Deep dive into Interledger: Understanding the Interledger ecosystem – Part 1 –

Lucian Trestioreanu, Cyril Cassagnes, and Radu State

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

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

Contents

List of Figures

List of Tables

1 What this document covers

The scope of these series 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. Our main goal 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.

To start our 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 the Interledger ecosystem. Consequently, tutorials and various other resources can become outdated after a few weeks.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 another distributed ledger.

The series is organized as follows:

In *Part 1* we are going to make an introduction of the Interledger ecosystem, the main concepts and components. We are going to present the Simple Payment Setup Protocol and the customer apps for sending and receiving payments, together with some examples.

In the next *Parts*, we are going to see some other important Interledger protocols like the *Streaming Transport for the Realtime Exchange of Assets and Messages* (STREAM), the *Interledger Protocol* (ILP) which is the core protocol of Interledger and the *Bilateral Transfer Protocol* (BTP), and also important infrastructure components like the Connectors and Ledgers. These are going to be practically illustrated with examples.

2 Who this document is for

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

3 The Interledger ecosystem

RippleNet[1] 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.

*"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.

[1]https://ripple.com/ripplenet/, accessed July 2019

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 currencies 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.

3.1 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.

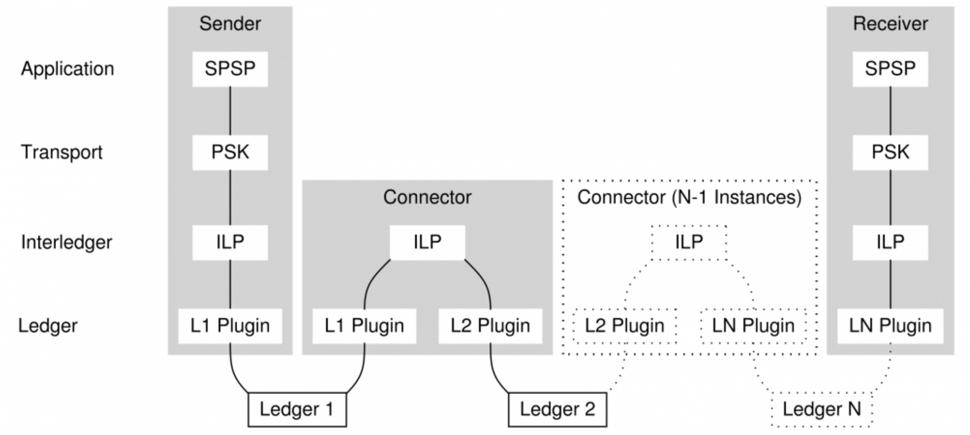

Fig. 1: Payment chain.

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 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.

3.2 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.

3.2.1 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.

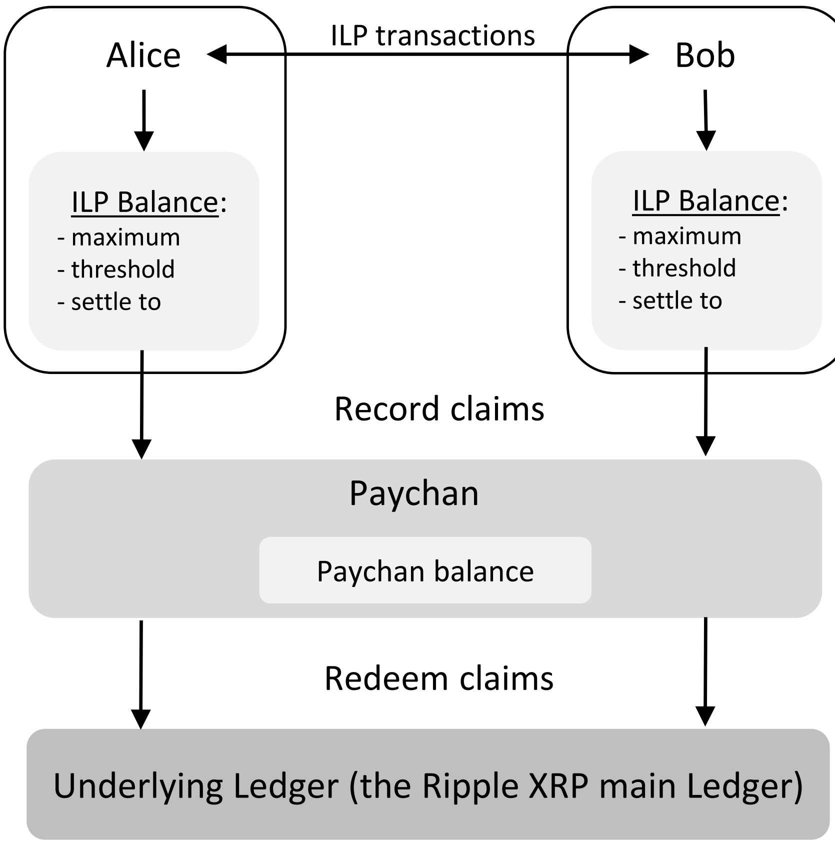

Fig. 2: The money transfer system.

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.

3.2.2 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[2].

Payment channels are opened only between direct peers. *Settlement*, presented in Section 3.2.3, 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 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

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

payment channel each claim is recorded individually, but they can be later redeemed individually or in bulk on the ledger [6].

3.2.3 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.

3.2.4 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.

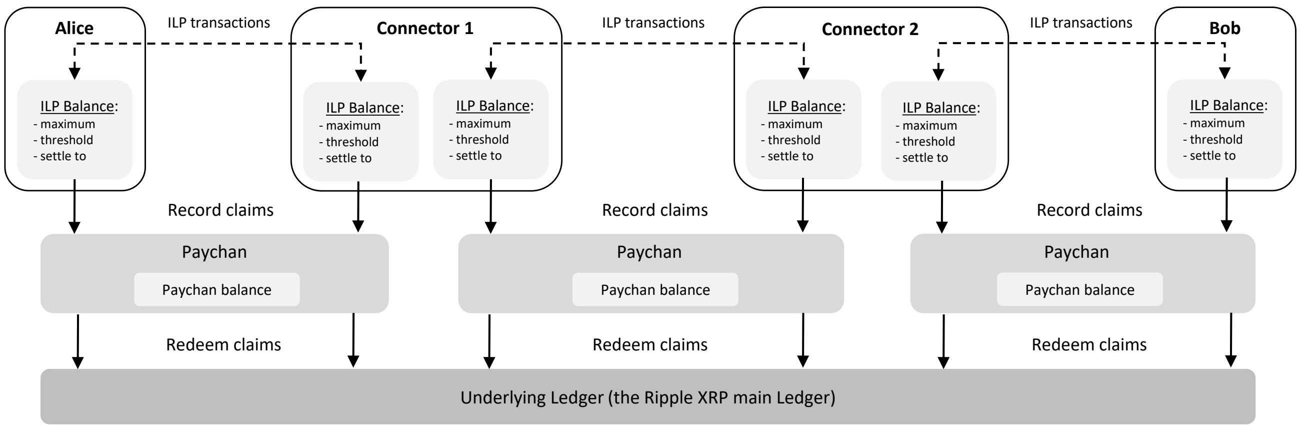

Fig. 3: The money transfer system in practice.

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.

3.3 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.

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

In this series examples, we are going to make use of BTP, ILP, STREAM and SPSP, a protocol suite also depicted in Figure 4.

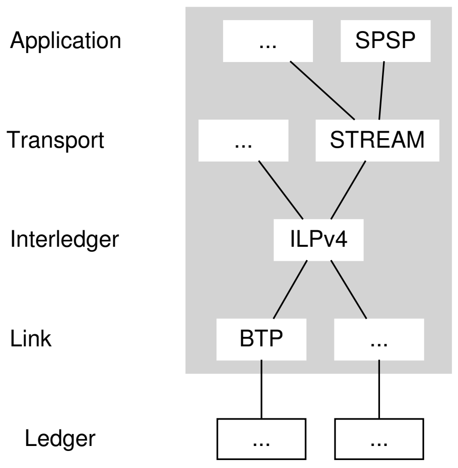

Fig. 4: The protocol suite. [10]

3.3.1 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 [11]. 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* [12]. 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 [13]. *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"* [13].

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"* [13]. Table 2 also explains how a Payment pointer is resolved to an SPSP endpoint.

Table 2: Payment pointer to Endpoint conversion. [13]

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.

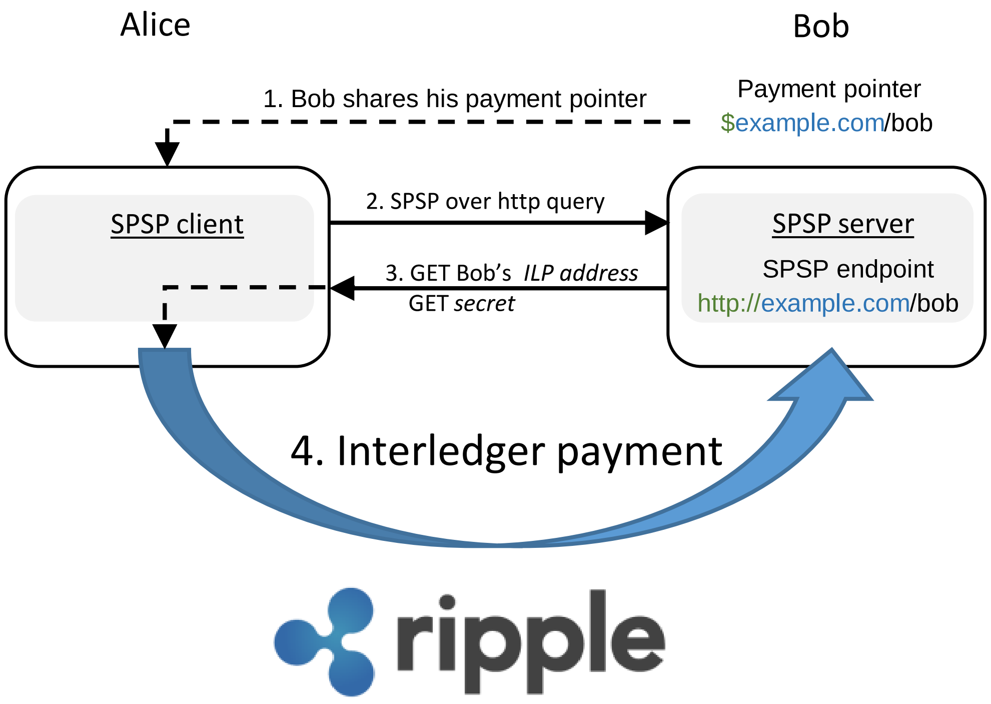

Fig. 5: Example 1: SPSP payment.

Example 1. Concerning the Figure 5:

- 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, $example.com/bob*, which he easily shares to Alice verbally. This information is easy for Alice to remember and use. (step (1) in Figure 5)
- Supposing the machine already set-up, at her computer, Alice starts up Moneyd, and is able to pay Bob just by introducing his *payment pointer address "$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:

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* [11].

4 Customer apps for money transfer

4.0.1 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.'* [14]

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[3], 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* [4]. Running on the same machine with Moneyd or with the js reference Connector, it can be installed with[5]:

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

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 6. 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 [15].

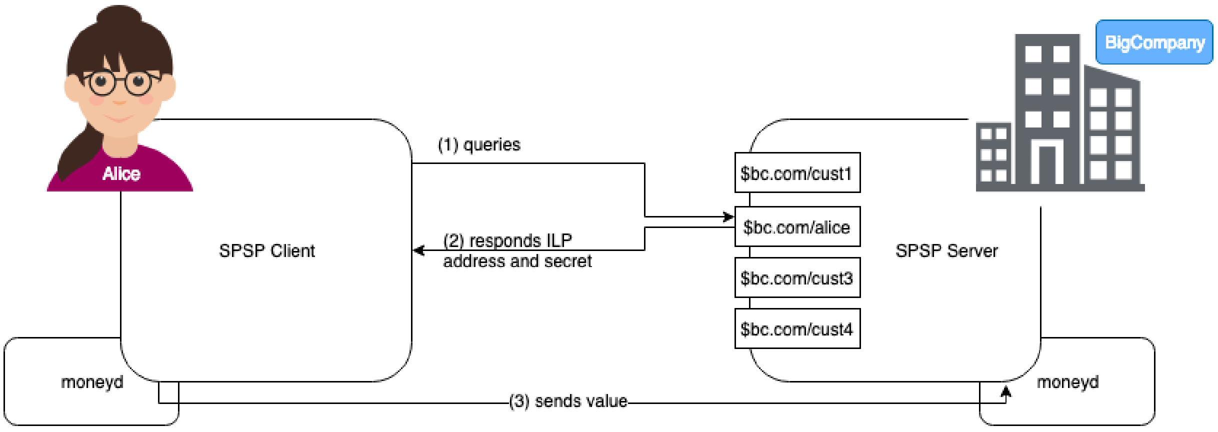

Fig. 6: SPSP and Moneyd [15]. Between the two Moneyd instances there are usually multiple connectors providing the ILP service.

The protocol follows like below [15]:

- 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 [16].
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[6] and a client[7] 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.

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

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.

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 [17].

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

We consider Figure 7 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.

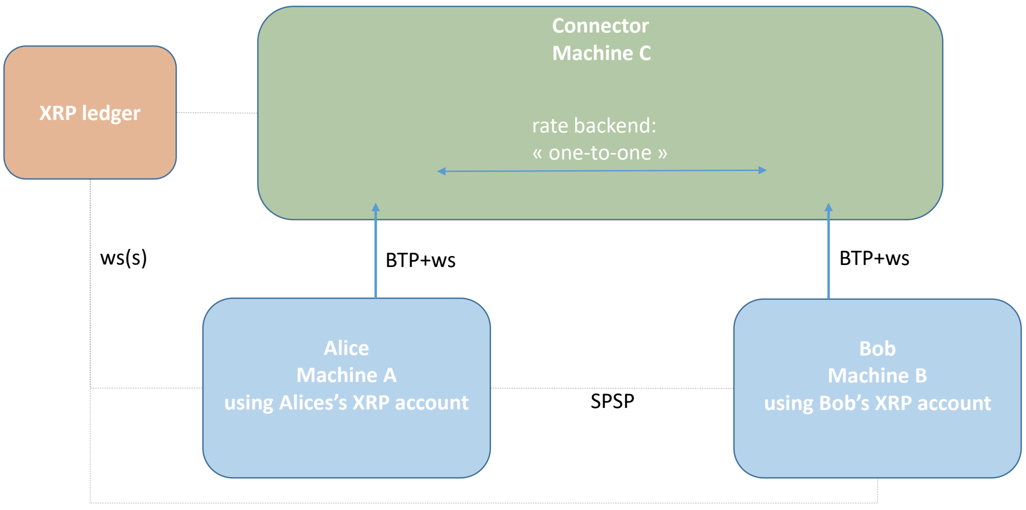

Fig. 7: Example 2: XRP payment.

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 8, 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.

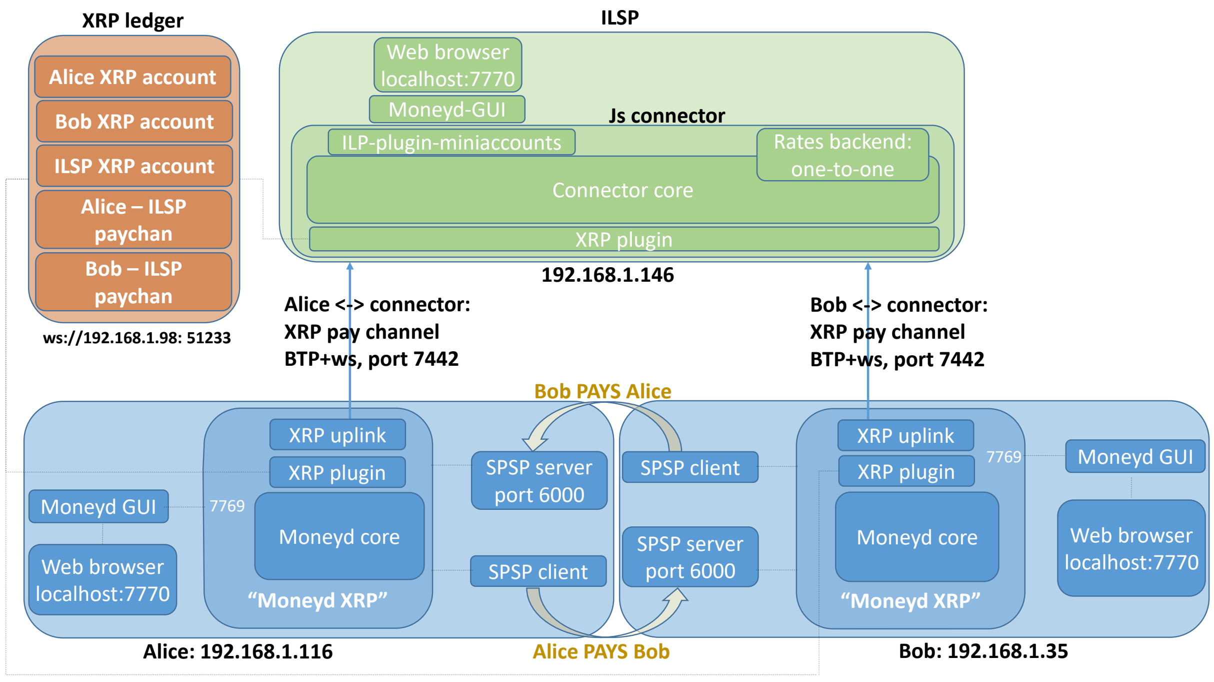

Fig. 8: Example 2: XRP payment, advanced.

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 9, and follows like below:

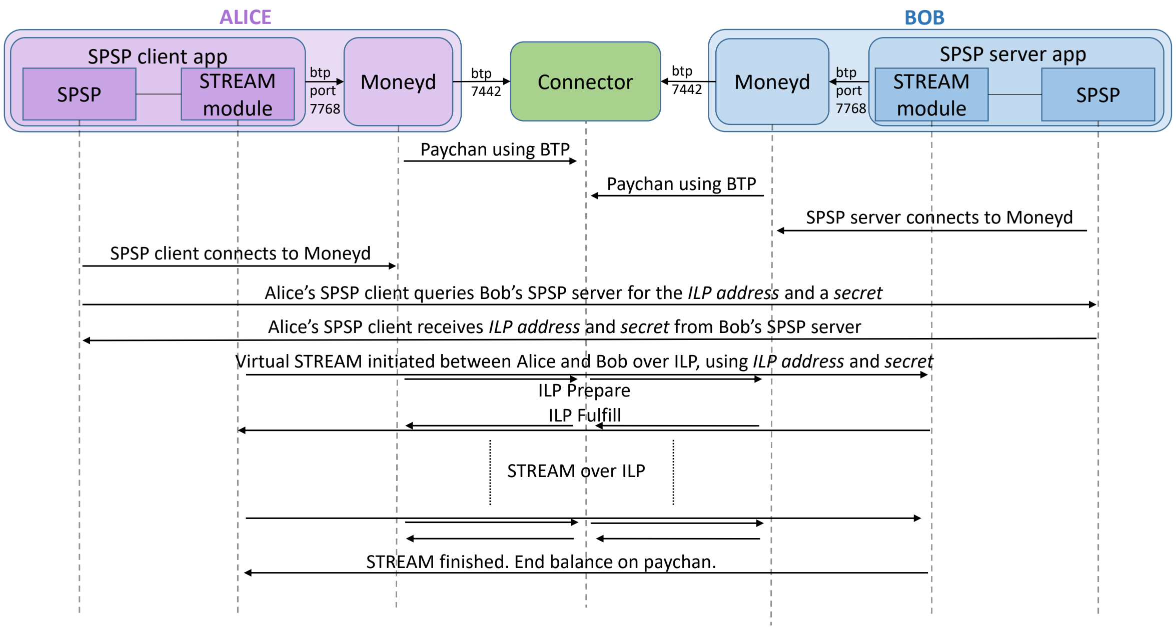
ALICE
BOB
SPSP client app
SPSP
STREAM module
btp port 7768
Moneyd
btp 7442
Connector
btp 7442
Moneyd
btp port 7768
SPSP server app
STREAM module
SPSP
Paychan using BTP
Paychan using BTP
SPSP server connects to Moneyd
SPSP client connects to Moneyd
Alice's SPSP client queries Bob's SPSP server for the ILP address and a secret
Alice's SPSP client receives ILP address and secret from Bob's SPSP server
Virtual STREAM initiated between Alice and Bob over ILP, using ILP address and secret
ILP Prepare
ILP Fulfill
STREAM over ILP
STREAM finished. End balance on paychan.

Fig. 9: Example 2: 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 segments the larger payments into packets and reassembles them. STREAM packets are sent as the data portion of an ILP packet that can be parsed 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+0APnOaBw5w5iUvvEggyB4LeOJ8Bjbav7FKGyJ6Ih95xT8
    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 [18, 19].

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.

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

```
{
  "version": 1,
  "uplinks": {
    "xrp": {
      "relation": "parent",
```

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

```
      "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": "100000"
      },
      "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"
      }
    }
  }
}
```

Next time we are going to discuss the STREAM protocol, the connectors, and are going to see a more advanced practical example.

Acknowledgements

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

Acronyms

API Abstract Programming Interface. 3, 6, 9, 23

BTP Bilateral Transfer Protocol. 10, 16–21

GUI Graphical User Interface. 12, 13, 16, 17, 21

ILP Interledger Protocol. 3–20

ILSP Interledger Service Provider. 6, 9, 16, 17

SPSP Simple Payment Setup Protocol. 10–21

Glossary

Moneyd An ILP provider, allowing all applications on an end-user computer to use funds on the live ILP network. 3, 6, 7, 12–21

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. 3

Switch API A SDK for cross-chain trading between BTC, ETH, DAI and XRP with Interledger Streaming. 3, 6

XRP Ripple's digital payment asset which is used for Interledger payments. 2, 4, 5, 7, 8, 12, 13, 15–17, 20, 21

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

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

[12] 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.

[13] 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.

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

[15] 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.

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

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

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

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