

MASARYK UNIVERSITY  
FACULTY OF INFORMATICS



# Security and Routing in the Ripple Payment Network

MASTER'S THESIS

**Jan Michelfeit**

Brno, 2011





---

## ZADÁNÍ DIPLOMOVÉ PRÁCE

**Student:** Bc. Jan Michelfeit

**Datum:** 15.11.2010

**Program:** Informatika

**Obor:** Bezpečnost informačních technologií

**Garant oboru:** doc. RNDr. Václav Matyáš, M.Sc., Ph.D.

**Vedoucí práce:** doc. RNDr. Václav Matyáš, M.Sc., Ph.D.

**Název práce:** Security and routing in the Ripple payment network

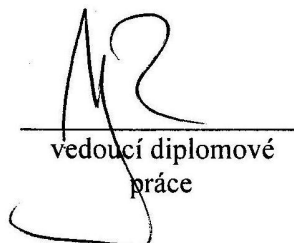
**Zadání:** The thesis will deal with Ripple, a decentralized payment network, and its security aspects, namely anonymity of users. The main goal of the thesis is to investigate various routing algorithms suitable for use in the system and point out their effects on anonymity, confidentiality and integrity of transactions. Depending on the achievements in the theoretical part, a practical component of the research will be agreed with the supervisor.

### Základní literatura:

- Ryan Fugger: *Money as IOUs in Social Trust Networks & A Proposal for a Decentralized Currency Network Protocol* (2004)
- Pranav Dandekar, Ashish Goel, Ramesh Govindan, Ian Post: *Liquidity in Credit Networks: A Little Trust Goes a Long Way* (2010)
- Ludwig von Mises: *The Theory of Money and Credit* (1953)

**Souhlas se zadáním (podpis, datum):**

  
\_\_\_\_\_  
student

  
\_\_\_\_\_  
vedoucí diplomové  
práce

  
\_\_\_\_\_  
garant oboru



## **Declaration**

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

**Advisor:** prof. RNDr. Václav Matyáš, M.Sc., Ph.D.



## **Abstract**

This thesis deals with the design of Ripple, a decentralized payment network proposed by Ryan Fugger. First, the nature of the system is analyzed from the point of view of economic science, in order to confront some common misconceptions about the system. The design of the distributed version of the system is then investigated, emphasizing the security aspects of the features of the system. Most attention is paid to the design of a distributed algorithm for finding paths in the network and the routing of messages used in the algorithm.





## **Keywords**

Ripple project, payment systems, distributed systems, onion routing, peer-to-peer networks



## Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction to Ripple</b>                                      | <b>3</b>  |
| 1.1      | <i>Ripple design overview</i>                                      | 4         |
| 1.2      | <i>Present status and potential users</i>                          | 5         |
| <b>2</b> | <b>The economics of Ripple</b>                                     | <b>7</b>  |
| 2.1      | <i>Money as a common medium of exchange</i>                        | 7         |
| 2.2      | <i>Indirect exchange as a routing problem</i>                      | 8         |
| 2.3      | <i>Modern money</i>  | 12        |
| 2.4      | <i>Electronic payment as a routing problem</i>                     | 14        |
| <b>3</b> | <b>Formal representation of a Ripple network</b>                   | <b>19</b> |
| 3.1      | <i>Money and accounts</i>  | 19        |
| 3.2      | <i>Network</i>   | 20        |
| 3.3      | <i>A simplified model based on credit limits</i>                   | 23        |
| <b>4</b> | <b>Core functionality of Ripple and its basic security aspects</b> | <b>27</b> |
| 4.1      | <i>Communication between neighbours</i>                            | 27        |
| 4.2      | <i>Communication between distant nodes</i>                         | 28        |
| 4.2.1    | <i>Onion routing</i>   | 28        |
| 4.2.2    | <i>Traffic analysis</i>  | 29        |
| 4.3      | <i>Book keeping and accounting</i>                                 | 30        |
| 4.4      | <i>Path discovery and payment</i>                                  | 31        |
| 4.5      | <i>Path discovery</i>  | 32        |
| 4.6      | <i>Payment</i>   | 34        |
| <b>5</b> | <b>Security and effectivity of path discovery</b>                  | <b>39</b> |
| 5.1      | <i>Security goals</i>  | 39        |
| 5.1.1    | <i>Circumventing trap paths</i>                                    | 39        |
| 5.1.2    | <i>Prevention of flooding</i>                                      | 40        |
| 5.1.3    | <i>Confidentiality of connections</i>                              | 40        |
| 5.1.4    | <i>Anonymity of users</i>  | 41        |
| 5.1.5    | <i>Confidentiality of account balances</i>                         | 42        |
| 5.2      | <i>Connectivity and capacity</i>                                   | 42        |
| 5.3      | <i>Concurrent path discoveries</i>                                 | 43        |
| 5.4      | <i>Broadcast—convergecast search</i>                               | 43        |
| 5.5      | <i>Pruning by price</i>  | 44        |
| 5.6      | <i>Limiting the number of messages</i>                             | 45        |
| 5.7      | <i>Avoiding circles</i>  | 46        |
| 5.8      | <i>Multi-path payments</i>   | 47        |
| 5.9      | <i>Existing routing schemes</i>                                    | 48        |
| 5.10     | <i>Summary</i>   | 49        |

---

|     |                                      |    |
|-----|--------------------------------------|----|
| 6   | <b>Ripple as a service</b> . . . . . | 51 |
| 6.1 | <i>Host-based routing</i> . . . . .  | 52 |
| 7   | <b>Conclusions</b> . . . . .         | 55 |

## Chapter 1

### Introduction to Ripple

The main motivation of the Ripple project is the original author's realization of the nature of today's electronic money as consisting of formal acknowledgements of debt issued by a particular debtor (in most cases electronic bank deposits issued by a bank) to a given creditor. Creditors—individuals and businesses—give credit to issuers of money, and are therefore connected with them by *credit relationships*. Financial institutions are connected with other financial institutions by clearing and settlement systems. All these entities and the credit relationships between them form a network, and any payment between two entities (vertices) is carried out over a path in this network by issuing new certificates of debt (or settling existing debt) between adjacent vertices.

The fact that any payment requires finding the aforementioned path, the related effort, and the inherent risk of dealing with many intermediaries has given the international money system the structure it has today. Digital transactions are carried out either through commercial banks connected through central banks, or using isolated micropayment systems. Speaking in graph terminology, the graph is a union of (not necessarily disjoint) trees of small depth. Payments follow leaf-to-root and root-to-leaf paths which rarely exceed the length of four hops.

It is exactly here where Ripple is supposed to step in. As Ryan Fugger, the author of Ripple, argues in [13], the problem of finding a path for payment is similar to routing a message in a communication network. With today's computing power and modern routing algorithms, finding a path in an arbitrary network should be a much simpler task than in history. Moreover, modern cryptography can help minimize the risks associated with payment and complement the expensive legal framework that ensures security today. The benefit of this, Ryan Fugger believes, could be that "Billions of trust relationships that exist outside the tightly-regulated global hierarchical currency network could be integrated into that network, removing single points of failure without harming the value of existing obligations. The resulting network would be more stable, and therefore require less regulation and be less expensive to use, while at the same time being more democratic and responsive to local concerns." [13]

Ripple would be a system accommodating the credit relationship network in an actual computer network. It would consist of the necessary communication protocols and server software that would facilitate the finding of paths for payment and securely committing the payments by exchanging notes between participants. The main goal of the Ripple

project is to decrease the transaction cost of payment, and possibly allow for a more flexible money system.

### 1.1 Ripple design overview

The design of Ripple is characterized by a few important features, some of them quite challenging, security-wise. It is yet unclear whether all these requirements can be satisfied at the same time. They are:

- **Decentralization**

It is assumed that the Ripple network will span over many servers connected in a peer-to-peer fashion and communicating with each other through insecure channels. While in reality most users would probably flock together on a relatively small number of servers (as is the case with today's micropayment and internet banking systems), Ripple must also be prepared (performance-wise) to handle a one-user-per-server scenario. Considering some of the following points, routing in such a network is quite a technological challenge compared to routing among users on a single server, which is trivial, of course.

- **Confidentiality**

The details of credit relationships and standing balance between users will not be published outside of the users' servers. Obviously, financial information is one of the most sensitive types of information, so it is logical that it is appropriately protected, however it presents an obstacle to efficient routing, because routing rules follow directly from this information. Perfect confidentiality is however also not an option. A malicious user will always be able to extract *some* information about credit relationships or balances simply from his ability or inability to route a payment of a chosen value to a chosen participant. Any implementation will therefore have to find a balanced trade-off between confidentiality and efficiency of routing.

During payment, intermediaries do not know who is participating in the payment except the previous and next intermediary in the chain. Even then they do not know whether the former is the payer or whether the latter is the payee. The payer and payee themselves do not know who the intermediaries are either (except for their immediate neighbours).

- **Honouring existing trust relationships**

Ripple must allow routing payments through an arbitrary path of *untrusted* intermediaries, while ensuring that every user can only be harmed by one of his *trusted* immediate neighbours in the network. Unlike today's money system, where payers are to some degree exposed to potential malicious behaviour of distant users, in Ripple any malicious behaviour should be contained on the level of neighbouring participants, who can presumably take each other to court.

- **Abstraction of contract details**

Every note (or any financial instrument) is in essence a contract and as such it can

contain a plethora of terms and parameters such as time to pay, interest rate, court of law etc. Ripple should be designed in such a way that it is completely agnostic to these details while offering enough information for potential legal process. That should be easily achieved by embedding contract terms in notes and ensuring their integrity, authenticity and non-repudiation.

## 1.2 Present status and potential users

Because the requirements for a system like Ripple are determined by potential users, it is worthwhile exploring Ripple's possible future step by step and noting the specific needs of various kinds of users.

Early adoption will most probably be driven by enthusiasm or social reformism rather than economic reasons, it will be based on friend-to-friend relationships between individuals, account for a minimum part of users' economic activity, and require settling most notes outside of the system. Users in this phase will most likely form highly connected communities and they will generally trust each other more than they trust the system itself—the opposite situation compared to a mature payment system. This translates to lesser need for non-repudiation of notes and confidentiality of credit relationships, but raises the proverbial bar for the perceived security of the human interface.

An answer to these requirements (and the only tangible output of the Ripple project so far) might already be in place—*ripplepay.com*, a web-based single-server implementation of Ripple. While being fully functional and generally acceptable for the aforementioned early adopters, lack of reputation prevents it from being used as more than a demonstration of Ripple's basic features. It also implements a rather simplistic concept of a credit relationship and corresponding routing rules. While they are probably sufficient for early adopters, who would not resort to complex valuation of notes, financial institutions would soon find them an obstacle to serious adoption.

Success of *ripplepay.com* or a similar service could inspire some commercial implementations of Ripple in existing closed communities, such as users of auction and shopping websites, but the full potential of Ripple lies in its decentralization. It could become a system connecting as yet isolated micropayment systems, or serve as a distributed *real time gross settlement* system. In any case, nothing less than adoption by at least some existing financial institutions and integration into existing financial infrastructure can be considered success for Ripple. In the following, we shall always assume that Ripple aims to offer enough flexibility to appeal at least to existing banks and payment systems, not just individuals.





## Chapter 2

# The economics of Ripple

It is easy for an economic layperson (to which the author of this thesis counts as well) to fall into misconceptions about both Ripple and the nature of money itself. To avoid these misconceptions, it is vital to have a basic understanding of the underlying economics. This chapter is therefore dedicated to explaining the origin and historical development of money, the differences between various types of money, and how money is related to routing.

### 2.1 Money as a common medium of exchange

“Where the free exchange of goods and services is unknown, money is not wanted. In a state of society in which the division of labor was a purely domestic matter and production and consumption were consummated within the single household it would be just as useless as it would be for an isolated man.”

As Ludwig von Mises, an Austrian economist, points out in his *Theory of Money and Credit* [5], the crucial condition for the emergence of money was division of labour. While it dramatically increased productivity of labour through specialization, it also created a problem—how should the produced goods be effectively transferred from producers to consumers. In the following section, we shall argue that this problem is essentially a routing problem and explore it as such, but for now, let us continue with its actual history.

The simplest solution to the aforementioned problem is *direct exchange*. When one party produces a commodity desired by another party and at the same time the other party is in possession of a different commodity desirable for the first one, they can exchange some of the commodities at a ratio that is suitable for both—such that each party values the given amount of the commodity it acquires more than the given amount of the commodity it surrenders. The problem with direct exchange is that finding a suitable partner for exchange is difficult in practice; such a partner need not even exist.<sup>1</sup> Being faced with this inconvenience, people in history have quickly turned to *indirect exchange*, which is illustrated in the following example:

*Alice desires a given amount of commodity b, owned by Bob, and she is willing to forfeit some of her commodity a. Unfortunately, Bob is not interested in having a, however*

---

1. More precisely, such an arrangement where it is impossible to find a partner for direct exchange would not evolve in the first place.

he desires commodity  $c$  owned by Carol, who in turn is interested in having  $a$ . In order to get  $b$ , Alice first approaches Carol and exchanges  $a$  for  $c$ , then exchanges  $c$  for  $b$  with Bob.

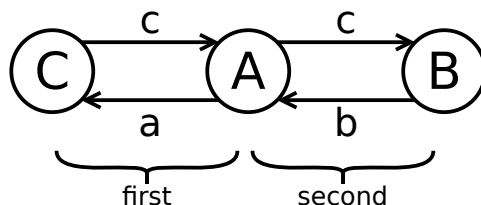


Figure 2.1: Indirect exchange

The main characteristic of indirect exchange is the use of a commodity (here the commodity  $c$ ) as a medium of exchange. This commodity need not have any use-value for Alice and is obtained not for consumption, but mainly for the purpose of a future exchange with Bob. In theory, any commodity can serve as a medium of exchange, but obviously, it holds that the more marketable a commodity is, the more apt it is to serve as a medium of exchange, and when it is used as a medium of exchange, it becomes even more marketable. [5, section 1.2] This development directs the market to select fewer and fewer commodities, ultimately one, as the common medium of exchange, or *money*. With money, indirect exchange can be easily carried out as a two-step process—selling your produce for money and buying consumption goods for money. Because everyone in the economy accepts money, no one has to solve the problem of finding the right partner for direct exchange anymore.

## 2.2 Indirect exchange as a routing problem

Let us forget this elegant solution for a while and take a closer look at indirect exchange. We will try to describe the problem with which the parties in the last example are faced, and try to generalize it in three logical steps. For the sake of simplicity, we shall refer to direct exchange simply as “exchange” and to indirect exchange as “transaction” from now on. (The word “transaction” also carries some fitting computer-science connotations.)

The whole transaction consists of two parts, or two exchanges: first, Alice and Carol exchange  $a$  and  $c$ , and then Alice and Bob exchange  $c$  and  $b$ . The order in which these exchanges take place and the whole arrangement are determined by two things:

- Before exchanging  $a$  for  $c$  with Carol, Alice may not have enough  $c$  that she could offer to Bob to get the desired amount of  $b$ . In this case, the exchanges cannot be carried out in a different order.
- The whole transaction is not perfectly atomic. The exchanges do not take place simultaneously and the transaction may fail after just one of them has happened. In our example, this would correspond to a scenario where Alice successfully deals

with Carol, but is for some reason unable to continue by exchanging with Bob. It is Alice as the initiator of the transaction who is exposed to the risk of being left with  $c$ , unable (in the extreme case) to exchange it for any other commodity. As we have learned,  $c$  need not have any use-value for Alice, therefore Alice would be left at loss. This risk of loss constitutes a *transaction cost* to any party in the middle of the transaction.<sup>2</sup> In practice, the initiator of the transaction is usually the one willing to take the risk, but this cost may be prohibitive to any other arrangement of exchanges.

As the first of our three steps, we will relax these two constraints and take a look how this may affect the options Alice has. First, let's assume every party enjoys an abundance of all kinds of commodities involved here. (This is a realistic assumption for example with own promissory notes, which one may produce on demand.) And second, let us assume the parties can somehow mitigate the aforementioned risk of mid-transaction failure (for example by employing some kind of *transaction controller* to ensure atomicity), or that they can agree on such exchange ratios that will compensate them for the risks they are taking. Then, Alice may be able to arrange the following transaction:

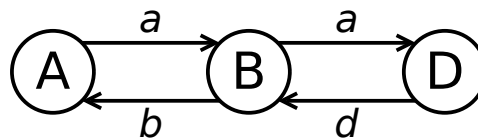


Figure 2.2: "Reverse" indirect exchange

Alice finds such a Dave who is willing to exchange  $d$  (desired by Bob) for  $a$  and informs Bob of that fact. Then either Bob accepts Alice's  $a$  in return for  $b$  with a future prospect of exchanging that  $a$  for  $d$ , or Bob exchanges some of his  $a$  for  $d$  with Dave first, then replenishes his reserve of  $a$  with Alice. (The order of the exchanges is not relevant under our first assumption.)

In our picture describing the transaction, we now see that Alice need not be placed in the middle. A question that emerges naturally is whether the transaction could involve both Carol and Dave, and if the chain of parties in the picture must always be a chain of three. Theoretically, the transaction may involve an arbitrary number of parties and form an arbitrarily long chain. Formally speaking, the transaction (indirect exchange between Alice  $A$  and Bob  $B$ ) can be described by a sequence  $[P_i]_{i=1}^n$  of  $n \geq 2$  exchanging parties such that:

- $\exists j, (1 \leq j \leq n - 1), A = P_j$  and  $B = P_{j+1}$  (Alice and Bob are neighbouring parties),
- $\forall k, (1 \leq k \leq n - 1), P_k$  exchanges  $c_{k,k+1}$  (a given amount of some commodity) for  $c_{k+1,k}$  with  $P_{k+1}$ ,

2. We are using the term *transaction cost* in its traditional economic sense here; not to be confused with our *transaction* as short for indirect exchange.

## 2. THE ECONOMICS OF RIPPLE

- $P_1$  values  $c_{2,1}$  more than  $c_{1,2}$ ,
- $P_n$  values  $c_{n-1,n}$  more than  $c_{n,n-1}$ ,
- and  $\forall l, (1 < l < n)$ ,  $P_l$  values  $(c_{l-1,l} + c_{l+1,l})$  more than  $(c_{l,l-1} + c_{l,l+1})$ .

(Of course, the last point only holds if we rule out the possibility of transaction failure. In reality, it is a little more complicated and depends on which one of  $P_l$ 's exchanges happens first.  $P_l$  has to evaluate the risk that the transaction fails between the two exchanges concerning  $P_l$  and the potential loss in that case—the risk exposure.)

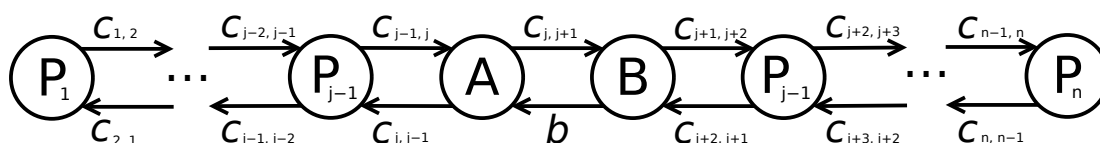


Figure 2.3: Multi-party indirect exchange with traditional counter-trades

As the second step in our generalization, we will abandon the notion of direct exchange as an atomic process. Obviously, each exchange consists of one transfer of commodity from one party to the other, and another one in reverse direction. (In our formalization, the transfer of  $c_{l-1,l}$  is paired with the transfer of  $c_{l,l-1}$ , and  $c_{l+1,l}$  with  $c_{l,l+1}$ ) We will now treat these two transfers independently of each other.<sup>3</sup> In our picture, the chain representing the transaction is now taking form of a circle. Most parties are occurring twice, on the upper arc and lower arc, only the two parties that were previously at either end are present only once. The thick arrows indicate the individual transfer pairs that are traditionally considered atomic.

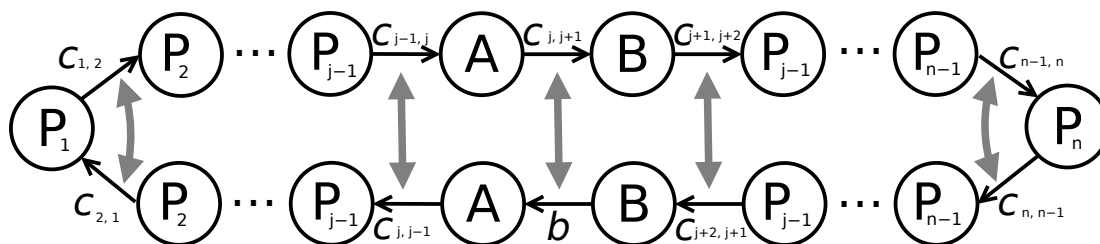


Figure 2.4: Atomicity of traditional counter-trades

Normally, the parties act (decide to participate or not to participate in the transaction) with some assurance of atomicity of individual exchanges (fraud prevention) and

3. It should be noted that it is out of economics' scope of interest to consider these two transfers separately (the case when only one transfer takes place—fraud—is more a legal, than economic, phenomenon). It will however prove to be fruitful to us, so we will take our analysis to a more general, praxeological level.

some assurance of the atomicity of the whole transaction (our theoretical transaction controller). To guarantee exchange atomicity is theoretically an easier task than to guarantee transaction atomicity, and more, if we do away with exchange atomicity, the more assurance of transaction atomicity will be required.<sup>4</sup> In order not to lose touch with reality and to maintain a *practically conceivable* concept of indirect exchange, we shall look for another security element that could compensate for the loss of exchange atomicity.

What lends itself is atomicity assurance for the pairs of adjacent transfers in the circle. (The transfer of  $c_{l-1,l}$  would be paired with that of  $c_{l,l+1}$ .) It is only a slight variation of the traditional “fraud prevention” of direct exchange, and therefore quite realistic. Also, if we assume that  $P_l$  values  $c_{l-1,l}$  over  $c_{l,l+1}$  for each  $l$ , there is absolutely no need for transaction atomicity anymore.

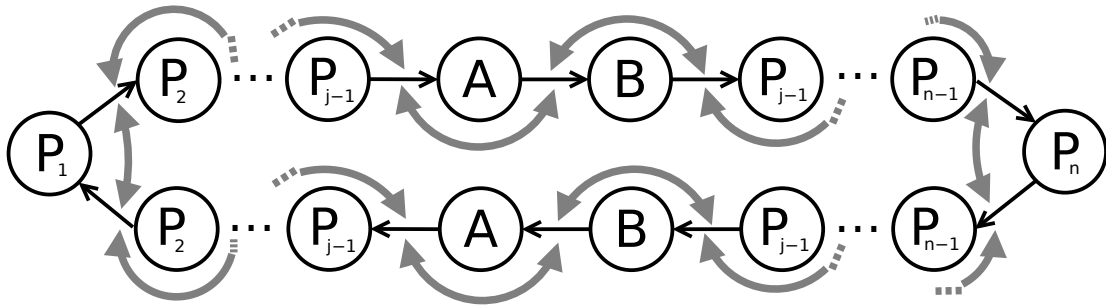


Figure 2.5: Atomicity of transitional exchange

It is now becoming clearer that by considering direct exchange as not atomic, the very concept thereof (two transfers between two parties) ceases to play an important role in our analysis. An alternative concept of exchange is emerging here that corresponds to the pairs of *adjacent* transfers and involves not two, but three parties, each having a different role. Whereas direct exchange is “symmetrical”, in this new concept one party plays a central role—it “exchanges” one commodity for another—, one party only gives, and one party only receives. From the viewpoint of the central party it is however very similar to traditional direct exchange, only now it deals with two partners, not just one. In the rest of this paper, we shall explicitly refer to this new concept of exchange as *transitive exchange* whenever in risk of confusion with direct exchange.

The third and last step of our generalization should be obvious from the picture illustrating the transaction. After abandoning the traditional concept of exchange, there is now no reason why most (all but two) of the participants in the transaction should take part in two transitive exchanges. In other words, there is no reason to assume the party  $P_l$  on the upper arc of the circle in the original figure is identical to  $P_l$  on the lower arc. This applies even to Alice and Bob; we can disregard one of the transfers happening between them. It is of course the one of  $a$ , directed from Alice to Bob, as the transfer of  $b$  is crucial to the transaction. Alice’s desire to get commodity  $b$  is the very motivation of the transaction, the reason why it is initiated.

4. The two complement each other. If we had perfect atomicity for exchanges (and each exchange was profitable unto itself, considering use-value only), there would be no need for transaction atomicity.

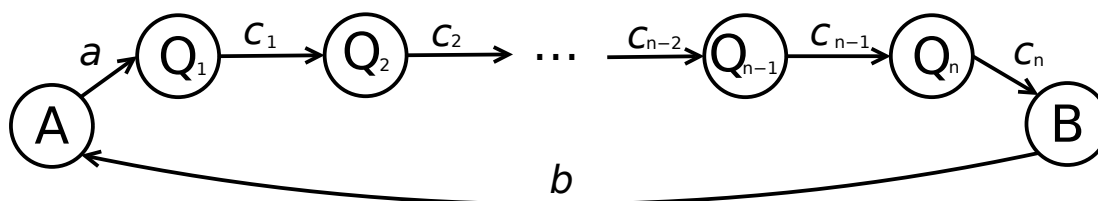


Figure 2.6: Indirect exchange is a routing problem

The formalization of the above is as follows: A transaction initiated by Alice  $A$  in order to get commodity  $b$  from Bob  $B$  is described by a sequence of parties  $[Q_i]$  such that:

- $A$  values  $b$  more than  $a$  (plus the risk exposure in case of fraud or transfer failure in the transitive exchange with  $B$  and  $Q_1$ ),
- $\forall i, (1 < i < n), Q_i$  values  $c_{i-1}$  more than  $c_i$  (plus risk exposure),
- and  $B$  values  $c_n$  more than  $b$  (plus risk exposure).

(Note that this model can easily be adapted to include direct exchanges as well with  $n = 0$ .)

Let us now picture the whole market as a directed graph, all the parties in the market represented by vertices and all the possible transfers between the parties by directed edges. Alice's problem of carrying out an indirect exchange and getting  $b$  is the problem of finding in this graph a path from  $A$  to  $B$  (the sequence of vertices  $[Q_i]$  and edges  $[c_i]$ ) that satisfies the above conditions—a *routing problem*.

It would be far-reaching to call Ripple a *tool for indirect exchange of physical goods*, but the principle can very well be used for a multi-party barter system.<sup>5</sup> What should also be kept in mind is that even though the Ripple is meant to be used for the transfer of money, it does not necessarily presuppose the existence of one common medium of exchange, and its operation need not be bound to one currency. Our detour into the economics of indirect exchange will also provide us with a nice parallel with today's money, and the concept of *transitive exchange* will show itself vital to understanding Ripple.

### 2.3 Modern money

Let us now continue with our introduction to money and its history. In the past, people throughout the world started to use various physical commodities as money: grain, livestock, but most prominently precious metals, such as gold and silver. The use of *commodity money*, however, has its drawbacks, namely the size, weight, and resulting difficulty

5. One such system based on the idea behind Ripple is already in use at [www.multiswap.net](http://www.multiswap.net)

of storage and transport. For these reasons, people gradually abandoned the commodity itself as the medium of payment and started storing the commodity in banks. Banks issued bank notes—claims to the stored commodity—to the owners, and the owners would then use these bank notes for payment, transferring ownership of the commodity to the payee.

Through the history, bank notes referring to commodity money, mostly gold and silver, were used in parallel with gold and silver coins and acted as substitutes for their face value's worth of the commodity. They are however fundamentally different from commodity money, as illustrated in the following quote:

“Claims are not goods; they are means of obtaining disposal over goods. This determines their whole nature and economic significance. They themselves are not valued directly, but indirectly; their value is derived from that of the economic goods to which they refer. Two elements are involved in the valuation of a claim: first, the value of the goods to whose possession it gives a right; and, second, the greater or less probability that possession of the goods in question will actually be obtained. Furthermore, if the claim is to come into force only after a period of time, then consideration of this circumstance will constitute a third factor in its valuation.” [5, section 1.3.1]

Bank notes therefore constitute another type of money—*credit money*. Users of bank notes give the issuing banks *credit*, and value the notes depending on the perceived probability of default, the note's maturity date, and other possible terms. Bank notes issued by banks with perfect reputation and payable on demand are mostly valued as high as the commodity they refer to, but in general, the person of the issuer and his financial standing can affect the value of the notes just as much as the value of the commodity does. This observation is crucial for our analysis, but before we pursue it further, we should make sure it is still applicable to today's money.

With *fractional reserve* banking, bank notes cannot be considered claims (strictly speaking, property titles) to goods anymore, as issuing a claim to nonexistent goods is nonsense or fraud). Instead, we must consider them mere *promissory notes*<sup>6</sup>. Despite the fundamental legal difference, the valuation of claims and promissory notes is essentially identical. It depends only on the probability of default, regardless of whether it is more affected by the risk of robbery or the risk of a bank-run. Therefore we need not differentiate between claims and promissory notes.

Another thing that must be pointed out is the significance of government intervention. Using legal tender laws, the governments first monopolized minting of coins in the era of commodity money; later, when bank notes became prevalent, they monopolized the issuance of notes.<sup>7</sup> At that time, government issued notes were not dramatically different from privately issued notes; both were credit money backed by precious metal.

6. A promissory note is a document, wherein one party (the issuer) makes a promise to pay a given amount of goods to another party (either specified by name, or the bearer of the note), on demand of the payee or after maturity date, under specific terms.

7. By “notes”, we mean both paper notes and token coins—coins made out of ordinary metals, which can be redeemed for precious metal.

That has, however, changed to this day. Most governments in the world have abandoned the metallic standard (convertibility of government currency to precious metal) by 1971, when the Breton Woods system collapsed.

This fact considerably affects our analysis, as we cannot consider government currency credit money anymore. In the absence of convertibility to commodity, government currency does not derive its value from the value of the commodity and the probability that it will be successfully redeemed. It only has the value it has because it is a legal tender and must be accepted as settlement of both public and private debt; it is not credit money, but *fiat money*—established by government fiat.<sup>8</sup> Fortunately, for us, who are not especially concerned with payment involving physical currency, this is not an insurmountable problem.

It is because during the course of the aforementioned developments, bank deposits have taken a form different from bank notes and they are still being used directly for payment. They are not represented by bearer promissory notes anymore, but rather exist only in book-entry form (physical or electronic). Unlike bearer notes, which are only bound to one person—the issuer—and can be traded, bank deposits always represent a relationship between two persons. They can be created or eliminated, but the person of the creditor (the client of the bank) and the debtor (the bank) remain constant. Not all bank deposits are used for payment directly, such as time-deposits and various savings accounts, but a large portion of *demand deposits*—money in current accounts (or “checking accounts”)—serves for *wire transfer* or *credit/debit card payments*.

With the advent of the Internet, various electronic payment systems came into being where users are able to “top-up” their accounts with cash or from traditional bank accounts, and then pay other users of the same service. These systems, though undoubtedly innovative in many ways, nevertheless share their characteristic features with both wire transfer and card payment. All of these methods of electronic payment, we shall argue, rely on a network of participants maintaining credit relationships between them, and operate by routing credit through that network.

### 2.4 Electronic payment as a routing problem

All electronic money—money that can be directly used for electronic payment—is credit money.<sup>9</sup> And of this credit money, an overwhelming majority is not represented by *bearer securities*.<sup>10</sup> With this kind of money, electronic payment cannot be performed by trans-

---

8. Ryan Fugger’s papers on Ripple often conflate fiat money and credit money, in order to capture both in a single network of (loosely defined) *trust relationships*. [12] It is somewhat justified, as the expected monetary policy (mostly the perceived probability of hyper-inflation) affects the valuation of government currency similarly to how the perceived probability of default affects the valuation of a promissory note. Both of these factors can be considered a kind of *trust*. However such generalization is not necessary for Ripple.

9. Probably the only exception being an alternative currency called Bitcoin, value of which is not based on the value of some underlying economic good, but rather on pure speculation.

10. Exceptions include the *Digital Bearer Certificates* issued by eCache, a bank operating in the Tor network.



ferring its ownership from payer to payee. Instead, it must be done by coordinating the creation and elimination of credit between the payer, payee and several intermediary parties.

Before we illustrate this mechanism with an example, let us recapitulate what we have learned about the nature of the money in question. Bank deposits as well as deposits with payment systems represent a *relation* between economic subjects—an obligation of the debtor to the creditor. This enables us to model the system of electronic money and its users as a directed graph, with vertices representing economic subjects and arcs representing existing obligations.

Any obligation can, of course, only arise between such a pair of subjects, where the creditor believes that the debtor is willing and able to fulfil the agreed terms of payment, or that a settlement can be legally enforced. That means potential creditors only attribute value to obligations from a limited number of potential debtors. Thus underneath the directed graph of actual obligations, there must be an undirected graph of *credit relationships* based on the creditworthiness of one person to another. Vertices remain the same, but instead of arcs representing obligations, edges exist between any two vertices where there is a *possibility* of obligation between the corresponding subjects. For purposes of electronic payment, these relationships are mutually acknowledged and formalized in the form of *accounts*.

Let us now illustrate what happens during payment, using bank-to-bank wire transfer as an example: Alice wants to pay \$100 to Bob by wire-transfer from her account with *AliceBank* to Bob's account with *BobBank*. Alice's \$1000 deposit in *AliceBank* is lowered to \$900, and Bob's \$1000 deposit with *BobBank* is increased to \$1100. What happens behind the scenes is that *AliceBank* initiates an inter-bank payment of 100 dollars to *BobBank* using a real-time gross settlement (RTGS) system operated by the central bank. This payment is done by lowering *AliceBank's* \$100100 deposit with the central bank to \$100000, and increasing *BobBank's* \$9900 deposit with the central bank to \$10000.

In the example, both the commercial banks and the central bank form a *chain of intermediaries*. They all increase their debt to the next link in the chain or decrease the next link's debt to them, and in turn decrease their debt to the previous link or increase the previous link's debt to them. The same mechanism is responsible for card payments, payments made using micropayment systems, or international wire-transfers. The important thing is that payer and payee need not be in any form of legal or economic relationship before or after the payment, they use an existing framework of relationships—the one represented by the credit relationship graph. For any payment to be made, a path in this graph must exist between the payer and payee.

The problem of finding a path in a graph is indeed inherent to this type of payment, and this is one of the reasons why the national banking systems have evolved into hierarchical structures, where the respective part of the credit relationship graph is a *tree* rooted in the central bank. Finding a path in a tree by following a known leaf-to-root and root-to-leaf path is a trivial task, of course.

The existence of a path between the payer and the payee is a necessary, but surely not a sufficient condition for payment to take place. The amount that can be paid out of a bank account is of course bounded by the balance of the account. But also the amount that can be paid *into* the account is limited, which may not be obvious. Based on our simplified example, the reader may be tempted to believe that the \$100 being paid have a constant value, regardless of the account where they are situated. It is, however, not that simple in general.

The characteristic feature of credit money—the fact that there is a debtor involved and the money’s valuation depends on their financial standing—is to a great degree obscured by the legal and regulatory environment in which the banking industry operates. Deposit insurance or the prospect of a potential government “bail-out” push the probability of a bank-run (or any event that would hinder the bank’s ability to fulfil its obligations) to a value close to zero. Even in the absence of these factors, only trustworthy banks tend to succeed in a free market. Because of the minuscule probability of default, most clients then resort to rational ignorance and treat deposits in the same way as currency. They disregard the debtor and value the deposits as high as cash.

However, in cases where the risk exposure is comparable to the cost of precisely evaluating the probability of default, doing so proves worthwhile. It is true for subjects that are dealing with big amounts of money, but it would also be true when dealing with potential debtors with a higher probability of default (compared to banks). Because introducing such subjects into the payment infrastructure is one of the goals of the Ripple project, we must take the more complex valuation of electronic money into consideration.

Equipped with these insights, we must treat money in different accounts as altogether different economic goods. If we then look at electronic payment again, we can describe the payment as an instance of the generalized *indirect exchange transaction* we defined above. The intermediaries engage in *transitive exchanges* of money in one account for money in another account, so their motivation can be understood as an effort to maximize their utility or profit. When the value of the money they receive (an increased debt of the previous intermediary to them, or their decreased debt to the previous intermediary) to them is higher than the value of the money they surrender (their increased debt to the following intermediary, or a decreased debt of the following intermediary to them), they agree to participate in the payment, otherwise they refuse.

This condition constitutes the other constraint on the feasibility of payment, along with connectedness. Because it is so broadly formulated, it conveniently covers all important aspects of payment, such as currency exchange or transaction fees. Because we understand the real value of credit money as independent of its nominal value, the denomination of an account is irrelevant to us. And the use of transaction fees in practice only exemplifies our understanding of the motivation of the intermediaries.<sup>11</sup>

---

11. It is fair to admit that in the case of banks that do *not* charge fees for wire-transfer, the motivation must be more exactly explained. Such banks offer wire-transfer as a *service*, which means that the costs associated with the transaction are compensated by client satisfaction, and, for example, the resulting ability to offer lower interest rates than their competitors.

We have previously mentioned that the amount of money that could be paid into an account is not unlimited. That was perhaps too bold a claim, as it is not possible to formally prove there is an effective *upper bound* on the balance of any account. However, the balance represents an *obligation*, and the creditor understands that the debtor is only able to fulfil an obligation up to a certain amount. Actually, the higher the outstanding balance is, the less value any further increase possesses. Therefore from a certain point the creditor would either demand payments of exorbitant nominal value when paid into the account, or refuse to accept payment through that account altogether. This point, called a “credit limit” in the literature on Ripple, is an important concept in the design of Ripple, at least in the current draft of the protocol. We shall, however, find this concept redundant; partly because in practice, there is no such discrete boundary, and also because our understanding of the valuation of money by participants already captures the phenomenon.

We have shown the similarities between the mechanism by which electronic payment is carried out and the problem of finding a path in a graph. We propose approaching the problem of payment in a distributed manner—similarly to packet routing.



## Chapter 3

# Formal representation of a Ripple network

### 3.1 Money and accounts

Before we lay down the the formal representation of the Ripple network itself, it is important to formalize the basic concepts, such as money, account, and balance. The formalization of money we choose should serve as the basis of representing money in the practical implementation, therefore the most useful way to describe money seems to be with a *registered promissory note*. A registered promissory note is a legally binding promise to pay an amount  $a$  of economic good  $g$  by debtor  $d$  to creditor  $c$  under terms of payment  $t$ , where  $a \in \mathbb{R}^+$  and  $c$  and  $d$  are economic subjects.

$$\langle a, d, c, g, t, \rangle_{\iota}$$

The unit of denomination  $g$  and terms of payment  $t$  are of great significance in practice, but for our purposes, they merely capture the complexity we choose to leave out of our analysis.  $\iota$  is a nonce—a unique identifier of the note used to prevent duplication, when the note is in electronic form.

Registered promissory notes are a financial instrument with a long tradition and firmly rooted in legislation. They can both represent existing balance—when kept—and transfer value—when issued. They are the simplest, yet the most flexible way of representing credit. Most of the “paperwork” surrounding bank accounts, such as statements, payment orders, receipts of deposit, etc., could be easily replaced by registered notes accompanied by specific contracts—a fact that can be taken advantage of to keep the design of Ripple simple and clean.

An important feature of the registered notes is their fungibility—the fact that two equivalent notes (different only in  $\iota$ ) are completely interchangeable and have the same value to the creditor.<sup>1</sup> Moreover, any two sets of notes have the same value, if the sums of nominal values ( $a$ ) of the notes in both sets are equal. Because of this, an account can be easily characterized by the ordered quadruple

$$\langle d, c, g, t \rangle.$$

A balance on the account would then correspond to the sum of nominal values of all notes  $\langle a_i, d_i, c_i, g_i, t_i, \rangle_{\iota_i}$ , such that  $d_i = d, c_i = c, g_i = g, t_i = t$ . In practice, it is of course possible to open multiple accounts with a single institution, in the same currency and with

---

1. This should come as no surprise, as the notes are discrete pieces of information with no physical aspects and no possibility, for example, of deterioration.

identical contract terms. Also in any practical implementation an account would have to be somehow identified. However, this is merely a technical detail. We have purposefully chosen registered notes as our formalization of money to capture all the possible legal aspects of accounts in  $t$ .

So far, we have only shown how an *increase* in balance on an account  $\langle d, c, g, t \rangle$  is represented—by a note  $\langle a, d, c, g, t \rangle_t$  in case of an increase by  $a$ . This, in current banking practice, would correspond to a receipt of deposit signed by the bank. But we also need to describe a *decrease* of balance. Today that would be documented for example by a payment order signed by the client. For simplicity, we can assume that a decrease in balance is again represented by a registered promissory note—for a decrease by  $a$  with  $\langle a, c, d, g, t \rangle_t$  (with the creditor and debtor swapped). It can be shown that  $\langle a, d, c, g, t \rangle_t$  and  $\langle a, c, d, g, t \rangle_t$  together have zero value to both  $c$  and  $d$ , so we can assume that such notes with counterparts would either be ignored or mutually acknowledged as settled. In this way, the balance of an account could be calculated as a difference between the sum of nominal values of notes issued by one party and the sum of nominal values of notes issued by the other party—in the same way as the balance of a bank account today is equal to the net of all credits and debits.

All of the above is aimed at maximum simplicity. But it should be noted that in theory, Ripple can be used to transfer *any* securities which can take electronic form, whose value is not affected by duplication, and which require a signature for validity. The only requirement that must be satisfied, with regard to feasibility of exchange, is that these securities be first presented in *unsigned form* (or mentioned in a message signed as a whole), so that a potential buyer can assess their value. It is likely that in case of a widespread adoption of Ripple-like systems, money and payments would be represented by special, standardized messages, instead of registered promissory notes, but the differences would not be vast, economically or legally.

In the current Ripple protocol, money is represented by **account-entry** messages, which are basically book entries (credits or debits) for a given account and informal acknowledgements of debt (“IOUs”). Their main difference from registered promissory notes is that they do not have legal power, and would require additional legal regulation to make obligations established in Ripple legally enforceable.

## 3.2 Network

A Ripple network can be formally represented in two different ways. The more natural and obvious one emphasizes the role of participating subjects and relationships between them, as well as the corresponding computer hosts and communication channels. The second one, closely related to the first, focuses on the role of money and enables us to describe the transfer of money similarly to a packet routing problem.

First, we shall model the Ripple network as a *multigraph*  $G_P = \langle P, A \rangle$  where the set of vertices  $P$  represents participants, or “Ripple nodes”, and the set of edges  $A$  represents

accounts. Note that we conveniently bend the classic definition of a multigraph: accounts are edges in the classical sense (unordered pairs of vertices) only after we abstract from  $\langle d, c, g, t \rangle$  to  $\{d, c\}$ ; only then the set of accounts becomes a *multiset*. We could more correctly describe the network as a *labelled* multigraph, but it would be at the expense of readability.

$G_P$  also reflects the topology of the computer network in which the Ripple network is implemented. For every participant, there must be a host performing the computations on their behalf, and for every account, a communication channel must exist between the two corresponding hosts. For the sake of generality, we are going to assume that every node has its own host, but in practice, multiple nodes may of course share a single host.

The second way to model a Ripple network is with *accounts*, not participants, represented by vertices. The directed graph  $G_A = \langle A, E \rangle$ , where vertices  $A$  represent accounts and arcs  $E$  represent *transitions between accounts*, can easily be constructed based on the first one: a pair of arcs (for each direction) exists between two vertices (accounts), if the edges corresponding to the accounts in the first graph share an incident vertex. For accounts  $\alpha_1 = \langle d_1, c_1, g_1, t_1 \rangle$  and  $\alpha_2 = \langle d_2, c_2, g_2, t_2 \rangle$ :

$$\langle \alpha_1, \alpha_2 \rangle \in E \Leftrightarrow \{d_1, c_1\} \cap \{d_2, c_2\} \neq \emptyset.$$

Naturally, every vertex of order  $o$  in  $G_P$  corresponds to  $o(o - 1)$  arcs in  $G_A$ .

By *transitions between accounts*, represented by arcs in  $G_A$ , we shall understand the conditions under which “money can be transferred” from one account to the other; or more precisely, conditions under which a transaction can be routed through the two accounts in the arc’s direction. Such a transfer constitutes a *transitive exchange* to the participant common to the two accounts and only takes place if it is profitable to the participant.<sup>2</sup>

Let us have two accounts  $\alpha = \langle d_\alpha, c_\alpha, g_\alpha, t_\alpha \rangle$  and  $\beta = \langle d_\beta, c_\beta, g_\beta, t_\beta \rangle$  such that

$$\{d_\alpha, c_\alpha\} \cap \{d_\beta, c_\beta\} = \{P\}.$$

To the arc  $\langle \alpha, \beta \rangle$  we shall assign an injective “credit-exchange” function

$$X_{\alpha, \beta} : \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow \{1, 0\}$$

such that  $X_{\alpha, \beta}(r, s) = 1$  if and only if  $P$  is willing to receive  $r$  balance in account  $\alpha$  in exchange for surrendering  $s$  balance in account  $\beta$ .<sup>3</sup> “Receiving balance” refers here to an

2. It should be stressed that understanding this process as “money being transferred” is very naïve, even if somewhat illustrative in a very specific case—when both of the accounts are *liability accounts* to the common participant, that is where this participant is the *debtor*. Then, naturally, the balance of the first account is decreased, while the balance in the second account increases. This is for example the case when the common participant is a bank and the payment goes from one its client to another. But if both accounts were *asset* to the common participant, that is if the participant were the *creditor* in both, the balance would *increase* in the first and *decrease* in the second. Similarly, for an asset—liability pair of accounts, it would be increase—increase, and decrease—decrease for liability—asset.

3. It is of course possible for two accounts to share both the creditor and the debtor ( $\{d_\alpha, c_\alpha\} = \{d_\beta, c_\beta\}$ )—an individual having both a current account and a savings account with one bank, for example. In this case, each of the parties would have a different credit-exchange function. In practice, however, one of them can always be deterministically selected depending on the context.

### 3. FORMAL REPRESENTATION OF A RIPPLE NETWORK

---

increase with an asset account or a decrease with a liability account. “Surrendering balance” means a decrease with an asset account or an increase with a liability account. With our representation of money by registered notes, that equals receiving  $\langle r, Q, P, g_\alpha, t_\alpha \rangle_t$  from  $Q$  in exchange for surrendering  $\langle s, P, R, g_\beta, t_\beta \rangle_t$  to  $R$ , where  $\{P, Q\} = \{d_\alpha, c_\alpha\}$  and  $\{P, R\} = \{d_\beta, c_\beta\}$ .

We can now properly define a payment transaction. A payment from  $A \in P$  to  $B \in P$  is routed along a path in  $G_P$ , representing a sequence of  $n \in \mathbb{N}$  accounts  $[\alpha_i]_{i=1}^n$  where  $\alpha_i = \langle d_i, c_i, g_i, t_i \rangle$ , and carried out by issuing a sequence of notes  $[\langle a_i, P_i, P_{i+1}, g_i, t_i \rangle_{t_i}]_{i=1}^n$ , where the following holds:

- $[P_i]_{i=1}^{n+1}$  is a sequence of participants corresponding to the accounts,
- meaning that  $\{P_i, P_{i+1}\} = \{d_i, c_i\}$  for  $\forall i, 1 \leq i \leq n$ ,
- $P_1 = A$  and  $P_{n+1} = B$ ,
- and  $X_{\alpha_i, \alpha_{i+1}}(a_i, a_{i+1}) = 1$  for  $\forall i, 1 \leq i \leq (n-1)$ .

At our level of abstraction, a curious question arises, which is: How much is being paid? Obviously,  $A$  is paying  $a_1 g_1$ , while  $B$  receives  $a_n g_n$ . It is similar to an international wire-transfer with automatic currency exchange. Since in the majority of cases today, transaction fees are covered by the payer, we shall refer to the nominal value *received by the payee* when speaking of the amount being paid, unless stated otherwise. Also, when discovering the path of payment algorithmically, the value received by the payee, not paid by the payer, would in most cases be fixed, and serve as a starting point for the algorithm.

Our credit-exchange function is not very practical for implementation, but fortunately, it has some useful properties stemming from the fact that it represents a value comparison. We can use these properties to adjust it for practical purposes.

It holds that if  $X_{\alpha, \beta}(r, s) = 1$ , then

$$\forall s' \in \mathbb{R}^+, s' \leq s : X_{\alpha, \beta}(r, s') = 1.$$

Naturally, if some party is willing to surrender  $s$  balance in  $\beta$  for  $r$  balance in  $\alpha$ , it would certainly be willing to surrender *less*. We can therefore define a “fixed-supply” credit-exchange function

$$X_{\alpha, \beta}^\triangleright : \mathbb{R}^+ \rightarrow \mathbb{R}^+,$$

such that

$$X_{\alpha, \beta}^\triangleright(r) = s_{max} \iff X_{\alpha, \beta}(r, s_{max}) = 1 \wedge \forall s' > s_{max} X_{\alpha, \beta}(r, s') = 0.$$

For any amount  $a$  offered through the account  $\alpha$ ,  $X_{\alpha, \beta}^\triangleright(a)$  is the maximum value that could be passed into account  $\beta$  in return, or undefined, if no such exchange is possible.

Similarly, when  $X_{\alpha, \beta}(r, s) = 1$ , then

$$\forall r' \in \mathbb{R}^+, r' \geq r : X_{\alpha, \beta}(r', s) = 1.$$



Based on this observation, we define a “fixed-demand” credit-exchange function

$$X_{\alpha,\beta}^{\triangleleft} : \mathbb{R}^+ \rightarrow \mathbb{R}^+,$$

such that

$$X_{\alpha,\beta}^{\triangleleft}(s) = r_{min} \iff X_{\alpha,\beta}(r_{min}, s) = 1 \wedge \forall r' < r_{min} X_{\alpha,\beta}(r', s) = 0.$$

Being a counterpart<sup>4</sup> to  $X_{\alpha,\beta}^{\triangleright}$ , for any amount  $a$  demanded in account  $\beta$ ,  $X_{\alpha,\beta}^{\triangleleft}(a)$  is the minimum value in account  $\alpha$  that constitutes an appropriate compensation, or is undefined if an exchange is impossible.

In a decentralized setting, where paths have to be discovered step by step,  $X^{\triangleright}$  and  $X^{\triangleleft}$  are exactly what needs to be calculated by nodes participating in the path discovery— $X^{\triangleright}$  when the algorithm is initiated by the payer,  $X^{\triangleleft}$  when by the payee. Of course, both approaches can be combined in a “meet-in-the-middle” fashion.

### 3.3 A simplified model based on credit limits

The model presented above is not, however, the one currently adopted by the authors of Ripple or reflected in the current draft of the protocol. It is rather a generalization thereof proposed by the author of this thesis. The differences are not fundamental, however.

Ripple revolves from the beginning around the idea of individuals granting credit to each other based on friend relationships, which are “symmetrical”. Therefore the original Ripple concept of an *account* does not really make a distinction between creditor and debtor. Both of the parties sharing the account are equivalent and the balance of the account can represent both an obligation of the first to the second as well as vice versa. It can be likened to a common example from the real world—a current account with overdraft, only with the same terms for both parties.

On the other hand, our concept of an *account* encompasses a *creditor* to whom the account is an *asset* account, and a *debtor* to whom it is a *liability* account. It also contains the *terms of payment*, a catch-all variable for the account’s legal aspects, and also for example the *interest rate*. With this model, a current account with overdraft would be represented by two accounts. For “positive balance”, an asset account to the client and liability to the bank, and vice versa for “negative balance” (actual overdraft). The accounts would most probably have different interest rates, too. It is true that going from “positive balance” to “negative balance” in one payment would not be possible unless the system supported forking paths of payment, but on the other hand, the implementation we are proposing does not really differentiate between creditor and debtor (payments in either direction take the same form of a registered note), therefore in practice, a “negative balance” would be possible.

4. We cannot assume much about  $X^{\triangleright}$  or  $X^{\triangleleft}$  with absolute certainty, but in practice, the functions would probably be non-decreasing, and therefore each other’s inverse.

Another concept that characterizes the original Ripple account is a *credit limit*. Every account has two credit limits, each decided by one party, serving as an upper bound to the balance that can be owed to them by the other party. We discussed the purpose of this limit in the last chapter. In our model, such a limitation could be easily implemented by properly limiting the range of  $X^\triangleright$  or the domain of  $X^\triangleleft$ .<sup>5</sup>

The importance of credit limits makes more sense when we consider the relative inflexibility of the (transitive) credit exchange that Ripple is supposed to allow according to the development wiki. [1, /Main/CreditExchange] Earliest sources indicate that finding paths would be limited to accounts denominated in a single currency and that participants would exchange only obligations of identical nominal value. Later documents do not mention this limitation, but still allow only the option of a *fixed exchange rate* between accounts—without a possibility to charge *flat* transaction fees, for example. Then, because creditors cannot mitigate the risk of default by flexibly valuating marginal debt, they must resort to limiting the total debt using the credit limit.

The difference can be nicely illustrated using  $X^\triangleleft$  (or similarly using  $X^\triangleright$ ). While we do not assume anything about the function, according to the Ripple wiki, users would be limited to defining it as a direct proportion

$$X_{\alpha,\beta}^\triangleleft(s) = e \cdot s$$

where  $e \in \mathbb{R}_0^+$  is the fixed credit exchange rate between accounts  $\alpha$  and  $\beta$ . The domain of  $X_{\alpha,\beta}^\triangleleft$  would be the interval  $(0, l - b]$ , where  $l$  is the credit limit on  $\alpha$  and  $b$  the outstanding balance.

The limitation of the credit-exchange function to direct proportion is quite restrictive. The limitations could be somewhat obviated by using multiple accounts between users, each with different terms (typically interest rate or maturity), but such a solution could be too complex for potential users to accept. The inability to charge flat fees could also be a problem, as the credit-exchange function should flexibly reflect transaction costs and risk exposure, which are not necessarily directly proportional to the amount of payment. Users would then have to choose between higher risk exposure with small payments or unnecessarily high fees with high-value payments. These restrictions could arguably be a big obstacle to Ripple's success, which is why we do not take them into account in this thesis, having a more general approach.

The assumption that credit exchange functions are direct proportions is, however, greatly taken advantage of in the current protocol draft. Under the assumption, it holds that if

$$X_{\alpha,\beta}^\triangleleft(s) = r$$

for an arbitrary pair of accounts  $\alpha, \beta$  and arbitrary  $s \in \mathbb{R}^+$ , then also

$$X_{\alpha,\beta}^\triangleleft(ks) = kr$$

---

5. A credit limit is of course a good way to let an individual user of Ripple conveniently configure the accounts with his friends, and as such it might be a useful feature in the human interface to Ripple. But it should not be a fundamental concept in the system.

for any  $k \in (0, 1]$ . (The same is true for  $X^\triangleright$ , of course.) Nodes participating in path discovery for a payment of a given value can use this fact to offer mediating a payment of a lower value, in case the full amount exceeds any credit limits. Because the currently proposed protocol allows payments along multiple paths, this feature is a necessity if such paths are to be found effectively. For details, see section 5.8.



## Chapter 4

### Core functionality of Ripple and its basic security aspects

In this chapter, the basic functions of Ripple are presented, as well as a brief overview of the most important security aspects relevant to them. These functions are to:

- set standards for communication between neighbouring nodes,
- enable the routing of messages between distant nodes,
- provide all data necessary to keep track of obligations between users (book keeping and accounting),
- discover possible paths of payment in the network,
- and coordinate payments along selected paths.

#### 4.1 Communication between neighbours

The choice of the basic communication channel between neighbouring nodes should be out of the scope of Ripple project itself. Enthusiast individuals would probably employ general purpose PCs and use the Internet to communicate, whereas businesses or financial institutions, being a high-value target, could opt for dedicated hardware and secure point-to-point channels. In any case, we should not make any assumptions about the channel and consider it insecure—with attackers able to eavesdrop on, intercept, emit, alter and replay communication. [11]

Securing the channel is of course in the interest of the owners of the endpoint nodes, but we should keep in mind that for distant nodes, who use the channel as well (although only aware of its existence, nothing else), the channel is untrusted by design, regardless of any security measures employed. *Confidentiality* is obviously desirable, as the information transferred is financial information, therefore highly sensitive. Mutual *authentication of nodes* is also needed, if it is not provided by the communication layer. And, as we will demonstrate, *non-repudiation of origin* must be ensured for some messages. Fortunately, when considering cryptographic means to satisfy these requirements, we can assume that the nodes' owners have a secure channel at hand for example for key exchange. When establishing the link between the nodes, the two parties are most likely entering a contract, which calls for a secure channel anyway. This secure channel would, of course, only be needed in the beginning and occasionally for example in cases of key expiry or key compromise, not during operation.

### 4.2 Communication between distant nodes

In most cases when distant nodes exchange messages, intermediary nodes are equal partners in the communication, not mere messengers. *Anonymity*<sup>1</sup> of the endpoints of the communication is desirable, so nodes must behave in the same way towards their neighbourhood regardless of whether they initiated the communication or not. The messages should be understood as tokens passed between neighbours, that are foremostly significant to the neighbouring nodes themselves. The messages are all related to path discovery and payment. In the path discovery phase, initiators of the algorithm (payer or payee) communicate with potential intermediary nodes in the network until a path (or paths) between the payer and payee is found, then the payer and payee coordinate the payment along one (or several) of the found paths.

We always assume that the payer and the payee have a secure channel outside the Ripple network at hand, and that they are able to use it for key exchange or any other purpose requiring authentication or possibly confidentiality. The existence of such a channel is absolutely necessary for payments to be feasible, as the network cannot ensure authentication of messages by itself.

The problem arises when we want to route a message to a distant node without revealing anything about the local topology of the network to other nodes. All the nodes in the network should ideally act as routers connecting as many subnetwork as they have neighbours, but be completely unaware of anything regarding the subnetworks themselves, from the structure of the connections to the mere number of hosts in them.

#### 4.2.1 Onion routing

In order for the nodes to be able to pass messages to distant nodes along exactly defined paths without learning the actual paths, *onion routing* can be employed. [6] In onion routing, every message carries with itself a recursive data structure known as a *routing onion*. Whenever a node in the network receives a message that should be routed further, the routing onion in the message is encrypted, and it can only be decrypted by the node itself. The decrypted routing onion consists of an identifier of the next node the message should be routed to and a smaller routing onion encrypted for this following node. This way, the routing onion is being “peeled” layer by layer, until the message eventually reaches its destination.

A routing onion can be created in two ways. First, it can be created whole by the sender of the message. In this case, the sender must be aware of all nodes in the path and their order, and encrypt every layer of the onion using the corresponding public key of each node. This is the case with *Tor* [10], the most popular implementation of onion routing, for example. Second, the routing onion can be created step-by-step by the routing

---

1. From the point of view of any observer node  $O$ , the Ripple network consists of several subnetworks, one for each neighbour of the observer node. By the *anonymity* of a node  $A$ , we shall understand the indistinguishability of  $A$  from any other node in the same subnetwork as  $A$ , to all observers  $O$ .

nodes themselves, if a message was sent along the same path in the opposite direction previously. In this case, every node adds another layer to the onion, and encrypts it for itself, therefore symmetric encryption can be used.

In Ripple, the first way cannot be used, unfortunately, simply because the routers themselves wish to stay anonymous.<sup>2</sup> Therefore we can only use onion routing for messages where some message has found its path from the recipient to the sender before. In other words, onion routing is used only during payment itself and during the later stages of path discovery, whereas the initial stages of path discovery must be carried out using a different scheme. After all, the word *discovery* already suggests that the paths are unknown in the beginning.

We must keep in mind that all the paths represented by routing onions can ultimately become a path of payment, which should specify exactly the accounts through which the payment is supposed to be done and the amounts on the accounts used for the payment. The routing onions must therefore contain not the identifiers of nodes, but the identifiers of specific accounts, so that no ambiguity is present in case two adjacent nodes in the path share more than one account. Every layer of the onion could optionally also include additional meta data about the path, such as the payment amount, although such information can easily be stored by the nodes in most cases.

Embedding payment-specific data in routing onions is closely related to their security. If the account identifiers and the encryption keys used to create the onions are persistent in time, and the layers do not contain any payment-specific information or random data (such as padding), then the routing onions representing a single path would be identical. This fact would enable nodes to make the link between payments and payment attempts originating with a single node, which is, of course, undesirable. To guarantee the *unlinkability* of payments, unique data should be inserted into every layer, such as nonces, random padding, or the aforementioned meta data. This unique data in the layer should ideally have a significantly varying size, in order to complicate any efforts to guess the length of the path from the size of the routing onion.

Unfortunately, the most desirable security goals—integrity of routing onions and the authentication of the endpoint node—cannot be achieved due to the very nature of how the routing onions are created. The payment sub-protocol successfully obviates this problem, but the path discovery mechanism is negatively affected in its vulnerability to denial-of-service attacks.

#### 4.2.2 Traffic analysis

Ripple should be considered a low-latency network, as it is theoretically supposed to enable point-of-sale payments, which can take seconds at most. Therefore the same problems with traffic analysis apply to Ripple as with other low-latency networks, such as Tor. A *global adversary* able to detect ongoing communication in a Ripple network with a low

---

2. Curiously, the Ripple wiki does mention using asymmetric encryption for routing onions, namely RSA encryption, but all proposed protocols assume building the routing onions using the second way. [1]

frequency of payment attempts may be able to identify the endpoints of the payment—the payer or the payee, whoever initiates the path discovery. Successful payments will also be easy to identify, as they require at least three passes of messages between the payer and the payee (at least one for path discovery and two for the actual payment), whereas unsuccessful paths will be traversed twice at most. The possible countermeasures that can be employed are not specific to Ripple; the techniques used in Tor can be used in Ripple as well—with all their downsides, of course.

### 4.3 Book keeping and accounting

Ripple must help keep a secure record of past transactions on two different levels. The higher level is the traditional payment level—keeping record of committed transactions for the benefit of payer, payee, and possibly other parties, such as government authorities. (For details, please refer to section 4.6.) The other level is the account level—keeping record of exchanged notes between neighbouring participants and tracking their obligations to each other.

In today's banking, book keeping and accounting reflect the hierarchical structure of the system. Banks act as trusted<sup>3</sup> accountants of their clients' accounts and central banks are trusted accountants for commercial banks. This means that both the central bank and commercial banks are actually in charge of recording *their own* obligations. This is only possible thanks to the legal and regulatory framework of banking.

Ripple can (and should) take advantage of this existing security framework, but it must also allow for an arbitrary (not just hierarchical) arrangement of users. It should therefore provide both parties sharing the account with the evidence of existing obligations—or more precisely, each party with evidence of the other party's obligations, because we cannot assume any incentive to record own obligations. The creditor must therefore be able to prove to the debtor that the balance of the account is *at least* what he claims it to be, and the debtor must be able to prove it is *at most* the amount he claims. To enable the participants to do this, the *integrity, authentication* of the issuer, and *non-repudiation of origin* must be ensured for every note in the system.

The above are the only security requirements we need to consider for book keeping and accounting in Ripple itself. They can be easily provided for by cryptographic means, such as a simple *digital signature* scheme. Signed notes can then serve for mutual accounting, as well as credit or debit entries in book keeping.

For legal purposes, the notes should satisfy a few conditions. The terms of payment must be either explicitly stated in the note or the note must refer to an existing contract. The identity of the issuer and the authentication method must be recognized by the tribunal in whose jurisdiction the note is issued. Using, for example, a certificate by a government-accredited certification authority in order to be able to take disputes to a

---

3. Not always absolutely trusted—for example, account statements signed by the bank can serve the client in case of a dispute with the bank.



government court might be useful. However, storing a general-purpose private key to such a certificate on a server hosting the Ripple node would be risky in case of own server, and outright impossible with a server operated by a second party.

As we already mentioned, the currently proposed protocol for Ripple operates with `account-entry` messages instead of promissory notes. Such a message constitutes a book entry for both parties involved and establishes an informal commitment to settle the corresponding debt. The legal aspects are not discussed in the existing literature on Ripple.

### 4.4 Path discovery and payment

Discovering the payment path and coordinating the payment itself are very closely related tasks. Theoretically, they could be performed together in a single step, which could be used to make the nodes financially accountable for their behaviour during path discovery, but it is better for many practical reasons to separate the tasks and carry them out consecutively.

First, the path discovery task can yield several results with varying consequences for the payment task. If no path between the payer and the payee is found, the process should be terminated at once, without the need to notify any other nodes that participated in it. In the case when multiple paths are found, then after the payer or the payee selects one (or finds all paths unacceptable), the nodes that are not situated on the selected path should not require (and wait for) any further communication. If path discovery and payment were carried out in a single step, some nodes could exchange legally binding messages which would then have to be revoked, in case the nodes in question were not on the selected path. Introducing such unnecessary complexity into the protocol would make it much harder to design it as secure.

The separation of path discovery and payment has more benefits with regard to security. As with any payment system, the wealth of the participants is the most important asset that must be protected. By separating path discovery, we are left just with the payment task—relatively simple and only involving a limited number of parties—to apply the corresponding security measures to. Ensuring that malicious behaviour can lead to financial loss only in the payment step also leaves much more freedom in the design of the path discovery mechanism.

Carrying out path discovery and payment in two separate steps also has its drawbacks, however. Because the messages exchanged during the path discovery part are not binding and because there is a time difference between path discovery and actual payment, it is possible for a node to refuse participating in a payment even if it agreed to do so during path discovery. The preferences of the nodes can change in the meantime for many different reasons. Typically, the balance on an account can change as a result of another payment, making a previously requested payment through that account impossible.

On one hand, such behaviour can be completely legitimate and the protocol should allow it, but it also enables malicious nodes to block payments. An attacker's node could offer an exchange rate extremely unprofitable to itself, therefore extremely profitable to the initiator of the payment. The cost-minimizing initiator would then most probably select a path containing the malicious node, who would consequently cancel the payment. It is not a critical vulnerability, as the initiator could continue by trying the second-best path, but it is still a waste of time, increasing the time difference between path discovery and payment for paths selected later. This in turn increases the probability of payment failure—even for paths only including honest nodes. But what is worse, it severely limits the options of allowing the nodes participating in path discovery to prune sub-optimal (not least expensive) paths. Even relatively expensive paths should be returned to the initiator in order to ensure that at least some of them circumvent a potentially malicious node.

The current protocol draft tries to minimize the occurrence of the incidents where an *honest* node cancels a payment it previously agreed to participate in. It does so by introducing a *credit guarantee time limit* until which the amount of money needed to carry out the payment is “frozen” in all accounts on the path. [1, /Main/Protocol] This means that for a limited period of time the nodes on the path reject any other payments “incompatible” with the payment in question—for example, payments after which the original payment would result in a negative balance on some account (for unilateral accounts). The actual length of the time limit would be comparable to the time needed to perform path discovery and optimal path selection—probably seconds in practice—, but the feature could still be abused for a denial-of-service attack. An attacker controlling one or several nodes could easily start initiating many instances of the path discovery task (without the intention to continue with the payments, of course), effectively “freezing” all the credit in the network and making payments impossible. Therefore, it is arguably better not to implement this feature at all.

### 4.5 Path discovery

Discovering potential paths for a payment is the core functionality of Ripple, and also the most challenging to implement. In a centralized setting, when the whole network is on a single server, it is trivial, but when the Ripple network is distributed among many servers, finding paths effectively and protecting the privacy of users become conflicting goals. As this matter is at the heart of this thesis, we shall save a more in-depth analysis for the following chapter, providing only a basic introduction to path discovery in this section.

Path discovery for a given payment is performed by executing a distributed algorithm initiated by the payee, the payer, or possibly both. The inputs are, respectively:

- the minimum amount to be received—provided by the payee for each of his or her accounts,

- or the maximum amount to be paid—provided by the payer for each of his or her accounts,
- or both.

The corresponding outputs are:

- a set of paths from the payer to the payee represented by routing onions, each with the minimum amount to be paid—returned to the payer,
- a set of paths from the payee to the payer represented by routing onions, each with the maximum amount that can be received—returned to the payee,
- or one of the previous outputs—returned to whoever is interested in the output. (Typically the payer, as we will show in the following section.)

We shall now present a very simplistic version of such an algorithm as an example, and also as a starting point for discussing the security aspects of path discovery in Ripple. The algorithm is aimed at maximum simplicity and protection of user privacy—to the point of being unusable in practice. The algorithm’s pseudo-code below corresponds to the first case, the *payee-initiated* path discovery.

Payee — Init:

```

for all  $\alpha \in \text{Accounts}$  do
  send  $\langle \text{path-request}, a_\alpha, E(\emptyset) \rangle$  to  $\alpha$ 
end for

```

Node — On receipt  $\langle \text{path-request}, a_\alpha, \text{onion} \rangle$  from  $\alpha$ :

```

for all  $\omega \in \text{Accounts} \setminus \{\alpha\}$  do
   $a_\omega := X_{\omega, \alpha}^{\leq}(a_\alpha)$ 
  if  $a_\omega \neq \perp$  then
    send  $\langle \text{path-request}, a_\omega, E(\text{onion} || \alpha) \rangle$  to  $\omega$ 
  end if
end for

```

Payer — On receipt  $\langle \text{path-request}, a_\alpha, \text{onion} \rangle$  from  $\alpha$ :

```

 $\text{Result} := \text{Result} \cup \langle a_\alpha, \text{onion} \rangle$ 

```

Payer — Code:

```

repeat
  wait
until timeout
return Result

```

(*Accounts* represents the set of accounts of a single node here. It constitutes the neighbour relation between nodes, while reflecting the possibility of multiple accounts existing between two nodes.)

A similar algorithm could be easily designed for the second case—when the path discovery is initiated by the payer. The payer and the payee would simply swap roles,  $X^\triangleright$  would be calculated instead of  $X^\triangleleft$ , and the payee could respond to the payer along any selected path. For the third case, we would have to introduce a flag specifying whether a **path-request** message originated with the payer or the payee and which one of  $X^\triangleright$ ,  $X^\triangleleft$  should be calculated. Nodes at which the **path-request** messages from both directions meet would then have to respond back to the payer and/or the payee using the partial paths they have learnt. (See section 5.4.) All three cases are *fundamentally* similar, however.

The algorithm presented above suffers from several major issues, which need to be resolved in the design of any practically usable algorithm:

- Multiple instances of the algorithm cannot run concurrently in a single network without the identification of payment or payee.
- It only returns the whole set of all available paths under the assumption that all computations and communications run in zero time and the *timeout* is greater than zero. In practice, it cannot be verified whether the resulting set is complete—including the case when an empty set is returned.
- If the network contains circles, **path-request** messages may theoretically circulate without end. Moreover, the algorithm is not sound in this case, as the output may include invalid paths containing circles.
- A *broadcast* routing scheme is used, which is not applicable to large, highly connected networks in practice.

The first three issues can be solved easily using various approaches, but all of the solutions have an impact on security. The last point—the need to find or design a scalable and effective routing scheme considerate of privacy—has not been satisfactorily solved yet, and remains the greatest obstacle to finalizing the design of distributed Ripple.

The current Ripple protocol does specify a large part of the path discovery algorithm. It takes advantage of the linear credit exchange of the current model, which makes it possible for nodes to offer paths for payment of a lower amount, if a path for the full amount of the payment cannot be found. [1, /Main/Protocol] The payment can then be carried out along *several paths*, even forking and joining ones. The algorithm uses a *unique payment identifier* to distinguish between messages for different payments, and avoids circles by recording the set of visited nodes using a *Bloom filter* in the **path-request** messages. These features will be discussed in greater detail in Chapter 5.

## 4.6 Payment

A Ripple payment is basically a distributed transaction, and it is highly desirable to ensure its atomicity and isolation. It encompasses changing balances on several accounts, which by themselves are atomic operations. Ensuring atomicity of the whole payment is

quite difficult, however. In other applications, a distributed transaction is usually coordinated by a trusted transaction manager. This is, obviously, not an option for Ripple, as the participants in a payment do not necessarily share any trusted party and are not supposed to enter into any relationships with anyone except for their immediate neighbours. Ensuring isolation of payments is also problematic, as it can make the system vulnerable to denial-of-service attacks. Regarding isolation, the current Ripple payment protocol tries to reach an optimal balance between isolation and availability. As for atomicity, the protocol does not support the traditional “roll-back” mechanism on payments, but rather focuses on clearly assigning responsibility for payment failure to participants.

We shall now present the payment sub-protocol, explaining the necessity of its features, along with their security aspects. We assume that path discovery has successfully taken place, and that the payer or the payee has selected the optimal path. The routing onion representing the path can be used to route messages from the payer to the payee and back, and in addition to that, the payer and the payee have a secure channel outside of the Ripple network at hand. At first, we are only going to consider payments along a single, undividing path.

Let the payment path consist of the payer  $P = I_0$ , the payee  $Q = I_n$ , and  $n - 1$  intermediaries  $\{I_i | i \in [1, n - 1]\}$ , all connected by  $n$  accounts  $\{\alpha_i | i \in [1, n]\}$ , such that  $\alpha_i = \langle I_{i-1}, I_i, g_i, t_i \rangle$  (or  $\langle I_i, I_{i-1}, g_i, t_i \rangle$ , as the creditor/debtor roles do not matter). The actual payment comprises  $n$  transfers of value for each of the accounts involved:  $I_i$  transfers  $a_i g_i$  to  $I_{i+1}$  for  $\forall i \in [0, n - 1]$ .

Because communication between nodes can only take place along the payment path, the individual transfers should take place sequentially in the order in which the accounts are situated on the path, either in the forward (payer to payee) or backward (payee to payer) direction. While it seems natural to choose the forward direction, the design of Ripple does not allow it. Let us assume the transfers take place in the forward fashion— $P$  transfers  $a_0 g_0$  to  $I_1$  first, then  $I_1$  transfers  $a_1 g_1$  to  $I_2$ , etc. If an intermediary node is first given the value, before it is supposed to transfer it further down the path, a strong incentive is created to disrupt the payment and keep the value. Let us say  $I_{k-1}$  transfers  $a_{k-1} g_{k-1}$  to a malicious node  $A_k$ . If  $A_k$  stops cooperating after this step, none of the neighbouring intermediaries are financially motivated to resolve the payment failure.  $A_{k-1}$  finished its credit exchange and is “even”, and so is  $A_{k+1}$ , which did not receive or surrender anything. Not even can the nodes claim with certainty that  $A_k$  disrupted the payment, as the nodes before  $A_k$  (closer to the payer) can only communicate with the nodes after it (closer to the payee) through  $A_k$ . What is most important, however, is that the payer was harmed by an anonymous distant node, which goes directly against the basic design of the system. We must therefore choose the opposite order of transfers— $I_{n-1}$  transfers  $a_{n-1} g_{n-1}$  to  $Q$  first, and  $P$  transfers  $a_0 g_0$  to  $I_1$  last. In this case, if an intermediary node does not transfer the expected value, it only harms the following intermediary node to which it is trusted.

Having the intermediaries surrender the required value before they receive anything in return makes acting as an intermediary risky, of course. For this reason, the protocol

introduces the concept of a *promise*. All intermediaries receive from the preceding intermediary a conditional promise (a `promise` message) to transfer value to them when they present a proof that they have transferred value to the following intermediary.<sup>4</sup> This in turn is done by producing a `receipt` message proving that the rest of the payment has successfully taken place. An exchange of the the promise and and the receipt messages should always take place between any two intermediaries before value is transferred between them. [1, /Protocol/Payment]

Both the promise and the receipt have an additional purpose. The promise also serves as an assurance of isolation, as it should “freeze” the respective amount on the account between the two nodes. After issuing the promise, all nodes should turn down all path requests for payments “incompatible” with the ongoing payment. It is very similar to the case described in section 4.4, only now the nodes are made accountable for freezing the credit by the binding nature of the promise. The receipt message serves as a proof to the payee that the payment has been made, .

The proof that the intended payee has been paid is performed using a digital signature. The payer and the payee establish a communication channel outside the Ripple network. The payee generates a pair of temporary signing keys and sends the public key to the payer through the secure channel. The payer then sends a promise to the first intermediary, along with the required receipt (unsigned) and the public signing key. The first intermediary then sends all of this to the second and so on until the last intermediary sends it to the payee. The payee responds with the provided receipt message, now signed with the private key. After the last intermediary node validates the signature on the receipt, it transfers the given value to the payee. Then, it presents the receipt message to the previous intermediary, and so on until the first intermediary presents the receipt to the payer and collects the value. Repudiation of the receipt of the receipt (this is not a typographical error) by one of the intermediaries is the only direct attack leading to a financial gain in the system—one that the design of the system purposefully does not address, as the attacker is trusted to the following, harmed intermediary.

The process does not strictly require synchronous communication outside of Ripple. The signing keys can be generated beforehand and sent (or even published) to the payer along with a unique payment ID. A payment completely without the payee’s cooperation is, however, impossible. Also note that the security of the payment from the perspective of the payer is determined by the security of the outside communication channel.

The main problem of this scheme is the “freezing” of balances, the fact that the promises make a part of the available credit temporarily unavailable for another payment, as all intermediaries are motivated to do so by having committed to fulfil the promise. If a payment fails as a result of malicious behaviour or network communication error, some promises cannot be redeemed and they block subsequent payments. For this reason, the protocol specifies exactly how promises start to degrade in value after a specified time<sup>5</sup>, until they eventually become worthless. In fact, what the protocol describes is a new

---

4. The promise can also be understood as a *forward contract* to buy the receipt, if it is handed over.

5. Of course, there is a possibility of time disputes, which cannot be easily resolved without a time-stamping authority.

*financial security* specifically tailored to the purposes of Ripple payment. It is arguably better to take a “laissez-faire” approach and make the protocol oblivious to what exactly is the content of a promise. After all, the harm in case of an error or attack is always contained between mutually trusted parties, who can enforce their own rules upon each other.

On the other hand, the existence of promises offers a significant deal of flexibility in regard to the representation of money in the system. Because the promise can specify exactly (in unsigned form) what will be provided as the fulfilment of the promise, any financial security (or any information that is only valuable when signed) can be used for payment.

An interesting feature of the system would be the possibility of routing a payment over several paths in parallel. The only problem is that the atomicity of such a multi-path payment cannot be effectively ensured. Such a payment would essentially consist of several single-path payments, and if one of them failed, the payer would have no way to undo the other ones. A more important question, however, is how the paths for such a multi-path payment should be found. See section 5.8 for details.





## Chapter 5

### Security and effectivity of path discovery

In the previous chapter, we have presented a simple path discovery algorithm, which we will now use as a starting point for discussing the security and effectivity of path discovery in general. The algorithm employed a trivial broadcast routing scheme and did not include as much as a payment identifier or a mechanism to avoid looping in circles. Any practical path discovery algorithm will have to include a number of such features to be effective, but we must keep in mind that anything we add for effectivity or convenience is most probably at the expense of privacy. The privacy requirements are of course detrimental to the accountability of users, which makes the detection and repression of malicious behaviour an almost impossible task. The design of the protocol must therefore focus on prevention, minimizing the incentives and opportunities for malicious behaviour.

#### 5.1 Security goals

We shall now list several security goals for the path discovery algorithm. They cannot be satisfied all at the same time, however. In practice, it is most probable that some of them would have to be sacrificed for effectivity. Some of the privacy goals can be taken to extreme measures, but we should not neglect even the seemingly “paranoid” requirements, as even small parts of information can be combined and abused. In practice, the users would most probably stick to standardized contracts (equivalents of current accounts) and to some degree recreate the current money system structure with banks playing a prominent role in the system. Nodes would therefore have some information about the local topology. The inability to perform a payment of a given value hints at the balances of the accounts of other nodes. All this information can be valuable for potential adversaries.

In practice, protecting the privacy of users will, however, not be as difficult as maintaining the availability of the network. Attackers can gain control of several nodes in different places in the network and use them to launch denial-of-service (DoS) attacks.

##### 5.1.1 Circumventing trap paths

One of the two basic DoS attacks which can be expected during path discovery is what we shall call the *trap path* attack. The strategy of the attacker is to insert a path containing

a node under the attacker's control into the set of paths returned by the algorithm and make the party responsible for selecting the actual path of payment select that particular path (by offering a low price). The path may or may not be a valid path of payment connecting the payer and the payee, as the paths are not authenticated and their validity cannot be verified. In case the trap path is selected for payment and the payment transaction is initiated, the attacker's node disrupts the transaction as soon as it receives a `promise` message. It can either cancel the payment according to the protocol, or it can cease to communicate and wait until all time limits specified in the promise expire. Cancelling the payment does not indicate maliciousness, as such behaviour can be completely legitimate. On the other hand, pretending that a communication error occurred may raise some suspicion, but it is difficult for upstream nodes to prove which node in particular is responsible for the error, as the attacker can blame nodes further down the path of payment for the failure. This attack not only blocks the payment in question, but keeps the credit reserved for the payment unusable for other attempted payments.

Without the ability to protect the integrity and authentication of routing onions, the only countermeasure to this type of attack is to let the path discovery algorithm return as many alternative paths as possible, so that payment can continue using other paths. However, this requirement severely complicates implementing some features which may be useful to the algorithm, such as depth-first search or pruning by price.

### 5.1.2 Prevention of flooding

Another typical DoS attack expected during path discovery consists of flooding the network with `path-request` messages. A single `path-request` message originating at a node under the attacker's control can easily translate into many more messages through the replication inherent to the algorithm. The goal of the attacker is to generate as much messages as possible using his own limited bandwidth. The design of the protocol must naturally strive for the opposite, while maintaining a sufficient spread of the messages needed for successful path discovery.

This attacks can, of course, be initiated from multiple nodes scattered in the network at the same time, in order to prevent nodes from finding paths around the affected area. Failure to prevent this attack would lead to the unavailability of the communication channels and draining of the computational power of the hosts, due to the cryptographic operations that need to be done with every message.

### 5.1.3 Confidentiality of connections

The connections in the network should to a maximum degree remain unknown to all participants. Ideally, all nodes would only be aware of the accounts they have with neighbouring nodes, unsure about the connections of nodes two or more hops away. However, even the most simplistic path discovery algorithm aimed at maximum privacy discloses partial information about the network to potential adversaries.

From the point of view of every node, every neighbouring node represents a subnetwork of the Ripple network, consisting of such nodes that a path exists between them and the corresponding neighbouring node. If a node initiates a path discovery algorithm by sending a `path-request` message to one of his neighbours and then receives a `path-request` message (belonging to the same attempted payment) from another neighbour, it is clear that the corresponding subnetworks are connected. This may not be a big problem in general, but if the algorithm prefers shorter paths, such a fact may hint at the existence of an account between the neighbouring nodes themselves. Depending on other features of the algorithm, such as circle avoidance, any node participating in the path discovery—not only the initiator—may be able to learn about the connectivity of the subnetworks.

The design of the path discovery algorithm necessarily encompasses many trade-offs between effectivity and security. In this regard, the confidentiality of connections may be the security goal most likely to be sacrificed, at least partially. After all, most people today do not worry about disclosing the fact that they have an account with a certain bank or a payment system. The connections between financial institutions or some big organizations are even public by law.

#### 5.1.4 Anonymity of users

As we already mentioned, from the point of view of any node, all other nodes (except for his neighbours in specific situations) should be indistinguishable from all other nodes in the same subnetwork. This seriously limits our options of using network-wide identifiers of nodes, which in turn makes traditional routing schemes difficult to use for our purposes. A partial solution may be using a single public identifier for routing purposes, a *routing ID*, for a set of nodes. Messages with this routing ID specified as a destination would either be routed to a single significant node in the set, or an arbitrary node in the set, and continue along a short path to its real destination. This solution would be particularly reasonable for nodes on a single host or nodes of the clients of a single financial institution. The anonymity of nodes would then be diminished to the indistinguishability from other nodes in the set sharing the routing ID.

An opposite approach, having multiple routing IDs assigned to each node and using each ID only once could also be useful. This way, the unlinkability of payments or path requests would be ensured even with a routing scheme in place. However, the overhead created by the need to constantly update the routing tables with new, unused routing IDs would arguably be unacceptable in practice, not to mention the ability of adversaries to use this feature to flood the network.

Another potentially useful feature of the path discovery algorithm that could diminish anonymity is limiting the distance to which messages may get away from their source—either to avoid circles or to avoid long paths, which would be deemed suboptimal. If such an optimal maximum distance is standardized and all nodes follow the protocol, the ability to find a payment path puts an upper bound on the distance be-

tween the payer and the payee. Nodes participating in path discovery would also be able to learn their distance to the initiator, which is particularly troubling in case the distance is one hop and the initiator is thus immediately identified.

### 5.1.5 Confidentiality of account balances

Keeping the existing financial obligations (and account parameters in general) confidential is an important security goal that must be considered in designing the path discovery algorithm. After all, for every path, the ability to accommodate a payment of a given amount follows directly from the balances of the accounts. If we disregard the transaction fees and potential currency conversions and assume that at least one of the accounts is unilateral and the payment goes in the direction from the creditor to the debtor, then the maximum amount of payment is lesser or equal to the balance of the account. Despite the fact that payment paths are confidential, any node with enough knowledge of the local topology could, in specific cases, identify the account most likely to set the limit for a payment amount along a given path. When a path discovery for a payment of a given amount is initiated in that specific part of the network, such nodes could learn whether the payment amount was greater or lesser than the balance of the account. An active adversary could even try to approximate the balance of the account by trial and error—initiating the path discovery algorithm multiple times with varying payment amounts and checking whether the path is found or not. In general, such an analysis would be complicated due to the existence of multiple paths and transaction fees, but in special cases, it may be a problem.

Unfortunately, this issue cannot be effectively resolved, as it is an integral part of the basic functionality of Ripple. We can only try to limit the opportunities to learn the account balances, when the conditions allow that. For example, the capacity of paths could be verified during path discovery only after the connection between the payer and payee is found. This way only the nodes on a path connecting the two nodes would be able to learn the attempted payment amount.

Closely related to this issue is also the proposed ability of the path discovery algorithm to offer paths with their maximum capacity in case the payment amount exceeds it. See section 5.8.

## 5.2 Connectivity and capacity

As we noted previously, there are two properties of paths between any two nodes in the network. One is the very existence or nonexistence of a path between the nodes—*connectivity*. The other is the ability or inability to make a payment along the path converting a given amount on the payer’s account to a given amount on the payee’s account—*capacity* (for lack of a better term; distantly related to “capacity” in flow networks). Connectivity is a long-term property stemming from the topology of the network. It only

changes when new accounts are established or old accounts are closed, but it is also occasionally affected when communication channels or hosts are unreachable. Capacity depends on current account balances on the accounts on the path, so it can change quite rapidly—with every completed payment going through any one of the accounts in the path.

Because the capacity of paths always changes, it is not feasible to maintain the information about capacities up-to-date (for example in some kind of routing tables). It is also not desirable, as the capacities follow directly from balances, which should remain confidential. And more, updating such information distributed in the network could disclose the fact that a payment was made using a given account and the amount of the payment.

Connectivity, on the other hand, is not such a sensitive information as capacity. Therefore a part of the topology could be publicly known, typically connections between financial institutions. Such nodes would have publicly recognized identifiers and paths to them (could be many) in the form of routing onions could be kept by individual nodes. The question how these paths are to be found and verified remains open, however.

### 5.3 Concurrent path discoveries

Because multiple path discoveries for different payments can take place in the network at the same time, it is clear that all messages corresponding to a single attempted payment must contain a unique identifier thereof, so that they can be distinguished from messages belonging to other attempted payments. Such an identifier is an absolute necessity for all of the advanced techniques described below. A direct consequence of its introduction is, however, that nodes will be able to learn about the existence of paths between their neighbours—the connectivity of the corresponding subnetworks.

In the distributed setting, there is no way how to ensure the uniqueness of the payment identifiers, so there will always be a possibility of conflict. Of course, such an incident would only lead to the failure of the path discovery part. If an invalid result is returned, it will always be discovered before an actual transfer of value takes place (at least with the proposed payment subprotocol). Nodes will be able to minimize the occurrence of conflicts by choosing the identifiers randomly from a large enough set or by caching and avoiding the identifiers they have received during path discoveries in the past.

### 5.4 Broadcast—convergecast search

In our simplified algorithm proposal, the payer or the payee initiated the algorithm and assumed that the `path-request` messages would eventually reach the other side. But neither of them could verify if all possible paths were tried and, in case no path was found, that none really exists. This problem can be solved by adding an additional step to the algorithm and letting the result of the search be reported back to the initiator.

First, the `path-request` messages would be sent outwards from the initiator, eventually spreading out along a spanning-tree-like structure<sup>1</sup>, then `path-reply` messages would converge along the same structure back to the initiator specifying whether the other endpoint was found or not. This approach is, of course, also used whenever the initiator of the algorithm is also interested in the result (covering the transaction fees and selecting the payment path) and when the path discovery is initiated simultaneously from both ends.

Using this approach, messages are sent both forward and backward for every existing path between the payer and the payee—`path-request` messages in one direction and `path-reply` messages in the other, depending on who is the initiator. This enables us to let the nodes check for the capacity of the paths only in one of the two passes. Preferably, the capacity would only be tested for *successful* paths (in terms of connectivity), so that only candidate intermediaries would learn the amount of the attempted payment.

When a broadcast or multicast routing scheme is employed, a choice must be made as to whether nodes should send all `path-request` messages at the same time (breadth-first search), or one account after another (depth-first search), waiting for a response from each account. Additionally, with depth-first search, a question arises if the accounts can somehow be ordered according to the expected chance of success.

The path discovery subprotocol proposed in the current protocol draft employs a depth-first search, where the nodes respond back immediately after receiving a positive reply from any account, ignoring the other accounts they would try in case of a negative reply. However, this feature may of course be exploited by attackers imposing the node which is searched for, because it effectively prevents the initiator from learning alternative paths. Depth-first search is therefore not the best choice and should be avoided.

### 5.5 Pruning by price

A useful feature of the path discovery algorithm would be the possibility for nodes to prune unpromising (too expensive) paths as soon as they are detected. After all, if the amounts requested for two different paths on one account are known to the node, they can be compared in a trivial manner. If we assume that  $X^{\triangleright}$  and  $X^{\triangleleft}$  are non-decreasing (a safe assumption indeed) for all nodes and all accounts, any node can identify the path which would be ultimately more profitable for the payer or the payee—depending on who is paying the transaction costs.

---

1. The algorithm is in this case basically a distributed graph traversal. As `path-request` messages spread into the network over time, they gradually build a set candidate paths. Theoretically, if the nodes forwarded the messages into all possible directions except for already visited nodes, the union of the candidate paths would eventually form a spanning tree of the network (which one would depend on the speeds of individual links). In practice, however, we are not only interested in reaching every node in the network, but in learning as many paths to every node as possible. For practical reasons, we also need to limit both the “breadth” of the structure using a routing scheme other than broadcast, and the “depth” using hop limits, for example. See the following sections for details.

The problem with pruning is that the communication during path discovery is asynchronous, which leads us to the question, when should such a comparison task take place and whether some messages must be delayed to make pruning possible. One solution could be using all time limits to their fullest, but that would undermine the very purpose of time limits. Another opportunity may be the waiting phase inherent to the broadcast—convergecast search described above. Nodes participating in this search wait for responses from all branches “downstream” they have contacted before replying “upstream” in the direction of the initiator.

Pruning seemingly sub-optimal paths has significant effect on security, however. Adversaries with a dumping strategy can carry out trap path attacks with or without pruning, of course, but pruning severely limits the opportunity of honest nodes to continue with alternative paths when they detect a trap path. This feature should therefore be only considered with great caution.

## 5.6 Limiting the number of messages

It is important to keep the number of messages that need to be exchanged during path discovery as low as possible. First, because bandwidth is expensive, and second, because sending every message encompasses at least one encryption operation for building the response routing onion and optionally a decryption operation for the forward onion, in case the ultimate target of the communication is known.

Apart from a more targeted routing scheme limiting the multiplication of messages, we should also limit the distance to which messages can diverge from their source. As a side effect, it prevents messages from roaming indefinitely in circles, but mostly it prevents the discovery of long paths. Such paths would in practice be more expensive than shorter paths, the payment itself would take longer—with a higher probability of timeouts, and more intermediaries always mean a greater risk of payment failure. This feature can be implemented by embedding a *hop count* in the `path-request` messages. The initiator of the algorithm would choose a hop limit if his liking, and all other nodes forwarding the path request would then decrease the hop count by one for all messages they send. Nodes receiving messages with zero hop count would refrain from forwarding the messages altogether. Such a feature does, however, have its consequences for security. If the hop limit is standardized or conventional, the actual hop count of a message sent to a node can hint at the distance of the node to the initiator. Additionally, in case the node in question happens to end up in the selected path of payment, the hop limit of the original message puts an upper bound on the distance of the node to the other payment endpoint. Both, of course, under the assumption that all nodes follow the protocol.

The total running time of the algorithm or its parts is also a measure we want to minimize. The longer the path discovery lasts, the greater is the probability that found paths will be rendered unusable due to other payments coming through the accounts in the meantime. We must also take into account that communication channels will fail sometimes, therefore some kind of time limits are necessary anyway, so that nodes do not wait

for responses indefinitely. Therefore it seems reasonable for `path-request` messages to contain a time-to-live information specifying the latest moment by which the initiators of the algorithm are willing to accept results and continue with the payment. Regarding security, if the time-to-live was somehow standardized and the latencies of the communication channels were known, the time could theoretically disclose some information about the path to the initiating nodes, but it is hard to imagine that it could be useful for a practical attack.

Both of these limits, the hop limit and the time limit, can also be used to counter flooding attacks. While the values should in general be respected by nodes, unreasonably high values could be a symptom of an attempted flooding attack. Nodes could therefore lower such impractically high hop limits or time limits into reasonable bounds on their own, or ignore the messages with such values altogether.

## 5.7 Avoiding circles

It is important to understand that, in theory, there is no reason why a path of payment could not contain circles. Actually, one of the types of payment typical in practice uses a circular path  $[A, B, A]$ —a client  $A$  transferring funds from one account to another (from a savings account to a current account, for example), both with a bank  $B$ . Circular debt could also be effectively resolved (from the point of view of one debtor) by making a circular payment to self. In general, circular paths for self-payments where the only node occurring twice in the path is the payer or the payee are similar to circle-free paths. The situation is a different, however, with paths where a mere intermediary node repeats in the path.

Circles in the network may pose a problem during path discovery for two different reasons. First, the `path-request` messages may loop in the circle infinitely (unless other countermeasures are employed), and second, a payment path containing a circle may reserve the same credit twice, unless the protocol prevents the occurrence of such conflicts. Infinite looping of messages in circles is not a significant problem, since many of the necessary features of the algorithm we have discussed above already solve this issue. Hop limits as well as time limits eventually stop the generation of new messages. Pruning by price also disadvantages paths with circles, as they suffer from a build-up of transaction fees. But even with these features implemented, we still need to address the second problem.

When nodes assess the conditions under which they are willing to act as intermediaries in the payment, they do so based on their current credit-exchange functions, which are in turn derived from the current balances on the accounts in question. For an intermediary node occurring twice in a payment path with the same account, the payment consists of two separate transitive exchanges. In the payment phase, the first transitive exchange will change the balance on the account, changing the credit exchange function for the second transitive exchange. The first exchange may even make the second one impossible. Therefore all nodes must be informed, when receiving a `path-request`



message during path discovery, whether they appear in the path back to the initiator or not—whether the candidate path already went through them.

This information must be explicitly stored in the messages, as it cannot be obtained from the routing onion for obvious reasons. The nodes have to save the information that an account is part of the path into every message they send, so that they can verify it later in case the message returns through a circle. Obviously, the information must not be readable by any other node than the one which saved it. The data structure holding this information should also not divulge any other information about the path, such as its length.

The current protocol draft proposes using a Bloom filter for this purpose [1]. Each account would be mapped to an  $m$ -bit string with exactly  $k$  bits set to 1, and these bit strings would be combined using bit-wise AND into an  $m$ -bit string constituting the Bloom filter. This way, the order of the accounts would be obscured, but all nodes would be able to learn the lower bound of the length of the path. Theoretically, a Bloom filter with  $K$  bits set to 1 would indicate a path of at least  $\lceil \frac{K}{k} \rceil$ . Practice is unfortunately even worse. With large enough  $\frac{m}{k}$ , the probability that two accounts would be mapped to values with a Hamming distance less than  $2k$  is relatively small, so any path of length  $l$  would, with a high probability, have a Bloom filter with  $lk$  bits set to 1 (or with a number very close to  $lk$ ). The  $\frac{m}{k}$  ratio must be kept high to prevent false positives, so that valid payment paths are not discarded by mistake. Therefore the nodes should also add a varying number of random values in each hop (especially before the first hop) to prevent other nodes from learning the exact length of the path. This would, however, raise the probability of false positives, of course.

In the opinion of the author of this thesis, using Bloom filters does not bring any particular benefits over, for example, a simple array of hashes of the account identifiers (or any random values that can be mapped back to the accounts by the nodes that created them). In case the values are not absolutely unique (to prevent the need to keep mapping tables in the memory of the nodes), the order of the values in the array can be simply randomized. Such a solution would be worse in revealing the upper bound of the length, rather than the lower bound, but this issue could easily be solved by inserting a varying number of random values in the same way as described above. This would, of course, introduce a possibility of false positives. The final solution should be probably decided primarily based on its performance impact.

## 5.8 Multi-path payments

In case a single path for a payment of a given value cannot be found, the currently proposed path discovery algorithm makes it possible for nodes to offer paths for payment of a lower amount, if the full amount of the payment exceeds their limits stemming from the account balance. The first problem with this approach is that it strictly requires all credit exchange functions to be linear. Otherwise the nodes cannot predict how much the other nodes in the path would charge for the payment. More importantly, however, we cannot

expect that nodes would be willing to disclose their limits for credit exchange that easily, as the limit can often be exactly equal to the balance of the account. Such information is, of course, sensitive, therefore the nodes could simply refuse to implement this feature.

Multi-path payment is, however, still possible. It only requires the payer or the payee to launch several runs of path discovery in a trial-and-error fashion until the required paths are found. The only issue left to resolve then is to specify in each path the payment identifiers of the other paths, so that the various branches do not reserve the same credit.

### 5.9 Existing routing schemes

Routing in the network in the later stages of path discovery and payment is left to onion routing, a scheme perfect for the purpose, but in the initial phase of path discovery, a routing scheme for the `path-request` messages would be useful, so that the whole network need not be traversed. Most traditional routing schemes designed for communication networks are, however, of little use for the purposes of Ripple, as they do not address the need for the local topology of the network and the parameters of the paths to remain confidential. Thanks to this, malicious behaviour in the building of the routing tables can also be detected and coped with in most cases, which would not be the case with Ripple. Additionally, these routing schemes often keep track of only a single optimal path between two nodes, whereas the routing scheme for path discovery in Ripple must be able to manage multiple (preferably many) paths between each pair of nodes.

An exception, at least in regard to confidentiality of connections, is the routing algorithm designed for the peer-to-peer *Darknet* network, part of the Freenet [3] project, by Ian Clarke and Oskar Sandberg [2][8]. (Note that this algorithm is being referred to as “Metropolis-Hastings routing” in the Ripple wiki [1].) The algorithm is specifically targeted at restricted-route networks with a small-world topology—where the average shortest path between two nodes is at most logarithmic in relation to the size of the network. This condition is satisfied by Ripple both in the friend-to-friend scenario without banks (social networks are small-world networks) and with the topology mimicking the current banking system (paths are mostly five hops long or shorter). In this scheme, nodes constantly maintain an abstract *position* value, regularly swapping these positions with distant nodes to minimize their *distance* (difference of positions) with their neighbours. The routing of messages is then carried out using a greedy algorithm minimizing the distance to the target. Such an algorithm is hardly usable in Ripple, where we need to find multiple paths. Adversaries trying to launch a trap path attack at a specific node could even take advantage of the location swapping to position themselves closer to the target. But apart from these objections, the routing algorithm was proven easily vulnerable to attack even from a small number of malicious nodes [7]. A similar attack in Ripple would not have deeper consequences as in Freenet (data loss), but it would nevertheless lead to the creation of ineffective paths, complicating or even disabling effective path discovery.

Two other routing schemes were at some time proposed for use in Ripple. The first is the hazy-sighted grouped link-state routing [1, /Protocol/GroupedLink-StateRouting].

The “grouped” property indicates the grouping of nodes into sets sharing a single routing identifier, and hazy-sightedness means that nodes are only aware of the local topology, not the whole network. The second scheme proposed for use in the algorithm is cell-structure routing [1, /Protocol/CellStructureRouting], using a hierarchical logical topology over the actual Ripple network. Both of these schemes, however, rely on establishing some sort of logical structure over the nodes, which cannot be easily accomplished in a fully peer-to-peer network. Most importantly, building such a structure requires the cooperation and honesty of all participating nodes, therefore such a process would be extremely vulnerable to byzantine errors leading to ineffective routing. The discussions of the schemes also include the advertising of account capacities into the network as part of the routing information. This should definitely not be required or allowed both for security and performance reasons. The capacities can often be equal to account balances, an information that should remain strictly confidential. Regarding performance, we should keep in mind that the capacity of the paths changes as payments are made in the network, therefore the capacities of the accounts would have to be constantly updated. Additionally, such updates would disclose some information about the payment which caused the update.

When the network is not fully peer-to-peer and contains connected hosts accommodating many nodes, such a property could, of course, be used to establish some sort of logical topology based on the structure of the hosts, making the aforementioned routing schemes more feasible. On the other hand, this arrangement could be used directly for a host-based routing scheme, as described in the following chapter.

## 5.10 Summary

The currently proposed path discovery algorithm has several flaws with regard to security. It implements the hop limits and the Bloom filters in a way that could disclose the length of the payment path and possibly identify the payer or the payee. The depth-first searching employed constitutes a trap path attack vulnerability. All the routing schemes proposed for use in the algorithm are prone to disruption by malicious nodes leading to ineffective routing.

The detailed discussion of the security goals and effectivity requirements laid down in this chapter hopefully illustrates the difficulty of designing a robust path discovery algorithm for an *arbitrary* network. The design of the algorithm must be tailored to specific cases, taking advantage of network topology and implementation details.



## Chapter 6

### Ripple as a service

So far we have made little or no assumptions about the practical aspects of the operation of a Ripple network, such as the security of the Ripple hosts, distribution of the nodes among the hosts, and the topology of the network. There can take a few forms in practice, each with far-reaching security implications. [1, /Main/DistributedArchitectures].

The first is the single server setting used in current Ripple implementations. In this case, all the nodes are located on single host. One of the main advantages of this solution is that it alleviates the need for any robust, distributed path discovery algorithm, enabling the host software to precalculate optimal paths between each two nodes, easily implement multi-path payments, and ensure the atomicity and isolation of payments. The main downside is the need for all users to trust the server operator, which ultimately limits the size of the network—even if many servers are in operation, the corresponding networks would be disconnected, which hardly fulfils the goals of the Ripple project. Placing the host out of the node owner’s direct control creates a need for a client—server interface, and opens up a whole new realm of the security of this interface. Clients would need to be able to issue payment commands, inform the server of incoming payments and their expected amounts, export the data about existing financial obligations (account statements) and set their credit exchange functions. Mutual authentication and the confidentiality of all communication will have to be ensured. What is most important, however, is how the legal enforceability of the financial obligations established using the system is maintained. Using signed promissory notes, a theoretical solution we proposed mainly for simplicity, is complicated due to the need to store the signing keys on the server. Unless the system is restricted to friend-to-friend accounts not requiring legal enforceability (the current work on Ripple goes exactly in this direction), additional legal regulation of Ripple would be needed, which might in turn make the system unattractive to potential users.

The second extreme is a completely peer-to-peer network with each node on its own host. Such a network would probably exist as an overlay network in the Internet, with the hosts being mostly personal computers or mobile devices *not dedicated* only to the purposes of Ripple. Such an arrangement is nicely simple in the fact that each user is responsible for his own host. All the legal aspects can then be left to the consideration of the users of neighbouring nodes. However, such a topology calls for a robust path discovery algorithm. As we have tried to illustrate in the previous chapter, designing such an algorithm as effective, scalable, considerate of privacy, and protective of the availability of the network is a difficult task. The most significant threat in the fully peer-to-peer network

are denial-of-service attacks. While the connections between nodes are generally established upon existing legal relationships, which prevents a massive entry of attackers into the network, the security of hosts would be a weak spot of the network. Arguably even a small portion of compromised hosts could be used to carry out a crippling denial-of-service attack on the network. Detecting the nodes responsible for the attack would very difficult, so large portions of the network may have to be disconnected in the attempt to counter the attack, possibly harming many honest nodes.

It is, however, quite unlikely that Ripple would be massively adopted in the peer-to-peer fashion. Users concerned about their privacy may choose to run their node on their own (hopefully dedicated) host, but the majority of users would settle for shared hosts. The situation would be similar to that of electronic mail. Most of the nodes will be serviced by a few (linearly less related to the number of nodes) servers, each with many nodes, while there will still be an open possibility for users to run their own server with a single node.

This hybrid solution should “take the best of both worlds”. The protocol must be able to handle a fully peer-to-peer network transparently (even if ineffectively), while taking advantage of the semi-centralization where possible. Node owners would have to bear the responsibility for the operator of their host in relation to the owners of all neighbouring nodes on other hosts. We cannot assume that the host operator will be in any legal relationship with anyone else than his clients. Of course, all the legal complications and the need for a secure client—server interface apply here as well as in the single server scenario. The basic idea is that banks and payment systems would run their own Ripple servers and offer Ripple as a service to their customers. In these cases the operator of the server would also be the owner of a central node connected to all the other nodes on the server. However, the server would also have to allow the client nodes to establish accounts with nodes on other servers, otherwise it would be pointless to use Ripple. Apart from these, the network could also contain servers without a central node. The advantages are obvious; there are less hosts that can be compromised, presumably most of them are dedicated and have better security. On the other hand, the servers hosting a large number of nodes constitute a high-value target.

### 6.1 Host-based routing

Hosts with many accounts are connected to other hosts by accounts having one incident node on one host and the other node on another. The hosts must, of course, have a communication channel between them for all of the accounts, but mutual authentication can be done merely on the node level. We can take advantage of the fact that many of such large hosts and their connections would be public, and use a routing scheme based on these host-to-host connections. The nodes on a single host would share a routing ID without the need for any unnatural grouping or leader election—the fact that they share the host would be used as a natural basis for the establishment of the routing entity.

The `path-request` messages can either be routed based on a destination host ID, in case such IDs are universally recognized among the hosts and they use some traditional routing scheme, or using a routing onion containing the host IDs of all hops, with each layer encrypted using the public key of the preceding host—using traditional onion routing with asymmetric encryption. Passing the path request on the host level would, on the node/account level, correspond to sending the `path-request` messages along all accounts connecting the two hosts, while the account-level routing onion would be created to represent the path back as usual.

This way, the sender of the messages (the initiator of path discovery) would remain anonymous in the traditional sense (see 5.1.4), whereas the recipient would be indistinguishable *at least* in the set of nodes on the destination host (unless the host-based onion routing is used). We must, of course, also enable routing to nodes on hosts that do not wish to be public. Nodes wishing to be routed messages to but not to disclose their location in the Ripple network would need to provide an ID of the public host closest to them, from where the messages would be routed using a broadcast scheme. That would, of course, be even more ineffective than in the peer-to-peer network due to the number of accounts, so the public hosts should enforce very strict hop limits on such messages. It would not be surprising if some of the host operators would forbid such blind routing altogether.

The advantage of this scheme is that it enables users to choose their own optimal trade-off between privacy and effectivity. We should, however, keep in mind that it is by no means a solution to the problems described in the previous chapter. It would merely allow building a bigger network (as for the number of nodes) than with the peer-to-peer architecture, due to a smaller number of hosts that can be compromised and theoretically a greater chance to localize attackers. However, the difference may be crucial to building a network large enough to attract users. Even if such a network would only be used by enthusiasts, with no legal enforceability of obligations, and mostly for non-serious purposes, the data that could be gathered from the operation of such network would be invaluable for further research.





## Chapter 7

### Conclusions

The main motivation for this thesis was the prospect of finding a simple, secure, and effective path discovery algorithm for a fully peer-to-peer Ripple network. This goal was, however, not achieved in the end. We have presented a number of difficulties the design of the algorithm would have to overcome, hopefully leaving future researchers with a good starting point for their efforts. We have undoubtedly repeated some of the undocumented work of the original author and contributors, but we also offered an independent analysis of the results of the design efforts until now, and investigated some previously unexplored possibilities. Apart from the work on the path discovery algorithm, we also tried to present a basic security analysis of the proposed system as a whole, and created an introduction into the economic theory of the system, in order to confront some recurring myths and misconceptions surrounding it.

The author of this thesis is left skeptical to the possibility of building a large, fully peer-to-peer Ripple network in the near future, although he would be delighted if proven wrong. Unlike the current contributors to the project, however, he sees no reason to restrain the effort to building merely a centralized version of the system. Even a poorly scalable path discovery algorithm can be used in building a network of connected Ripple servers for testing purposes. Attracting members of the academia and other enthusiasts—preferably including ones with malicious intentions—to participate in such a network with their own implementations would provide valuable data on the real-world distribution of nodes among hosts, and the real severity of the theoretical threats. Learning these facts could incite a new, more serious discussion about the system and ultimately lead to a better, more scalable design usable in practice.



## Bibliography

- [1] RippleWiki, home page of The Ripple Project. [<http://ripple-project.org/>] (last checked Dec 13, 2010).
- [2] Ian Clarke, Oskar Sandberg. Routing in The Dark: Scalable Searches in Dark P2P Networks, 2005.
- [3] Ian Clarke, Oskar Sandberg, Brandon Wiley, Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *International workshop on Designing privacy enhancing technologies: design issues in anonymity and unobservability*, 2007.
- [4] Jon Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective, 1999.
- [5] Ludwig von Mises. *The Theory of Money and Credit*. 1953.
- [6] Michael G. Reed, Paul F. Syverson, David M. Goldschlag. Anonymous Connections and Onion Routing. *IEEE Journal on Selected Areas in Communication Special Issue on Copyright and Privacy Protection*, 1998.
- [7] Nathan S. Evans, Chris GauthierDickey, Christian Grothoff. Routing in the Dark: Pitch Black. *23rd Annual Computer Security Applications Conference (ACSAC 2007)*, 2007.
- [8] Oskar Sandberg. Distributed Routing in Small-World Networks, 2005.
- [9] Pranav Dandekar, Ashish Goel, Ramesh Govindan, Ian Post. Liquidity in Credit Networks: A Little Trust Goes a Long Way, 2010.
- [10] Roger Dingledine, Nick Mathewson, Paul F. Syverson. Tor: The Second-Generation Onion Router. *Proceedings of the 13th USENIX Security Symposium*, 2004.
- [11] Roger M. Needham, Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21:993–999, December 1978.
- [12] Ryan Fugger. Money as IOUs in Social Trust Networks & A Proposal for a Decentralized Currency Network Protocol, 2004.
- [13] Ryan Fugger. Payment as a Routing Problem, 2006.