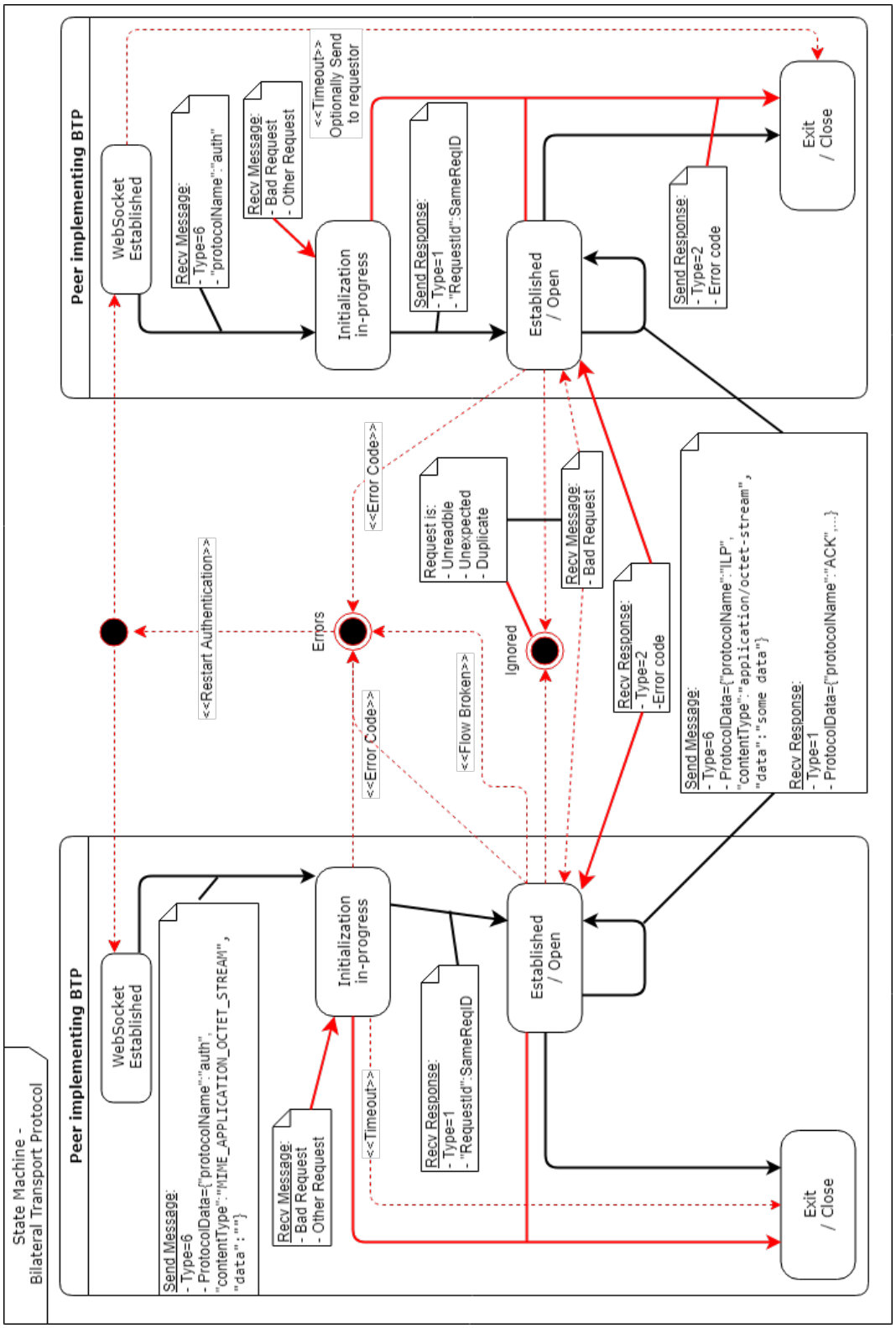

Fig. 13: BTP: the finite state machine diagram.

The finite state machine of the BTP protocol is presented in Figure 13.
The relationship between protocols, and especially the STREAM protocol, can be best un-

derstood by referring to Figure 28 at the end and reading the thorough explanations provided by [20, 21].

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 [20].

In the examples that follow, we are going to make use of BTP, ILP, STREAM and SPSP, a protocol suite also depicted in Figure 14.

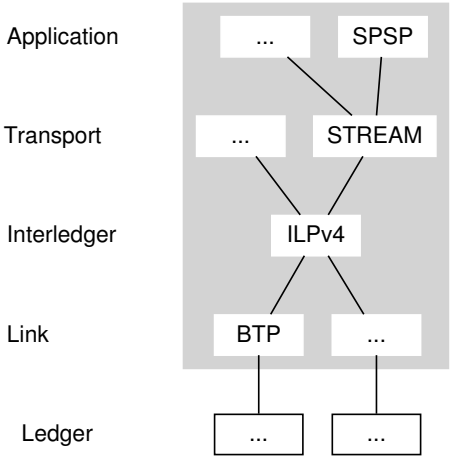

Fig. 14: The protocol suite. [16]

5 Customer apps for money transfer

5.0.1 A customer environment comprising Moneyd, Moneyd-GUI and SPSP client/server.

Moneyd: *'Moneyd provides a quick on-ramp as the "home router" of the Interledger, giving apps on your computer access to send and receive money. Although Moneyd can just as easily connect to the production Interledger, it is not currently designed for heavy production use, so it lacks features like budgeting that would let you give apps different spending limits and permissions levels.'* [23]

Moneyd connects to the Interledger and sends and receives ILP packets for you. It is a simplified, end-user version of a connector, and as such, it will not send or receive routes as a regular connector does. Apps running on your machine that require access to ILP will connect to the Interledger through Moneyd.

When you fire-up Moneyd:

- Moneyd loads *~/.moneyd.json* and instantiates an ILP Connector
- The connector (i.e. "Moneyd") opens a web socket connection to its parent connector (configured in *~/.moneyd.json*) and creates a payment channel, which can be in XRP, Ethereum,.. depending on the uplink connection

- The connector (i.e. "Moneyd") listens on the local port 7768 to process ILP payments. All your local apps (like SPSP) will connect here for ILP.

To install Moneyd for XRP[21], just type the following into the terminal:

npm install -g moneyd moneyd-uplink-xrp

The best way to understand it is to configure it in advanced mode and start in DEBUG mode.

Configure it using: *'moneyd xrp:configure –advanced'*. A configuration file will be created in the user root folder, as *~/.moneyd.json*. If needed, this file can be inspected and manually updated afterwards.

Start in DEBUG mode using: *'DEBUG=* moneyd xrp:start –admin-api-port 7769'*. The option *'–admin-api-port 7769'* will open the port 7769 as Moneyd admin port, such that Moneyd-GUI can connect to it and administration can be performed in a web browser.

Moneyd-GUI *is a frontend for Moneyd (or the reference js connector), which displays statistics, sends and receives money, and helps with troubleshooting* [22]. Running on the same machine with Moneyd or with the js reference Connector, it can be installed with[23]:
npm install -g moneyd-gui.
It will connect automatically to the port above, after being started with:
'npm start' in *'/home/user/moneyd-gui'*.
In a web browser, it can be accessed by default at http://127.0.0.1/7770.

SPSP: Another business case illustrating the way SPSP and Moneyd/ILP are working together (because Moneyd is implementing ILP for settlement) was well illustrated in Figure 15. Both Alice's and BigCompany's SPSP client and server are connected to ILP through Moneyd. Alice's SPSP client connects to BigCompany's SPSP server which exposes an HTTPS endpoint abstracted as a payment pointer [24].

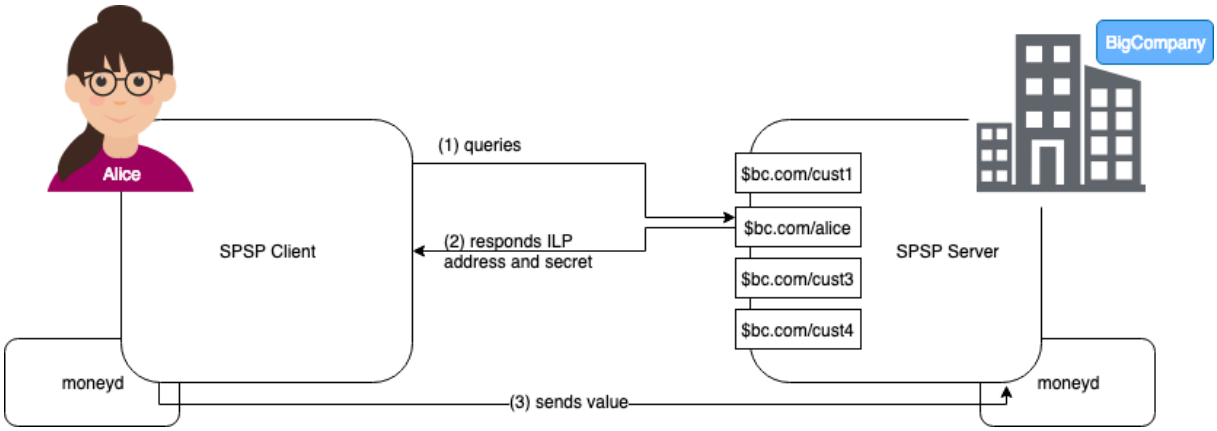

Fig. 15: SPSP and Moneyd [24]. Between the two Moneyd instances there are usually multiple connectors providing the ILP service.

The protocol follows like below [24]:

[21] https://github.com/interledgerjs/moneyd, accessed June 2019
[22] https://medium.com/interledger-blog/use-interledger-with-moneyd-gui-21dee0dc8ba0, accessed June 2019
[23] https://github.com/interledgerjs/moneyd-gui, accessed June 2019

- Alice queries her customer payment pointer (1), and
- receives the ILP *destination_account* and a *shared_secret* (2)
- Using the data obtained, Alice's SPSP client starts a STREAM connection and sends the money to BigCompany (3).

'The SPSP module calls the Interledger module with the address and other parameters in the Interledger packet to send a payment. The Interledger module would send a transfer to the next connector or destination account along with the Interledger packet and according to the parameters given. The transfer and Interledger packet would be received by the next host's Interledger module and handled by each successive connector and finally the destination's SPSP module' [25].

As a side note, SPSP first started with PSK and was later upgraded to STREAM.

Using the below procedure, one can send and receive money by writing a single line in the terminal. An SPSP server[24] and a client[25] should be installed on the recipient's and sender's machines, respectively. They work in pair and have as pre-requisite Moneyd but can also work alongside the reference connector.

The following lines are provided for the case of a private independent network using IPs as payment address identifiers. For real-life situations where addresses in the form of *'$sharafian.com'* can be provided, the original github guidelines can be used.

In order to transfer money, we should first have a receiver address, i.e. **start the server** which listens for an incoming payment. Using 'DEBUG' will provide more info on what happens behind.

SPSP 'server' listening for an incoming transfer
Install the server with:
'DEBUG= ilp-spsp-server –localtunnel false –port 6000'*
'–localtunnel false': forces the use of IP as the address instead of creating a tunnel and obtaining an address like *'$mysubdomain.localtunnel.me'* (analogue to *"$example.com/bob"*).
'–port 6000': the server will listen for incoming payment from an SPSP 'client' on this port.

An SPSP 'client' sending money to the server
The money can be then **sent** by another user/machine having a similar setup, with:
'DEBUG=ilp ilp-spsp send –receiver http://192.168.1.116:6000 –amount 100'.*
Where *'http://192.168.1.116:6000'* is the receiving server's IP and port.

To conclude this section we provide the following very nice explanation on SPSP, ILP, Moneyd by Ben Sharafian on the ILP forum, which we feel helps consolidate and clarify the larger picture:

In Interledger, the sender and receiver don't have any settlement relationship nor do they have any trust relationship.

The only relationship is to your direct peer. If you're connecting to Interledger through Moneyd, we would call this peer your "upstream connector" or "uplink". When you send packets through your upstream connector, Moneyd keeps track of the amounts and settles them.

[24] https://github.com/interledgerjs/ilp-spsp-server, accessed June 2019
[25] https://github.com/interledgerjs/ilp-spsp, accessed June 2019

SPSP isn't doing any of the tracking for settlements, that all lies at the Interledger layer. This lets you save a lot of headache when implementing high level protocols: settlement is always abstracted away.

Moneyd does settle on-ledger. The configuration that Moneyd uses to connect to its upstream connector determines the currency that this happens in. (for XRP, moneyd-uplink-xrp).

The currency that SPSP pays in depends on what currency is used by the Moneyd it connects to. If you use a Moneyd uplink in XRP, then the currency will be XRP [26].

Example 5. XRP payment using Moneyd, SPSP and a connector.

We consider Figure 16 a good starting example because while involving a relatively simple structure, it still illustrates the main structural components of the Interledger ecosystem:

- A ledger, i.e. the Ripple ledger
- A connector
- Customers:
  - Alice, operating Machine A with Moneyd-XRP and SPSP
  - Bob, operating Machine B with Moneyd-XRP and SPSP
- The main protocols: BTP, ILP, STREAM and SPSP.

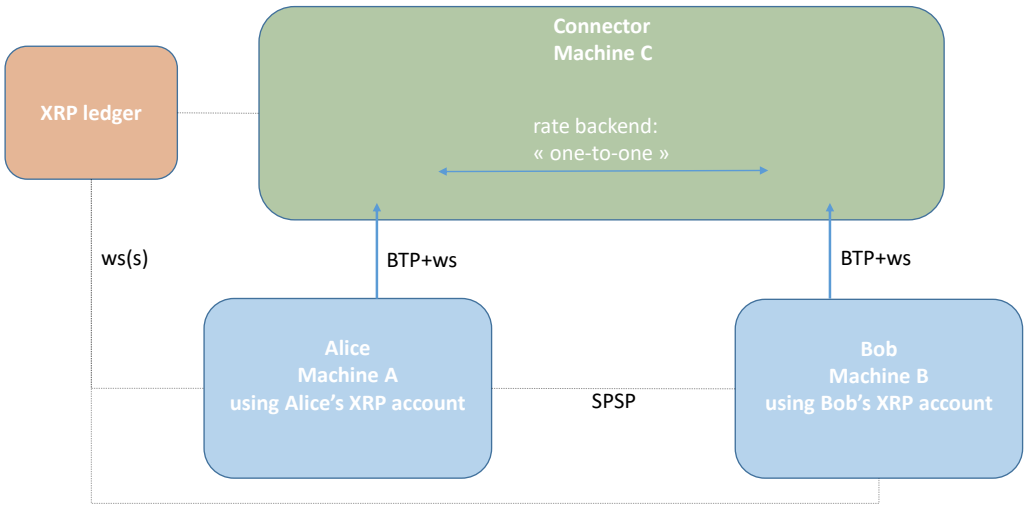

Fig. 16: Example 5: XRP payment.

Alice, Bob and the Connector are connecting to the ledger over a web socket connection to settle their obligations and exchange ledger-related data with the ledger. In order to access ILP, Alice and Bob are also connecting to the Connector through BTP+ws connections. The Connector works like an ILP and payment bridge between them, and because everything happens in XRP, the exchange rate applied is 1, and the backend used by the connector is "one-to-one".

Further, Alice and Bob can initiate exchanges of value, i.e. XRP payments, in this case by using SPSP.

The case is detailed in Figure 17, where we can identify:

- The Interledger Service Provider (ILSP) machine, containing:
  - The connector, where we can identify:
    - The connector core
    - Plugins
    - The rate backend, set as "one-to-one"
  - Moneyd-GUI
  - A web browser connecting locally to Moneyd-GUI as a visual admin interface on http://127.0.0.1:7770
- The XRP ledger, accessible over a web socket connection, usually at port 51233
- Alice's machine, comprising:
  - Moneyd-XRP:
    - Moneyd-core
    - XRP plugin
    - XRP uplink
  - Visual admin interface:
    - Moneyd-GUI
    - Web browser
  - SPSP client and server
- Bob's machine, with a similar setup.

Using Moneyd-XRP, Alice "dials-up" to, and opens a paychan with the Connector, thus enabling ILP access on her machine. The connection with the Connector is made over BTP+ws. Between other functions, BTP acts as a "carrier" for ILP. The paychan is recorded on the ledger. Alice is able to administer Moneyd through a web browser, using Moneyd-GUI. The SPSP app running on her machine will get ILP access through Moneyd. Bob's situation is similar.

The ledger holds Alice's, Bob's and ILSP's XRP accounts, and the paychans corresponding to the pairs Alice - ILSP (Connector) and Bob - ILSP (Connector).

When Alice sends a payment addressed to Bob, what happens is that the connector pays Bob on behalf of Alice (if Bob is able to provide the payment condition), and this transaction is recorded between the Connector and Bob. Further, the Connector presents to Alice the payment condition he just got from Bob, and Alice pays the Connector in exchange. This transaction is also recorded between Alice and the Connector. The value has moved from Alice to Bob, and in turn, two transactions have been recorded: Alice to Connector, and the Connector to Bob. Further, it is up to these pairs (Alice-Connector and Connector-Bob) to settle these transactions according to conditions agreed between each other (using the pay channels).

From a technical point of view, the sequence is illustrated in Figure 18, and follows like below:

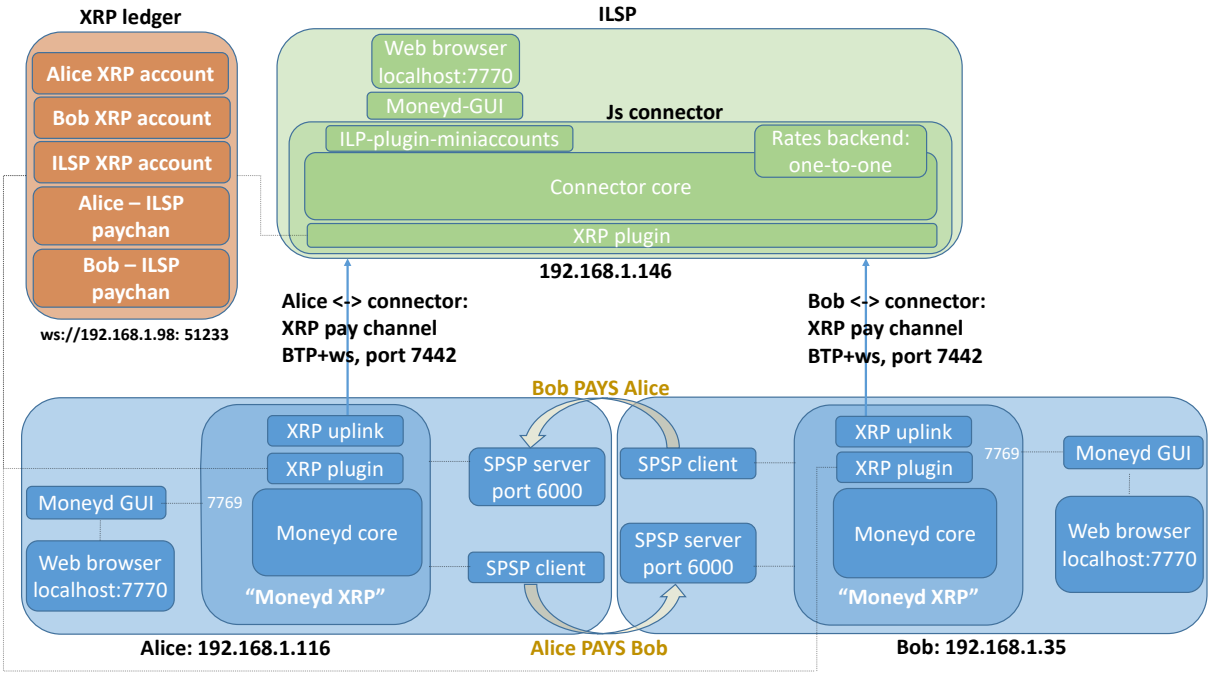

Fig. 17: Example 5: XRP payment, advanced.

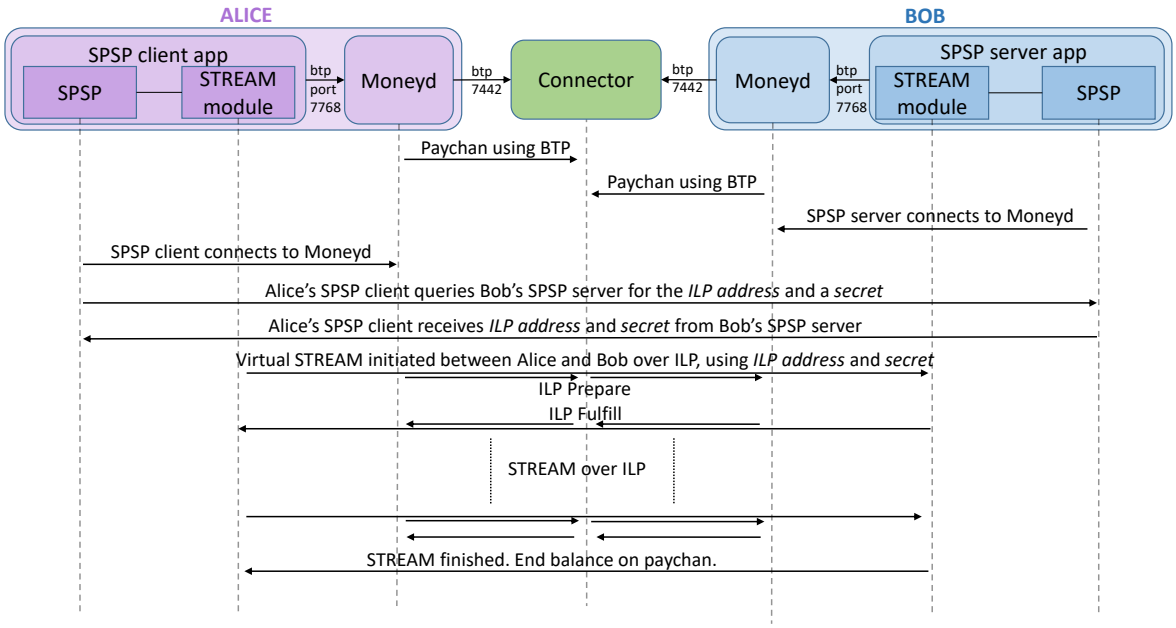

Fig. 18: Example 5: Protocol sequence. BTP, ILP, STREAM and SPSP are being used.

- Alice and Bod start-up and connect their Moneyd instances to the ILP connector. Each of them opens dedicated payment channels with the connector. Their transactions will be performed on the payment channels through Moneyd and ILP.

This is an example of a BTP packet sent between Moneyd and the Connector in order to open a paychan:

```
btpPacket: { type: 6,
   requestId: 1890145753,
   data: { protocolData: [ [Object], [Object], [Object] ] } } ; data: {
       protocolData:
    [ { protocolName: 'channel',
        contentType: 0,
        data:
         <Buffer 15 8b 1c 3d a9 97 e5 b6 af 10 2d 90 51 b2 cd 3d 8c 58 96 55 02
             ec dc 3a 07 02 f9 2c b7 88 61 1b> },
      { protocolName: 'channel_signature',
        contentType: 0,
        data:
         <Buffer 62 68 63 4c af b5 72 0f 2f 38 58 4a 81 03 cc 68 5a 62 ef 9a 43
             91 80 73 40 31 03 8f 49 f8 f5 ae 88 64 fd 7f 14 39 39 d7 5b c3 a2 17
             e1 7f c7 d2 49 a4 ... > },
      { protocolName: 'fund_channel',
        contentType: 1,
        data:
         <Buffer 72 70 4e 33 55 50 6a 59 61 45 72 74 34 52 57 34 67 41 69 71 75
             63 4d 43 66 4c 35 6e 4a 4a 43 34 59 7a> } ] } ;
```

- Bob starts his SPSP server application, which connects to his local Moneyd instance and starts listening for incoming connections from an SPSP client.
- Using her SPSP client, Alice queries the HTTP payment pointer presented by Bob.
- Alice receives an ILP address to use as *destination account* for the transaction and a *shared secret*.
- Using this information, Alice pays Bob by starting a STREAM payment over ILP. The ILP service is provided by Moneyd. The *shared secret* is used by STREAM to authenticate and encrypt multiple packets, as well as to generate the conditions and fulfillments.
- STREAM divides the larger payments into packets and reassembles them. The packets are encapsulated inside, and sent along with, ILP packets in the data field, and will be retrieved by the STREAM endpoint. During the STREAM connection, one of the parties acts as a client (the initiator) and the other acts as a server (the one accepting the connection). As STREAM is implemented by the SPSP application, the STREAM server will be already listening for a STREAM client connection.

Below is an example of a STREAM prepare packet carried by an ILP packet sent over BTP:

```
ILP-PLUGIN-BTP: handleIncomigWsMessage: binaryMessage:
<Buffer 06 1f 9d 97 68 81 e9 01 01 03 69 6c 70 00 81 e0 0c 81 dd 00 00 00 00 95
    02 f9 00 32 30 31 39 30 36 31 39 30 39 34 33 30 31 35 30 39 45 04 2b e1 cd
    98 ... >
ILP-PLUGIN-BTP: handle incoming packet from: ; btpPacket: { type: 6,
requestId: 530421608,
```

```
data: { protocolData: [ [Object] ] } } ; data: { protocolData:
    [ { protocolName: 'ilp',
        contentType: 0,
        data:
         <Buffer 0c 81 dd 00 00 00 00 95 02 f9 00 32 30 31 39 30 36 31 39 30 39
             34 33 30 31 35 30 39 45 04 2b e1 cd 98 68 71 95 50 c5 de 4a f2 1c f1
             eb 4e 79 6e 95 cb ... > } ] } ;
ilp-plugin-xrp-paychan: received btp packet. type=TYPE_MESSAGE
    requestId=530421608 info=ilp-prepare
ILP_PACKET: binary: <Buffer 0c 81 dd 00 00 00 00 95 02 f9 00 32 30 31 39 30 36
    31 39 30 39 34 33 30 31 35 30 39 45 04 2b e1 cd 98 68 71 95 50 c5 de 4a f2
    1c f1 eb 4e 79 6e 95 cb ... >
ILP_PACKET: type: 12
ILP_PACKET: contents: <Buffer 00 00 00 00 95 02 f9 00 32 30 31 39 30 36 31 39 30
    39 34 33 30 31 35 30 39 45 04 2b e1 cd 98 68 71 95 50 c5 de 4a f2 1c f1 eb
    4e 79 6e 95 cb d8 f6 5a ... >
ILP_PACKET: deserializeIlpPrepare:
{
amount= 2500000000 ,
executionCondition= RQQr4c2YaHGVUMXeSvIc8etOeW6Vy9j2WlDZYKIZUbM= ,
expiresAt= 2019-06-19T09:43:01.509Z ,
destination= g.conn1.ilsp_clients.mduni.local.NL8f2khL-VmasfzfA-
    w_ds5F15J063Tn4oxDwoXTjGw.gHvuhB1r5GN0UQikoCGahPsj ,
data= YFwVZXQYK7pDTrprLcFOYbyt9qGQm+0APnOaBw5w5iUvvEggyB4Le0J8Bjbav7FKGyJ6Ih95xT8
    lss4BCQ==
}
```

The first byte "06" of the BTP packet is the BTP packet type, 6 in this case.
The next 4 bytes "1f 9d 97 68", are the request_id in hex - 530421608 in this case.
We can also infer from the ILP-packet header that "0c" is the packet type - 12 in this case, and from the body, that "95 02 f9 00" represents the *amount* = 2500000000.
This confirms the packet type as STREAM, because it is type 12 [14, 22].

Below is a second example of a BTP packet carrying a STREAM fulfill:

```
{ type: 1,
   requestId: 1054375881,
   data: { protocolData: [ [Object] ] } } ; data: { protocolData:
    [ { protocolName: 'ilp',
        contentType: 0,
        data:
         <Buffer 0d 5e 78 d3 d3 3e 33 27 b9 44 a1 45 92 f8 d8 98 28 8c 96 e2 20
             00 af 8f bd eb 0d a3 24 04 79 0f 9b 75 3d 12 ef 89 a6 79 c1 a5 cc 53
             ef c6 0f c1 60 8a ... > } ] } ;
```

The first byte "0d" is the packet type - 13, meaning fulfill. The packet structure is like below:

```
{
type: 13,
   typeString: 'ilp_fulfill',
   data:
    { fulfillment:
       <Buffer 78 d3 d3 3e 33 27 b9 44 a1 45 92 f8 d8 98 28 8c 96 e2 20 00 af 8f
```

```
        bd eb 0d a3 24 04 79 0f 9b 75>,
    data:
     <Buffer 12 ef 89 a6 79 c1 a5 cc 53 ef c6 0f c1 60 8a 71 38 b5 72 70 a8 f7
        54 16 1c 30 65 f5 f1 9e fd 8f ee a6 d0 63 85 36 49 fd ab 5e 18 a6 d9
        40 04 d4 5a 61 ... > }
}
```

- when the transfer is finished, the STREAM connection is closed.
- at this moment the balances are updated on the payment channels opened between the parties involved, accordingly. The value can be redeemed out of each payment channel by each participant.
- after claiming the funds, the payment channel can be either closed or used for other transactions.

The Interledger balance between Alice and Bob is continuously updated. If for some reason the STREAM connection is interrupted before the total amount is transferred, the amounts already transferred are not lost.

"Once peered, the two connectors both track the Interledger account balance and adjust it for every ILP Packet successfully routed between them." [5]

Regarding the setup and configuration, for this example, we are going to consider the XRP Ledger and the connector black boxes, and provide instructions for Moneyd and SPSP.

- On both Alice's and Bob's machines:
  - If not already installed, install *node.js*
  - Install Moneyd, Moneyd-GUI, SPSP server and SPSP client apps:
    - *"npm install -g moneyd moneyd-uplink-xrp"*
    - *"npm install -g moneyd-gui"*
    - *"npm install -g ilp-spsp-server ilp-spsp"*
  - Configure Moneyd:
    - *"moneyd xrp:configure –advanced"*
      There will be four questions:
      - *? BTP host of parent connector:*
        We are going to use the IP:port(7442) of the connector.
      - *? Name to assign to this channel:*
        Can keep the autogenerated proposal or enter custom name.
      - *? XRP secret:*
        Alice's/Bob's XRP account secret: "sXXXXXXXXXXXXXXXXXXXXXXXXX"
      - *? Rippled server:*
        The IP:port of the local Ripple server (ws://192.168.1.98:51235) or for example, "wss://s1.ripple.com".
  - Start Moneyd, Moneyd-GUI and the browser interface:
    - Start Moneyd:
      "DEBUG= moneyd xrp:start –admin-api-port 7769"*

    - Start Moneyd-GUI by issuing:
      "npm start" in /home/user/moneyd-gui
    - Start a web browser and go to:
      http://127.0.0.1:7770
      In order for all Moneyd-GUI's graphical interface elements to load, you should also have internet access.

- On Bob's machine, start the SPSP server:
  DEBUG= ilp-spsp-server –localtunnel false –port 6000*

- From Alice's machine initiate the SPSP transfer:
  DEBUG=ilp ilp-spsp send –receiver http://192.168.1.116:6000 –amount 100*
  Obviously, the above IP is just an example and you should use here Bob's machine's IP, according to your network setup.

Additional information and useful commands for Moneyd can be found on Moneyd's github page. We provide the Table 3 below, for general reference.

Table 3: Useful Moneyd commands.

command	effect
moneyd xrp:configure –advanced	configure Moneyd in advanced mode
moneyd xrp:start –admin-api-port 7769	enable the admin api port for Moneyd-GUI use
moneyd start –unsafe-allow-extensions	allow web browser payments
moneyd xrp:info	XRP account balance and outstanding paychans
moneyd xrp:cleanup	close paychans (get the money back from paychans)
moneyd help	list of Moneyd flags
moneyd help <command>	info on a specific command

For reference, we also provide an example of a Moneyd-XRP configuration file, *.moneyd.json*:

```
{
  "version": 1,
  "uplinks": {
    "xrp": {
      "relation": "parent",
      "plugin":
          "/home/user/.nvm/versions/node/v10.15.3/lib/node_modules/moneyd-uplink-xrp
           /node_modules/ilp-plugin-xrp-asym-client/index.js",
      "assetCode": "XRP",
      "assetScale": 9,
      "balance": {
        "minimum": "-Infinity",
        "maximum": "20000000000",
        "settleThreshold": "50000000",
        "settleTo": "1000000"
      },
      "sendRoutes": false,
      "receiveRoutes": false,
      "options": {
        "currencyScale": 6,
```

```
        "server":
            "btp+ws://mduni:85941fd308ac69bfe7a4f6b9726430ea9ee6e6e654bb19b40a419c7a029b6fa7
            @192.168.1.146:7443",
        "secret": "ssVe1jBi2SUU5HQX6YPoTpRdRDG69",
        "address": "rpN3UPjYaErt4RW4gAiqucMCfL5nJJC4Yz",
        "xrpServer": "ws://192.168.1.98:51233"
      }
    }
  }
}
```

Example 6. XRP-ETH ILP payment using Moneyd, SPSP and a connector

We will discuss the configuration presented in Figure 19. 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.

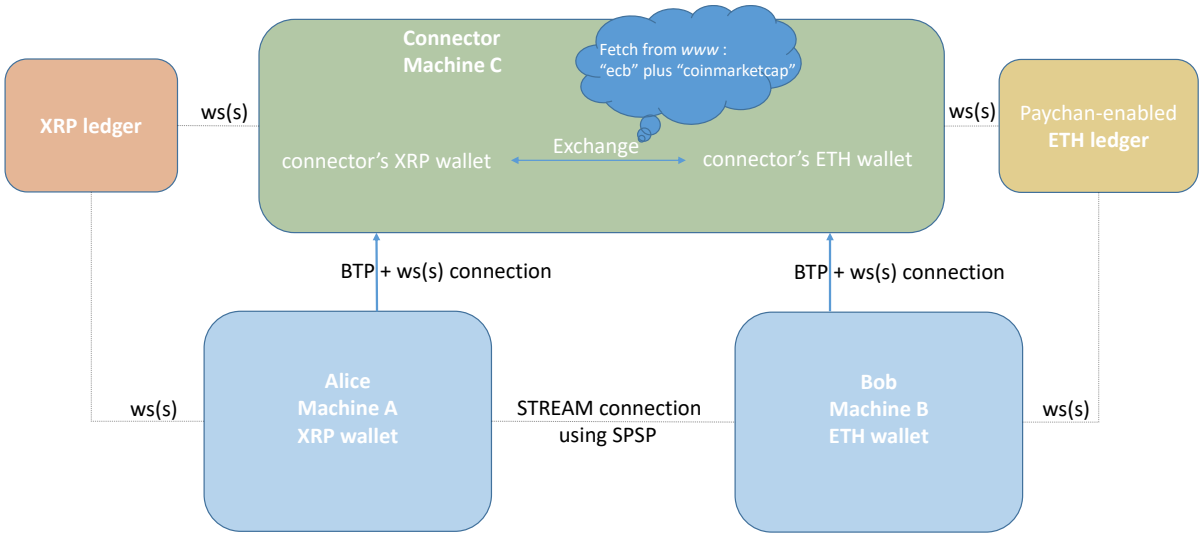

Fig. 19: Example 6: Interledger payment.

A more advanced representation of the same setup is provided in Figure 20 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*.

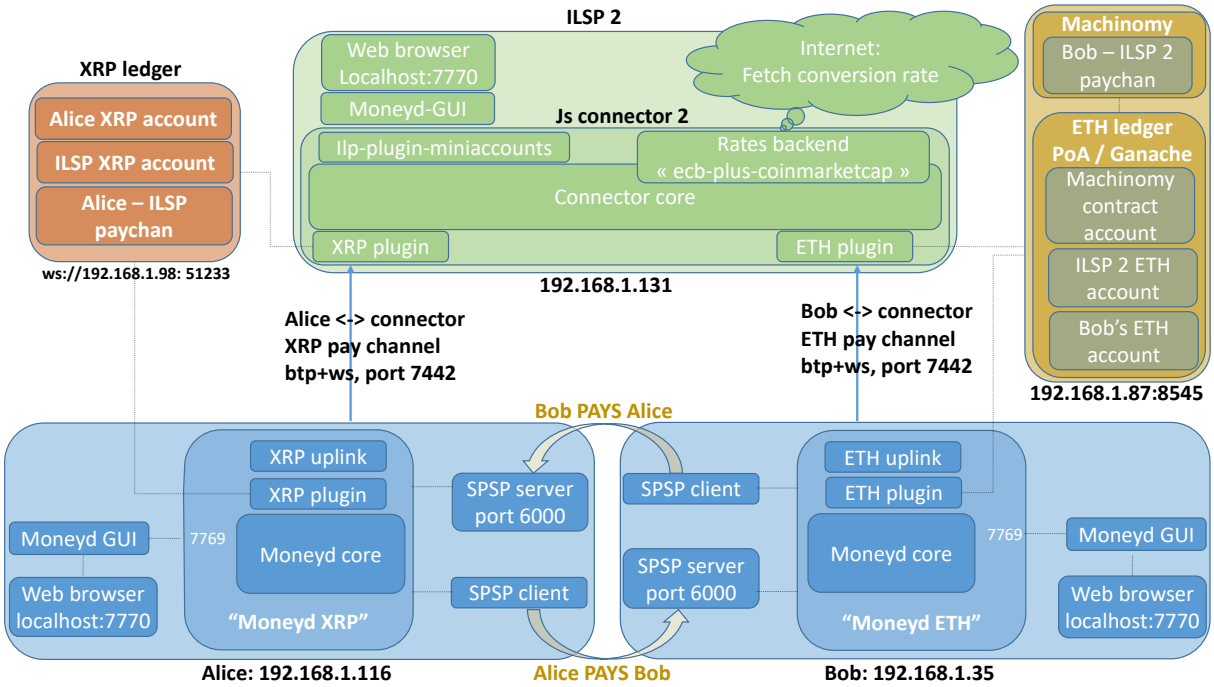

Fig. 20: Example 6: Interledger payment, advanced.

- *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
    - 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.

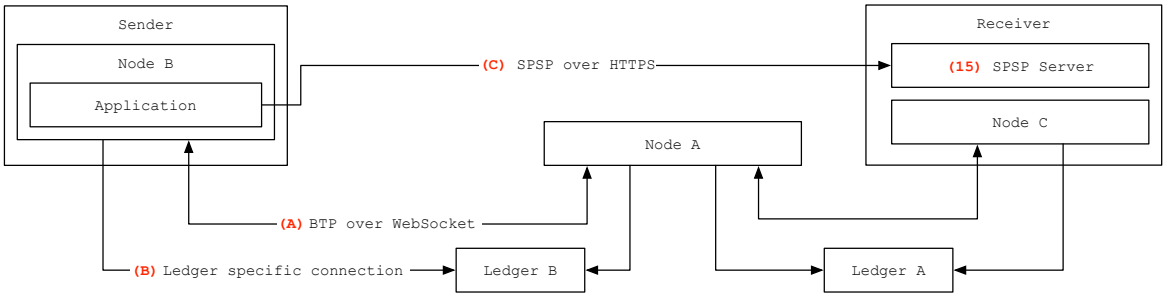

Fig. 21: Perspective: connections. [20]

Into perspective, the situation can be represented as in Figure 21, 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
- 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 [21]. 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"
      }
    }
  }
}
```

5.0.2 @Kava-Labs: Switch API

Switch API[26] 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 [27, 28].

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[27], while for the Ethereum uplink we use a dedicated Ethereum plugin[28].

To handle the ETH settlement, Machinomy contracts have to be deployed on the ETH network, as explained in Section 7.2. We have tested a stream payment between XRP and ETH

[26] https://github.com/Kava-Labs/ilp-sdk/blob/master/README.md, accessed June 2019
[27] https://github.com/Kava-Labs/ilp-plugin-xrp-paychan, accessed June 2019
[28] https://github.com/interledgerjs/ilp-plugin-ethereum, accessed June 2019

using Ganache[29] 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 27, AND
  - in the same file **on the machine running the Switch API** app

```
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: {
```

[29] https://truffleframework.com/ganache, accessed June 2019

```
                '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 [30].

  – 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
```

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

```
        }
```

  - 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 [27]:

```
const { connect } = require('@kava-labs/switch-api')
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[31].

6 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 22. 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 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

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

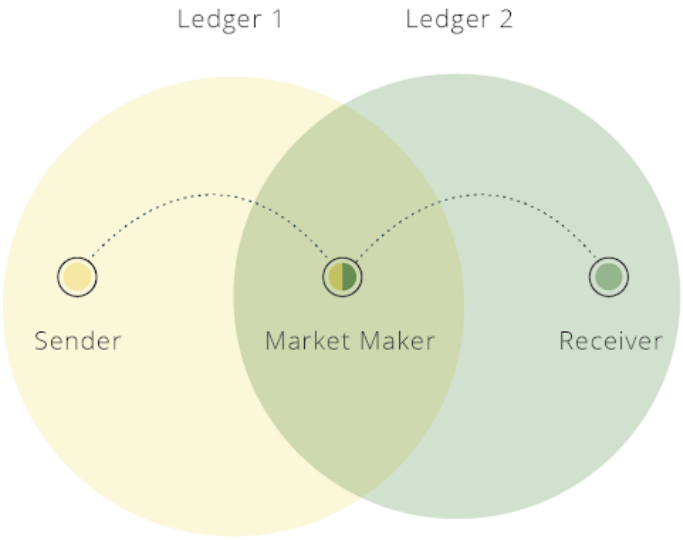

Fig. 22: A connector (money maker) holding two wallets on two different networks.

facilitating the payment between the two ledgers.' [29]

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[32] 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[33] or Switch API[34] are examples of 'customer' apps connecting to their preferred ILSP and sending requests, as shown in Figure 27.

'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.'[25]

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 con-

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

figured according to purpose. This architecture is illustrated in Figure 23.

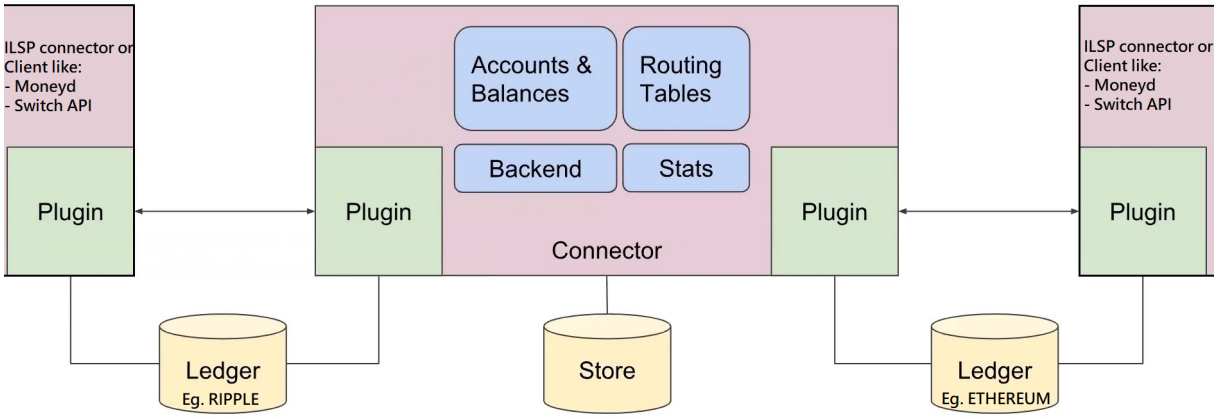

Fig. 23: Architecture overview. [30]

Some plugin examples would be:

- **ilp-plugin-xrp-paychan**[35] 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**[36] 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**[37] 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**[38] makes use of *ilp-plugin-xrp-asym-client*.
- **ilp-plugin-mini-accounts**[39] 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**[40] and **ilp-plugin-ethereum**[41]. 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 payment through the connectors with the use of SPSP, the ILP protocol, some of the plugins above and the servers from Section 7.1 which form the XRP ledger for example, is provided in Figure 24, with some other ledger examples being 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.

[35] https://github.com/interledgerjs/ilp-plugin-xrp-paychan, accessed June 2019
[36] https://github.com/interledgerjs/ilp-plugin-xrp-asym-server, accessed June 2019
[37] https://github.com/interledgerjs/ilp-plugin-xrp-asym-client/, accessed June 2019
[38] https://github.com/interledgerjs/moneyd-uplink-xrp, accessed June 2019
[39] https://github.com/interledgerjs/ilp-plugin-mini-accounts, accessed June 2019
[40] https://github.com/Kava-Labs/ilp-plugin-xrp-paychan, accessed June 2019
[41] https://github.com/interledgerjs/ilp-plugin-ethereum, accessed June 2019

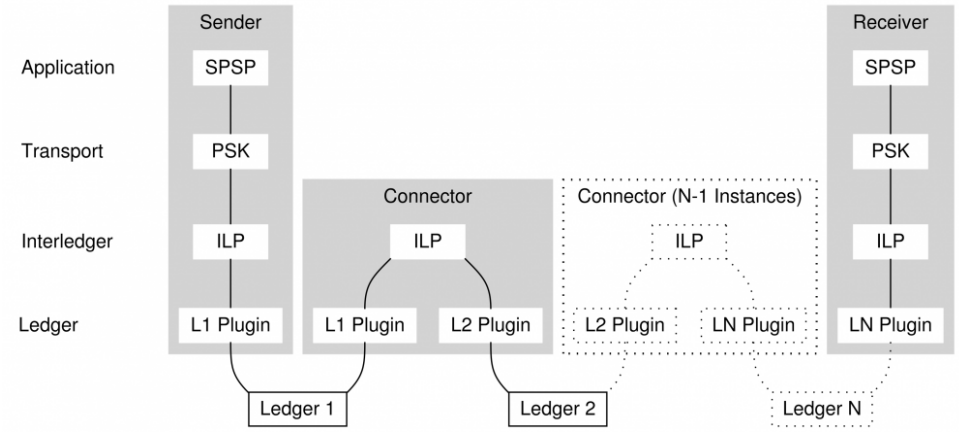

Fig. 24: The protocol stack in the payment chain.

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, 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.'* [31]

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.

Example 7. An advanced XRP-ETH setup using two connectors

Alice, having an XRP account, connects to the ILSP 1 connector and establishes a paychan using the XRP plugin and uplink. Bob, having an Ethereum account, connects to the ILSP 2 Ethreum enabled connector using an Ethereum plugin and uplink. The two connectors are peered over XRP and establish a paychan with the *ilp-plugin-xrp-paychan*. When Alice sends money to Bob, the payment goes through ILSP 1 and ILSP 2. Alice will settle her balance with ILSP 1; ILSP 1 will settle its balance with ILSP 2, and ILSP 2 will settle with Bob.

Because ILSP 1 works exclusively in XRP, its exchange rate will always be 1, so the backend used will be "one-to-one". On the other hand, ILSP 2 works on two ledgers and performs "the currency exchange" so the backend will make use of the "ecb-plus-coinmarketcap" option, and the rate will be fetched from the internet.

On the XRP ledger we find Alice's, ILSP 1 and ILSP 2's accounts, together with the XRP paychans established between the pairs Alice - ILSP1 and ILSP1 - ILSP2. Accordingly, on the ETH network we find ILSP2's and Bob's accounts along with the paychan Bob - ILSP2.

For this configuration, we will assume the ledgers are already up and running on the same Local Area Network hardwired or Wi-Fi. Also, internet access is available. Then, the deployment sequence is:

- Start the connectors, one by one, and wait for them to establish a payment channel between

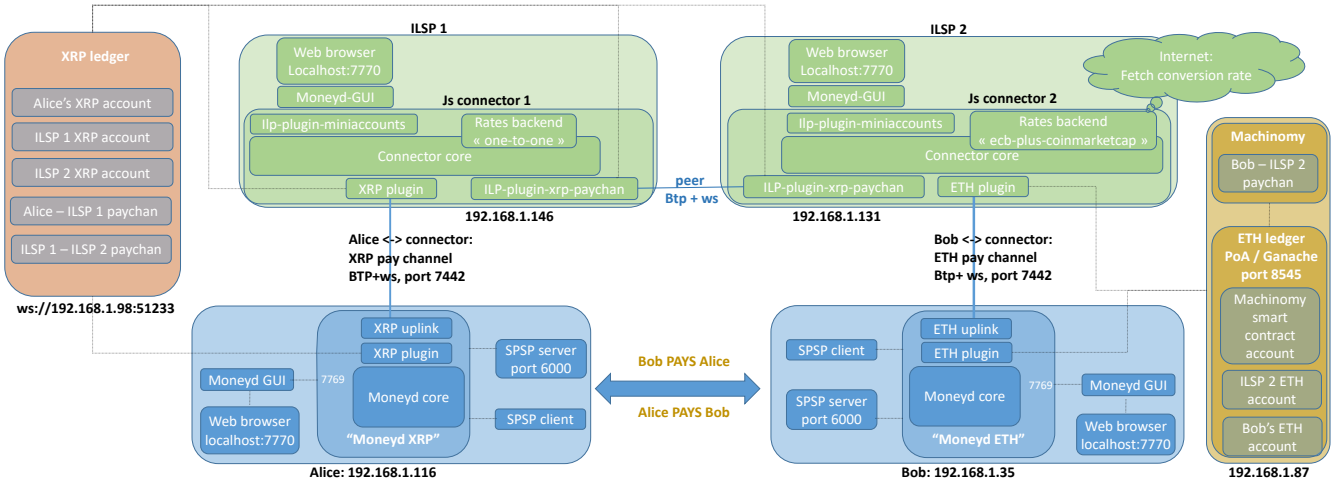

Fig. 25: Example 7: advanced Interledger payment.

them. This will be shown in the connector logs. If using *pm2*, logs can be enabled with *"pm2 logs connector"*.

- Start the Moneyd instances for Alice and Bob and wait for each to establish paychans with the corresponding connector.
- Start Bob's SPSP server.
- From Alice's terminal, use the SPSP client to initiate an SPSP transaction.
- Optionally, Moneyd-GUI and a web browser can be used to administer Moneyd and the connectors.

The diagram presented in Figure 26 further illustrates how the nodes and protocols interact.

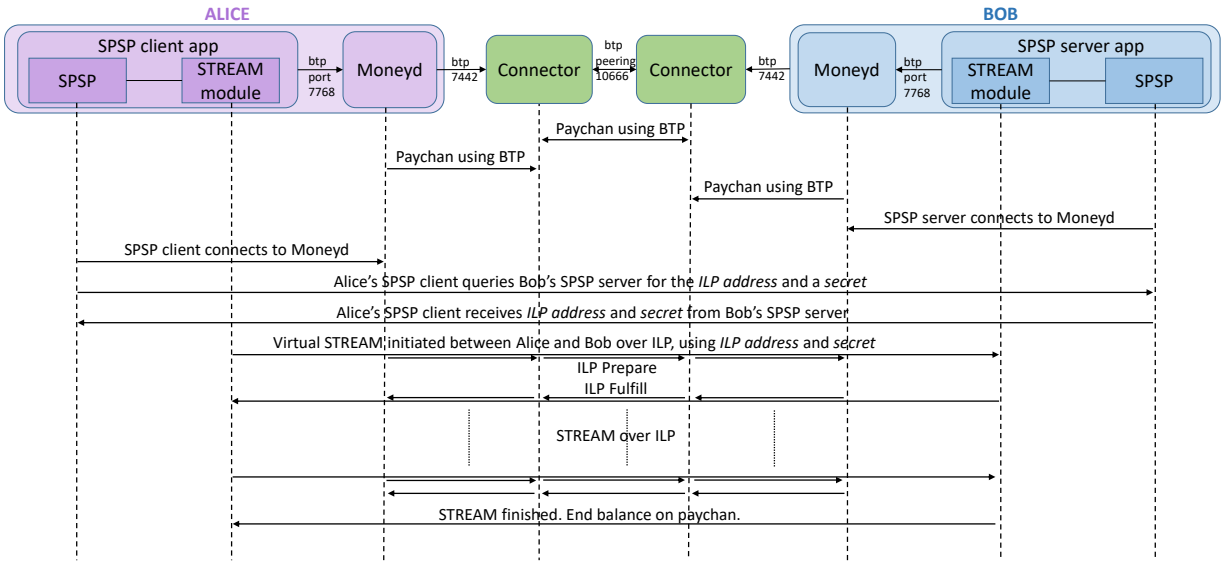

Fig. 26: Example 7: interaction diagram.

The setup for Alice and Bob's machines is similar to Examples 1 and 2.

In regards to installation and set-up of connectors, a new guide[42,43] 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[44] 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"[45]
- 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'.

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

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

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 ILSP1 Connector from Figure 27. 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 in Figure 27, 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 23. 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. More on SPSP in Section 5.0.1.

```
'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 Section 5.0.1.

Other connectors, functional but still in development, are the *Rafiki*[46] connector and the *Rust*[47] connector. A basis for a Java connector is also in the works, as *Quilt*[48].

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 [32].

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[49]. 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

[46] https://github.com/interledgerjs/rafiki, accessed June 2019
[47] https://github.com/emschwartz/interledger-rs/tree/master, accessed June 2019
[48] https://www.hyperledger.org/projects/quilt, accessed July 2019
[49] https://interledger.org/rfcs/0038-settlement-engines/, accessed January 2020

configuration need to be provided[50].

7 The ledgers

7.1 The Ripple ledger

The Rippled XRP ledger is made up of two types of servers: *"trackers"* (or *stock servers*) and *"validators"*. They run the same piece of software [33], just with a different configuration. Although they can answer user queries, validators should ideally just process the transactions they receive from the trackers.

'Ideally, validating nodes are clustered with at least two stock nodes, to prevent DoS attacks and to preserve availability while updating the stock nodes. This configuration enables the validating node to be cut off from the internet, except for messages to/from other trusted nodes in the cluster and SSH connections via a LAN connection. Using two stock nodes provides redundant communication to the validating node, which is useful in case one of the stock nodes crashes or goes offline. However, this means a validating node has 3x the cost, 3x the monitoring, and 3x the time commitment of a stock node. Production validating nodes should have at least 32 GB of memory as well as a 50 GB+ solid state drive. I encourage operator to refrain from making API calls (monitoring excepted) on validating nodes'. [34]

Validators participate in the consensus process and vote on fees and amendments. Trackers are meant to be placed in-between validators and the rest of the network, pick-up traffic and forward it to the validators. They work as relays, protecting the validators. They also can hold the full history of the ledger and answer queries about old ledgers. On the other hand, the validators can work with minimal stored history.

Below is the procedure to build and cold-start an independent local validators cluster. One aspect to keep in mind is that *'there is no rippled setting that defines which network it uses. Instead, it uses the consensus of validators it trusts to know which ledger to accept as the truth. When different consensus groups of rippled instances only trust other members of the same group, each group continues as a parallel network. Even if malicious or misbehaving computers connect to both networks, the consensus process overrides the confusion as long as the members of each network are not configured to trust members of another network in excess of their quorum settings.'* [35]

7.1.1 Preparation

To build a parallel Rippled servers (validators, trackers) network, which, in its entirety is also called the *"Rippled Ledger"*, the minimal required hardware resources[51] need to be planned in advance. Depending on the available resources, each Rippled server can be deployed on a different physical machine or not. However, for high traffic use-cases, in order to streamline I/O, each server could have its own physical SSD.

To install the pre-packaged Rippled server, the instructions on the Ripple developer portal should be followed[52]. Then, another guide is disseminated on the developer portal to install

[50] https://github.com/interledger-rs/interledger-rs/tree/master/examples, accessed January 2020
[51] https://developers.ripple.com/system-requirements.html, accessed June 2019
[52] https://developers.ripple.com/install-rippled.html, accessed June 2019

Rippled from the source code[53]. After that, the following steps must be followed:

- Build the validators keys and tokens, using the published documentation[54] and code[55].
  - Create keys:

```
~/validator-keys-tool/build/gcc.debug$ ./validator-keys create_keys
```

  - Create tokens:

```
~/validator-keys-tool/build/gcc.debug$ ./validator-keys create_token --keyfile
    /home/user/.ripple/validator-keys.json
```

- Add generated [*validator_token*] to 'rippled.cfg'.
- Add generated [*validators*] public keys to 'validators.txt'. Comment the rest of 'validators.txt'.
- In 'rippled.cfg', add the peer validators' IPs in the field [ips_fixed] in the form of IP:port (51235).
- Check that 'validator.txt' file name is the same with the name referenced by 'rippled.cfg'.
- Configure clustering as per the Ripple documentation[56], using the validation_create[57] method:

```
~/rippled/ccabuild$ ./rippled --conf /home/user/rippled/cfg/rippled-example.cfg
    validation_create
```

7.1.2 Start up

In the case when Docker images have been used, after creating the Docker image of the Rippled server, this can be loaded and started on each physical/virtual server machine with the following:

```
- 'sudo docker load -i /path/to/your_docker_image.tar'
- 'sudo docker images' - to check the image name
- 'sudo docker run -ti -u root --network host --name <container_name> <image_name>'
- 'sudo docker exec -ti -u root <container_name> bash'. To open a second terminal
    to the container, just run the command again into a fresh terminal window.
```

With the docker images loaded on each Rippled server machine, the actual Rippled validators servers network can be cold-started as follows:

- Start the first Rippled server with 'quorum 1' and wait a few minutes for it to stabilize:

```
./rippled --conf /home/user/rippled/cfg/rippled-example.cfg --quorum 1
```

- Start the remaining servers with the same command, waiting for each to stabilize, first.
- Restart the servers in the same order, waiting a few minutes for each to stabilize before starting the next, with 'quorum 2':

[53] https://developers.ripple.com/build-run-rippled-ubuntu.html, accessed June 2019
[54] https://developers.ripple.com/run-rippled-as-a-validator.html#enable-validation-on-your-rippled-server, accessed June 2019
[55] https://github.com/ripple/validator-keys-tool, accessed June 2019
[56] https://developers.ripple.com/cluster-rippled-servers.html, accessed June 2019
[57] https://developers.ripple.com/validation_create.html, accessed June 2019

```
./rippled --conf /home/user/rippled/cfg/rippled-example.cfg --quorum 2
```

In a new terminal window handling the Docker container, use the "stop" command in Table 4 to gracefully stop the servers before restart.

In this minimal set-up though, if any of the servers is restarted, it will lose previously kept ledger history - even with full history enabled. This won't stop it working after restart, as validators do not need full history to work properly. To be able to access previous ledger history, tracking servers should be also set up. Data API[58] is a useful history tool which could also be set-up if desired, although setting it up on a private network seems not too obvious.

'Ripple API'[59] provides the means to interact with the server. For example, in Ripple, all the money are created in the beginning, and stored in an account with a hard-coded address, called the "Genesis account'. One can check the 'Genesis account' with:

```
./rippled --conf /home/user/rippled/cfg/rippled-example.cfg account_info
    rHb9CJAWyB4rj91VRWn96DkukG4bwdtyTh
```

Some other useful server commands are provided in Table 4. These can be entered from a separate terminal window handling the docker container.

Table 4: Useful Rippled server commands.

command	effect
wallet_propose	create a new wallet with random seed credentials (inactive until funded)
stop	gracefully stop the server
restart	restart the server
server_info	various easy-to-read info about the server
server_state	almost same info as above, but easier-to-process instead of easy-to-read
peers	info on peer validators: connected? ledger sequences available? ...

Immediately after creating and starting the validators cluster network (which form the XRP ledger), one can open a few accounts with the 'wallet_propose' command above, and fund them using for example the following simple procedure. Regarding the wallets, they can be sometimes classified as 'hot' or 'cold' wallets. The difference is that 'hot' wallets are connected to the internet, while 'cold' wallets are not. 'Hot' wallets provide the advantage of quick access but lower security, like anything connected to the internet. 'Cold' wallets are slower to access (need to connect) but more secure due to generally not being online. It is generally recommended to hold only the amounts necessary for daily operation in the 'hot' wallet, while the bulk of the money would be kept offline.

- Install Ripple-API for javascript[60].
- Place the two example scripts in the app folder: *'/home/user/ripple_api/get-account-info.js'*.
- Run them with *'./node_modules/.bin/babel-node get-account-info.js'*. The code should run on one of the Ripple servers.

Example script - get account info:

[58] https://developers.ripple.com/data-api.html, accessed June 2019
[59] https://developers.ripple.com/rippleapi-reference.html, accessed June 2019
[60] https://developers.ripple.com/get-started-with-rippleapi-for-javascript.html, accessed June 2019

```
//GET ACCOUNT INFO
'use strict';
const RippleAPI = require('ripple-lib').RippleAPI;

const api = new RippleAPI({
  server: 'ws://localhost:6006'
});
api.connect().then(() => {
  /* begin custom code ------------------------------------ */
  const testAddress = 'rHb9CJAWyB4rj91VRWn96DkukG4bwdtyTh';

  console.log('getting account info for', testAddress);
  return api.getAccountInfo(testAddress);

}).then(info => {
  console.log(info);
  console.log('getAccountInfo done');

  /* end custom code -------------------------------------- */
}).then(() => {
  return api.disconnect();
}).then(() => {
  console.log('done and disconnected.');
}).catch(console.error);
```

Example script - fund an account:

```
 //Account funding
 const RippleAPI = require('ripple-lib').RippleAPI

// SENDER - ADDRESS 1
const ADDRESS_1 = "rHb9CJAWyB4rj91VRWn96DkukG4bwdtyTh"
const SECRET_1 = "snoPBrXtMeMyMHUVTgbuqAfg1SUTb"

// RECEIVER - ADDRESS 2
const ADDRESS_2 = "rMqUT7uGs6Sz1m9vFr7o85XJ3WDAvgzWmj"

const instructions = {maxLedgerVersionOffset: 5}
const currency = 'XRP'
const amount = '20000000' // this is not 'drops' but XRP

const payment = {
  source: {
    address: ADDRESS_1,
    maxAmount: {
      value: amount,
      currency: currency
    }
  },
  destination: {
    address: ADDRESS_2,
    amount: {
      value: amount,
```

```
      currency: currency
    }
  }
}

const api = new RippleAPI({
  //server: 'wss://s1.ripple.com'                 //MAINNET
  //server: 'wss://s.altnet.rippletest.net:51233' // TESTNET
  server: 'ws://localhost:6006'                   // Localhost
})

api.connect().then(() => {
  console.log('Connected...')
  api.preparePayment(ADDRESS_1, payment, instructions).then(prepared => {
    const {signedTransaction, id} = api.sign(prepared.txJSON, SECRET_1)
    console.log(id)
    api.submit(signedTransaction).then(result => {
      console.log(JSON.stringify(result, null, 2))
      api.disconnect()
    })
  })
}).catch(console.error)
```

The known amendments seem not to be automatically enabled after cold starting a private network. In order to force them, we added the *[features]* stanza in each validator's config file. Otherwise, the validators would apparently work, but, when trying to open for example a paychan, would throw the error "logic not enabled' - because the paychan amendment is not enabled. According to documentation[61], for an amendment to become enabled, it needs the support of 80% of validators' votes for two weeks. If it loses this support, the amendment is temporarily disabled, and it can be re-enabled after it re-gains this support.

[features]
PayChan
Escrow
CryptoConditions
fix1528
.....

Below is an example config file for a private network cluster of 3 validators. The file is located in *'home/user/rippled/cfg'*. We used a docker container with a compiled version of Rippled.

```
[server]
port_rpc_admin_local
port_peer
port_ws_admin_local
port_ws_public
port_public

[port_rpc_admin_local]
```

[61] https://developers.ripple.com/amendments.html, accessed June 2019

```
port = 5005
ip = 127.0.0.1
admin = 127.0.0.1
protocol = http

[port_peer] //talk to other validators
port = 51235
ip = 0.0.0.0
protocol = peer

[port_ws_admin_local]
port = 6006
ip = 127.0.0.1
admin = 127.0.0.1
protocol = ws

[port_ws_public]
port = 6005
ip = 127.0.0.1
protocol = wss

[port_public] //connectors, moneyd, switch API will connect here
ip = 0.0.0.0
port = 51233
protocol = ws

[node_size] //required for full history
huge

# This is primary persistent datastore for Rippled. This includes transaction
# metadata, account states, and ledger headers. Helpful information can be
# found here: https://ripple.com/wiki/NodeBackEnd
# delete old ledgers while maintaining at least 2000. Do not require an
# external administrative command to initiate deletion.
[node_db]  //NuDB type required for full history
type=NuDB
path=/var/lib/rippled/db/nudb
#open_files=2000 //these are not needed for NuDB
#filter_bits=12
#cache_mb=256
#file_size_mb=8
#file_size_mult=2
#online_delete=2000
#advisory_delete=0

[ledger_history] //although enabled, full history seems not to work
                 //correctly for validators, will need trackers for this.
full

[database_path]
/var/lib/rippled/db

# This needs to be an absolute directory reference, not a relative one.
# Modify this value as required.
```

```
[debug_logfile]
/var/log/rippled/debug.log

[sntp_servers] // servers for time sync
time.windows.com
time.apple.com
time.nist.gov
pool.ntp.org

# File containing trusted validator keys or validator list publishers.
# Unless an absolute path is specified, it will be considered relative to the
# folder in which the rippled.cfg file is located.
[validators_file]
validators-example.txt

# 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": "trace" } //verbose logging

# If ssl_verify is 1, certificates will be validated.
# To allow the use of self-signed certificates for development or internal use,
# set to ssl_verify to 0.
[ssl_verify]
0

[ips_fixed]
192.168.1.97 51235 //IPs and ports of the other 2 peer validators
192.168.1.132 51235

[peer_private]
1

[node_seed]
shEm9dGAs2aq6MMe9XsXYXKrPmqft

[cluster_nodes]
n9LPJFoTLxVbTtdWADZzPpCwACwC3aLAYGhFcNNR61fD9DTc2w5L ripdbg1
n9KUMms9ZrDgHU7rN9pRTRGMKEWy5Ghk3qj53aCPAbJRur2sTqwp ripdbg3

[validator-token]
eyJtYW5pZmVzdCI6IkpBQUFBQUZ4SWUwYkVlUVp1bGNsKzRadk44cGhXUWJNNWhlV3RKYOhN
YUVKcUpadWVRWm9jWE1oQXYvVWY3MmlaQOVQZndPZTdOTjNaYOV1UnFDd2Q3U2JkU3hPTnJq
TXlsNWlka2N3UlFJaEFKc3IzL3g2UORiRGprOHc0Mks2eU91M1FPbW4vNjVIeTM4bkxjbnJa
c1ROQWlBSnRlRTRpdjVqSjRJMytvSOVseEFjTmFUL3VoQnRlSVFyK29RdmVoemJESEFTUU53
RnpLN21kV3lUaTZoTWY4SUJTRUxmZHI1cjhuMFdIeE5BSGNHSXJURDV1NO9BK3FKZWZLMzkw
Smx3aE5ydGVLLO9LWS8rQldDUHoOejQ4VXptaHd3PSIsInZhbGlkYXRpb25fc2VjcmVOX2tl
eSI6IkJGMTcyRjJBMzNGQTZDOTdBQOJBODhBNTAONThGQzZFRURENzBCNjEwMzdEMjcwNjgz
RTQ3MzRBNUY2OURGRkMifQ==

[features]
PayChan
Escrow
CryptoConditions
```

```
fix1528
DepositPreauth
FeeEscalation
fix1373
MultiSign
TickSize
fix1623
fix1515
TrustSetAuth
fix1513
fix1512
fix1571
Flow
fix1201
fix1523
fix1543
SortedDirectories
EnforceInvariants
fix1368
DepositAuth
fix1578
```

7.2 The ETH ledger. Connecting the XRP and ETH ledgers through 'Machinomy'

For the scope of this work, we will assimilate the ETH network to a black-box holding the ETH wallet accounts, executing commands and providing immediate response. For testing purposes, such a friendly 'black-box' can be 'Ganache'[62], previously called 'TestRPC'. After download, Ganache can be started directly:

```
cd /Downloads
./ganache-1.3.1-x86_64.AppImage
```

'Machinomy'[63] is used to connect the XRP and ETH ledgers. It achieves this by deploying a specific contract on the ETH ledger. One contract manages all the channels for Ether micropayments (all the sender-receiver pairs). Thus, Machinomy creates the settlement capability when ILP payment interacts with the ETH ledger.

'Machinomy is a micropayments SDK for Ethereum platform. State channels is a design pattern for instant blockchain transactions. It moves most of the transactions off-chain. As transactions do not touch the blockchain, fees and waiting times are eliminated, in a secure way.' [36]

Machinomy should be installed[64] on the same machine with the ETH provider, in this case, Ganache. After installing Machinomy, a contract can be deployed on the ETH network using the following:

```
    cd machinomy/node_modules/@machinomy/contracts
    yarn truffle migrate --reset
```

[62] https://truffleframework.com/docs/ganache/quickstart, accessed June 2019, accessed June 2019
[63] https://machinomy.com/, accessed June 2019
[64] https://github.com/machinomy/machinomy, accessed June 2019

Checking back in Ganache after Machinomy contract deployment, you will notice that a small amount of ETH has been subtracted from the first account, and in the *Transactions* tab, the contract has been deployed.

After a Ganache restart, the *'–reset'* option has to be used because Ganache is not persistent. The contract will be deployed on the first Ganache account. The other accounts can be used by ETH client wallets.

After deploying the Machinomy contract on the ETH network, apps like Switch API can be used to exchange XRP and ETH back and forth. The plugins should be set to access Ganache using *http://ganache_IP:ganache_port*. A detailed explanation on Switch API is provided in Section 5.0.2.

8 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.

9 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 supported by the Luxembourg National Research Fund (FNR) through grant PRIDE15/10621687/SPsquared. In addition, we thankfully acknowledge the support from the RIPPLE University Blockchain Research Initiative (UBRI) framework for our research.

Acronyms

API Abstract Programming Interface. 4, 8, 11, 37–43, 52, 54, 60, 61

BTP Bilateral Transfer Protocol. 11, 21–24, 27–32, 36

FSM Finite State Machine. 2, 17

GUI Graphical User Interface. 13, 25, 28, 32, 33, 35, 43, 45, 47, 49

ILP Interledger Protocol. 4–12, 14–16, 18, 19, 21, 22, 24–30, 32, 34, 42, 43, 46, 59, 60

ILSP Interledger Service Provider. 8, 11, 28, 36, 38, 42–44, 47, 49

SPSP Simple Payment Setup Protocol. 11–15, 19, 21, 22, 24–30, 32–36, 43, 45, 49, 51, 60

Glossary

Moneyd An ILP provider, allowing all applications on an end-user computer to use funds on the live ILP network. 4, 8, 10, 13, 15, 21, 24–30, 32–37, 42, 43, 45, 60

Proof of Work (PoW) A consensus protocol introduced by Bitcoin, known as mining, which involves answering to a mathematical problem that requires considerable work to solve, but is easily verified once given the answer. 6

Switch API A SDK for cross-chain trading between BTC, ETH, DAI and XRP with Interledger Streaming. 4, 8, 37–43, 60

XRP Ripple's digital payment asset which is used for Interledger payments. 3–7, 9, 10, 13, 19, 24, 25, 27–29, 32–38, 40, 41, 43, 44, 46, 49, 54, 59, 60

References

[1] Dr. Demetrios Zamboglou. *ripple |Explained*, [Online] Accessed: June 6, 2019. https://medium.com/datadriveninvestor/ripple-explained-4df46678e0bd.

[2] Ripple. *ILP Connector*, [Online] Accessed: June 6, 2019. https://github.com/interledgerjs/ilp-connector#what-is-this.

[3] Ripple. *ILP Addresses - v2.0.0*, [Online] Accessed: June 21, 2019. https://interledger.org/rfcs/0015-ilp-addresses/.

[4] Adrian Hope-Bailie. *Running your own ILP connector*, [Online] Accessed: April 10, 2019. https://medium.com/interledger-blog/running-your-own-ilp-connector-c296a6dcf39a.

[5] Ripple. *Peering, Clearing and Settling*, [Online] Accessed: June 18, 2019. https://interledger.org/rfcs/0032-peering-clearing-settlement/.

[6] Ripple. *Payment Channels*, [Online] Accessed: June 17, 2019. https://xrpl.org/payment-channels.html.

[7] Adrian-Hope Bailie. *Settlement Architecture*, [Online] Accessed: June 18, 2019. https://forum.interledger.org/t/settlement-architecture/545.

[8] Ben Sharafian. *Moneyd - payment channels explanation*, [Online] Accessed: June 20, 2019. https://forum.interledger.org/t/moneyd-payment-channels-explanation/374/3.

[9] 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.

[10] Evan Schwartz. *Thoughts on Scaling Interledger Connectors*, [Online] Accessed: June 14, 2019. https://medium.com/interledger-blog/thoughts-on-scaling-interledger-connectors-7e3cad0dab7f.

[11] Ripple. *Simple Payment Setup Protocol (SPSP)*, [Online] Accessed: June 6, 2019. https://github.com/interledger/rfcs/blob/master/0009-simple-payment-setup-protocol/0009-simple-payment-setup-protocol.md.

[12] Ripple. *Payment Pointers and Payment Setup Protocols* , [Online] Accessed: June 14, 2019. https://github.com/interledger/rfcs/blob/master/0026-payment-pointers/0026-payment-pointers.md.

[13] 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.

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

[15] Ripple. *Ripple InterLedger Protocol's role in realizing the Internet of Value [IoV]*, [Online] Accessed: June 11, 2019. https://bcfocus.com/news/ripple-interledger-protocols-role-in-realizing-the-internet-of-value-iov/19033/.

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

[17] D. Appelt, A. Hope-Bailie, M. de Jong, E. Schwartz, B. Sharafian, S. Thomas, and B. Way. *ILP v4: Version 4 of the Interledger protocol, April 2018*, [Online] Accessed: June 13, 2019. https://github.com/interledger/rfcs/blob/2052575084cdeeb94c0e9bd2e3c37960b732fa2d/whitepaper/interledger.pdf.

[18] Evan Schwartz. *Trustlines Explanation*, [Online] Accessed: June 13, 2019. https://forum.interledger.org/t/trustlines-explanation/358.

[19] Ripple. *Hashed-Timelock Agreements (HTLAs)*, [Online] Accessed: June 11, 2019. https://interledger.org/rfcs/0022-hashed-timelock-agreements/#simple-payment-channels.

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

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

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

[23] Ripple. *Interledger Check-in*, [Online] Accessed: April 15, 2019. https://developers.ripple.com/blog/2019/interledger-checkin.html.

[24] Sabine Bertram. *Introducing Pull Payments to the Interledger Protocol*, [Online] Accessed: June 6, 2019. https://medium.com/interledger-blog/introducing-pull-payments-to-the-interledger-protocol-d763e21af6ec.

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

[26] Ben Sharafian. *SPSP payments*, [Online] Accessed: June 6, 2019. https://forum.interledger.org/t/spsp-payments/310/7.

[27] 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.

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

[29] 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.

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

[31] 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.

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

[33] Ripple. *Install Rippled*, [Online] Accessed: April 10, 2019. https://developers.ripple.com/install-rippled.html.

[34] Rabbit. *The Interledger Protocol*, July. 2018. [Online] Accessed: April 10, 2019. https://xrpcommunity.blog/rippled/.

[35] Ripple. *Parallel Networks and Consensus*, [Online] Accessed: April 10, 2019. https://mduo13.github.io/ripple-dev-portal/tutorial-rippled-setup.html.

[36] Sergey Ukustov, Andrei Riaskov, Alexander Burtovoy, and Matthew Slipper. *Machinomy*, [Online] Accessed: April 10, 2019. https://machinomy.com/.

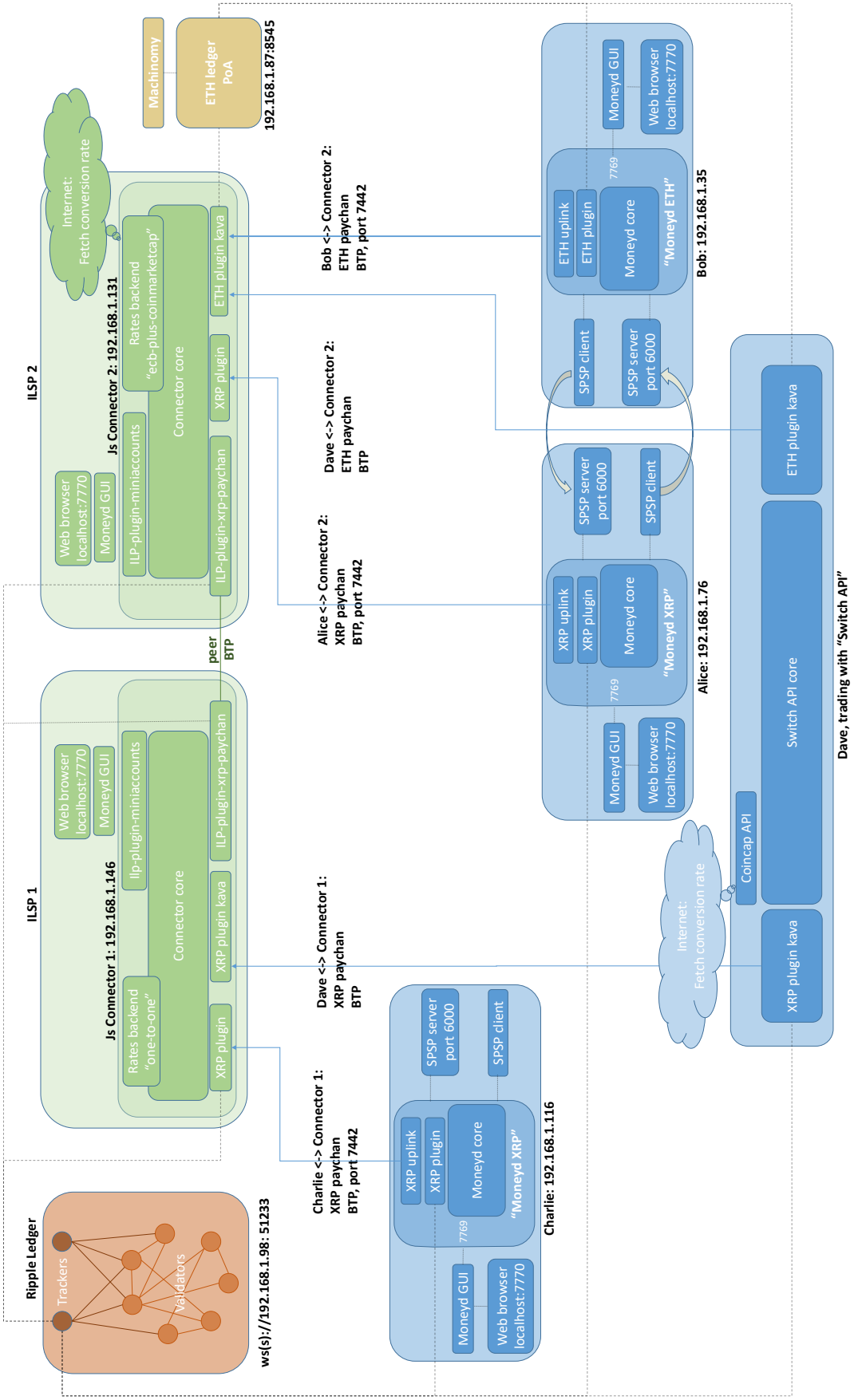

Fig. 27: Ecosystem overview. The machines involved are time-synchronized using time servers. The ETH gas price and the currency rates are fetched from online. To keep the diagram readable we didn't illustrate all plugin connections to the ledgers; each plugin provides for connection to the appropriate ledger using wss or ws.

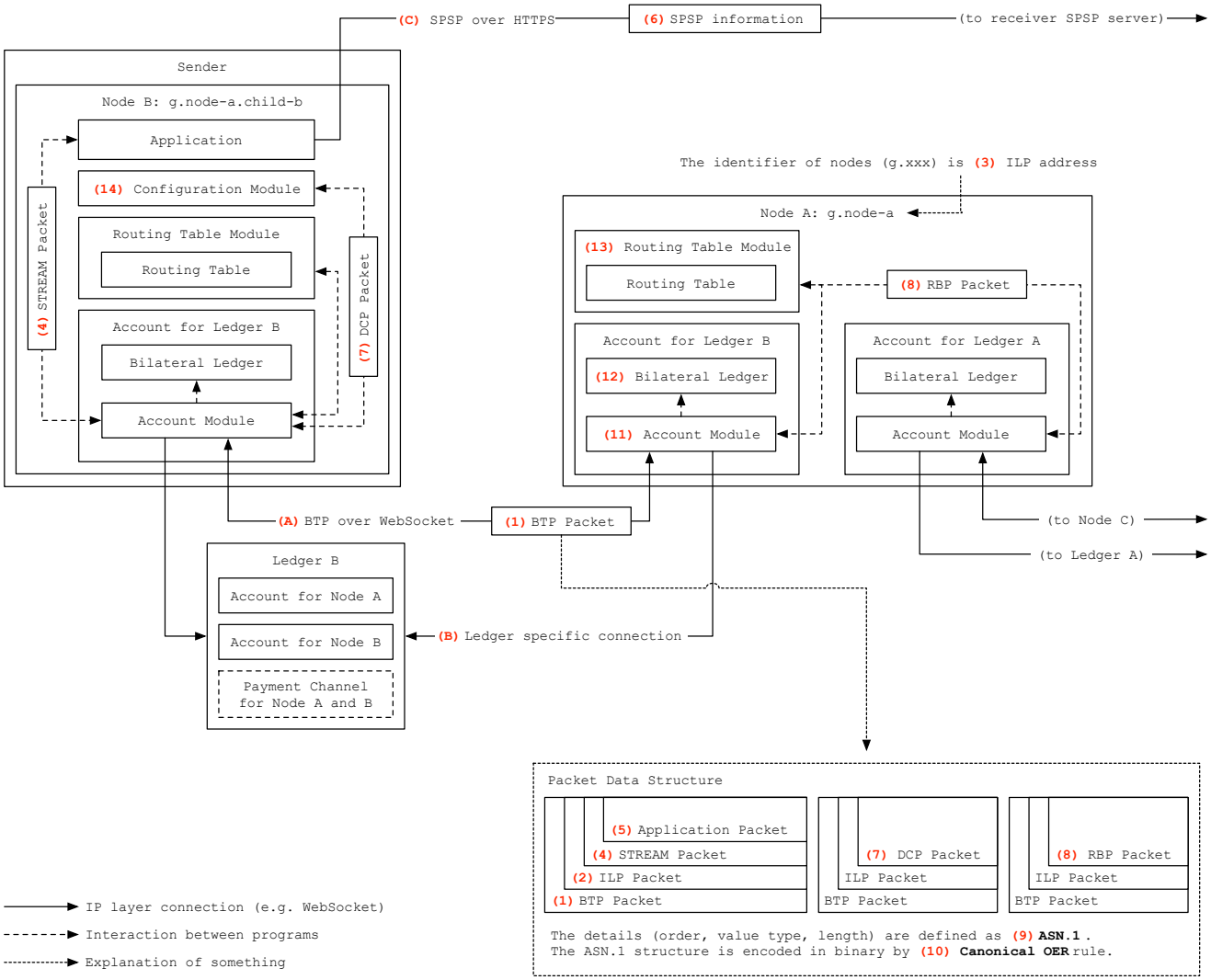

Fig. 28: Protocols and details. Advanced diagram. [20, 21]