BRIDGING BLOCKCHAINS

Kirill Shatilov
Jakob Heyder
Zac Wellmer

Abstract

We have recently seen an explosion of blockchains in various fields experimenting with different parameters to achieve different objectives. From Chains with focus on decentralized applications, such as Ethereum, to privacy-first ones like Z-Cash, tokens for filestorage with filecoin, proof of knowledge for educational certificates or corporate chains like hyperledger, azure or quorum. The growth of this innovation is stunted because of the fact that all these systems are isolated. They do not have a native way to interact with each other. Interactions between different chains is difficult due to the consensus logic of such chains. Interoperability is especially difficult for open chains because you need to verify the current global state to be able to interact with it. We will propose in our research project a first implementation of a bridging concept between two chains. The chain will be an ethereum based blockchain using proof of authority for its consensus and UST Token as its currency. It is the hypothetical and totally symbolic coin of an university internal system. The goal of the project is to build a chain to the already set ethereum testnet KOVAN which is also a proof of authority network controlled by several companies in that field. To interact between both chains we will use a break-in and break-out contract that allows users to create "external transactions" and deposit money. The fees will be handled from the bridging partner or validators. Our concept tries to integrate into the overall framework of the polkadot project, as such that you could build the relay chain between the direct bridge to extend the system.

1. Introduction

With blockchain technology rapidly spreading it is easy to envision a future where there is a large ecosystem of blockchains. Currently this blockchain ecosystem is mostly just a large number of chains operating independently. However, transactions and messages taking place in one chain still may contain information that is useful to other chains. Emphasizes is on transactions and messages, since not all blockchains revolve around an underlying token.

A basic example of a blockchain not based on an underlying token would be a chain used for recording the shipping process from sellers to buyers. These token free blockchains likely contain information that is particularly useful to currency based chains. For example, imagine a token free blockchain that is used to record the path of an item purchased off the internet and being shipped to the buyer. A currency based chain could have a smart contract that is "listening" to the token free item shipment chain and when the purchased item is guaranteed to have been delivered then the money initially committed to the smart contract by the buyer is sent to the seller. This guarantee of being delivered is determined by the state of the token free shipment chain. In our work we are particularly interested in the process of how "listening" to the token free chain could be done.

In the previously mentioned example we went over how a token based chain can benefit from listening to a token free chain. However, it's important to recognize that all blockchains can benefit from listening to one another regardless if it's two token based chains, a token based chain with a token free chain, or even two token free chains. One of the not so obvious examples that we will later explore in this paper is between two token based blockchains. The premise of the problem we address is the following: Imagine three users. User 1 and user 3 both of accounts affiliated with Chain A and B. User 2 only has an account associated with chain B. However, User 1 would like to transfer tokens belonging to Chain A to user 2. Obviously they could use the not so elegant solution of going through an exchange. But a more desired approach is a way to manage this transaction without user 2 ever opening an account in chain A. In our example we will later detail how a full sequence of transactions from user 1 to user 2 to user 3 takes place and is validated.

Now that a motivational foundation has been set in place we will explore some of the related work and background technologies necessary for understanding how the experiments work and why they are important. Specifically our experiments require background knowledge of Ethereum[1] and proof of authority consensus. While the experiments shown only deal with two proof of authority style blockchains interoperating with one another. It is possible for two chains with different consensus mechanisms to work in this context as well. However, for ease of testing purposes we choose proof of authority style chains.

In Section 2 we discuss other generally more solid approaches to interchain communication. As far as we experimenting over Ethereum network, Section 3 will superficially cover aspects of this technology, meanwhile giving some insight into our choices of software environment. Next, in Section 4, different consensus mechanisms are described to provide a theoretical background for our experiments over Proof-of-Authority networks. It is clearly a matter of future work to scale interchain communication over networks with other consensus mechanisms. Section 5, in details discusses the key component of studied solution - bridge between networks; following up, Section 6 provides with a details of conducted experiments, covering network setup and description of ui application. Conclusion to our experiments is presented in final Section 7.

2. Related Work

The problem of interchain communication has been discussed and looked into by some projects in blockchain industry. In this section we will shortly discuss Polkadot, Plasma and Interledger. As far as industry is growing extremely fast, concepts of chain interoperability doesn't limit to this small list.

2.1. Polkadot

Polkadot[5] is a scalable heterogeneous multi-chain. Polkadot relies on a system of actors referred to as validators, collators, nominators, and fisherman to perform trust-free interchain transactability. Along with another bonus of pooled security allowing all the parachains to benefit from the security of the base relay chain.

2.2. Plasma

Plasma has quite a few similarities to Polkadot. The largest similarity between Polkadot and Plasma is that they both produce a hierarchy of blockchains. However, instead of Polkadot's structure where the “fishermen” ensure block accuracy, Plasma constructs a series of child blockchains which enforce state via merkle proofs[6].

2.3. Interledger

Interledger [7] is a bit different than the two previously mentioned approaches to interchain communication. Interledger is closer being described as a decentralized exchange protocol. At a high level it works by finding a short path from the source chain to the destination chain and offers security to both nodes along the path and source/destination nodes by using trusted escrow.

3. Ethereum

This section covers aspects of blockchain app platform Ethereum, which (Ethereum test networks, smart contracts and distributed application concept) we used for our experiments.

The purpose of Ethereum is to provide an infrastructure for building decentralized applications. The infrastructure offers an environment well suited for rapid development time, and the ability for vastly different applications to communicate with each other. Ethereum achieved this by building a blockchain with a Turing-complete programming language. Offering the freedom for anyone to write smart contracts and decentralized applications where they can decide their own rules for ownership, transaction formats, and state transition functions.

3.1. Accounts

Every Ethereum account has four features:

- Nonce: making sure transactions are only processed once
- Ether Balance: account balance
- Contract Code: not always present
- Storage: An accounts storage is empty by default

There is generally two different types of Ethereum accounts. The first being externally owned and the second being contract owned. Contract owned accounts are controlled by the contract code while externally owned accounts are controlled by private keys.

Contract code is executed every time a contract receives a transaction or message. It is a common questions to wonder “who” is the one executing the contract code. Contract code execution is part of the state transition function, which is part of block validation. If a transaction is added to a given block, the contract code associated with the transaction will be executed by every node that validates the block.

3.2. Transactions

An Ethereum transaction has 6 fields associated with it:

- Recipient
- Signature identifying the sender
- Optional data
- Maximum number of computation steps
- Amount fee per computation step

The first four serve an obvious purpose while the last two may not immediately make sense. The purpose of gas is to prevent malicious actors from causing denial of service to the network. Gas makes users pay fees proportional to the amount of resources they use. If excess gas is provided in a transaction it is returned to the sender. If there is a shortage of gas provided then the transaction stops and the state is rolled back.

3.3. Messages

Messages are similar to transactions except that messages are produced by contract owned accounts and not by externally owned accounts. Contracts are able to communicate and have relationships with each other by passing messages. A contract has the following 5 features:

- Sender of message
- Recipient
- Amount of Ether to transfer alongside the message
- Optional data field
- Gas value

It is important to recognize that this infrastructure is useful for a wide variety of different decentralized applications. These applications could range from being financially focused(ex: a decentralized hedge fund) to applications that have no relation to finance(ex: decentralized voting). In a future where there is a vast ecosystem of decentralized applications it is easy to imagine that these applications must speak to each other. In our project we will heavily rely on this aspect of Etherium to show how can contract encapsulate mechanism for transactions using tokens, locked in another blockchain.

3.4. Parity and Ethereum

Blockchain technology nowadays in its infant stage. Yes, we have reliably running most notorious Bitcoin network. But most of the others networks and protocols are in some kind of experimental stage, being developed and maintained by enthusiasts funded by venture capital. Parity software [8] is a perfect example for this statement. Although developers state their idealistic goals of making this world a better place with distributed applications and writing fully documented and tested code, the software provided by them is truly experimental and relentlessly unstable with clunky web interfaces and infinite amounts of opened issues on github. Despite all this acrimonious negativity, the Parity web client comes with a vast functionality, including a framework for developing and testing distributed applications. Moreover, bridge application is developed by Parity software, being basically modified ethereum client with some additional features.

4. Consensus

Our goal is to build a UST Token network, every transaction on which is approved by a set of central authorities. This one of the approaches to build a blockchain network, usually referenced as Proof-of-Authority (PoA) consensus mechanism. In general, there are other ways in which cryptocurrency blockchain network can achieve a distributed consensus, including, but not limited to:

- Proof-of-Work (PoW)
- Proof-of-Stake (PoS)

In this section we give short overview of this consensus mechanisms, bearing in mind that PoA concept facilitates the difficulties of building and network itself and interchain communication as well. More complex solutions, like discussed Polkadot, while being still in development, tackle the problem of connecting heterogeneous chains.

4.1. Proof-of-Work (PoW)

Originally designed as a countermeasure for denial of service attacks and the other service abuses for distributed service systems [2], this principle found its way into cryptocurrencies protocols, including Bitcoin, where solving Proof-of-work puzzle is basically reversing hash function (SHA-256) [3].

To seal the current block each network participant (in general, each full node in practice) looks for a value (nonce of block) that when hashed among with other block attributes (header, merkle tree root, etc) the hash begins with a certain amount of zeros. This amount of zeros is defined by puzzle difficultness, which is dependent on overall network hashing rate and adjusted to meet average block mining expectation time. The majority decision is represented by the longest chain, which has the most amount of computations invested in it.

Central assumptions in PoW is that the amount of honest nodes in the network is at least 51 percent.

PoW concept based on two principles:

- One-CPU-one-vote. Not only does this principle introduce equality (in theory; in practice *many* CPUs can be controlled by a single authority), giving an equal opportunity to vote for every network participant without any direct investments, but also it prevents so called sybil attack, when a malicious actor can subvert a voting in a network, based for example on one-IP-one vote, by allocating huge amount of IPs.
- Puzzle trait: hard to solve - easy to check. The probability of mining next block (finding suitable nonce value) follows the poisson distribution, also meaning that the success if the current trial doesn't depend on outcomes of the previous ones. In practice the amount of computations to find the right nonce is so big that the whole bitcoin network consumes an amount of electricity comparable to that of a small country. But checking (which is just computing SHA-256 of given message) of correctness of the solution takes milliseconds on any device.

Most of Proof-of-work systems require nodes to do useless computations, wasting electricity and depleting hardware resource. However, there are some exceptions, e.g. PrimeCoin [12], which requires to find prime numbers of certain types, which might have (arguably) some useful side-applications. Anyway amount of wasted resources is among the reasons why other Proof-of-... systems exist (or being developed).

4.2. Proof-of-Stake (PoS)

Main idea of PoS cryptocurrencies systems is that creator of the next block is chosen randomly, but the chance of being chosen is proportional to staked tokens (which can be generalised to any matter, including reputation points, inner currency, amount of time spent contributed to participating in the network, age of coins, etc) [4].

PoS approach brings many advantages, including:

- Obviously a huge saving of energy - it is not needed to perform heavy computations to secure blockchain.
- As a consequence, there are less reasons to incentivise network participants, mostly miners, by issuing new coins. This may lead to more stable economy.
- Arguably, PoS mechanizm will prevent participants from uniting in ~~mining~~ staking pools. Large mining pools in PoW network are serious threats to very concept of decentralization.
- Additional rules in PoS, like burning of (un)intentionally wrongly staked coins, will discourage attacks on such networks.

- Self-contained economics - all tokens are coming from inside network. Let us consider PoW network, ExampleCoin. Theoretically, adversary, who wants to crash ExampleCoin, can buy enough computational resources to overthrow network security and perform 51% attack. After such kind of event, ExampleCoin will crush, its capitalisation will go beyond zero values, but attacker will be left with his mining farms, which still posses high value in offline world. But in order to perform the same attack on PoS network adversary has to invest into economy of this network to buy staking tokens. This will inevitably lead to high increase of tokens conversion rate to other (crypto)currencies. Firstly, it is economically unreasonable for adversary to crush economy in which he has invested. Secondly, such attacks can be detected during the phase, when attacker tries to buy as many tokens as he can, so considerate network participants can refuse selling their tokens.

On the other hand, PoS concept has some, arguable disadvantages:

- Higher implementation difficulty. Theoretically, idea is quite simple, but practically such systems might be hard to deploy and maintain.
- PoS system has to consider reliable source of randomness to persuade every participant of network in fairness of 'lottery'. Such source of randomness must be publicly observable, and practically non influenceable by anyone interested in certain outcome.
- Self-contained economics might impose some limitations on new participants entering network, while heavily relying on active participation of set of influential old players.

4.3. Proof of Authority(PoA)

Proof of Authority is a replacement for proof of work consensus and can be used is usually found in private chain setups[5]. The biggest difference between proof of authority and the two previously mentioned consensus mechanisms is that proof of authority doesn't allow just any user to participate in determining consensus. Unlike proof of work, there is no need to solve arbitrarily hard math problems. Unlike proof of stake, users don't need to stake tokens. Proof of authority relies on predetermined authority nodes to create new blocks and secure the blockchain. A new block is required to be signed by a majority of the predetermined "authority" nodes in the network.

In the consortium setting, a chain shared between several companies, there is not downside of using proof of authority as opposed to proof of work. It is more secure, because outside attackers have no say in block generation or securing the chain. There is no computational waste from computing meaningless hashes. It has faster performance with respect to transaction performance time. Proof of authority chains are also predictable since the consensus mechanism doesn't rely on a stochastic process like proof of work. Proof of authority chains are popular in the enterprise setting but can still play nicely with chains following other consensus mechanisms.

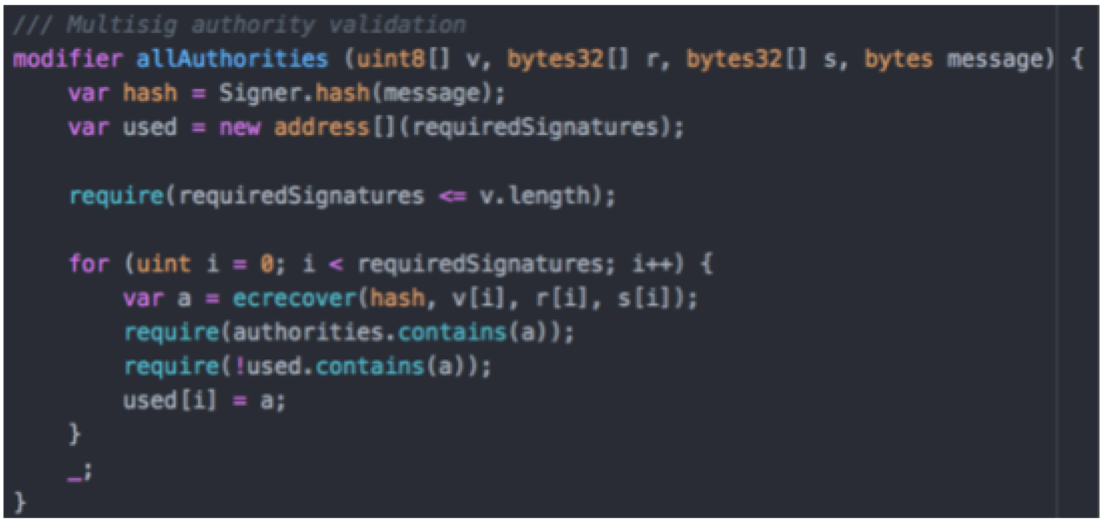
5. Bridge

Current Section describes key component of our experiment bridge application, which is observing state of selected contracts on different networks and automatically transfers messages between them.

The architecture of the bridge built comes from a framework provided by Parity[10]. Some parts of this repo were half broken and this was solved by alerting the Parity team of some issues and even merging pull requests of our own to solve some of the more minor problems. Encountering this kind of roadblock is exciting because it is a sign that this field of work is still in progress. It is exciting to contribute to an early stage project that will inevitably grow to become more mainstream in the future.

A Parity bridge in the context of our work has two crucial features. These two features are the break-out contract or home contract and the foreign contract or break-in contract. These two contracts operate in a similar fashion, however, the naming scheme and subtle differences are based on the context of which chain is home chain and which is the foreign chain. In the case drawn in Figure 2 UST Token is the home chain and Ethereum is the foreign chain.

The home contract has two main functions: withdraw and payable. Payable would be used to deposit tokens. The withdraw function would be used to extract a message or token which was received as a result of an event that took place in the foreign chain. The home chain must contain enough information to have a consensus of the home chain. In the case of proof of authority this would be the contract authorities, number of required signatures, and a multisig verification from the minimum number of required authorities. This last part is a key concept, in Figure 1 multisig is explicitly outlined.



```
/// Multisig authority validation
modifier allAuthorities (uint8[] v, bytes32[] r, bytes32[] s, bytes message) {
    var hash = Signer.hash(message);
    var used = new address[](requiredSignatures);

    require(requiredSignatures <= v.length);

    for (uint i = 0; i < requiredSignatures; i++) {
        var a = ecrecover(hash, v[i], r[i], s[i]);
        require(authorities.contains(a));
        require(!used.contains(a));
        used[i] = a;
    }
    _;
}
```

Figure 1: Solidity multisig authority validation modifier

Anytime a withdraw attempts to take place in the home contract the modifier is invoked. A modifier in Solidity essentially just specifies a set of conditions to be satisfied in order for a function to execute. In the case described the multisig is the set of conditions and the function is the withdraw. An example of what a withdraw could be sending a token to someone from chain A to someone who does not have an account on chain A. However, this person must have an account on chain B.

The foreign contract has the following four main functions: deposit, transfer, and submitting signatures. Deposit is used to record a payable interaction from the home chain. Transfer does exactly as the name suggests and passes a token between two users. Submitting signatures is used as a synchronization tool when passing messages to the home chain. The foreign contract also involves a few consensus specific operations. In the poof of authority case in particular it is necessary to validate that all incoming messages are sent from a authority. Specifically there are two cases. First: when a deposit has been submitted to the foreign chain it is necessary to validate that the message came from a authority. Second: when signatures are being collected to relay a message to the home chain. For a more in depth description of the creation of home and foreign contracts please refer to the solidity script for building bridges found in our references[11].

A high level view of a how these two contracts operates is the following. A user in the chain affiliated home contract would like to transfer a token to a user in the foreign chain. The user deposits money into the home contract. This message is passed to the foreign contract where the message is validated to have come from an authority node. The token now belongs to the second user according to the foreign contract. It is essential to realize here that what we are referring to here is NOT an exchange. At no point in this scenario is a token from chain the home chain converted into tokens from the second chain. Now the second user would like to send the token from the home chain to a different user who belongs to the home chain. The tokens are reassigned to the third user who then withdraws from the home token the system by validating with authority nodes in the Foreign chain where once validated sends the message to the home chain where once again the minimum number of authorities must validate before the home token is withdrawn to the third user.

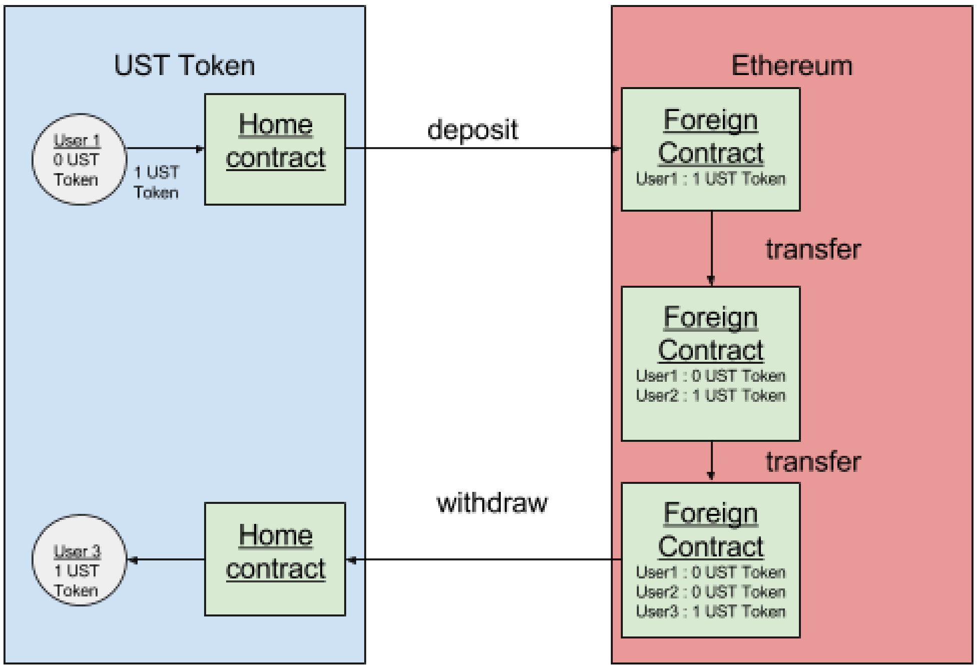


Figure 2. Networks overview

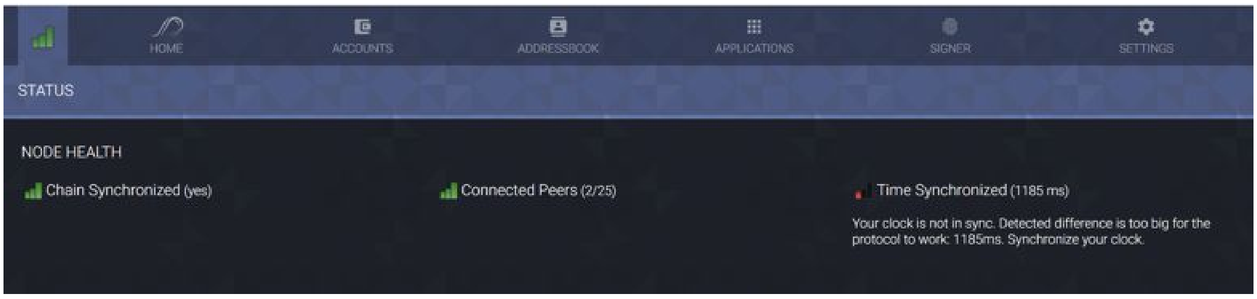
6. Experiments

6.1. UST Token Setup

To set up and run UST Token network we used Parity demo PoA tutorial [8] with our custom parameters. For proper simulation and testing we configured 3 Parity nodes, on different machines running Windows, MacOS and Ubuntu.

After building a blockchain that follows proof of authority consensus we connected three nodes which ran this system. What follows is a few screenshots from a small user interface that is friendly for letting participants visualize network information.

Network overview from ui:



NODE HEALTH

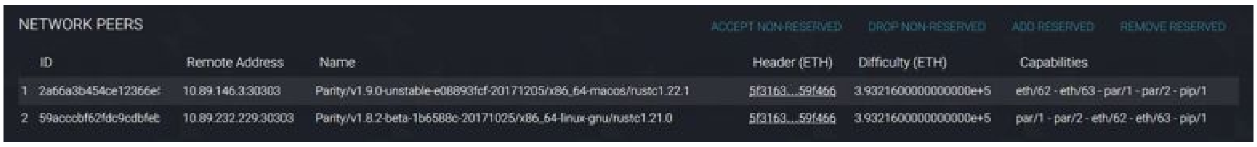
Chain Synchronized (yes)

Connected Peers (2/25)

Time Synchronized (1185 ms)

Your clock is not in sync. Detected difference is too big for the protocol to work: 1185ms. Synchronize your clock.

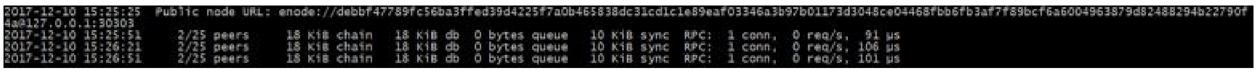
Below is a snapshot of two nodes running different operating systems and different versions of parity client:



NETWORK PEERS

	ID	Remote Address	Name	Header (ETH)	Difficulty (ETH)	Capabilities
1	2a66a3b454ce12366e	10.89.146.3:30303	Parity/v1.9.0-unstable-e08893fcf-20171205/x86_64-macos/rustc1.22.1	5f3163…59f466	3.93216000000000000e+5	eth/62 - eth/63 - par/1 - par/2 - pip/1
2	59acccbf62fdc9cdbfeb	10.89.232.229:30303	Parity/v1.8.2-beta-1b6588c-20171025/x86_64-linux-gnu/rustc1.21.0	5f3163…59f466	3.93216000000000000e+5	par/1 - par/2 - eth/62 - eth/63 - pip/1

Connection information on console output:



```
2017-12-10 15:25:25  Public node URL: enode://debbf47789fc56ba3ffed39d4225f7a0b465838dc31cd1c1e89eaf03346a3b97b01173d3048ce04468fbb6fb3af7f89bcf6a6004963879d82488294b22790f
4a@127.0.0.1:30303
2017-12-10 15:25:51     2/25 peers     18 KiB chain    18 KiB db   0 bytes queue    10 KiB sync   RPC:  1 conn,   0 req/s,   91 µs
2017-12-10 15:26:21     2/25 peers     18 KiB chain    18 KiB db   0 bytes queue    10 KiB sync   RPC:  1 conn,   0 req/s,  106 µs
2017-12-10 15:26:51     2/25 peers     18 KiB chain    18 KiB db   0 bytes queue    10 KiB sync   RPC:  1 conn,   0 req/s,  101 µs
```

Our experiment included exchange of UST Tokens between accounts, which was processed successfully.

What beared a significant difficulty for us in this step:

- Configuring and running nodes on different OS. Each node uses two configuration files: one for network, another for node settings including ports, running services etc. Eventually every machine had a slightly different config files.
- Connectivity issues. Software and network instability overlaps each other, and given particular error message it's hard to come up with certain diagnosis what is wrong in the moment. For example, the Windows node was constantly falling off and losing connection to other nodes. Relative stability was achieved by running nodes with administrative privileges with disabled firewall, which is obviously not safe.
- Another issue were different chain states which lead to an unrecognized hard-fork so that the clients do not connect at all anymore.

6.2. Koval Testnet Setup

Connecting nodes to Kovan testnet is relatively easy job. The Parity client has a built-in configuration for the testnet. Users have to run the following command (on any of Windows/OS X/Ubuntu operating systems):

parity --testnet

What turned into a difficulty for us in this step:

- After a node connects to the testnet it has to synchronize its blockchain with the rest of the network. Correct work during synchronization period (which takes up to several hours and consumes CPU resources) is obviously not guaranteed; that fact brought significant delay to our work.
- Accounts on testnet should have some balance to pay gas fees for calling contract methods. Also it was required by our experiment that accounts should have some balance to exchange ether with each other.

From now on we will refer to the UST Token network as the home network and the test network (or second network, where home tokens are exchanged inside contract) as a foreign network for ease of understanding.

6.3. Bridge Setup

Experimenting with a bridge requires two networks, home and foreign, to be up and running. The bridge application itself has be run in a single instance. It requires some tedious configuration, the most important aspects of which are:

- Address and compiled instance of bridge contract, deployed in home network.
- Address and compiled instance of bridge contract, deployed in foreign network.
- Accounts of authorities of home network, who are eligible to verify transactions and trigger certain contract functions.
- Gas deposit for interacting with a contracts on both network.

What beared a significant difficulties for us in this step:

- Bridge application doesn't come as executive file. It has to be compiled from sources from github. It is also not a cross platform codebase. It can not be run on Windows systems. As a result of this, the set of machines where bridge experiments can be conducted is immediately limited.
- The client is highly customizable which made it possible to run different networks and applications on the same computer over different ports. Networking port, RPC (Remote Procedures Call), IPC (Inter Process Communication), WebSockets, port for web application to be hosted and so on. Port management is really hard and tedious, especially when running three instances of the parity client on a single machine. Moreover, Windows has issues of its own; two parity clients can not be run on a Windows machine with the exact same port configuration as on Ubuntu.
- Additionally parity has two different API modes, a secure API where you need to verify requests with a token and a non-secure one. This calls made from the UI and dapps, which itself use different ports to communicate with the parity instance, use different channels which lead to a bug for dapps to run seamlessly on any non-default port.

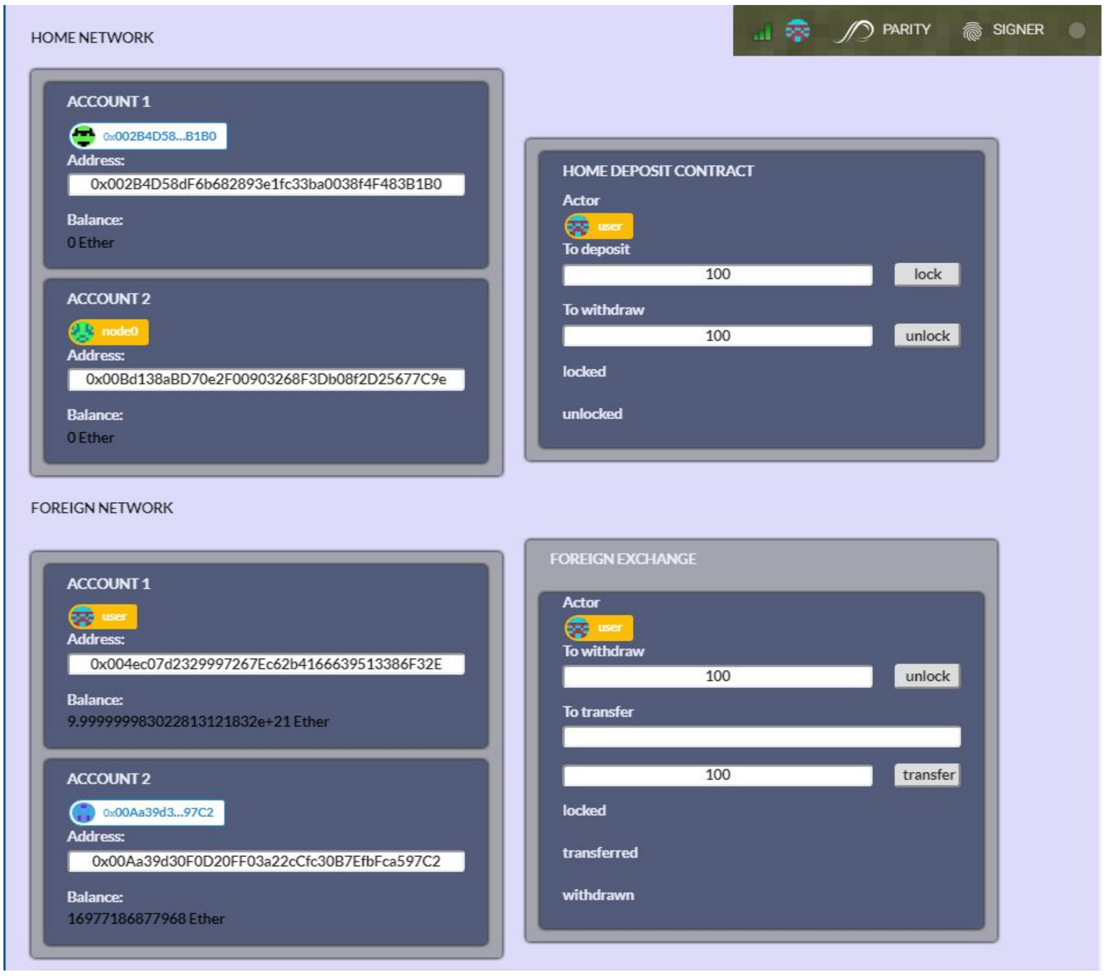
Bridge configuration is available on:
[https://github.com/kaikun213/COMP6311E_BridgeChain/tree/master/examples]

After having the bridge configured and running; network commands could be triggered from the console. Interchain operations could be observed with curl commands and their return codes. It was logical to develop a distributed application for visual and demonstrative overview of processes happening.

6.4. Observation

For controlling and illustrating the flow of interchain communications we developed a distributed application for the parity ethereum client, which is available on github [https://github.com/kshatilov/BCCCProject] and also included in the supplementary material.

Overview of the application interface is presented below:


HOME NETWORK

PARITY SIGNER

ACCOUNT 1
0x002B4D58...B1B0
Address:
0x002B4D58dF6b682893e1fc33ba0038f4F483B1B0
Balance:
0 Ether

ACCOUNT 2
node0
Address:
0x00Bd138aBD70e2F00903268F3Db08f2D25677C9e
Balance:
0 Ether

HOME DEPOSIT CONTRACT
Actor
user
To deposit
100 lock
To withdraw
100 unlock
locked
unlocked

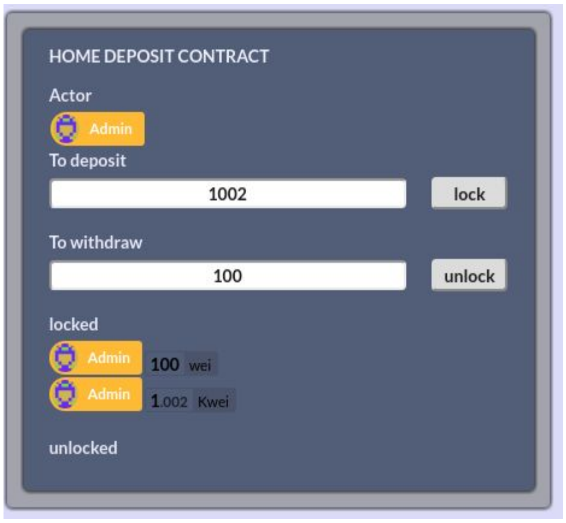
FOREIGN NETWORK

ACCOUNT 1
user
Address:
0x004ec07d2329997267Ec62b4166639513386F32E
Balance:
9.9999999830228131218 32e+21 Ether

ACCOUNT 2
0x00Aa39d3...97C2
Address:
0x00Aa39d30F0D20FF03a22cCfc30B7EfbFca597C2
Balance:
16977186877968 Ether

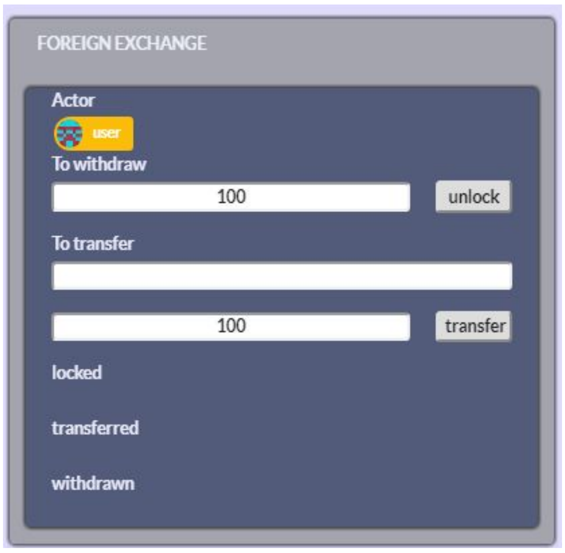
FOREIGN EXCHANGE
Actor
user
To withdraw
100 unlock
To transfer
100 transfer
locked
transferred
withdrawn

The same application is used for both home and foreign networks. Running in each network, the application displays the balance of predefined accounts, which are supposed to be actors in interchain communications. Addresses can be changed at runtime, and account widgets will display balances for newly input correct address immediately.


HOME DEPOSIT CONTRACT
Actor
Admin
To deposit
1002 lock
To withdraw
100 unlock
locked
Admin 100 wei
Admin 1.002 Kwei
unlocked

In the Home network, exists a widget for the deposit account (screenshot on the left). It displays the current active account (Actor), which will be used for depositing home tokens. The active address can be switched in parity client. To deposit tokens, the actor after specifying an amount has to press the “lock” button. Technically speaking, the application will initiate a payment to the contract address. After the transaction is signed using standard parity client signing tool, locked tokens will be displayed in the bottom part of deposit contract widget. As far as the widget observes all deposit events in contract, all currently locked tokens and aliases of addresses, who locked them, will be displayed under “locked” section.

The parity bridge will track all Deposit events as well, and will dispatch a transaction in Foreign network. Balances inside foreign exchange contract will change accordingly.



Using Foreign Exchange widget (on the left), accounts, who are eligible to control locked funds can perform following operations:

- Unlock their home tokens. Call of according contract method will dispatch event of withdrawal. Which in its order will be observed by parity-bridge. Forthcoming transaction issued by bridge will unlock home tokens.
- Transfer their home tokens (represented by inner structure of contract) to another address in foreign network. Receiver of this transfer can further transfer this tokens or withdraw them, if he is a participant of home network or interested in becoming one.

This widget observes all events in foreign exchange contract as well.

Challenges we met on this step:

- Although there is a nice tutorial [13] for javascript DApps development, code from it sometimes doesn't work.
- Node package manager was invented to make web-developers' lives easier. However, dependencies from the parity developers cannot be installed automatically (see init script for npm, where each dependency installed manually). Also, the project failed to be deployed multiple times on Ubuntu, making Windows only platform for DApp development.

7. Conclusion

In our work we constructed UST Token, a token that could be used internally at HKUST. We then built bridge between UST Token and Ethereum's Kovan testnet. The bridge allows UST Token to execute transactions as a result of a guarantee that an event took place on Ethereum's Kovan testnet. Blockchain interoperability is one of the first steps towards a future where blockchain plays a major role. We hope that our work will act as a guide to others attempting to build blockchain bridges. Even though we foresee our bridge example being made obsolete in the coming future as intermediary chains or relay chains are produced. Developing this project required us to collaborate with Parity's team, notify them of software issues, and merge updates of our own into their bridge framework.

A limitation of our work is that we do not explore using an intermediary relay chain between the two blockchains that can listen to each other. In the examples we explored an intermediary chain does not offer benefits in terms computational complexity since only two blockchains are being considered. However, as more chains are added the number of bridges required to allow chains to listen to one another grows at a rate of n^2. There exist approaches to introduce an intermediary chain, such as polkadots relay-chain setup which could possibly scale to the order of n. It should also be noted that the introduction of a relay chain would introduce more benefits than just a reduced computational complexity.The two largest benefits stemming from introducing a relay chain are pooled security and interchain transactability. For more on this checkout the awesome work being done on polkadot by parity technologies[5].

References

1. Ethereum White Paper. https://github.com/ethereum/wiki/wiki/White-Paper
2. Dwork, Cynthia; Naor, Moni (1993). "Pricing via Processing, Or, Combatting Junk Mail, Advances in Cryptology". CRYPTO’92: Lecture Notes in Computer Science No. 740. Springer: 139–147.
3. Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf
4. Proof of Stake. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ
5. Polkadot project. https://github.com/w3f/polkadot-white-paper/blob/master/PolkaDotPaper.pdf
6. Plasma. https://plasma.io/plasma.pdf
7. Interledger. https://interledger.org/
8. Parity. Demo PoA tutorial. https://github.com/paritytech/parity/wiki/Demo-PoA-tutorial
9. Parity. Ethereum client. https://github.com/paritytech/parity/wiki
10. Parity-bridge. https://github.com/paritytech/parity-bridge
11. Bridging contract. https://github.com/paritytech/parity-bridge/blob/master/contracts/bridge.sol
12. Primecoin. http://primecoin.io
13. DApp development tutorial. https://github.com/paritytech/parity/wiki/Tutorial-Part-1