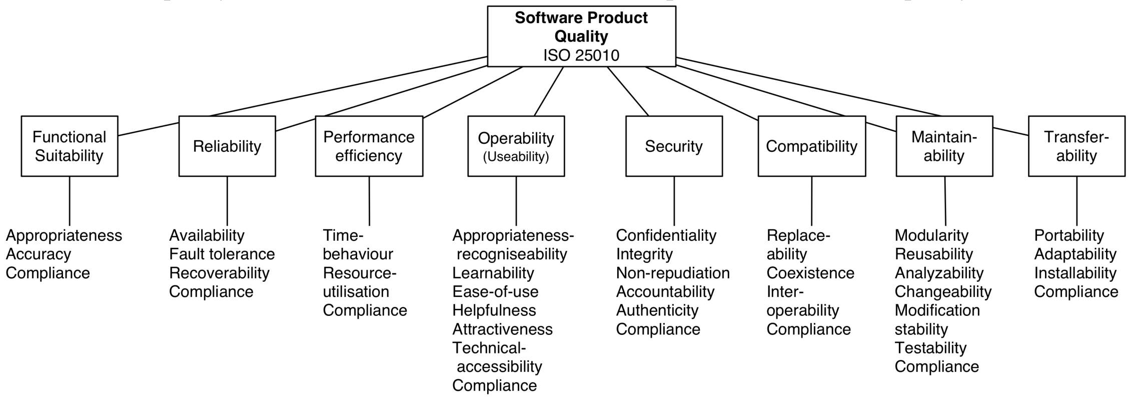
1 Assessing Ripple

In this essay, we will focus on the means to safeguard the quality and architectural integrity of the underlying system. First, We will overview the software quality characteristics. Then we will look into Ripple's GitHub repository to see if Ripple is applying any tests to achieve better software quality. Finally, we will discuss the analysis of Ripple's source code using SIG.

1.1 What do we mean by software quality

With key aspects of Ripple's architecture described previously, it is important to understand its software quality - specifically, what constitutes a good code and what are the different ways we can judge it. Even though quality is somewhat a subjective attribute that may be understood differently by different people, Hruschka and Starke[1] describe a quality tree checklist with which we can derive pointers to assess the quality of the



project.

Ripple does not have explicit documentation on how they measure the quality of their product. However, from their GitHub contributors Guidelines[2], test scripts and modules[3][4], and their travis-ci page[5], we were able to add the missing pieces together. Whenever new code is pushed into Ripple or its sibling repositories, the request triggers automated unit and integration tests, flow type checking, ESLint checks, automated testing of the documentation, as well as code reviews by several developers.

1.1.1 Main Checkpoints and Test Analysis

If we take a look at their test script file[6], we can find that it calls 14 manually written tests. Some of the tests that are compatible with the tree checklist mentioned above are discussed here—the `ripple.consensus.ByzantineFailureSim` script undergoes a fault tolerance test on the ledger. Every time the code belongs to the ledger is changed, this script invokes the Byzantine Failure Simulation and makes sure that the system will be reliable in production. Similarly, the `beast.unit_test.print` assess the functional accuracies of each component, and the integration test addresses the interoperability between the components. Since Ripple works on three main OS, the transferability of the product must be assessed. In Travis CI, we can find tests that assess the reproducibility of Ripple in Linux, macOS, and Windows

[1] https://docs.arc42.org/home/
[2] https://github.com/ripple/xrpl-dev-portal/blob/master/CONTRIBUTING.md
[3] https://github.com/ripple/ripple-lib/tree/develop/test
[4] https://github.com/ripple/rippled/blob/develop/bin/ci/test.sh
[5] https://travis-ci.com/github/ripple/rippled
[6] https://github.com/ripple/rippled/blob/develop/bin/ci/test.sh

systems. Whether or not Ripple assesses the other checklists mentioned in the diagram above is not clear because such information is not publicly available.

Rather than manually running all these tests, Ripple leverages the power of Travis Continous Integration service. Whenever a new push/pull request is submitted, Travis CI will check out the relevant branch and run the commands specified in the `.travis.yml` file, which usually builds the software and runs any automated tests. Ripple adopts a five-stage CI check that has 33 jobs in total. An infographic of the result of their CI test for March is shown below.

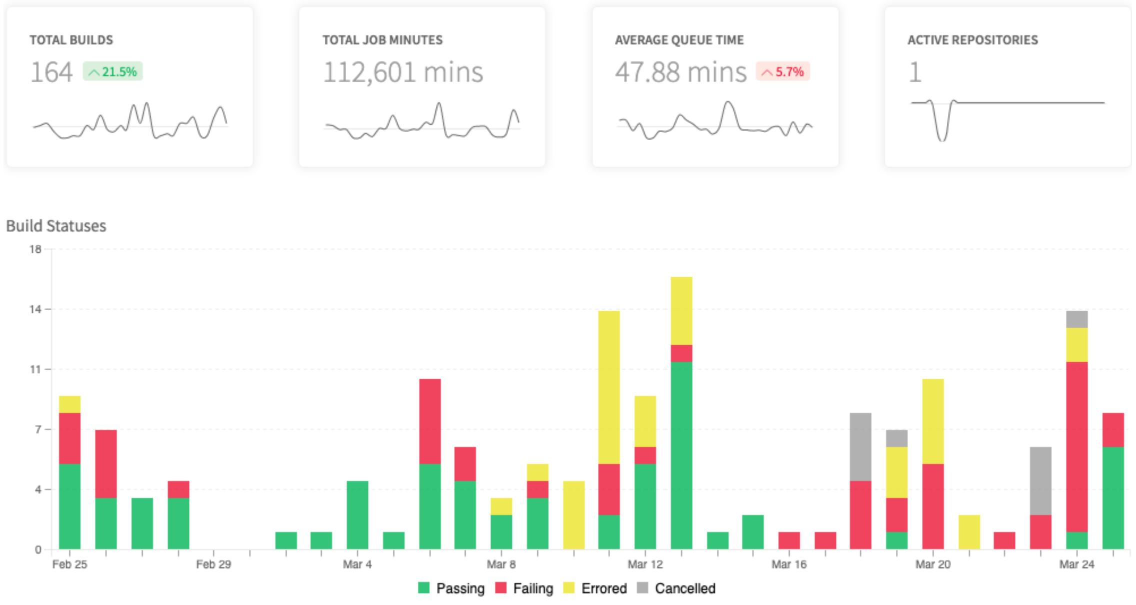


Figure 1: travisbuild

1.1.2 How much do they care about tests

From the infographic above, we find that most of the builds are either passing or failing. By comparing these builds to the several pull requests at GitHub, we find that failed or even error builds are not merged to the branch. This implies that the Ripple dev team takes these tests very seriously and will only merge if the CI tests pass.

From the Codecov site[7], we see that Ripple has 70% code coverage. Even though they measure it, we haven't found any discussion between developers where they address it. Even though there is a coverage bot that displays the code coverage for every pull request, the participants don't address it. One reason why it might be just 70% is that the Ripple repo contains code for all three major OS, and maybe some tests cases don't apply to a specific platform.

[7]https://codecov.io/gh/ripple/rippled

1.1.3 A mapping of recent coding activity on key architectural components

As we discussed in our architectural components before, we decomposed the system into three layers with several components. In this section, we look into how the Ripple team has brought them alive through code.

1.1.3.1 Blockchain layer

- 2020-01-13: Ripple reorganizes its SQL databases to remove some unused data, which provides better data management for the **ledger**.
- 2016-07-19: The TrustSetAuth is enabled in the **ledger** to authorize other addresses to hold their issued currencies and build connections to the **shamap**.
- 2014-06-26: A new signature scheme is implemented based on DSA in the **crypto** to ensure data security for the **ledger**.

1.1.3.2 Network layer

- 2019-10-02: The fixMasterKeyAsRegularKey is implemented in the **proto** to set regular key pair to master key pair.
- 2017-03-16: Gateway Bulletin is implemented to ensure a trust line quality and is realized in the **RPC, server, client** components.

1.1.3.3 Architecture layer

- 2018-11-07: Node store is upgraded to shard store in the **node store**, providing reliable paths toward ledger history across the XRP Ledger Network. So it is connected to the **ledger**.
- 2014-04-28: Ripple's Naming system is used to standardize the configuration. It is checked in the **conditions**.

Most of our representative code activities are serving the **blockchain** layer, especially the **ledger** component, performing better currency transactions and encryption. The XRP Ledger is the focus of programmer development, and other layers are providing technical support for XRP in the blockchain layer.

1.1.4 Mapping of the system's roadmap onto architectural components

In this section, we extract key events in our roadmap analysis and map it according to architectural layers in the graph below.

1. Database enhancement:
   - Since the database is the basis of the Architectural layer, a robust database ensures smooth operation of other architecture layers.
   - It is mainly implemented in the **nodestore** and **resources**.
2. Better login system with Ripple Names:
   - As an entrance, the login system belongs to the App layer, providing better interactions for Ripple users.
   - The naming system is more of a written specification.
3. Debit card Linking:
   - Payment handling is a key topic of the blockchain layer, whose job is mainly to oversee the operation and payment of digital currencies.
   - As for codes, **all the components** in the Blockchain layer is responsible for payments.
4. Downloadable and improved web client:

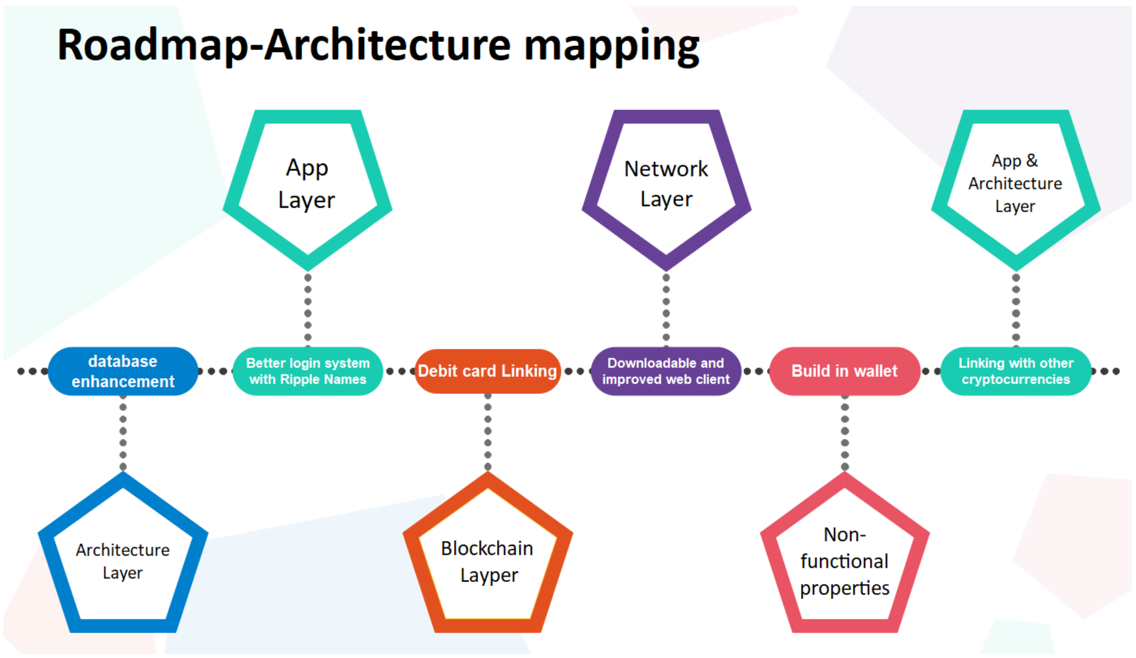

Roadmap-Architecture mapping
App Layer
Network Layer
App & Architecture Layer
database enhancement
Better login system with Ripple Names
Debit card Linking
Downloadable and improved web client
Build in wallet
Linking with other cryptocurrencies
Architecture Layer
Blockchain Layper
Non-functional properties

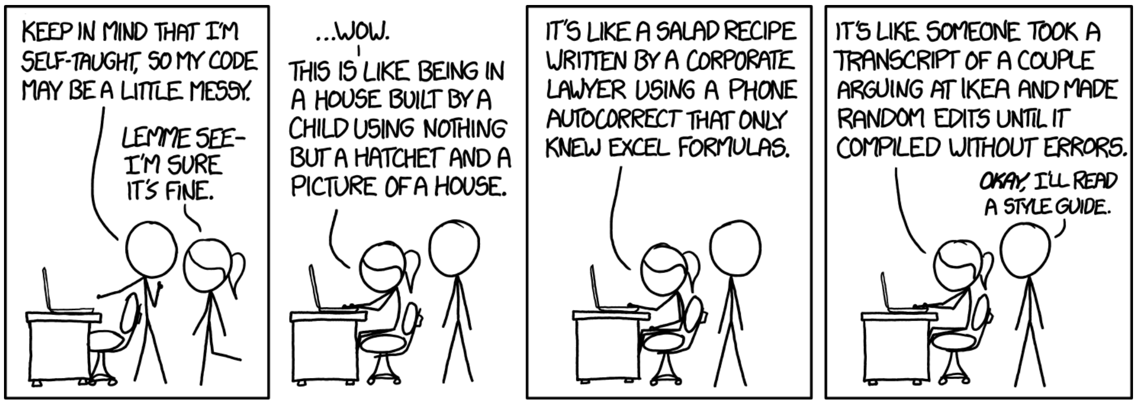
Figure 2: Roadmapping

- Ripple introduced several WebSocket amendments and gateway services APIs, providing straightforward calls that clients can use to route payments appropriately. This reliable client serves for the Network layer.
- **All the components** in the Network layer work for this client and WebSockets.

5. Build-in wallet:
   - This is realized by Ripple's XRP project. This project is a goal of Ripple NFR, which is a key component of non-functional properties.
   - The wallet is mainly implemented in the **ledger**.
6. Linking with other cryptocurrencies:
   - This improvement is beneficial to both the upper-level Blockchain transaction architecture and App layer.
   - Codes are integrated into **app**.

1.2 Code Quality and Maintainability

In this section, first, we will discuss the code quality and maintainability and then analyze the results from Sigrid to see how much Ripple is applying the theory in practice.


KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE– I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

Source[8]

Code quality can have a significant impact on software quality, on the productivity of software teams, and their ability to collaborate. But how can one measure code quality?[9]

Here are some of the main attributes that one can use to determine it:[10]

Attributes	Describtion
Clarity	Easy to read and oversee for anyone who isn't the creator of the code.
Maintainablity	A high-quality code isn't overly complicated. Anyone working with the code has to understand the whole context of the code if they want to make any changes.
Documentation	The best thing is when the code is self-explaining, but it's always recommended to add comments to the code to explain its role and functions. It makes it much easier for anyone who didn't take part in writing the code to understand and maintain it.
Well-tested	The fewer bugs the code has, the higher its quality. Thorough testing filters out critical bugs ensuring that the software works the way it's intended.
Efficiency	High-quality code doesn't use unnecessary resources to perform a desired action.

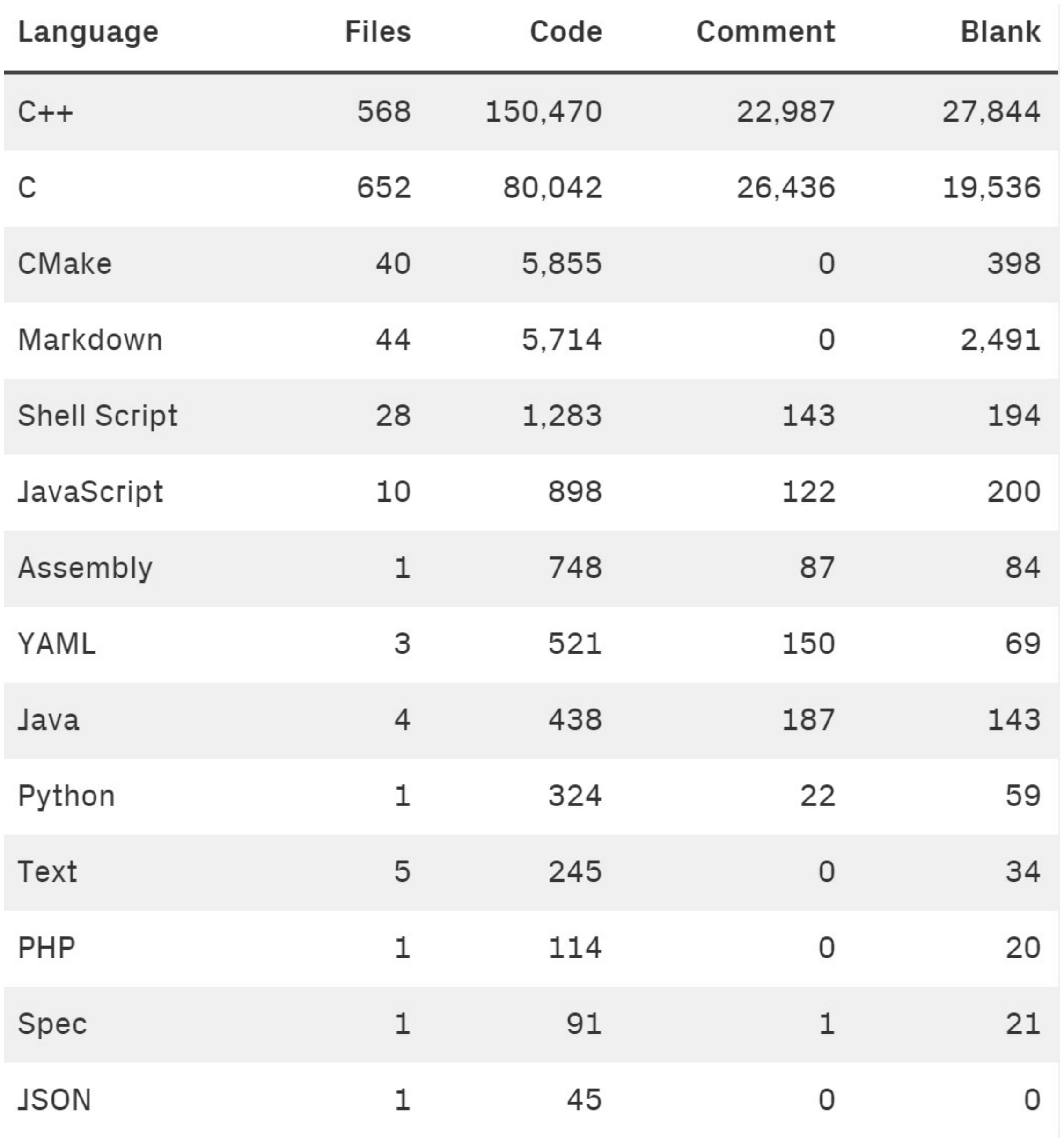
1.2.1 Refactoring Suggestions

To Analyze the code quality and maintainability of Ripple, we used a behavioral code analysis tool, namely Sigrid.

The distribution of code volume over language are as follows:

Ripple achieve a good score in the SIG assessment system due to its high unit interfacing(4.6). But it falls

[8]https://xkcd.com/1513/
[9]https://www.sealights.io/code-quality/code-quality-metrics-is-your-code-any-good/
[10]https://codingsans.com/blog/code-quality



Language	Files	Code	Comment	Blank
C++	568	150,470	22,987	27,844
C	652	80,042	26,436	19,536
CMake	40	5,855	0	398
Markdown	44	5,714	0	2,491
Shell Script	28	1,283	143	194
JavaScript	10	898	122	200
Assembly	1	748	87	84
YAML	3	521	150	69
Java	4	438	187	143
Python	1	324	22	59
Text	5	245	0	34
PHP	1	114	0	20
Spec	1	91	1	21
JSON	1	45	0	0

Figure 3: codelines

heavily in the unit size(0.5) and unit complexity(0.5) sections. The duplication score (3.7) can also be improved further.[11]

Some of the maintainability violations found are listed in the table below.

Violation type	Instances in Ripple
Unit size	632
Unit complexity	239
Unit interfacing	12
Duplication	934

The unit size and unit complexity are all belonged to the unit test level.[12]

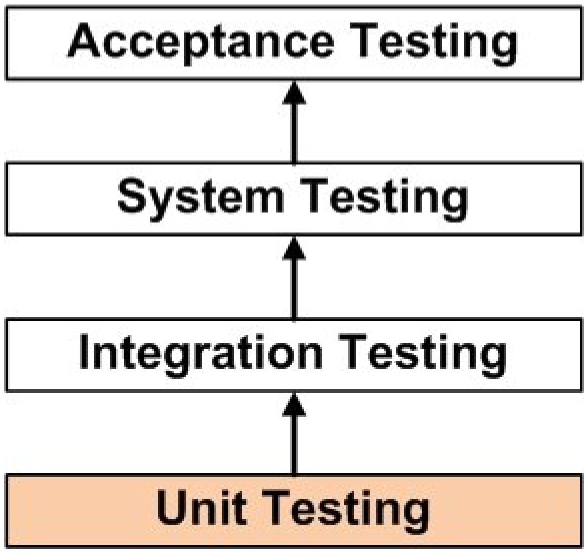


Figure 4: Test layer

1.2.1.1 Duplication

Duplication inside the codes means how many codes are repeated. It can be checked easily through some existing tools. To improve it, duplicated codes can be stored in separate functions or files to be reused. But it will also add some dependencies inside the program.

1.2.1.2 Unit size

Unit size is a parameter that belongs to testability to the software's maintainability[13]. During the unit test of software, individual functions and procedures need to be tested separately to ensure each unit works properly[ref]. The unit size means how much codes inside each size. According to the SIG results[14], the unit size of the ripple projects is seen as large and complex. During testing, these large units may lead to an insufficient unit test.

[11] https://sigrid-says.com/maintainability/tudelft/rippled/refactoring-candidates
[12] http://softwaretestingfundamentals.com/unit-testing/
[13] http://softwaretestingfundamentals.com/unit-testing/
[14] https://sigrid-says.com/maintainability/tudelft/rippled/refactoring-candidates

To improve this part, some large functions may split into multiple simpler functions. By doing this splitting, some functions can be reused to enhance the duplication. But this split will bring more functions inside the program and make it harder to read.

Ripple's unit size keeps increasing during its further developments. They are adding more features, more services, and making the software more secure and robust to meet the requirements of markets. To reduce the unit size, developers need to review their design and do some split to different functions. Splitting some functions to become separate units.

1.2.1.3 Unit complexity

The complexity of each unit has some relation to the unit size but contains more than just size. It also shows the complexity of control logic and code complexity. Cyclomatic complexity invented in 1976 by Thomas McCabe Snr is the most widely accepted metric for evaluating the code complexity. By measuring and reducing the code complexity, it brings several advantages like better tests, reduced risks, lower costs, and greater predictability. It can also help the developer to improve their programming skills.[15]

To improve this part, use smaller methods and reducing the if/else statements will help.[16] This may need a lot of effort to read through all the existing codes.

To achieve security and stability in Ripple's working scenario, methods, functions, and control logic are designed to be robust and complicated. Those codes are growing more and more complex during development. It will be useful to revisit their codes and design to see whether there are redundancy codes or meaningless control logic. This review can be handy in reducing unit complexity.

1.2.1.4 Module coupling and Component independence

Scores for both Module coupling and Component independence on Sigrid are 5.5 and (N/A), respectively. One can easily argue that something must be wrong with these scores since zero interdependencies between components of the software as huge as Ripple seems impossible. As soon as we can use other code behavioral analysis tools, we will analyze these factors as well.

1.2.2 An assessment of technical debt

Ripple, as blockchain technology variations, aims to solve some real-world problems. [17] Ripple wants to build a global system of payments, settlements ad exchange. XRP is the cryptocurrency they use to solve the problem.

1.2.2.1 Debt about the blockchain technology

The blockchain technology is a hot debt topic these years, mainly because of its decentralized design, anonymous system, and cryptocurrency.

1.2.2.2 Debt about the ripple approach (DLT and XRP ledger)

Ripple as one of the blockchain variations, they chose to go their technical solutions. The DLT (Distributed Ledger Technology) they use is an open-source protocol hosting a shared and public ledger, using a consensus mechanism to secure security[18]. The DLT provides faster transaction speed and fewer transactions fee. The consensus is achieved by those validators, which is not anonymous and need to be proved by the ripple lab(The company who

[15] https://blog.codacy.com/an-in-depth-explanation-of-code-complexity/

[16] https://www.axelerant.com/resources/team-blog/reducing-cyclomatic-complexity-and-npath-complexity-steps-for-refactoring

[17] https://medium.com/@siddharth.sitpure/a-closer-look-at-ripples-blockchain-technology-and-xrp-9e036e1bf019

[18] https://www.bitdegree.org/tutorials/ripple-coin/

made Ripple). This idea against the decentralized idea inside the blockchain. In a word, Ripple is trading their decentralized for speed and fee.

1.2.2.3 Debt about the consensus algorithm The consensus algorithm ripple use is their RPCA algorithm. To achieve a fast speed of consensus, they using a UNL(Unique Node List) design among all servers. In this design, it requests several requirements, including the fault nodes is decreased to less than 20%; the network topology needs to avoid a single connection between two blocks.

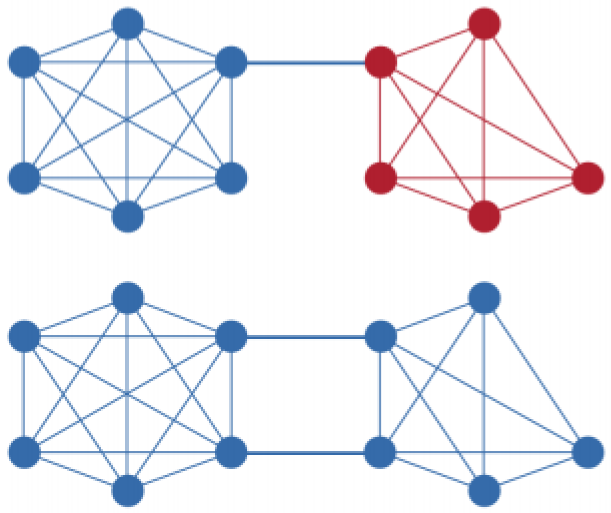


Figure 5: Network

For all those algorithms, the failure of the physical layer is hard to avoid. The Ripple approach requires monitoring of the global network structure to ensure some boundary conditions.[19]

[19]https://ripple.com/files/ripple_consensus_whitepaper.pdf