Proposta de Modelagem de Democracia Líquida utilizando a Blockchain da plataforma Ethereum

Marcos Aurélio C de S Filho, Rafael G Damasceno, Rodrigo C R de Jesus

{113112418, 114009017, 114039371}

Escola Politécnica - Departamento de Engenharia Eletrônica e de Computação
Universidade Federal do Rio de Janeiro

{maasouza, rdamasceno, rjesus}@poli.ufrj.br

Resumo: *O objetivo deste relatório é estabelecer uma proposta de solução de problemas associados ao método atual de escolher representantes, através de uma implementação de democracia líquida em uma plataforma de blockchain. Durante esse projeto, foi implementada uma prova de conceito utilizando a plataforma Ethereum, e foram demonstradas suas relações com sistemas de comunicação e padrões de segurança envolvidos. Além disso, este relatório procura apontar as estruturas escolhidas durante o desenvolvimento dessa prova de conceito e o funcionamento final do sistema, levantando os possíveis cenários finais da solução e as vantagens deste frente ao modelo atual.*

1. Introdução

Com a disseminação da tecnologia nos mais diferentes ambientes é normal pensar em sua implementação para resolver questões recorrentes de nossa sociedade. Uma dessas questões está relacionada a forma como escolhemos quais caminhos devem ser percorridos pela nossa sociedade, e a forma mais comum de pensar sobre isso é avaliando a maneira com que fazemos política, ou seja, através da democracia representativa. Esse sistema permite que, através de eleições regulares, a sociedade delegue a um representante o direito de representá-la, ou seja, um número determinado de representantes é escolhido para exercer o poder político delegado a ele pelos eleitores durante um mandato.

Além de considerações políticas, podemos dar ênfase em questionamentos no campo de segurança da informação na maneira como esses representantes são eleitos, ou mais especificamente no fato da dessas eleições necessitarem de total confiança no processo do Tribunal Superior Eleitoral (TSE), dado que essa única instituição é responsável por elaborar o processo de votação, executar esse processo, e ser também o único órgão capaz de iniciar e ditar as regras de

possíveis auditorias, sejam elas internas ou com a participação de terceiros. Essa centralização torna a organização um ponto único de falha no processo, ou seja, caso ela seja comprometida, todo o sistema é comprometido. O trabalho documentado neste relatório busca solucionar o problema de centralização em processos eleitorais através da implementação de um sistema em que qualquer pessoa possa auditar esse processo. Para isso, estabelecemos também as funções necessárias para a implementação de uma democracia líquida.

A democracia líquida é um projeto misto de democracia direta e indireta, que permite a votação direta ou delegação de um voto a um representante, que pode ser feita de maneira geral, por tema a ser debatido ou em uma única proposta. A implementação deste conceito de maneira descentralizada é possibilitada pela tecnologia de blockchain, que permite um registro imutável de dados armazenado sem que seja necessário um centro lógico de controle. Além disso, os protocolos mais comuns de blockchain, entre eles o da plataforma Ethereum, utilizado para esse trabalho, exigem que as transações feitas em uma rede sejam autenticadas por todos os nós da rede, fazendo com que a descentralização se torne uma medida de segurança e permitindo que a democracia líquida seja um conceito mais factível, com cada nó participante na rede agindo como um auditor para garantir a idoneidade da votação.

Este artigo está elaborado em 7 seções, organizadas da seguinte forma: A seção 2 aborda o cenário atual observado e a motivação para a elaboração deste projeto; a seção 3 procura destacar os pontos principais que foram encontrados na bibliografia selecionado; a seção 4 contém a fundamentação teórica das plataformas utilizadas dentro deste projeto; a seção 5 explicita a modelagem e implementação realizada, levantando também as decisões de projeto; a seção 6 faz uma breve descrição dos testes executados durante o desenvolvimento desse trabalho e mostra a interface através da qual interagimos com a rede; por último a seção 7 conclui este artigo e explora suas possíveis aplicações e desenvolvimentos futuros.

2. Motivação

O sistema de eleição atualmente usado é alvo de grande questionamento pela população. No Brasil, em específico, o funcionamento da urna eletrônica e a manipulação na apuração são algumas das maiores preocupações, dado o desconhecimento geral sobre o processo utilizado e frequentes questionamentos trazidos à tona pelos poucos especialistas que recebem autorização do TSE para analisar a segurança desse processo, dado que o tribunal considera que o próprio processo deve ocorrer em segredo, indo contra as boas práticas de segurança ao depender de segurança por obscuridade. Essas práticas fazem com que seja praticamente impossível a existência de auditorias completamente independentes

do sistema de eleição, e consequentemente obriga na confiança em um sistema centralizado.

Um dos especialistas que obtiveram autorização para realizar testes na urna eletrônica utilizada pelo TSE, o professor Diego F. Aranha, da Universidade Estadual de Campinas (UNICAMP), se tornou um dos críticos mais ferrenhos ao sistema atual, tendo publicado um artigo[1] sobre as falhas de software encontradas na urna eletrônica brasileira por ele e sua equipe. Entre essas falhas, podemos destacar a utilização da mesma chave criptográfica para todas as urnas eletrônicas, que é armazenada sem criptografia na mídia da urna, a utilização função resumo SHA-1, que desde 2007 não é mais considerada segura, e a utilização da medida do tempo em resolução de segundos como fonte única de entropia.

Dado esse panorama, e considerando os avanços recentes em técnicas de implementação de sistemas descentralizados criptograficamente seguros, incluindo a proposta de blockchain[2] introduzida por Satoshi Nakamoto em 2008 e a construção de uma plataforma Turing-completa[3], proposta por Vitalik Buterin em 2013, possibilitam um sistema de eleição descentralizado em que não seja necessário a confiança em nenhuma das partes que participam da rede.

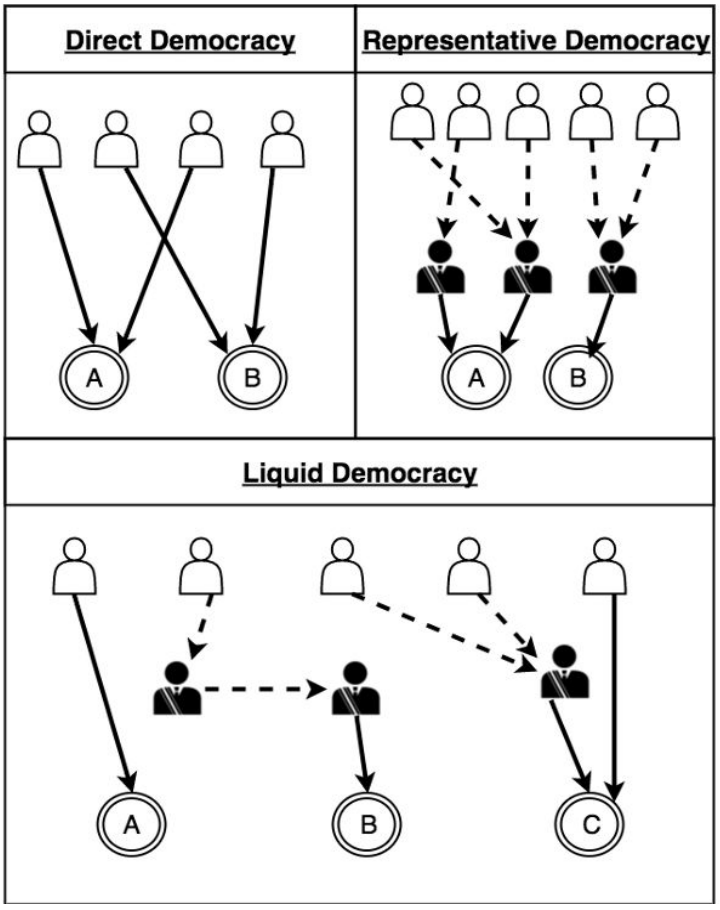
3. Revisão Bibliográfica

O trabalho seminal em blockchain, considerado como base para todos os outros vieram depois, é o artigo[2] compartilhado por Satoshi Nakamoto (pseudônimo) em um grupo sobre criptografia em 2008. Este artigo constrói em cima do algoritmo de prova de trabalho inicialmente proposto por Dwork[3] e Naor em 1992 para especificar uma moeda digital completamente descentralizada, chamada de Bitcoin, que resolvia os problemas clássicos dos esforços prévios nesse campo, sendo o principal dele o problema do *double spending*, ou seja, como garantir, sem uma autoridade central, quem é o dono atual de um conteúdo puramente digital, e assim impedir que alguém gaste a mesma moeda mais de uma vez. A solução proposta para esse problema foi associar cada bloco de dados ao hash do bloco anterior, formando uma cadeia de blocos, de onde vem o nome *blockchain*. Cada bloco dessa cadeia contém, além do hash do bloco anterior, diversas transações que representam a mudança de propriedade de uma moeda e uma assinatura indicando quando o bloco foi validado (*timestamp*). A forma como essa validação acontece será explicitada na seção 4, em que explicamos o funcionamento de um algoritmo de prova de trabalho. O nó responsável por resolver o algoritmo de prova de trabalho fará um broadcast para os outros nós com o resultado e as transações contidas naquele bloco, e os nós receptores validarão o resultado e as transações antes de adicionar o bloco na cadeia canônica, como é chamada a cadeia com maior número de blocos associados.

Os avanços propostos por Satoshi foram a base para que, em 2013, Vitalik Buterin e Gavin Wood[4] propusessem uma plataforma de computação Turing-completa funcionando de maneira completamente descentralizada, através da utilização de uma máquina virtual nos nós validadores da rede. Essa plataforma, chamada de Ethereum, funciona como uma máquina de estado baseada em transações, ou seja, existe o estado global da rede e cada transação altera o estado global da rede. Diferentemente do Bitcoin, entretanto, a transação não necessariamente se limita a mudança de posse de uma moeda, sendo na verdade descrita por um programa que funciona dentro da máquina virtual da plataforma. O funcionamento desta plataforma será melhor descrito na seção 4.2.

Além disso, outro artigo consultado foi *Liquid Democracy: Potentials, Problems, and Perspectives*[5], escrito por Blum e Zuber em 2015, que mostra um modelo básico de democracia líquida que utilizamos como base para o desenvolvimento do projeto proposto. A democracia líquida permite uma tomada de decisão coletiva combinando democracia direta e representativa, com todos os eleitores podendo votar em projetos, mas também sendo possível delegar seu voto a alguém de confiança, cujo julgamento o eleitor considere apto.

Figura 1: Ilustração do funcionamento das diferentes formas de democracia.



Fonte: Dominik Schiener em medium.com como organizer-sandbox.

O artigo também leva em consideração alguns benefícios, como a não existência de mandatos políticos, o que permite que o eleitor possa recuperar seu poder político imediatamente caso o delegado tomar decisões que forem contra o modo de pensar do eleitor, e malefícios desse sistema, dentre os quais podemos destacar a possibilidade da centralização de poder de voto na mão de poucas pessoas.

4. Fundamentação teórica

4.1. Funcionamento de uma Blockchain

A blockchain pode ser descrita como uma estrutura de dados composta por uma sequência de blocos, em que cada bloco aponta para o bloco anterior e tem em seus dados as transações que estão sendo validadas. Cada transação é formada por uma origem, destino e a quantidade transferida de *UTXO* (*Unspent Transaction Output*), além de uma taxa que será transferida para a conta do nó responsável por validar o bloco no qual a transação está inserida. Para efetuar a transferência o proprietário atual assina digitalmente com sua chave privada um hash da transação anterior do *UTXO* combinado com a chave pública do destinatário, esse hash é utilizado para conectar o bloco atual com o bloco anterior.

Para um bloco ser validado, ele precisa conter em seu header a informação do *nonce*. *Nonce* é o nome do dado que, junto com os dados do bloco anterior da cadeia, formará um hash que comece com o número de 0 especificados na dificuldade da rede; essa dificuldade é atualizada dinamicamente para que o tempo entre os novos blocos adicionados na blockchain seja constante. O Bitcoin utiliza o algoritmo SHA-256 para calcular esse hash. Como a única maneira de descobrir qual é o dado correto é através da força bruta, isso se torna extremamente custoso; para construir um incentivo para que os nós da rede façam as computações necessárias para a validação de um bloco, cada bloco gerado traz com ele uma determinada fração de Bitcoin, que, junto das taxas das transações presentes naquele bloco, é atribuída à conta do nó responsável por gerar aquele bloco. O ato de buscar o *nonce* é chamado de mineração.

A dificuldade da rede é proporcional ao número de 0 que deve existir no começo do hash gerado. O fato dessa dificuldade ser ajustada dinamicamente serve o propósito duplo de evitar que a blockchain cresça exponencialmente, o que impediria que a maioria das pessoas participassem ativamente da validação da rede e anularia uma das principais vantagens desse sistema, enquanto limita também a velocidade com que são geradas novas moedas, regulando a economia gerada em torno delas.

Ao validar um bloco, ou seja, achar o *nonce* e checar a validade das transações no bloco, o minerador deve compartilhar o bloco com outros nós validadores. Esses nós, quando recebem o bloco, devem checar que o hash do nó

anterior junto ao nonce começa pelo número de 0 estabelecidos pela dificuldade da rede. Sendo assim, o algoritmo básico que rege o funcionamento de uma blockchain pode ser descrito como sendo:

1. Novas transações são enviadas aos nós validadores
2. Cada nó coleta as transações em um bloco
3. Cada nó tenta achar o nonce necessário para resolver o problema de prova de trabalho para seu bloco
4. Quando um nó acha o nonce correto, envia essa informação para os outros nós
5. Nós aceitam o bloco apenas se todas as transações no bloco forem válidas
6. Nós representam aceitação de um bloco utilizando o hash desse bloco no header do próximo bloco da cadeia

O algoritmo de prova de trabalho é necessário para evitar fraudes, ou seja, para tornar os dados da blockchain imutáveis. Isso acontece porque os dados presentes em blocos anteriores da rede só podem ser mudados se os blocos em todos os blocos subsequentes tenham seus *nonces* recalculados, uma vez que ele dependerá do hash do bloco anterior. Considerando que a cadeia canônica (ou seja, aquela aceita como correta pela rede) deve ser sempre a cadeia mais longa de blocos, para um atacante modificar dados da blockchain seria necessário poder computacional maior do que todo o resto da rede, dado que o atacante teria que recalcular todos os *nonces* posteriores e mais os *nonces* que forem achados durante o tempo em que o atacante precisou recalcular os *nonces* antigos. Outra consequência do uso da cadeia mais longa como sendo a canônica é que existe uma probabilidade, mesmo que baixa, de um bloco válido ser substituído por outros dois blocos, também válidos, de uma segunda cadeia. O mecanismo desse acontecimento foge ao escopo deste trabalho, mas o resultado prático é que a maioria dos sistemas baseados em blockchain determinam um número determinado de blocos que devem existir depois do bloco que valida uma transação para considerar essa transação como tendo acontecido. O mais comum é utilizar 6 blocos para isso.

Esse funcionamento faz com que todo nó validador de uma blockchain, ou seja, todo nó que participe do processo de mineração, esteja na verdade fazendo uma auditoria da rede, com qualquer tentativa de fraude ficando imediatamente aparente. Qualquer pessoa interessada em verificar a idoneidade de um serviço baseado em blockchain precisa apenas conectar um nó validador a essa blockchain.

4.2. A plataforma Ethereum

Tendo estabelecido o funcionamento de uma blockchain, podemos detalhar as especificidades da blockchain da plataforma utilizada como base para esse trabalho, o Ethereum, que é descrito no artigo de seu cofundador Gavin Wood como

sendo uma máquina de estados baseada em transações. Enquanto o Bitcoin (e a maioria das outras chamadas criptomoedas) utilizam um sistema baseado em transferência de posse de moedas, o Ethereum faz com que a transação seja na verdade um programa. Ou seja, a validação de uma transação passa a ser na verdade a validação do estado que ocorre depois que determinadas instruções são aplicadas ao estado global da rede.

Como seria impraticável, além de extremamente inseguro, exigir que as instruções rodassem como um programa comum nos nós validadores, o Ethereum usa uma máquina virtual específica, chamada de Ethereum Virtual Machine (EVM), onde as instruções são aplicadas. Programas feitos para a EVM devem ser desenvolvidos em Solidity, uma linguagem construída para se assemelhar a JavaScript,que conta com a maior parte do suporte existente, ou em Serpent, uma linguagem construída para se assemelhar a Python, menos utilizada. Tanto Solidity quanto Serpent implementam apenas frações das linguagens em que são baseadas, dadas as necessidades da EVM. Pelas possibilidades criadas ao seguir essas instruções, o Ethereum é comumente referido como uma plataforma de contratos inteligentes, nome esse que também é dado ao análogo das classes que é implementado em Solidity. Por isso a programação na plataforma Ethereum é considerada como sendo orientada a contratos (*contract oriented programming*, ou simplesmente COP).

Outra diferença significativa da plataforma Ethereum para a blockchain utilizada no Bitcoin é a economia por volta do pagamento de taxas. Enquanto cada transação no Bitcoin tinha uma taxa associada, e os nós validadores tinham um incentivo econômico para colocar no bloco as transações que pagavam as maiores taxas, no Ethereum nem toda transação é construída de maneira igualitária, ou seja, transações diferentes podem exigir tempos de computação diferentes.

O Ethereum utiliza o conceito de *gas*. Gas, assim como a gasolina de um carro, é o combustível de um contrato, ou seja, o que fará com que esse contrato seja executado pelos nós validadores; *gas* é definido como uma fração da moeda Ether (ETH), e pode ser recalculado a medida que a moeda se valorizar ou desvalorizar. Contratos de maior complexidade computacional precisam de maior *gas* associado para serem executados. O mecanismo de como o *gas* é pago aos nós validadores e as mudanças de estados decorrentes disso seguem duas regras básicas: Caso a quantidade de *gas* associada a um contrato seja superior à quantidade de *gas* necessária, a sobra é devolvida ao dono do contrato;entretanto, caso a quantidade de *gas* seja insuficiente, as alterações de estado causadas pelo contrato até o ponto em que este ficou sem *gas* serão revertidas, mas o *gas* gasto não será devolvido ao dono do contrato.

Para a implementação do trabalho proposto neste artigo, optamos por construir uma rede privada, conceito que permite que qualquer um crie uma rede aos moldes da rede canônica do Ethereum, com a diferença de que ignoramos seu histórico (que começa com um bloco genesis criado por nós, cuja especificação está

disponível no Anexo 1 deste relatório) e que todos os nós verificadores da rede estão sob nosso controle, além do *gas* utilizado na rede não ter valor monetário real. Além disso, a dificuldade da rede passa a ser ajustada baseada no poder computacional dos nossos nós, ao invés de levar em consideração toda a rede Ethereum. A implementação dessa rede privada é descrita em maiores detalhes na seção 5.2.

5. Proposta de Modelagem e Implementação

5.1. Democracia Líquida

Como foi descrito no final da seção 3 - Revisão Bibliográfica, os preceitos de uma democracia líquida deveriam garantir a possibilidade de se criar uma proposta que pode ser votada diretamente ou através de votos delegados para outros participantes. Além disso, para este projeto foi pensado na possibilidade de requerer os votos delegados de volta e consequentemente desfazer um voto, desde que a proposta ainda não tenha sido encerrada. Foi necessário também em uma forma de realizar a autenticação dos votantes de forma que seja independente da *blockchain,* já que armazenar dados na *blockchain* é extremamente custoso, além de não ser possível modificar esses dados. Inicialmente optamos por utilizar a biblioteca *Oraclize*, que permite que dados externos a *blockchain* seja lido por ela através do conceito de oráculo. Oráculo, na terminologia utilizada pela plataforma Ethereum, é um agente externo à blockchain que verifica informações do mundo real e insere essas informações na blockchain. Entretanto, por limitações de tempo e complexidade, a integração com o *Oraclize* não foi implementada na prova de conceito descrita neste relatório. Sendo assim, a função de autenticação foi implementada sem verificações, apenas fornecendo um voto ao usuário que pedisse para ser autenticado.

A modelagem para este sistema foi possível com a declaração de estruturas *Proposal* e *Voter*, que representa uma proposta que será votada e um possível eleitor, respectivamente. Para permitir o funcionamento de uma eleição foram criados relacionamentos entre o endereço da carteira de uma pessoa e um *Voter*, para garantir que ela só vote uma única vez, e também entre um id e uma *Proposal*, para poder recuperar resultados de votações anteriores e saber qual votação está ativa no momento.

Quando um votante é autenticado, ele tem seu endereço mapeado em um *Voter* e ganha o direito de ter um voto. A decisão por implementar uma função de autenticação ao invés de inicializar cada endereço com um voto se deu pelo fato de ser possível cada pessoa gerar um número ilimitado de endereços.

Os principais métodos implementados para o funcionamento esperado do contrato foram *createProposal*, *vote*, *undoVote*, *transfer*, *transferBack*, *askBack*.

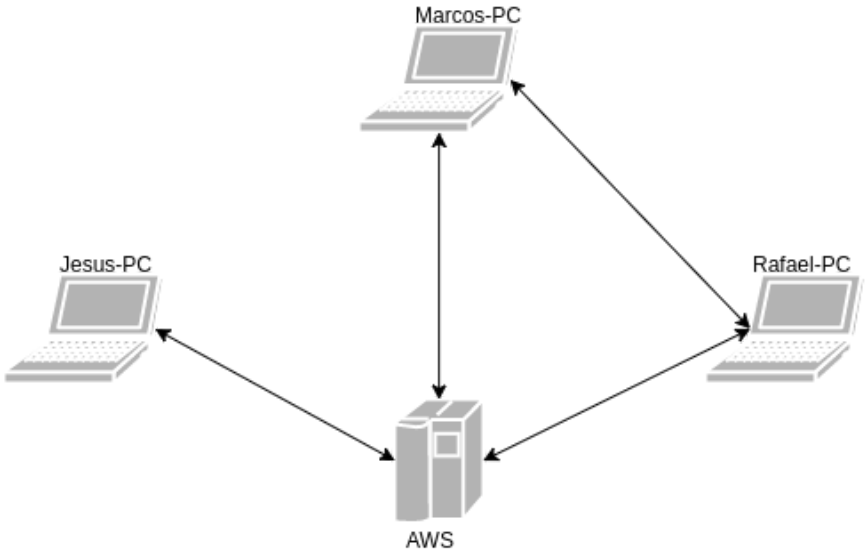
Além dos nomes relativamente descritivos, mais sobre o que faz cada função está nos comentários do código, disponível no repositório[7] deste trabalho.

O desenvolvimento do contrato foi realizado com a linguagem *Solidity*, durante esta elaboração algumas dificuldades surgiram inerentes à linguagem ou as condições de complexidade exigidas. A primeira delas, foi a falta de suporte a *arrays* multidimensionais de tamanho variável, que seria utilizado para armazenar as possíveis opções de votos de uma *Proposal*. Além disso a linguagem também não permite a criação de funções com complexidade ciclomática maior que 7.

5.2. Redes privadas

Para este projeto criamos uma rede privada utilizando nós nos nossos computadores privados e com um nó externo na *Amazon Web Services (AWS)*. A topologia da rede utilizada para realizar os testes descritos na seção 6 pode ser visualizada na imagem abaixo.

Figura 2: Topologia da rede privada.



Fonte: Autoria própria.

Para construir nossa própria rede privada tivemos que criar um bloco genesis, nome comumente dado ao primeiro bloco de uma blockchain, através da descrição de um arquivo JSON da rede. Os parâmetros desse arquivo são:

- **nonce**: Um hash de 64 bits que junto com o mix-hash, prova que uma quantidade de trabalho computacional foi feita nesse bloco, a combinação dele com o mix-hash deve satisfazer as condições descritas no artigo de Gavin Wood[4].

- **timestamp**: Número de segundos do sistema em que o bloco foi gerado, ele é usado para ajustar a dificuldade da mineração do próximo bloco. Ele também pode ser utilizado para verificar a ordem dos blocos na cadeia.

- **parentHash**: Um hash Keccak (SHA-3) de 256 bits que aponta para o bloco anterior ao bloco atual, como este é o bloco inicial, esse valor corresponde a 0.

- **gasLimit**: Quantidade máxima de *gas* que pode ser gasto por bloco.

- **difficulty**: Dificuldade atribuída para a descoberta do valor do nonce, quanto maior a dificuldade mais cálculos um minerador terá que realizar para descobrir um novo bloco.

- **mix-hash**: Um hash de 64 bits que junto com o nonce, prova que uma quantidade de trabalho computacional foi feita nesse bloco, a combinação dele com o mix-hash deve satisfazer as condições descritas no artigo de Gavin Wood[4].

- **coinbase**: Endereço da carteira do minerador que minerou o bloco.

- **alloc**: Permite pré-definir uma lista de carteiras inicializadas com Ether.

O bloco genesis utilizado foi gerado com *nonce* e uma id de rede aleatórios. Para existir conexão entre nós da rede, é preciso não só que os dois nós estejam com o mesmo id de rede mas que eles concordem com uma mesma blockchain. A construção de um bloco genesis próprio, junto a escolha de um id de rede relativamente baixo, nos permitiu ver, através de uma das configurações de verbosidade do cliente que utilizamos para nos conectar à rede, sucessivas tentativas de conexão com nós de fora da rede, e consequentes divergências por discordarem sobre o bloco genesis, ou seja, por estarem com blocos genesis diferentes. Esse funcionamento observado é resultado de outros computadores utilizando a mesma ID para outra rede privada, e se conectando aos computadores utilizados para esse trabalho em uma tentativa de fazer parte da mesma blockchain. Como é necessário que exista um consenso sobre a cadeia canônica para que dois nós participem da mesma rede, os nós que não tinham nosso bloco genesis eram rapidamente descartados.

Além desses parâmetros, para a conexão dos nós da rede privada geramos um arquivo static-nodes.json que descrevia alguns dos nós da rede. A partir do momento que um dos nós se conecta a outro nó da rede, o Ethereum tem um protocolo de descobrimento de nós por meio do qual os nós compartilham informações entre si.

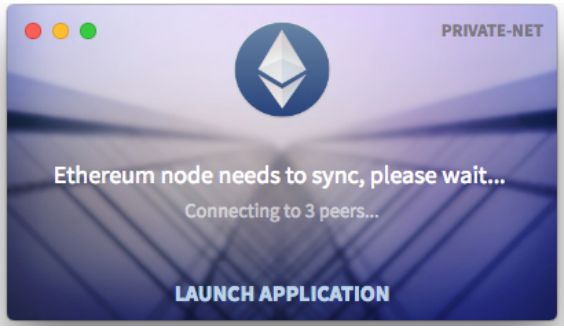
A conexão com a rede privada foi feita através do cliente *geth,* que é o cliente principal do Ethereum. Além disso, foi utilizado também o browser Mist e a Ethereum Wallet, aplicações que provém uma interface gráfica para fazer a interação com a rede e os contratos inteligentes disponíveis nela. A conexão com a rede estabelecida pelo *geth* é feita através de um arquivo IPC gerado quando o geth é chamado.

6. Funcionamento

6.1. Interação com a carteira e o contrato

Como descrito na seção anterior, a Ethereum Wallet oferece uma interface gráfica para interagir com a rede e os contratos inteligentes. Ao iniciar a aplicação, após a conexão ter sido estabelecida pelo *geth*, é possível criar uma nova conta e fazer uso da modelagem proposta por este projeto.

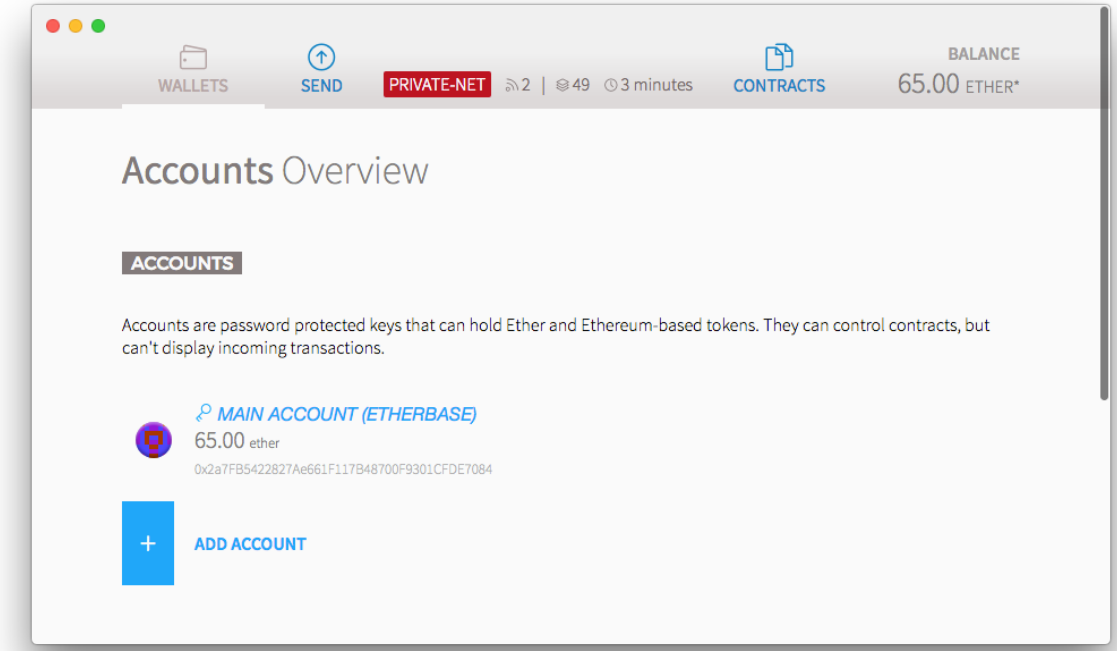
Figura 3: Tela de carregamento da Ethereum Wallet.



Fonte: Captura de tela da aplicação Ethereum Wallet.

Após iniciar a aplicação é exibida a opção de criar uma nova conta com uma senha que será utilizada para aprovar qualquer interação com outros membros da rede ou com os contratos inteligentes. A tela inicial exibe as principais informações da conta e a carteira associada, como o endereço, o saldo de Ether e as últimas transações.

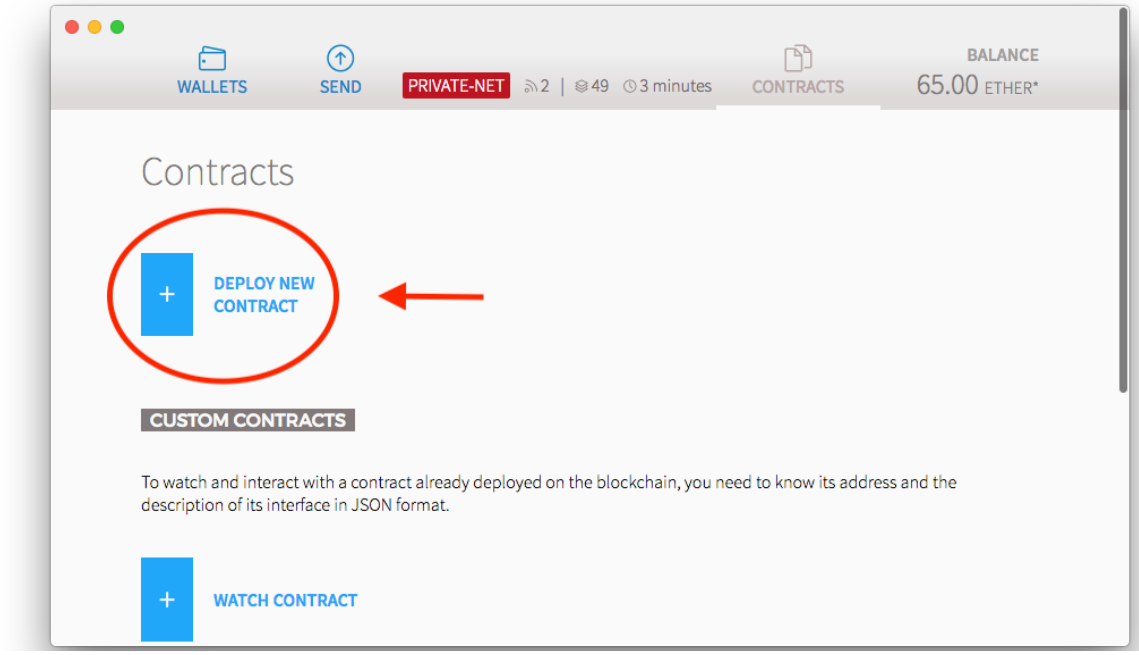
Figura 4: Visão geral da carteira em uma conta na rede privada.



Fonte: Captura de tela da aplicação Ethereum Wallet.

Na aba *Contracts* é possível acompanhar todos os contratos que o usuário iniciou ou se inscreveu, assim como os tokens personalizados destes contratos, que são as moedas construídas na plataforma Ethereum. Na barra superior é possível ver em qual rede estamos conectados (no caso, uma rede privada), o número de nós aos quais estamos conectados, qual é o número do último bloco da rede, e quanto tempo se passou desde que o último bloco foi minerado. Para demonstrar o funcionamento do contrato desenvolvido neste projeto é preciso inicialmente realizar o *deploy* do novo contrato através da opção em destaque da Figura a seguir.

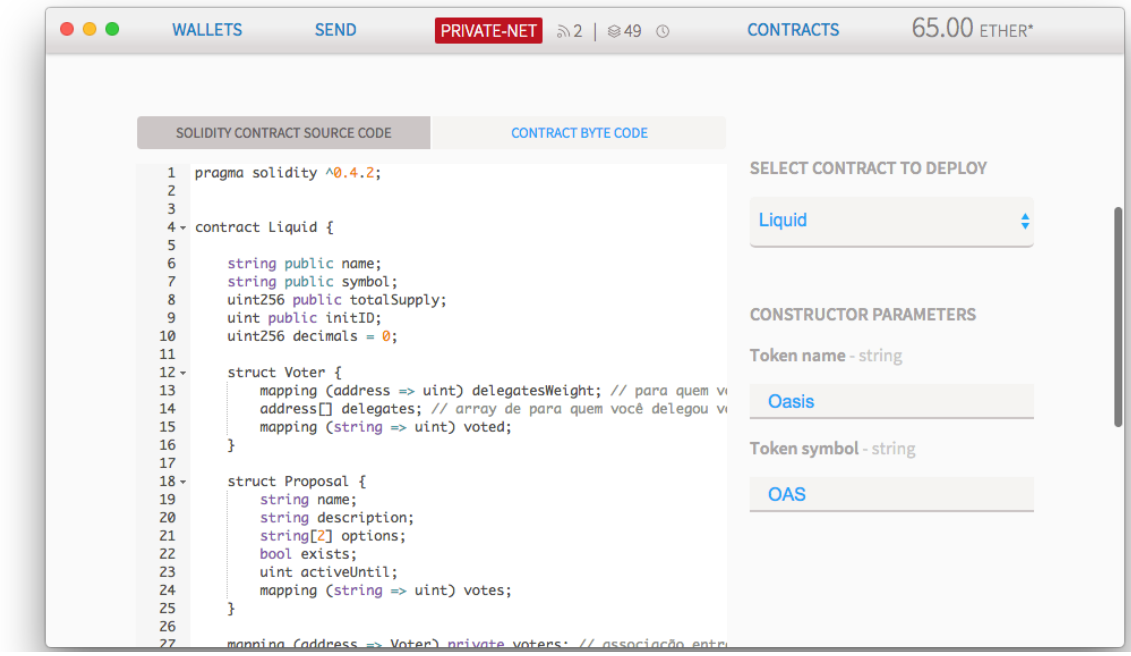
Figura 5: Opção de iniciar um novo contrato na rede privada.



Fonte: Captura de tela da aplicação Ethereum Wallet.

Para a criação de um contrato o método mais fácil e utilizar diretamente o código não compilado. Entretanto é possível utilizar a IDE Remix (https://remix.ethereum.org) para desenvolver o contrato e gerar o *bytecode* que pode ser diretamente importado para aplicação. Ao adicionar o código na opção de *deploy* da Ethereum Wallet, são exibidos os parâmetros construtores, que foram definidos no código, a serem preenchidos.

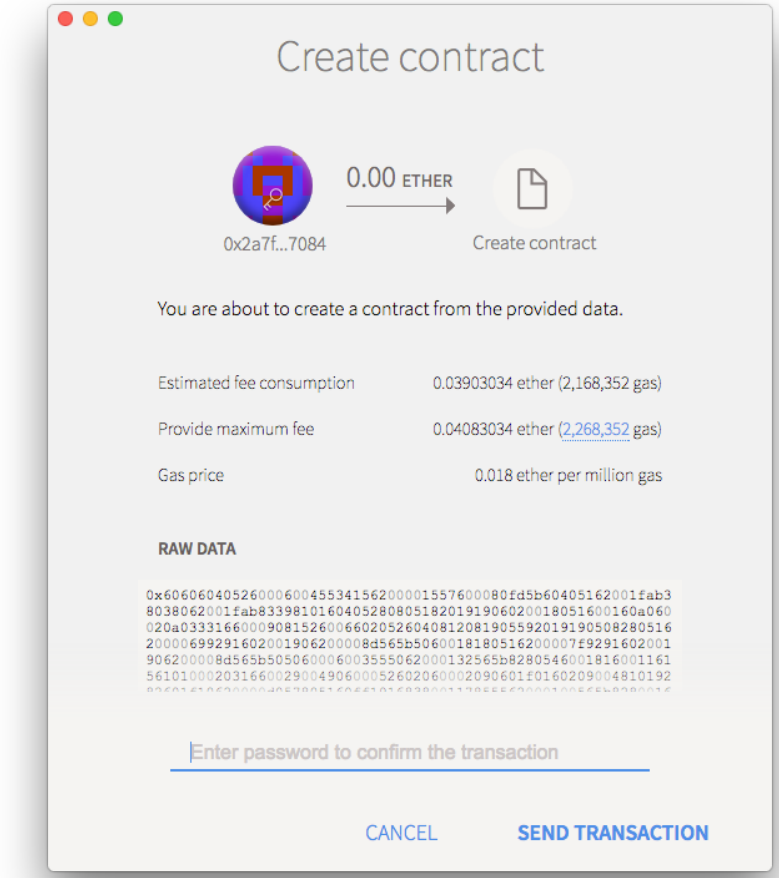
Figura 6: Iniciando a criação de um contrato a partir do seu código.



Fonte: Captura de tela da aplicação Ethereum Wallet.

Ainda na opção de *deploy* de um contrato inteligente é possível selecionar a taxa máxima que poderá ser utilizada para processar as transações, ou seja, o limite superior de *gas* que poderá ser utilizado antes que a transação seja abortada. Como dito anteriormente, caso o custo efetivo de *gas* seja inferior a esse limite, o excedente é devolvido à conta originadora da transação. Após selecionada a opção de deploy é exibida uma tela de confirmação, destacando o gasto previsto de *gas* e já pré visualizando o endereço do contrato e a sua imagem. Essa imagem é gerada automaticamente a partir do endereço, e é comumente utilizada como confirmação que seus dados estão sendo enviados para o endereço correto. Sendo assim, erros que ocorreriam como consequência de um *typo* podem ser evitados.

Figura 7: Tela de confirmação de transação para criar o contrato.

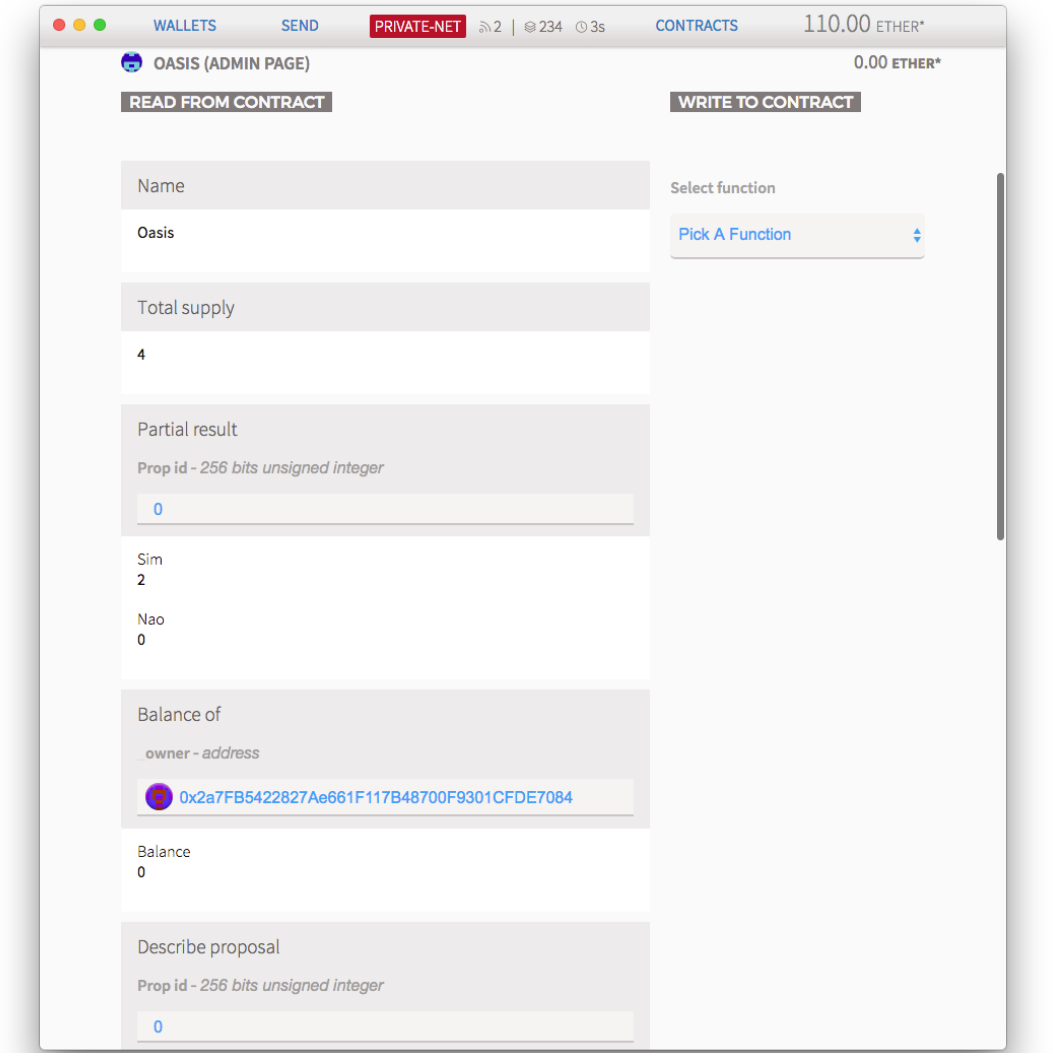


Fonte: Captura de tela da aplicação Ethereum Wallet.

Após confirmar a criação do contrato já é possível interagir com o mesmo. Para que outros usuários sejam capazes de visualizar e utilizar com o contrato é necessário o endereço do mesmo, assim como o *Contract JSON Interface* que explicita para a carteira as interações disponíveis naquele contrato como, por exemplo, chamar funções e receber dados.

A plataforma divide as funções em funções de leitura, que são as que possuem algum retorno e não alteram o estado da *blockchain*,exibidas ao longo da página, e funções de escrita, que normalmente geram eventos e alteram o estado da *blockchain*. Essas, quando escolhidas pelo seletor de função, exibem os parâmetros que devem ser preenchidos e geram um pedido de confirmação de transação similar ao da criação de contrato. Funções de escrita necessitam de *gas* para serem processadas, enquanto funções de leitura são processadas localmente.

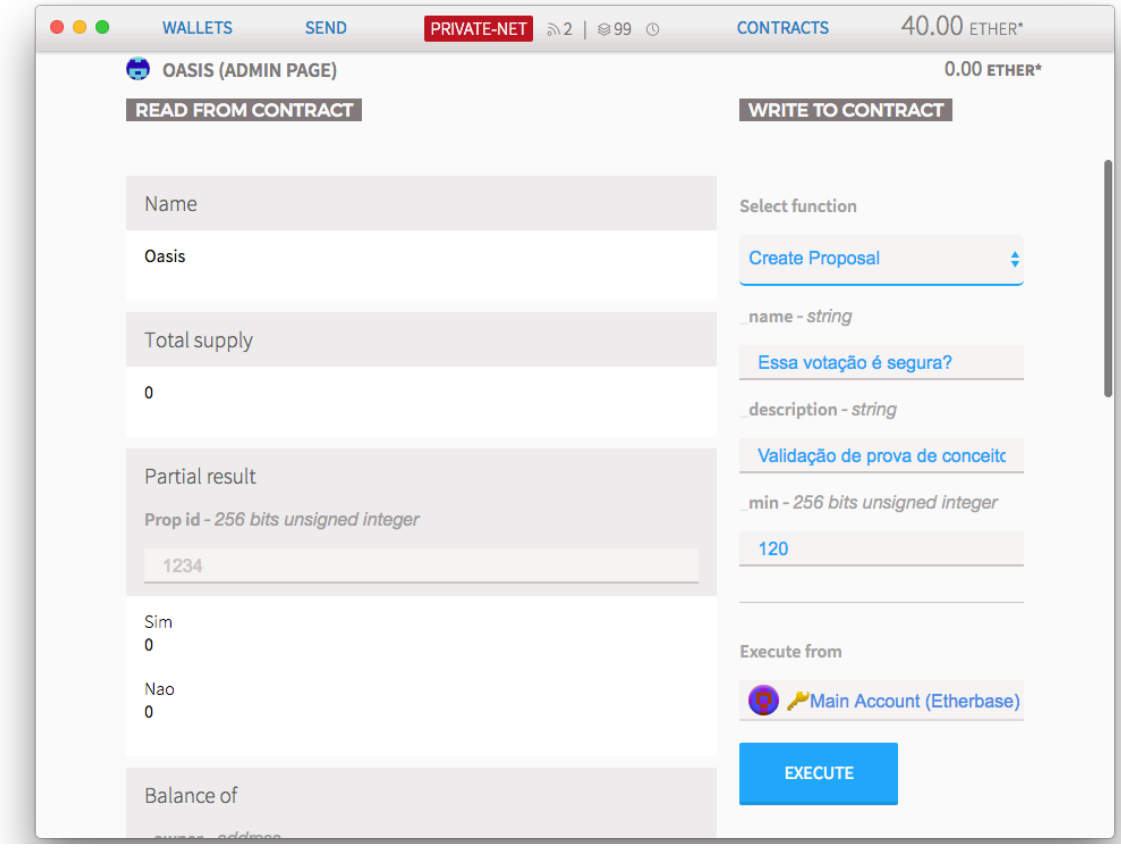
Figura 8: Visualização de algumas funções existentes no contrato



Fonte: Captura de tela da aplicação Ethereum Wallet.

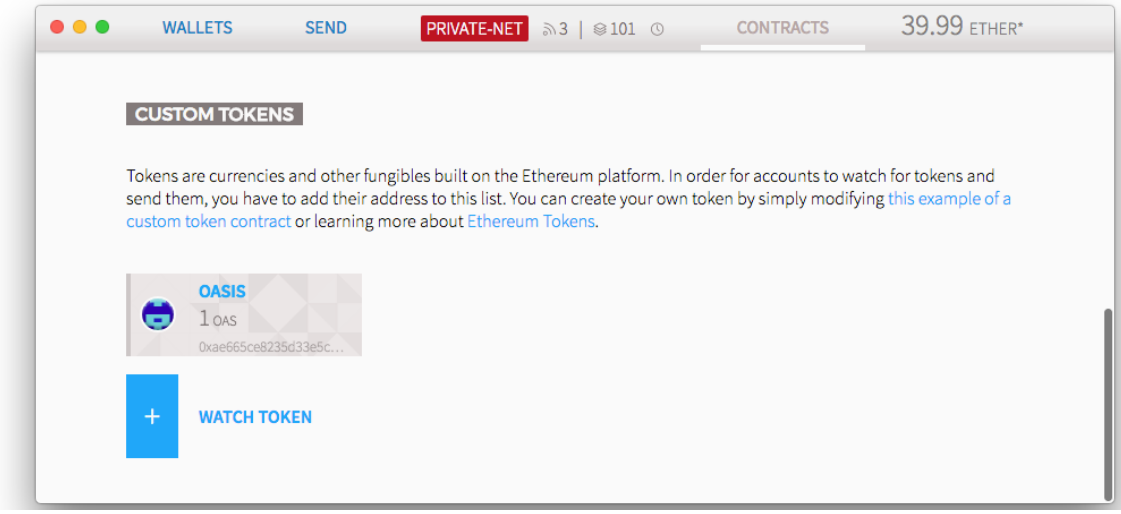
No contrato desenvolvido neste projeto, inicialmente os usuários não possuem *tokens*, sendo necessário executar a função de autenticação desenvolvida para que uma unidade seja obtida. Assim como para a criação do contrato, uma tela de confirmação é exibida para autorizar a transação. Cada token corresponde ao direito de um voto. Os tokens, depois de recebidos, podem ser delegados, pedidos de volta, ou utilizados para votar em propostas. Criar propostas não exige tokens específicos.

Figura 9: Função de criação de proposta dentro do contrato.



Fonte: Captura de tela da aplicação Ethereum Wallet.

Figura 10: Visualização do *token* personalizado do contrato.

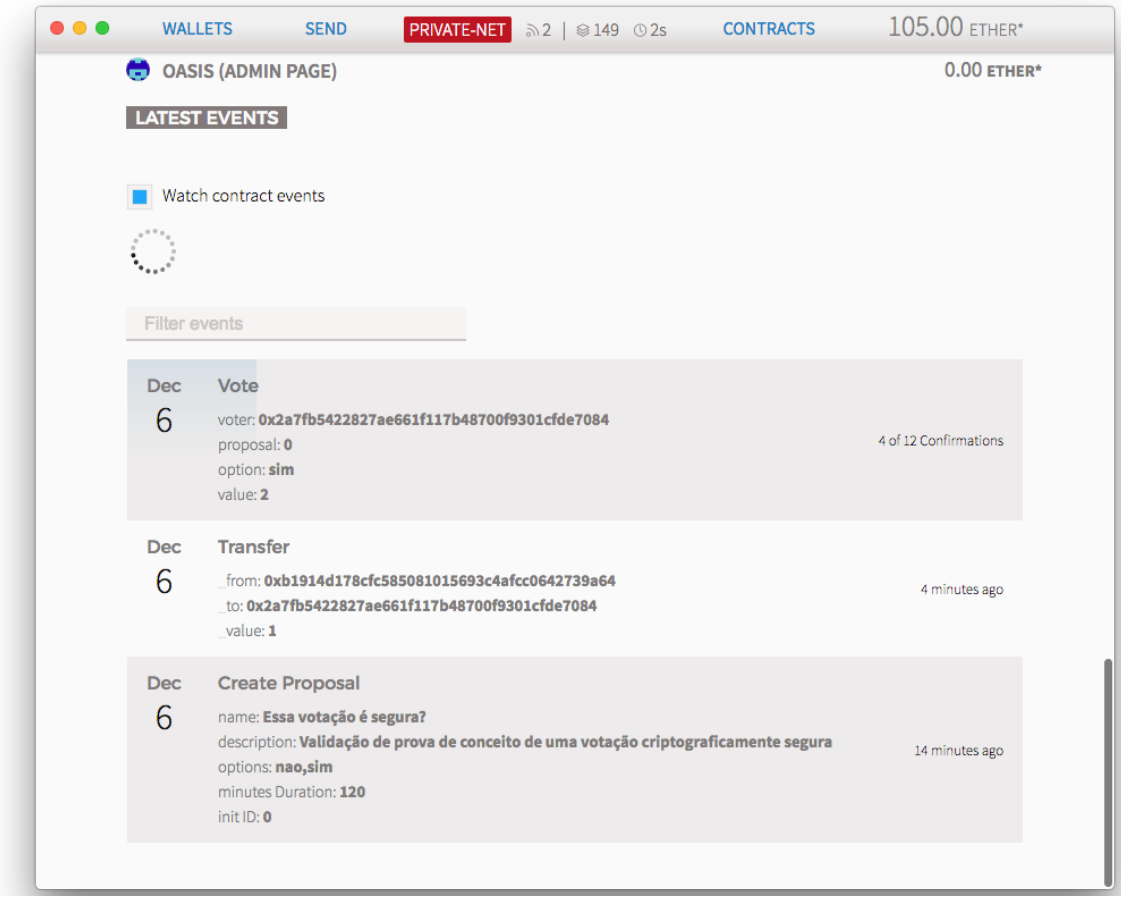


Fonte: Captura de tela da aplicação Ethereum Wallet.

Ao executar uma função de escrita, um ou mais eventos podem ser gerados. Esses eventos são definidos no contrato, e normalmente descrevem o que ocorreu

na transação. Para um evento ser efetivado, pelo menos uma confirmação deve ser realizada pelos nós validadores. Eventos que ocorreram na blockchain são exibidos em uma visualização separada na página do contrato, como pode ser observado na figura a seguir.

Figura 11: Visualização dos últimos eventos ocorridos na blockchain



Fonte: Captura de tela da aplicação Ethereum Wallet.

6.2. Testes

Para validar o que foi proposto, foi realizado o *deploy* do contrato que está disponível no repositório[7] deste projeto e simulado uma votação com uma proposta ativa e três eleitores que após a autenticação ganharam direito a um voto cada. Após a autenticação ser confirmada,diferentes testes foram executados com o objetivo de garantir que o sistema funcionasse como esperado. Segue abaixo uma lista dos testes e comportamentos obtidos:

- Cada eleitor votou diretamente e requisitou seu voto de volta. O sistema funcionou como planejado, devolvendo o voto se a proposta ainda estivesse ativa e avisando que não seria possível realizar a transação de devolução se a mesma já estivesse concluída;
- O eleitor ***A*** delegou seu voto ao eleitor ***B***, e em seguida requisitou seu voto de volta. O sistema funcionou como planejado, o eleitor ***A*** recebeu seu voto

de volta e o eleitor ***B*** teve seu número de votos diminuído, mesmo que o eleitor ***B*** tivesse delegato o voto a um eleito ***C***. Neste caso, o eleitor ***B*** requisitou de volta ao eleitor ***C*** o número de votos requisitados pelo eleitor ***A****;*

- O eleitor ***A*** delegou seu voto ao eleitor ***B***, ao receber o voto delegado o eleitor ***B*** votou em uma proposta, e em seguida o eleitor ***A*** requisitava o seu voto de volta. O sistema funcionou como planejado, desfazendo primeiro os votos realizados pelo eleitor ***B*** (buscando desfazer o mínimo de votos possível, porém removendo em pares caso as duas opções possíveis tenham sido votadas), para então o eleitor ***B*** realizando a devolução ao eleitor ***A***.

7. Conclusão

As tecnologias exploradas por este projeto demonstram a possibilidade de elaborar uma democracia líquida que se apoia em um sistema de eleição criptograficamente seguro através do uso de uma plataforma de blockchain. Os conceitos necessários para a implementação de uma democracia líquida foram modelados de acordo com nosso entendimento de quais eram as características essenciais para uma prova de conceito.

O funcionamento bem sucedido dentro de uma elaboração relativamente simples demonstra a viabilidade de sua implementação em diferentes cenários, mesmo diferentes da política nacional, principalmente pela segurança oferecida e a possibilidade de auditar uma votação. No cenário da universidade existem diversas votações, como eleição do reitor ou de grêmio estudantil que poderiam fazer uso deste sistema, inclusive utilizando redes privadas com nós institucionais da universidade. Essas redes poderiam disponibilizar as informações necessárias para a implementação de nós verificadores que permitiriam que qualquer um auditasse essas eleições.

Como continuação este projeto poderia ser integrado a uma plataforma web de interface amigável e mais ilustrativa, onde seria possível votar, delegar votos, ou consultar resultados de propostas.

Agradecimentos

Gostaríamos de agradecer ao professor Luis Felipe Magalhães de Moraes, responsável pela disciplina de Tópicos Especiais em Sistemas de Comunicação, e o doutorando Evandro Luiz Cardoso Macedo por compartilharem seus conhecimentos.

Bibliografia

[1] ARANHA, Diego F. et al. Vulnerabilidades no software da urna eletrônica brasileira. **Relatório Técnico**, v. 18, p. 19, 2012.

[2] NAKAMOTO, Satoshi. Bitcoin: A peer-to-peer electronic cash system. 2008.

[3] DWORK, Cynthia; NAOR, Moni. Pricing via processing or combatting junk mail. In: **Annual International Cryptology Conference**. Springer, Berlin, Heidelberg, 1992. p. 139-147.

[4] WOOD, Gavin. Ethereum: A secure decentralised generalised transaction ledger. **Ethereum Project Yellow Paper**, v. 151, 2014.

[5] BLUM, Christian; ZUBER, Christina Isabel. Liquid Democracy: Potentials, Problems, and Perspectives. **Journal of Political Philosophy**, v. 24, n. 1, 2015.

[6] SIRI, Santiago et al. The Social Smart Contract. https://github.com/DemocracyEarth/paper Acessado em: 25/11/2017

[7] Repositório Democracia líquida. https://github.com/maasouza/EEL840

Anexo 1

Arquivo *genesis.json* utilizado para inicializar a rede privada utilizada para os testes descritos neste relatório.

```
{
    "nonce": "0x0103955044184486",
    "timestamp": "0x0",
    "parentHash":
      "0x0000000000000000000000000000000000000000000000000000000000000000",
    "gasLimit": "0x8000000",
    "difficulty": "0x420",
    "mixhash":
      "0x0000000000000000000000000000000000000000000000000000000000000000",
    "coinbase": "0x0000000000000000000000000000000000000000",
    "alloc": {
    },
    "config": {
        "chainId":840840
    }
}
```