USO DE BLOCKCHAIN EM SISTEMAS DE ELEIÇÃO

TCC - ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - 2019/01

Elias Appio Mezalira [1]
Instituto Federal de Educação Ciência e Tecnologia do Rio Grande do Sul - Campus Farroupilha
Rafael Viera Coelho [2]
Instituto Federal de Educação Ciência e Tecnologia do Rio Grande do Sul - Campus Farroupilha

Abstract: With the adoption of electronic voting systems, the security and transparency surrounding elections has become the subject of many questions. The blockchain comes up to solve these problems, but with a focus on the financial sector. This article makes a study about the use of blockchain to develop an election system, seeking to evaluate the feasibility of using the technology for this purpose. The search for the objective is based on analysis of the use cases, raising the advantages and difficulties that involve the use of blockchain for elections, comparing the most popular tools on the market and developing a simple application, based on the use of the Hyperledger Fabric framework and addressing the characteristics raised with the analysis of use cases.

Keywords: Blockchain, election and P2P.

Resumo: Com a adoção de sistemas de votação eletrônica, a segurança e transparência que envolvem eleições se tornou motivo para diversos questionamentos. O *blockchain* surge para resolver estes problemas, porém com foco no setor financeiro. Este artigo faz um estudo sobre o uso de *blockchain* para desenvolvimento de um sistema de eleições, buscando avaliar a viabilidade do uso da tecnologia para este fim. A busca pelo objetivo se dá a partir de analises dos casos de uso, levantando as vantagens e dificuldades que envolvem o uso do *blockchain* para eleições, comparação das ferramentas mais populares existentes no mercado e desenvolvimento de uma aplicação simples, tendo como base o uso do *framework Hyperledger Fabric* e abordando as características levantadas com a analise dos casos de uso.

Palavras-chave: Blockchain, Votação e P2P.

1 INTRODUÇÃO

Em 2008, o pseudônimo Satoshi Nakamoto publica o artigo "*Bitcoin, A Peer-to-Peer Eletronic Cash System*", apresentando um sistema de moedas virtuais seguro, distribuído, auditável e sigiloso e criando a moeda virtual *Bitcoin*. Por utilizar um sistema descentralizado e criptografado, as transações utilizando o *Bitcoin* são verificados por todos os nós da rede, porém os envolvidos na transação não são identificados, tornando inviável que uma entidade central ou governamental influencie aspectos da moeda. Essa particularidade fez com que o *Bitcoin* fosse amplamente adotado no mercado ilegal.

Como é possível observar na Figura 1, no início de 2018, a moeda virtual *Bitcoin* ganha fama. Este acontecimento faz com que as tecnologias que envolvem o sistema recebessem atenção, sendo estudadas e recebendo um grande número de adeptos. Entre estas tecnologias, o *blockchain* foi a que ganhou maior destaque. Conforme os dados encontrados no gráfico 1 tanto o *bitcoin* quanto o *blockchain* tiver uma queda de interesse em suas pesquisas.

[1] E-mail: eliasappio@gmail.com - Discente
[2] rafael.coelho@farroupilha.ifrs.edu.br - Orientador

Através do *blockchain* é possível criar um livro razão imutável, o que fez com que diversos setores da indústria adotassem este sistema. Segundo Marion (1985), o livro razão é responsável por manter um registro com todas as movimentações financeiras de uma entidade.

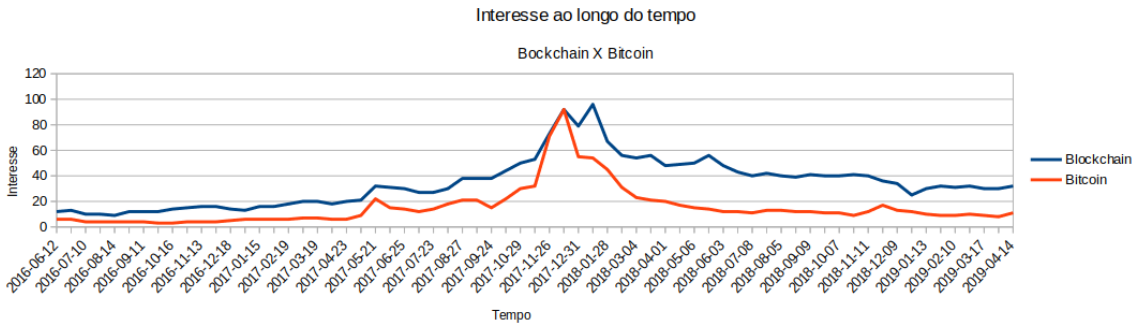

Figura 1 – Interesse ao longo do tempo: Blockchain X Bitcoin. Fonte: Google Trends, 2019

No início de 2019, a empresa Stackoverflow divulgou sua pesquisa anual, mostrando que o real uso *blockchain* não segue a mesma tendencia de interesse sobre o tema. Conforme pesquisa publicada, 15,6% dos desenvolvedores questionados sobre o uso do *blockchain* acreditam que o uso do *blockchain* é "um uso irresponsável de recursos", enquanto 29.2% acreditam que "o *blockchain* pode ser utilizado em diversos domínios e alterar nossas vidas em diversos aspectos".

Na mesma pesquisa, é possível encontrar dados sobre o uso do *blockchain* em empresas. Estes dados mostram que 80% das empresas em que os desenvolvedores trabalham não utilizam o *blockchain* em suas aplicações, sendo que 12% utilizam a tecnologia para usos não envolvendo moedas e 7,4% utilizam o blockchain em aplicações que envolvem moedas. Através das análises desses dados, é possível observar que o uso do *blockchain* está sendo utilizado em setores diferentes do de sua origem (STACKOVERFLOW, 2019).

Um dos usos para o *blockchain* que vem ganhando cada vez mais espaço é sua utilização em sistemas de eleições. Por ser um sistema descentralizado, auditável e imutável, o uso da tecnologia se mostra útil para tal aplicação. Especialmente em situações onde a forma que eleições ocorrem geram desconfiança quanto a segurança, contagem e imutabilidade dos votos (DUNIETZ, 2018).

Em relatório apresentado em 2013 por participantes da 2ª Edição dos testes Públicos de Segurança do Sistema Eletrônico de Votação organizados pelo Tribunal Superior Eleitoral (TSE), o sistema brasileiro de votação demonstrou falhas que permitem a violação do sigilo do voto, impossibilita ou torna inviável uma auditoria dos votos computados e dificultam a verificação da integridade dos resultados (ARANHA et al., 2013).

Junto com estes resultados, pesquisas indicam que a uma grande parcela da população brasileira desconfia ou desconhece como funciona o sistema brasileiro de votação. Junto com essas informações, acusações de fraude nos resultados de eleições por candidatos à presidência fragilizam ainda mais o confiança no atual sistema de eleições brasileiro (CRUZ; RIBEIRO, 2018).

Apesar do uso das urnas eletrônicas estar ligado a eleições nacionais, estaduais e municipais, a resolução TSE nº 22.685/07 permite com que o uso da urna para eleições comunitárias se dê para "entidades públicas organizadas, instituições de ensino e, excepcionalmente, a critério do TRE-PE, outras entidades". Tendo esta resolução como base, as desconfianças geradas pelo uso do sistema de eleição atual também se estendem a diversos outros setores e entidades da sociedade brasileira.

Este artigo tem como principal finalidade avaliar a viabilidade do uso da tecnologia *blockchain* em eleições, expondo os pontos positivos e negativos de seu uso para este fim e as ferramentas que podem ser utilizadas para desenvolver tal sistema.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 REDES P2P

Independentemente da versão, algoritmo ou arquitetura, sistemas *blockchain* dependem de uma rede de computadores P2P (*peer-to-peer*). Segundo Kurose e Ross (2014) uma rede P2P possuem dependência mínima de se manter ligados servidores com infraestrutura. Os hospedeiros de uma rede P2P são denominados pares. Neste tipo de rede, os pares se comunicam entre si. Os pares não são de propriedade de um provedor, mas sim de dispositivos controlados por usuários.

Diferente de programas baseados em uma estrutura cliente-servidor, onde a aplicação que roda no lado do servidor realiza a maior parte do processamento enquanto o lado cliente é focado em recolher e apresentar os dados, os participantes de uma rede P2P devem rodar programas semelhantes, pois eles cumprem a função de servidor para outros nós da rede além de cumprirem a mesma função do cliente.

Uma prática muito comum é a mescla entre arquiteturas cliente-servidor e P2P. Nesta arquitetura hibrida, frequentemente um servidor mantém um atualizada uma lista com os *IP's* dos usuários conectados à rede, porém mensagens continuam sendo trocadas pelos pares que compõem a rede P2P, sem passarem pelo servidor.

Um dos principais fatores que levam a escolha da arquitetura P2P é sua auto escalabilidade, onde cada par adicionado na rede aumenta sua capacidade de atender os serviços do sistema. O custo de redes *peer-to-peer* também tendem a ser menores devido ao fato não haver a necessidade de uma grande infraestrutura de servidores e largura de banda.

Apesar das vantagens, *softwares* desenvolvidos utilizando comunicação em redes P2P necessitam levar em consideração algumas desvantagens destas redes: segurança, provedores de internet e incentivos para os usuários.

Conforme Barcellos e Gaspary (2006), o fato de aplicações P2P serem distribuídas e não haver um controle de quem se conecta ou desconecta da rede, não há possibilidade de garantir a intenção dos nós conectados à rede. De forma com que este tipo de aplicação possua grandes vulnerabilidades no quesito de segurança. O incentivo é outro fator determinante para o sucesso de uma aplicação P2P, tendo como principal função manter os usuários conectados para que a

aplicação funcione adequadamente (KUROSE; ROSS, 2014).

Como a maioria dos fornecedores de internet residencial oferecem soluções de banda assimétrica, a banda para *download* tende a ser maior que a de *upload*. Essa característica pode ser geradora de gargalos no sistemas, o que força com que o desenvolvimento de aplicações P2P sejam pensadas para funcionarem neste cenário desfavorável. Diferente da arquitetura cliente-servidor, aplicações P2P tendem a ter quantidade de *download* e *upload* semelhantes (XIE et al., 2008).

A comunicação entre computadores em uma rede segue alguns protocolos específicos. Para o desenvolvimento deste trabalho, o principal meio de comunicação na rede se dá através do gRPC. O gRPC é um framework RPC (*Remote Procedure Calls*) inicialmente desenvolvido pela Google com base no HTTP2. *Frameworks* RPC são utilizados para realizar chamadas de procedimentos remotos. O uso do gRPC é recomendado em cenários que demandem alta escalabilidade, baixa latência e sistemas distribuídos (GOOGLE, 2009).

Como é possível visualizar na Figura 2, a chamada de procedimentos utilizando-se do RPC é semelhante à chamada de procedimentos locais. No processo RPC, uma *thread* controla dois processos: *caller's process* (processo de chamada) e *server's process* (processo de servidor). O processo de chamada inicialmente envia uma chamada com os parâmetros do procedimento para o processo servidor e aguarda uma mensagem de retorno para extrair o resultado. Enquanto aguarda uma resposta do servidor, o processo está sendo executado no *caller* é interrompido, retornando apenas quando há uma resposta (MICROSYSTEMS, 2009).

2.2 BLOCKCHAIN

O *blockchain*, também conhecido como cadeia de confiança, surgiu junto com o *Bitcoin* em 2008. Utilizando o princípio de sistemas distribuídos, a arquitetura criada por Nakamoto implementa uma versão eletrônica do livro razão, também denominado *ledger*, de forma pública e descentralizada (NAKAMOTO, 2008).

Os primeiros usos do *blockchain* se dá através do surgimento de cripto-moedas como o *Bitcoin*, servindo apenas como base de dados e mantendo um índice global de transações. Os registros gerados pelas transferências do *blockchain* ficam armazenadas em um livro razão virtual. Através do *blockchain*, foi possível resolver problemas como o dos gastos duplos, sem haver necessidade de autoridades confiáveis ou um servidor central. Com o passar do tempo, percebeu-se que a cadeia de confiança possuía uma vasta aplicação como registro distribuído e evoluiu ganhando novas funcionalidades.

A segunda versão do *blockchain* surge através do *Ethereum*, uma plataforma de dados descentralizado. O projeto *Ethereum* trouxe ao *blockchain* a possibilidade de executar *smart-contracts* e transformar o *blockchain* em uma plataforma global para transferência de diversos ativos, possibilitando o desenvolvimento de diversas aplicações, desde redes sociais a aplicações financeiras (ETHEREUM, 2019).

Redes *blockchain* podem ser agrupadas em duas categorias principais: (A) *blockchain*

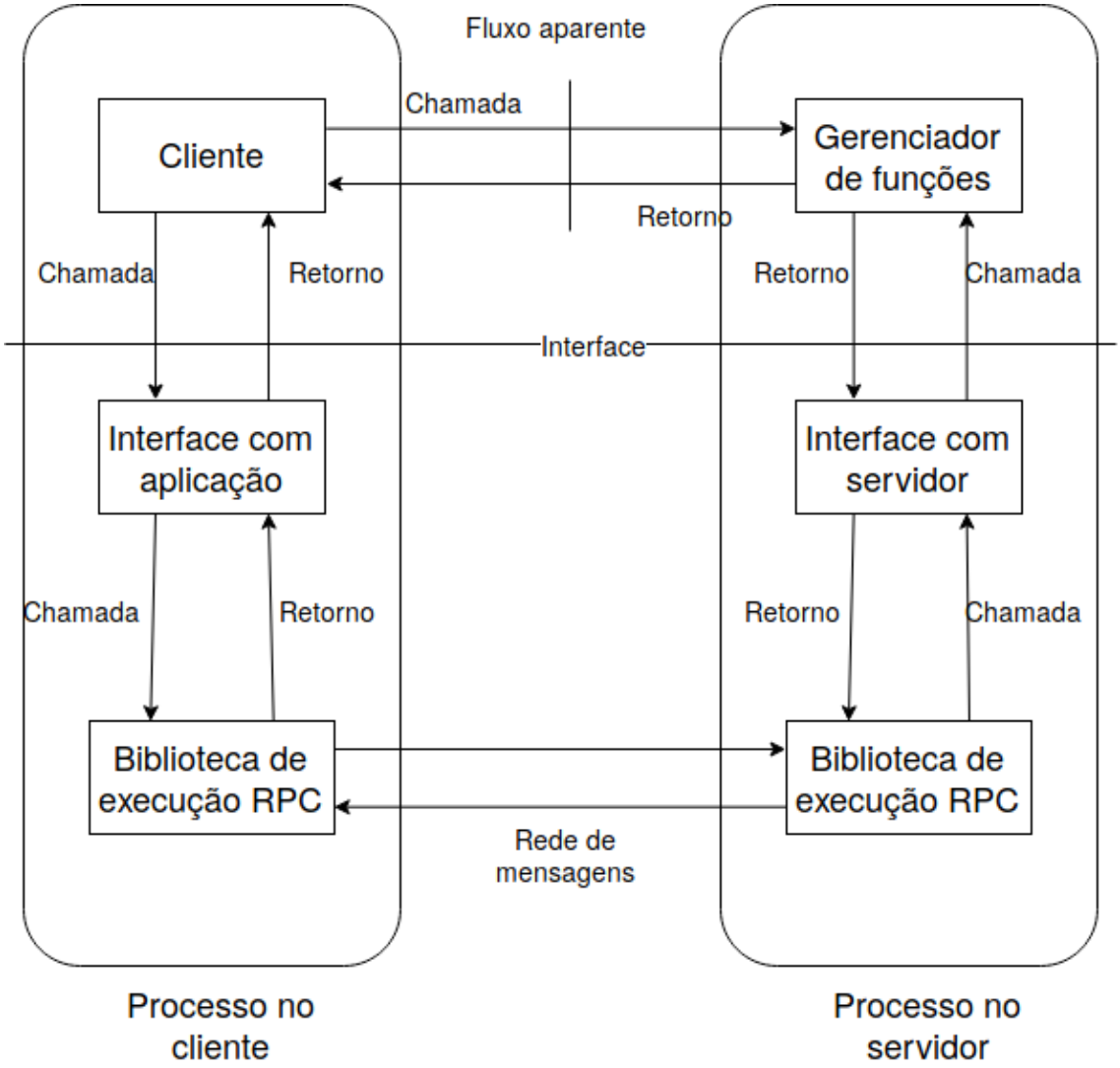

Figura 2 – Fluxo RPC. Fonte: IBM

pública e (B) *blockchain* privada. Tanto o *Bitcoin* quanto o *Ethereum* se utilizam do conceito de uma *blockchain* pública ou sem permissão. Neste tipo de cadeia, os nós que participam da rede P2P que mantém a *blockchain* não necessitam de uma identificação, permitindo que qualquer participante entre ou saia da rede em qualquer momento. Já nas *blockchains* privadas ou permicionadas, os participantes das redes são identificados, autenticados e autorizados. Em geral, os nós destas redes possuem papéis bem definidos, podendo se organizar em grupos. Nesta categoria, os nós que desejam ingressar ou sair da rede estão sujeitos a permissões e validações (GREVE et al., 2018).

Apesar da evolução e das diferentes categorias, o funcionamento base do *blockchain* continua o mesmo: para funcionar adequadamente, uma cadeia de confiança necessita de uma rede P2P. Quando gerada uma nova transação, a mesma será transmitida para todos os nós da rede, onde cada nó irá adicionar a nova transação em um bloco. Após lotar um bloco, os nós terão a responsabilidade de encontrar, através de um consenso, uma prova de trabalho para o mesmo e assim que a encontrarem deverão transmitir o bloco para toda a rede. O bloco será adicionado a cadeia apenas se todas as suas transações forem válidas. Caso ocorra de 2 nós fecharem um

bloco, apenas o bloco mais velho será validado como representado na Figura 3 (IBM, 2019).

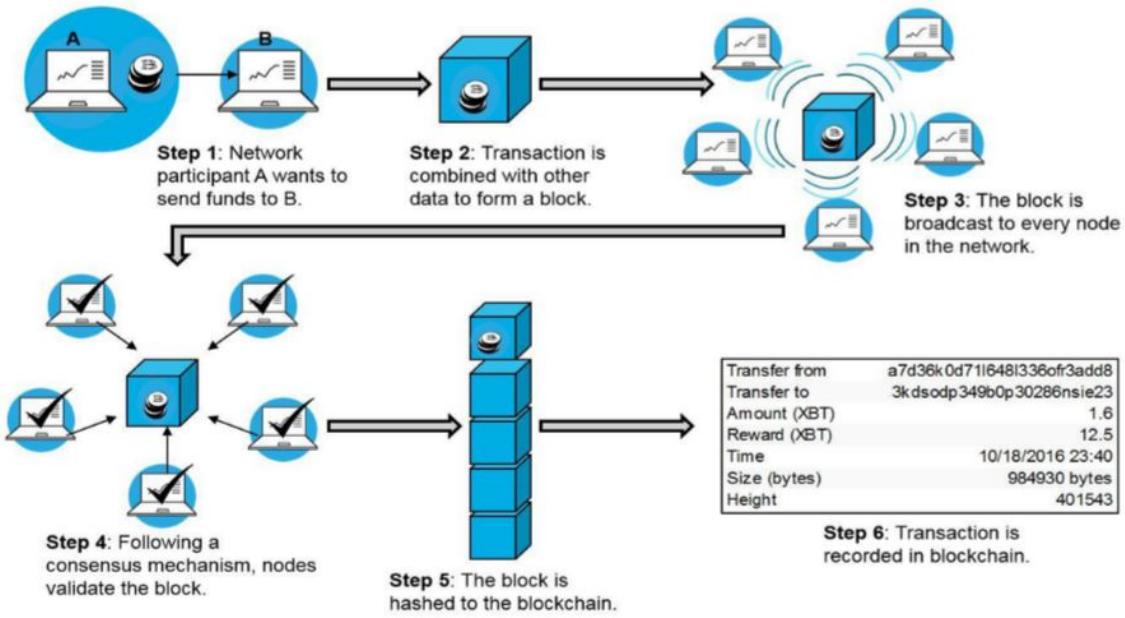

Figura 3 – Fluxograma de funcionamento blockchain. Fonte: Bloomberg New Energy Finance

O *blockchain* trouxe consigo diversas tendências e novidades. A ideia de *tokenização de ativos* foi a que possibilitou o uso das cadeias de confiança em diversos setores da economia. Um ativo é *"tokenizado"* através de um contrato digital ou *token*. Este contrato digital é de alguma forma ligado ao documento jurídico que representa o ativo real, com isso, um ativo pode ser representado digitalmente, tendo a mesma representação de um documento físico.

Atualmente, muitos ativos possuem uma forma de representação digital. Os ativos podem variar do tangível (imóveis e *hardware*) ao intangível (contratos e propriedade intelectual) (MARION, 1985). Ao serem representados através de *tokens*, os ativos podem ser adicionados em cadeias de confiança como o *blockchain*.

Com o surgimento do *Blockchain 2.0*, qualquer coisa pode ser considerada um ativo e ser transacionada na rede e registrada em um *ledger* (ETHEREUM, 2019). Seguindo este raciocínio, um voto também pode ser considerado um ativo para uma rede *blockchain*, permitindo o uso da tecnologia para aplicações de eleições.

Sempre que é necessário realizar a transação de um ativo, entra em ação o *smartcontract*. Os *smartcontracts* ou *chaincodes* são códigos, escritos em alguma linguagem de programação, a nível de aplicação. Estes códigos têm como principal função gravar e atualizar os registros no *ledger*.

Através dos *chaincodes*, é possível estabelecer regras e validar se as transações poderão ser realizadas ou não. Os *chaincodes* são executados automaticamente ao solicitar uma movimentação de ativos na rede e podem definir o estado do mesmo. Os *smartcontracts* iniciam com uma proposta de transação e finalizam com a gravação do registro chave-calor no *ledger*, que pode ser enviado aos nós da rede para aprovação da transferência através do consenso.

Os contratos inteligentes são importantes pois permitem que arquitetos e desenvolvedores de contratos inteligentes definam os principais processos de negócios e dados que são compartilhados entre as diferentes organizações que colaboram em uma rede *blockchain*.

Um dos mais importantes conceitos do *blockchain* é o consenso. O consenso é responsável por garantir a imutabilidade do ledger e coordenar as ações dos nós participantes da rede (GREVE, 2005).

Em uma rede *blockchain*, o algoritmo de consenso é um processo de tomada de decisão, com a finalidade de resolver o problema dos generais bizantinos, onde cada indivíduo constrói e apoia uma decisão que funcionará para todos integrantes da rede. Há diversos algoritmos de consenso existentes, cada um possui suas características, vantagens e desvantagem. A escolha dos algoritmos que garante um consenso entre os participantes da rede *blockchain* deve ser escolhido ou desenvolvido com cautela, pois o mesmo tem grande influência no desempenho da rede e a sua capacidade de realizar e aprovar transações com agilidade e segurança. Os algoritmos de maior presença nas redes são: *(1)Proof-of-work, (2)Proof-of-stake, (3)Proof of Elapsep Time* e algoritmos (4)BFT (*Byzantine fault tolerance*) e (5)CFT (*Crash fault tolerance*).

Algoritmos do tipo *proof* tem como princípio a solução de um problema computacional que demande um grande tempo poder de processamento ou tempo de espera para validar os blocos dessa forma, os custos envolvidos no processo afastam pares mal-intencionados. A desvantagem deste processo é o alto custo e necessidade de grande poder de processamento. Algoritmos do tipo BFT buscam formas alternativas de garantir a segurança da tomada de decisão. Geralmente, algoritmos alternativos tem um melhor desempenho, porém é necessário um maior controle dos pares que participam da mesma (ARAUJO, 2018).

A forma com que a imutabilidade é garantida independe do algoritmo utilizado e pode ser explicada pelo seguinte exemplo.

> "(···) supondo que em uma cadeia de blocos, estamos no número 940. De acordo com o funcionamento do blockchain, o bloco 940, após a sua validação e transação gerou um *hash* único, onde está copiado em todos os *peers* da rede. Esse *hash* será utilizado pela validação da transação do bloco 941 gerando um outro *hash*. Vamos assumir que essa rede é composta por 11 *peers* (P1 a P11). Supondo que um hacker tente modificar o bloco de número 900, conseguindo inicialmente infectar um peer (P4), ao modificar o bloco de número 900, toda a cadeia ficará diferente nesse *peer*, a partir do número 901 até o número 940 inclusive. No momento em que a cadeia de blocos tentar realizar a validação da transação do bloco 941, o algoritmo de consenso somente validará a transação se obtiver o consenso de 50% + 1 dos *peers* na rede. Se seis *peers* obtiverem o mesmo valor de consenso a transação é validada entre os peers que efetivaram o consenso e uma cópia do *ledger* é enviado para todos os onze *peers* da cadeia, sobrescrevendo o valor que cada um tinha anteriormente. Ou seja, o peer P4 não conseguirá efetivar a transação, portanto não poderá modificar um bloco que já foi transacionado e escrito no *ledger*. "(ARAUJO, 2018)

2.3 DOCKER E TESTES UNITÁRIOS

Diferente dos sistemas de virtualização tradicional, onde o sistema operacional é virtualizado de forma completa e isolada, o *Docker* virtualiza apenas recursos isolados, compartilhando os recursos do *kernel* do sistema operacional.

Como o *Docker* é baseado em contêineres, que não passam de um empacotamento de aplicações ou mesmo um ambiente inteiro, os sistemas empacotados se tornam portáveis para qualquer sistema que possua o *Docker* instalado. Além de portável, outra característica do *Docker* é a redução do tempo de *deploy* dos contêineres, pois não há a necessidade de ajustes de ambiente (BOETTIGER, 2015).

O fato dos serviços que são disponibilizados pelo *docker* executarem de forma desacoplada do sistema operacional, torna a ferramenta útil para o uso em testes. Como erros tendem a ocorrer com mais frequência em ambientes de teste, utilizar uma ferramenta que pode ser desconfigurada e substituída de forma rápida e automatizada é de grande utilidade, principalmente em testes de unidade.

Testes unitários tem como principal objetivo aferir o código em sua menor parte. Para linguagens orientadas a objetos, a menor parte de um código pode ser os métodos de uma classe. Os testes unitários são aplicados a partir da criação de classes de teste. Estas classes tem como objetivo executar métodos utilizados no código e verificar se seus retornos são os esperados. Como os testes são geralmente executados antes de se compilar o código, ao ser contado uma irregularidade, é possível cancelar a compilação do fonte. Através desta metodologia de teste, é possível testar os códigos escritos de forma automatizada, garantindo a consistência do código fonte (DELAMARO; JINO; MALDONADO, 2017).

Há diversas ferramentas para a criação de testes unitário. Para uso no projeto, foi utilizado a biblioteca *jUnit*, da família *Unit*. A *jUnit* está entre a mais utilizada para a construção de testes unitários em Java pelo fato de ser de fácil integração com diversos *softwares* de desenvolvimento e possuir código aberto com um grande número de contribuintes.

3 METODOLOGIA

Como forma de se alcançar os objetivos citados na introdução, a metodologia empregada neste artigo tem como base três etapas etapas principais: pesquisa, análise das ferramentas disponíveis e desenvolvimento de uma aplicação. Seguindo o fluxograma apresentado na Figura 4.

O início do desenvolvimento do artigo, leva em consideração uma pesquisa envolvendo o uso do *blockchain* em sistemas de eleições. Através da pesquisa, busca-se elencar casos de uso onde o *blockchain* já esteja sendo utilizado em aplicações de eleição, as vantagens e desvantagens que envolvem o uso da tecnologia bem como os resultados alcançados nestes casos. Junto a esta etapa, analisa-se o emprego do uso do *blockchain* nos casos estudados, com a finalidade se levantar dados para posterior desenvolvimento.

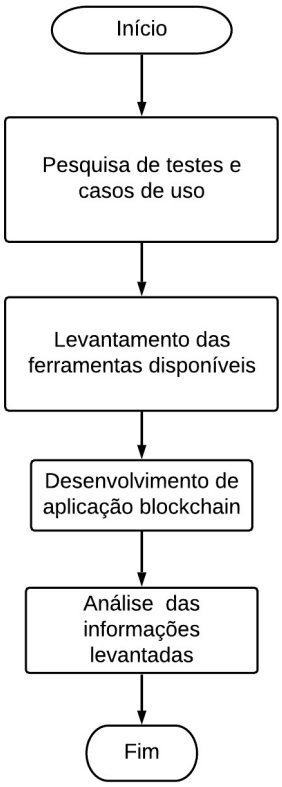

Figura 4 – Fase 3 da transação. Fonte: Hyperledger Fabric

Durante a etapa de análise das ferramentas disponíveis, é realizada uma comparação das opções mais conhecidas e usadas para o desenvolvimento de *softwares* baseados em *blockchain*. Nesta etapa ocorre a avaliação de quais os principais aspectos que devem ser levados em consideração quando tratamos de aplicações que envolvem blockchain. Com a conclusão dos estudos, é possível iniciar a terceira e última etapa proposta para atingir os objetivos.

A etapa de desenvolvimento busca como resultado uma aplicação simples, que explore os conceitos fundamentais de uma aplicação blockchain. Durante o desenvolvimento, busca-se utilizar as vantagens e, se possível, corrigir as desvantagens levantadas na primeira etapa utilizando-se das ferramentas analisadas na segunda etapa. Junto das etapas anteriores, é possível levantar os dados necessários para avaliar a viabilidade de se utilizar a tecnologia blockchain em eleições.

4 DESENVOLVIMENTO

4.1 TESTES JÁ REALIZADOS

Mesmo sendo uma tecnologia nova, o estudo e uso de *blockchain* já é realidade em algumas nações. No final de agosto de 2018, a cidade de Tsukuba, já conhecida por ser um centro cientifico no Japão, realizou testes do uso do blockchain em seu sistema de eleições, onde os eleitores podiam escolher qual iniciativa mereceria mais apoio financeiro pelo governo

(TSUKUBA..., 2018). Outro caso que ganhou repercussão aconteceu em Zug, na Suíça. A forma como o *blockchain* foi utilizado no pais europeu é muito semelhante ao uso no Japão. Os testes foram realizados com um pequeno número de eleitores que puderam escolher quais questões sociais deveriam ter uma maior representatividade naquela sociedade.

Em ambos os testes, o uso da tecnologia se mostrou bem aceito e promissor e os sistemas se mostraram semelhantes em alguns pontos. Os participantes dos testes tinham como pré-requisito possuir um documento de identificação, nos testes realizados em Tsukuba o *My Number Card* já em Zug os eleitores deveriam possuir uma certificação digital, também emitida por um órgão governamental. Outro ponto semelhante nos casos é o fato das votações serem realizadas para eleger questões sociais.

Apesar do sucesso do *blockchain* na Suíça e no Japão, a forma como a tecnologia foi aplicado em Serra Leoa e na Virgínia Ocidental deixaram a desejar. Em 7 de março de 2018, a empresa Suíça Agora alegou que a eleição presidencial em Serra Leoa foi a primeira a utilizar a tecnologia *blockchain* no processo eleitoral.

Após as votações terminarem, foi descoberto que na verdade, a empresa apenas realizou testes com a tecnologia. Após os escândalos, a empresa se retratou informando como os testes foram realizados, o que gerou discussões sobre quem deve ter a responsabilidade do desenvolvimento dos sistemas e como devem ser aplicados. Já em Virgínia, o uso do *blockchain* para votos se dá através de um aplicativo, utilizado por cidadãos e membros das forças armadas que estão no estrangeiro. No caso americano, a aplicação apresentou inconsistências e falhas no processo da autenticação.

Mesmo a maioria dos casos de uso terem sido desenvolvidos por empresas privadas, diversos relatórios e análises de especialistas foram geradas. Com estes documentos é possível avaliar os problemas que envolvem as aplicações e suas vantagens.

Dentre as vantagens encontradas vale destacar a facilidade de implementação, redução dos custos envolvido quando projetados a longo prazo e acompanhamento em tempo real dos resultados. Segundo o prefeito de Tsukuba... (2018), o novo formato de votação teve grande adesão por se tratar de uma tecnologia que garante mais transparecia e auditabilidade no processo de eleição.

Relatórios realizados pelo Instituto Brookings apontaram pontos que precisam de observação no aplicativo americano. Segundo Desouza e Somvanshi (2018), é fundamental que uma aplicação *blockchain* com foco em votações seja *Open-Source* desta forma é possível que se garanta a transparência e auditoria do sistema utilizado, bem como sua evolução.

Osgood (2016) afirma que a segurança de uma aplicação que utilize *blockchain* depende também da segurança da infraestrutura que mantêm a aplicação. Em sua pesquisa, o autor aponta que falhas de segurança em protocolos de transferência entre outras podem levar transações não autorizadas e em alguns casos a quebra do anonimato dos usuários.

Em todos os testes realizados houve relatos de dificuldades quanto ao uso da tecnologia. Nos testes realizados nos EUA, a forma como ocorria a autenticação no aplicativo se dava a partir de reconhecimento facial, o que gerou problemas pois as falhas na identificação abriam

brechas com falsos positivos e a eficiência do sistema dependia da qualidade dos *smartphones* dos usuários. Os testes realizados no Japão também apresentaram problemas com autenticação. No sistema oriental, os usuários utilizavam além do código fornecido pelo governo uma senha pessoal, fazendo com que houvessem diversos relatos de pessoas que esqueceram suas senhas e tiveram problemas para se identificar no sistema de votos.

4.2 FERRAMENTAS DISPONÍVEIS

Durante o desenvolvimento desta pesquisa, as principais plataformas utilizadas no desenvolvimento de aplicações *blockchain* são: *Ethereum* e *Hyperledger*. Ambas plataformas possuem características em seus *frameworks* distintas e que devem ser levadas em consideração para o desenvolvimento da aplicação.

A plataforma *Ethereum* foi lançada em 2015 oferecendo aos desenvolvedores um *blockchain* público facilmente programável. Com o *Ethereum* é possível desenvolver *smartcontracts* que executam múltiplas tarefas e rodam em uma rede com um protocolo generalizado. *Hyperledger*, por sua vez é um projeto colaborativo *Open Source* que implementa a tecnologia *blockchain* com base na implementação de diversos módulos. Diferente do *Ethereum*, onde há uma única *blockchain*, o *Hyperledger* permite desenvolver *blockchains* personalizadas e privadas que buscam atender diversas necessidades.

Como melhor explicado na seção 2.2, apenas pessoas autorizadas podem acessar a rede e a participação dos pares na aplicação não serem as mesmas, a confidencialidade de que envolve as transações na rede é superior a encontrada nas aplicações desenvolvidas com *Ethereum*. Em contrapartida, como todos os detalhes de um projeto são mantidos em domínio público, as aplicações que utilizam *Ethereum* tendem a ser mais transparentes.

Para fazer parte de uma aplicação que utilize *Hyperledger*, um par deve ter uma autorização previa, fornecida através de um servidor de certificados. Em redes *Hyperledger*, os nós participantes da rede não necessariamente apresentam a mesma função, diferente do que ocorre em redes *Ethereum*.

O desempenho e taxa de transferência nas duas plataformas também possuem diferenças. A principal causadora disto é o algoritmo de consenso presente nas plataformas. As principais diferenças entre as plataformas Ethereum e Hyperleder podem ser visualizadas na tabela 1.

Tabela 1 – Comparação Hyperledger X Ethereum

	Ethereum	**Hyperledger**
Tipo de Blockchain	Pública	Permissionada
Consenso	Prova de Trabalho	Plugável ou sem consenso
Linguagens Suportadas	Solidity	Java, JavaScript e GoLand
Token	Ether	Não tem, mas há possibilidade
Escalabilidade	Complexa	Simples
Confidencialidade	Criptografada	Transparente

4.3 HYPERLEDGER

Hyperledger é um projeto *open-source* colaborativo iniciado pela Linux Fundation em 2015 focado em desenvolver ferramentas para o uso do *blockchain* em diversos segmentos da indústria. O projeto conta com a participação de grandes empresas como Cisco, Airbus e IBM e já disponibilizou diversas ferramentas para o desenvolvimento e uso de *blockchains*. Entre elas está o *Hyperledger Fabric*.

O *Hyperledger Fabric* consiste em um *framework* modular e configurável e *open source* para desenvolvimento de *blockchains* permisionadas. Desenvolvido inicialmente para funcionar com a linguagem GO, o *framework* está em sua versão 1.4 e possui suporte às linguagens GO, JavaScript e Java.

Por ser um dos primeiros projetos do *Hyperledger*, o *framework* possui uma vasta documentação e uma comunidade ativa, que busca um constante aperfeiçoamento do *framework*. Baseado nisto, as informações presentes nesta seção são retiradas da documentação oficial do projeto. Segungo (IBM, 2019), as informações presentes no manual sempre estarão atualizadas e garantidas pelos colaboradores do projeto.

4.3.1 Pares e Chaincodes

Como apresentado na seção sobre *Blockchain* 2.2, os pares são os principais elementos da rede. Eles mantêm uma cópia do livro razão e do *smartcontract*. No *fabric*, os pares podem hospedar mais de um *chaincode* e mais de um *ledger*. Por definição, um par ou nó pode possuir um ou mais *ledgers* e nenhum *chaincode*, o contrário já não é possível, tendo em vista de o *chaincode* é a única parte da aplicação que possibilita a leitura e escrita no *ledger*. Lembrando que um *ledger* pode ser acessado por mais de um *chaincode*.

Uma aplicação que utilize o *Hyperledger Fabric* não precisa estar rodando em uma máquina nó, porém, sempre que a aplicação desejar fazer uma consulta o ou modificação no *ledger*, a mesma deve solicitar que um nó da rede a faça. Este processo ocorre em 3 etapas explicadas na seção Consenso e *Order Service* 4.3.2.

As redes *blockchain* do *fabric* são organizadas e mantidas por "organizações". Uma organização é formada a partir de uma série de pares, nós *Order* e um nó que representa a organização e fornece a identidade dos componentes que à pertencem. Uma melhor visualização da estrutura pode ser visualizada na Figura 5. A identificação dos nós dentro da rede se dá através de certificados fornecidos pelos nós autenticadores. Os pares dentro de uma organização só podem receber certificações emitidas pelo nodo certificador da própria organização, sendo que esta identificação segue regras preestabelecidas pelos desenvolvedores da rede.

4.3.2 Consenso e Order service

O algoritmo de consenso utilizado pelo *fabric* executa em nós denominados *Orders* ou nós de ordenação, durante uma transação, esses nós são considerados como centrais. Assim

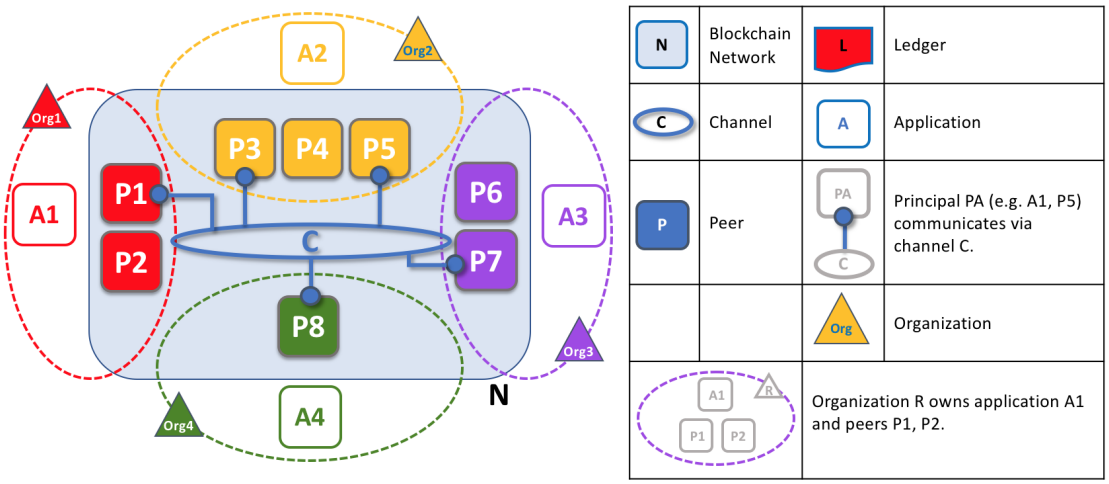

Figura 5 – Organização da rede. Fonte: Hyperledger Fabric

como os pares comuns, os nós de ordenação pertencem a uma entidade e também recebem uma identidade única de um *CA*. Os *Orders* são nós específicos e não realizam transações nem mantêm uma cópia do *ledger*, apenas executam os pedidos de transação e executam o algoritmo de consenso. A vantagem deste tipo de organização é que o consenso executado em uma transação pode ser modular, portando é possível utilizar o algoritmo de consenso que melhor se adapta a aplicação.

Os algoritmos que podem ser utilizados devem ser determinísticos e implementar ao menos um algoritmo CFT ou um algoritmo BFT, mais utilizado quando há a possibilidade de haver atores maliciosos na rede. Por padrão o *fabric* utiliza 2 implementações de algoritmos CFT, que alternam conforme a estrutura da rede.

Além do consenso, os *Orders* também têm a capacidade de criar canais, manter uma lista de quem pode solicitar a criação de novos canais e suas configurações. Um canal funciona como um túnel, onde apenas quem tem autorização pode enviar e visualizar dados que por ele trafegam, no *fabric*, cada canal possui um único *ledger*.

O processo de transação ocorre em trem fases. A primeira parte do processo é denominada proposta. Uma proposta consiste em um par comum solicitar que outros pares executem o *chaincode* e retornem uma resposta de execução. Neste momento, é possível verificar se algum par está retornando alguma mensagem inconsistente e descartar a transação. Como é possível observar na Figura 6, a aplicação *A1* envia uma proposta de transação *T1P* aos pares conectados ao canal *C*. Por sua vez, os pares executam o *chaincode* e enviam a resposta *T1R* novamente para a aplicação solicitante.

Durante a segunda fase, as respostas recebidas e a transação são enviadas ao nó *Order* representado por *O1*, que executa o algoritmo de consenso e organiza as transações dentro de blocos (B2), que serão enviados a todos os nós do canal e adicionados ao *blockchain*, conforme Figura 7. Caso um par esteja desconectado do canal no momento, o mesmo receberá uma versão atualizada do *ledger* ao se comunicar com algum nó da rede. O tamanho do bloco é definido

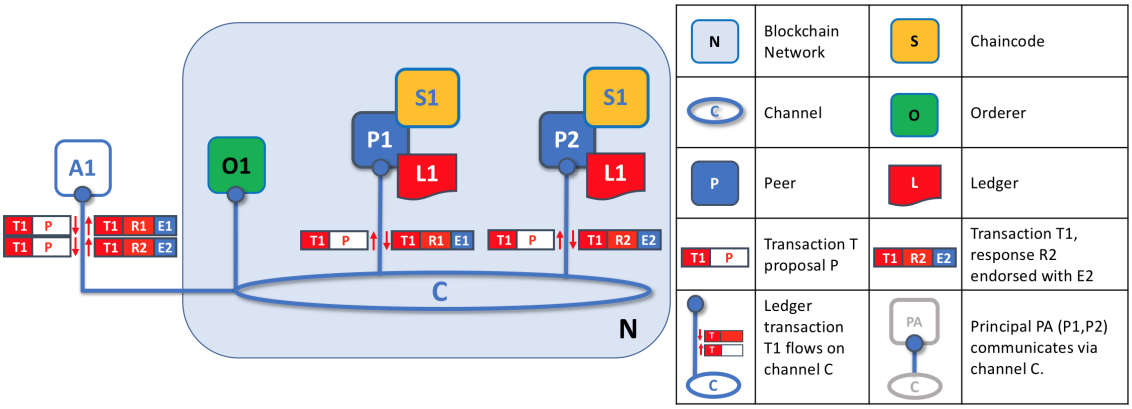

Figura 6 – Fase 1 da transação. Fonte: Hyperledger Fabric

com base de 2 parâmetros, a quantidade máxima de transações ou o tempo de vida do bloco. No *Hyperledger fabric*, os blocos gerados pelos *Orders* sempre são finais e imutáveis e como os nós *Orders* não mantêm uma versão do *ledger*, não interessa ao mesmo modificar o conteúdo das transações.

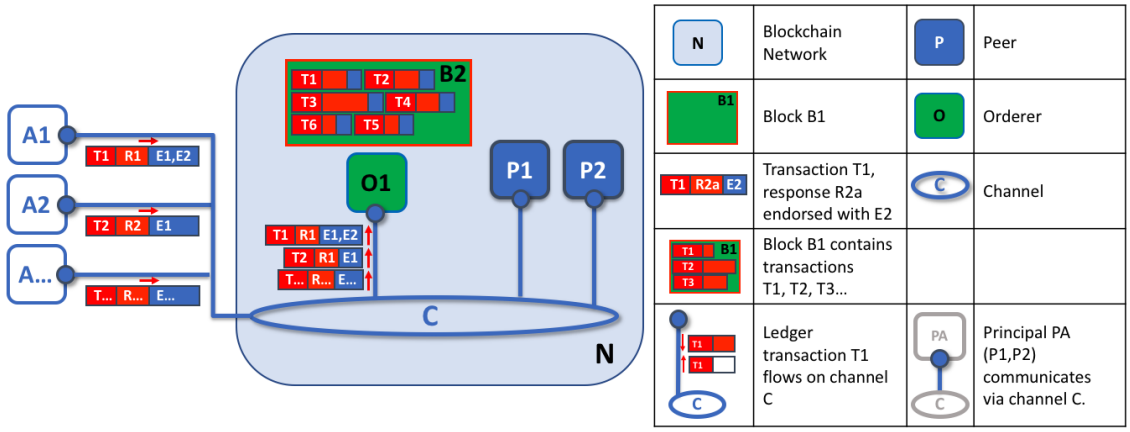

Figura 7 – Fase 2 da transação. Fonte: Hyperledger Fabric

A terceira e última fase do processo de transação, representada pela Figura 8, inicia com o *Order O* enviando os blocos gerados para os pares conectados no canal (P1 e P2). Por sua vez, os pares validam as transações dos blocos, adicionando os mesmos a cadeia de blocos mantida em seu livro razão (L1). Após realizarem este processo, os pares emitem uma resposta de conclusão, informando ao nó que solicitou a transação que seu *ledger* está atualizado.

4.4 VOTECHAIN

Sendo a última etapa do desenvolvimento da pesquisa, esta sessão demonstra os processos necessários para o desenvolvimento de uma aplicação de votos baseada em *blockchain*. Como o tempo, mão de obra e infraestrutura são escassos, o software possuirá apenas os processos fundamentais para o funcionamento do *blockchain*. Como forma de teste de funcionamento dos códigos desenvolvidos, será utilizado um *framework* para testes unitários.

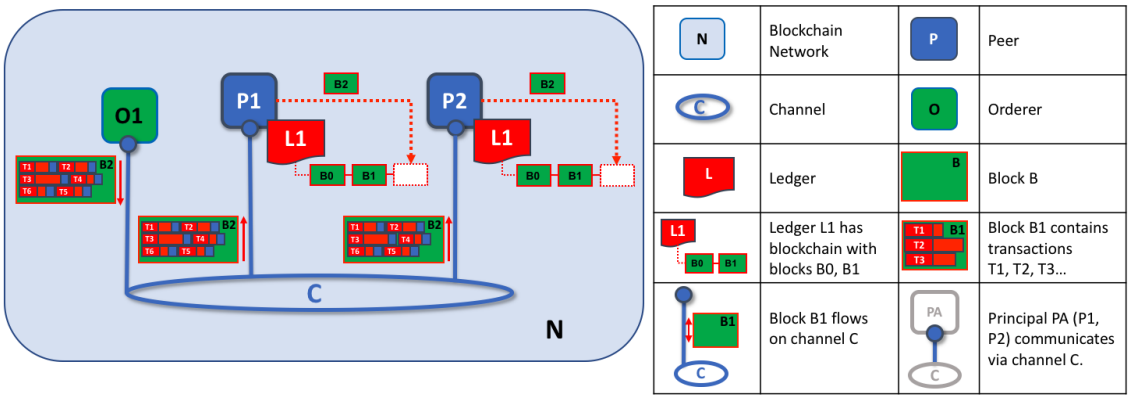

Figura 8 – Fase 3 da transação. Fonte: Hyperledger Fabric

O desenvolvimento do software a ser estudado se utilizará da linguagem Java em sua versão 8. Para auxiliar o controle das dependências, construção e documentação do projeto, foi escolhido a ferramenta Apache Maven. Tendo em vista a vasta documentação, compatibilidade com diversas tecnologias e a ampla gama de ferramentas para auxílio ao desenvolvimento do projeto, a IDE escolhida para o desenvolvimento do Votechain foi a Eclipse IDE Photon.

Tendo em vista o pouco tempo disponível para o desenvolvimento do sistema, a metodologia escolhida foi a *eXtreme Programming*, também conhecida como XP. O XP segue os preceitos estabelecidos no manifesto ágil, visando desenvolver um software funcional e de qualidade em um curto prazo, dando mais foco ao código e as pessoas do que na documentação.

Como os estudos apresentados na seção 4.1, é possível definir um escopo de desenvolvimento. O software deve ter a capacidade de inserir, consultar e adicionar votos a uma "Opção de voto". Desta forma, será possível avaliar como se dá o processo de desenvolvimento de uma aplicação baseada em *blockchain*. Conforme levantamento das ferramentas disponíveis 4.2 para desenvolvimento da aplicação baseada em *blockchain*, será utilizado o *framework Hyperledger Fabric*.

4.4.1 Ambiente de Testes

O desenvolvimento do ambiente de testes foi desenvolvido com imagens pré-configuradas para Docker v18.09, *framework* jUnit para realizar testes unitários automatizados que virtualizavam a rede. O computador utilizado para desenvolvimento e realização de testes possui como sistema operacional Ubuntu 19 x64. As imagens utilizadas para o desenvolvimento do ambiente de testes foram: *hyperledger/fabric-ca*, *hyperledger/fabric-orderer* e *hyperledger/fabric-peer*, ambas estão disponíveis no Docker Hub do projeto Hyperledger

A imagem *hyperledger/fabric-ca* tem como principal função ser um fornecedor de certificados (CA) de autorização para que pares possam interagir com a rede da aplicação. A comunicação com dos pares com o servidor de CA se dá através de uma API REST, utilizando-se do protocolo HTTPS.

Responsável por virtualizar os pares que compõem a rede, a imagem *hyperledger/fabric-peer* mantém o ecossistema necessário para rodar os *chaincodes* desenvolvidos. Para facilitar os testes, os contêineres que rodam a imagem *fabric-peer* possuem um volume apontando para uma pasta contendo o *chaincode*, assim, sempre que atualizado o *chaincode*, é necessário modificar apenas um arquivo.

A contêiner *fabric-order* implementa um nó do tipo Order. Como a rede desenvolvida será utilizada apenas como ambiente de teste, não há necessidade de mais de um nó do tipo na rede. A comunicação entre os pares que compõem a rede o os nós Order se dá pelo uso do gRPC.

A rede utilizada para os testes foi virtualizada com a ferramenta Docker. A rede virtualiza um total de 7 nós que utilizam imagens mantidas pelo projeto *Hyperledger*. Entre os nós virtualizados, 4 representam os pares da rede, sedo que, destes 4, 2 representam uma organização *org1* e outros 2 a *org2*. Dois nós foram reservados para cumprirem a função de servidores de CA e um nó com a função de *order*. Na Figura 9, é possível visualizar de modo gráfico a estrutura de rede virtualizada.

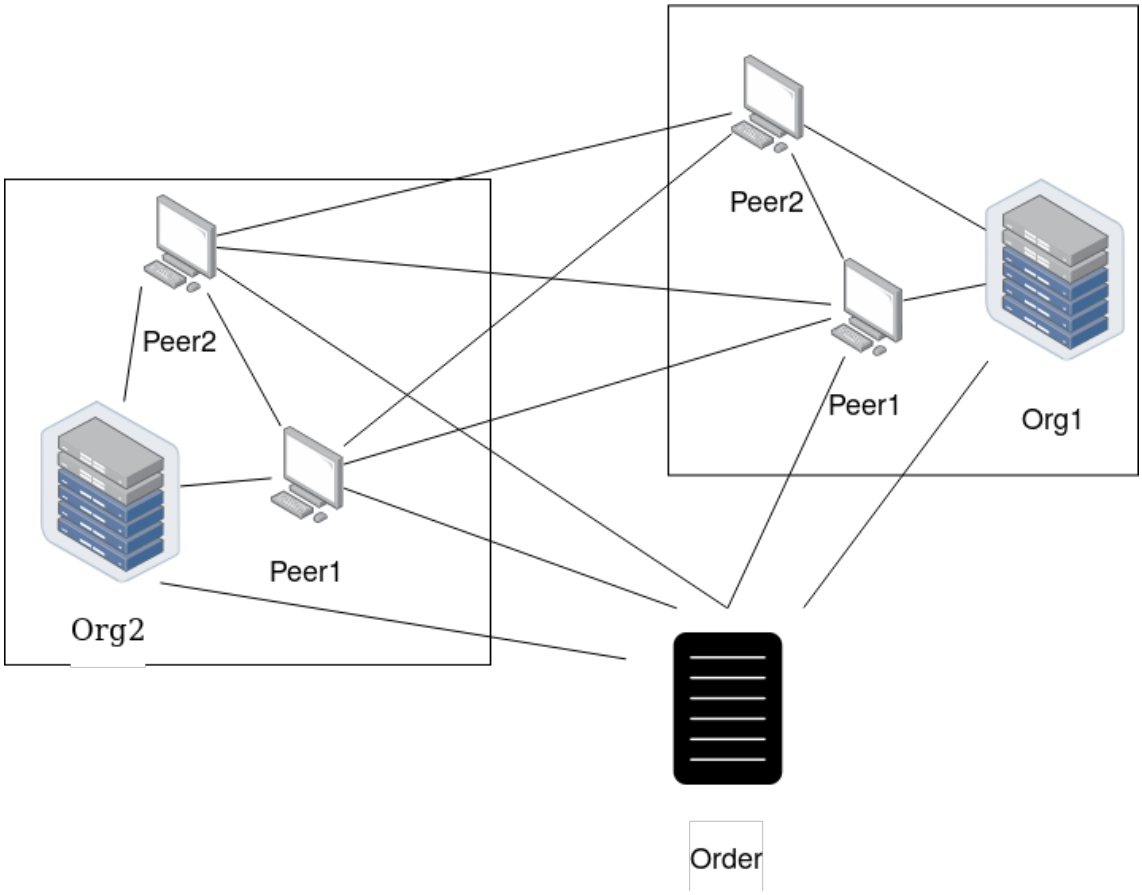

Figura 9 – Estrutura da rede Virtualizada. Fonte: O próprio autor, 2019

Para gerar os certificados utilizados nas entidades da rede, é utilizado a ferramenta *Cryptogen*. O *cryptogen* tem como função gerar certificados digitais e chaves de assinatura para os participantes de uma rede. Sua configuração e atuação se baseia em um arquivo que define a topologia da rede para gerar os certificados. Cada organização definida no arquivo consumido

pelo gerador recebe um CA mestre e único. Os pares conectados a esta organização recebem apenas os componentes vinculados a este certificado. Adotando esta estratégia, é possível definir que apenas pares previamente autorizados por uma organização autorizada possam participar de um canal na rede.

Para facilitar o gerenciamento da rede, foram desenvolvidos os *scripts inicia.sh* e *finaliza.sh*. O primeiro *script* é responsável por reiniciar e iniciar a rede, verificando a integridade dos arquivos utilizados pelo *Cryptogen*. O segundo *script*, tem como função desligar os contêineres dos pares e remover os dados gerados pelos testes, incluindo o *chaincode* instanciado nos nós pares.

4.4.2 Chaincode

Para o desenvolvimento do *chaincode* utilizado nos pares da rede, foi utilizado a linguagem Java 8, a biblioteca GSon para serializar e deserializar objetos e a API Shin. A API Shin fornece uma série de classes, *interfaces* e rotinas que facilitam o desenvolvimento de *chaincodes*. Com o auxílio da Shin, a programação do chaincode pode ser realizada utilizando 3 classes: Util, OpcaoDeVoto, VoteChaincode.

A classe OpcaoDeVoto tem como função representar uma opção que pode receber votos. Um objeto OpcaoDeVoto possui atributos para identificação, descrição e contagem de votos. Como as informações a serem gravadas ou recuperadas do *ledger* devem estar em formato jSon, os objetos modelos passam constantemente por processos de serialização e deserialização. A classe responsável por fazer estes procedimentos é a Util, a classe conta com métodos que utilizam as funções da biblioteca Gson para transformar os objetos OpcaoDeVoto em textos no formato jSon e vice-versa.

A classe Votechain implementa a *interface* ChaincodeBase e é responsável por manter o código do *chaincode*. Dentro do *chaincode*, existem 2 funções de implementação obrigatória: *init* e *invoke*. A função *init* é executada apenas uma vez ao iniciar o *chaincode*, nela é possível realizar operações para, por exemplo, verificar a integridade do nó e se todos os processos estão funcionando de forma correta, retornando erros para a aplicação que instanciou o *chaincode*.

O principal método do *chaincode* é o *invoke*. Este método é executado sempre que ocorre uma solicitação de leitura ou gravação no *ledger*. A função recebe como parâmetro um objeto contendo informações da requisição como método a ser chamado e os dados para realizar a ação, como observado na Figura 10.

Como é possível observar, cada método executa uma determinada função dentro do *ledger*. A definição de qual função será invocada e qual os dados para realizar a função são informados pela aplicação. O algoritmo demonstrado na Figura 11 mostra um exemplo de gravação e leitura do *ledger*. Pode-se observar que a gravação do *ledger* se dá através de uma String com o conteúdo, porém também há a opção de armazenar as informações em *bytes*.

Como levantado na sessão 4.3, tem-se a possibilidade de, se necessário, implementar mais de um *chaincode* para interagir com o mesmo *ledger*. Como exemplo, seria possível desenvolver

```
@Override
public Response invoke(ChaincodeStub stub) {
  try {
    List<String> param = stub.getParameters();
    String metodo = stub.getFunction();
    if (metodo.equals("adicionaVoto")) {
      return newSuccessResponse(adicionarVoto(stub, param));
    } else if(metodo.equals("criarOpcao")) {
      return newSuccessResponse(criarOpcao(stub, param));
    }else if (metodo.equals("buscaOpcao")) {
      return newSuccessResponse(buscarOpcao(stub, param));
    }else if(metodo.equals("buscarTodasOpcoes")) {
      return newSuccessResponse(criarOpcao(stub, param));
    }
    return newErrorResponse("Erro ao invocar a funcao. Expecting one of: [\"set\", \"get\"");
  } catch (Throwable e) {
    return newErrorResponse(e.getMessage());
  }
}
```

Figura 10 – Método Invoke. Fonte: O próprio autor, 2019

```
private String buscarOpcao(ChaincodeStub stub, List<String> param) {
  if (param.size() != ARGS_BUSCA_VOTO_SIZE) {
    throw new RuntimeException("Argumentos invalidos");
  }
  String value = stub.getStringState(param.get(0));
  if (value == null || value.isEmpty()) {
    throw new RuntimeException("Asset not found: " + param.get(0));
  }
  return value;
}

private String adicionarVoto(ChaincodeStub stub, List<String> param) {
  String value = buscarOpcao(stub, param);
  OpcaoDeVoto opcao = (OpcaoDeVoto) new Util().deserialize(value);
  opcao.addVoto();
  stub.putStringState(opcao.getId(), opcao.serialize());
  return "vote add";
}
```

Figura 11 – Métodos de escrita e leitura. Fonte: O próprio autor, 2019

um *chaincode* que permitisse leitura e escrita e outro apenas com a capacidade de ler os registros do livro razão. Com isso seria possível permitir que apenas urnas criassem registros e os demais pares apenas mantenham uma cópia do livro razão.

4.4.3 Testes

As rotinas para realizar os testes seguem a mesma sequência necessária para uso em uma aplicação normal, porém sem a necessidade de uma interface gráfica e interação humana com o código. Todos os processos geram logs de sistema e ao ocorrer erros o sistema é interrompido, indicando qual função obteve o resultado inesperado. Cada etapa só é executada se a etapa anterior for concluída com sucesso.

Para melhor controle, os testes foram divididos em 5 etapas: configuração da rede, instanciação do *chaincode*, cadastro do usuário, inserção e alteração de registro e busca de

registro. O *framework* utilizado para desenvolvimento dos testes foi o jUnit.

Na primeira etapa do processo, é necessário iniciar a rede subindo os contêineres docker, para isso, será utilizado o *script inicia.sh*. Após o processo de inicialização da rede começa a etapa de configuração da rede. Este processo consiste em criar instancias que representam as organizações e adicionar os certificados necessários, após, deve-se instanciar um novo par Order, setando as configurações que o mesmo necessita para o correto funcionamento do *blockchain*. Com o Order devidamente configurado, podemos criar o canal e adicionar os pares que o compõem. O código abaixo Figura 12 demonstra o processo de criação de um canal.

```
Usuario org2Admin = new Usuario();
File pkFolder2 = new File(Config.ORG2_USR_ADMIN_PK);
File[] pkFiles2 = pkFolder2.listFiles();
File certFolder2 = new File(Config.ORG2_USR_ADMIN_CERT);
File[] certFiles2 = certFolder2.listFiles();
Enrollment enrollOrg2Admin = Util.getEnrollment(Config.ORG2_USR_ADMIN_PK, pkFiles2[0].getName
    (),Config.ORG2_USR_ADMIN_CERT, certFiles2[0].getName());
org2Admin.setEnrollment(enrollOrg2Admin);
org2Admin.setMspId(Config.ORG2_MSP);
org2Admin.setName(Config.ADMIN);
FabricClient fabClient = new FabricClient(org1Admin);
Channel mychannel = fabClient.getInstance().newChannel(Config.CHANNEL_NAME);
Orderer orderer = fabClient.getInstance().newOrderer(Config.ORDERER_NAME, Config.ORDERER_URL);
Peer peer0_org1 = fabClient.getInstance().newPeer(Config.ORG1_PEER_0, Config.ORG1_PEER_0_URL);
Peer peer1_org1 = fabClient.getInstance().newPeer(Config.ORG1_PEER_1, Config.ORG1_PEER_1_URL);
Peer peer0_org2 = fabClient.getInstance().newPeer(Config.ORG2_PEER_0, Config.ORG2_PEER_0_URL);
Peer peer1_org2 = fabClient.getInstance().newPeer(Config.ORG2_PEER_1, Config.ORG2_PEER_1_URL);
mychannel.addOrderer(orderer);
mychannel.addPeer(peer0_org1);
mychannel.addPeer(peer1_org1);
mychannel.addPeer(peer0_org2);
mychannel.addPeer(peer1_org2);
mychannel.initialize();
```

Figura 12 – Criando canal. Fonte: O próprio autor, 2019

Tendo o canal e os pares devidamente configurados, é possível instalar o *chaincode* nos pares da rede que fazem parte de um determinado canal. Quando feito via software, é necessário que o programa do *chaincode* já esteja armazenado no computador par. As informações necessárias para se instalar o *chaincode* são: nome do *chaincode*, linguagem utilizada, versão, local do arquivo na máquina e em qual par da rede o *chaincode* deve ser instalado. Apenas pares pertencentes a uma organização certificada podem tem um *chaincode* instalado de forma remota. Uma proposta de instalação pode ser realizada conforme fonte presente na Figura 13.

O processo de cadastro de usuário é simples e consiste em solicitar ao nó representante da rede uma autorização. A partir deste momento, toda ação realizada pelo usuário ficara registrada como realizada por uma organização. Uma vez cadastrado, o usuário não poderá estar vinculado a outra organização. Com autorização concedida, o usuário tem permissão para iniciar uma proposta de transação em um canal ou solicitar uma informação do *ledger*.

Para solicitar uma proposta de transação os processos anteriores devem ter sido concluídos com sucesso. Uma proposta de transação consiste em solicitar a todos os pares do canal que executem determinado *chaincode* com determinados parâmetros. Se a resposta dos pares que

```
public Collection<ProposalResponse> deployChainCode(String nomeChaincode, String
    chaincodeLocal, String codepath, String linguagem, String vercao, Collection<Peer> peers){
  InstallProposalRequest request = instance.newInstallProposalRequest();
  ChaincodeID chaincodeID = ChaincodeID.newBuilder().setName(nomeChaincode).setVersion(vercao)
      .setPath(chaincodeLocal).build();
  Logger.getLogger(FabricClient.class.getName()).log(Level.INFO, "Instalando chaincode " +
      nomeChaincode + " using Fabric client " + instance.getUserContext().getMspId()+ " " " +
      instance.getUserContext().getName());
  request.setChaincodeID(chaincodeID);
  request.setUserContext(instance.getUserContext());
  request.setChaincodeSourceLocation(new File(codepath));
  request.setChaincodeVersion(vercao);
  Collection<ProposalResponse> responses = instance.sendInstallProposal(request, peers);
  return responses;
  }
```

Figura 13 – Instalando chaincode. Fonte: O próprio autor, 2019

participam da transação for positiva, é solicitado então que os mesmos gravem o registro no *ledger*, conforme Figura 14.

```
public Collection<ProposalResponse> sendTransactionProposal(TransactionProposalRequest request
    ){

  Collection<ProposalResponse> response = channel.sendTransactionProposal(request, channel.
      getPeers());
  for (ProposalResponse pres : response) {
    Logger.getLogger(ChannelClient.class.getName()).log(Level.INFO, "Proposta de transacao no
        canal" + channel.getName() + " " + pres.getMessage() + " "  + pres.getStatus() + " id
        da transacao:" + pres.getTransactionID());
    }

  CompletableFuture<TransactionEvent> cf = channel.sendTransaction(response);
  Logger.getLogger(ChannelClient.class.getName()).log(Level.INFO,cf.toString());
  return response;
}
```

Figura 14 – Envia proposta de transação. Fonte: O próprio autor, 2019

Como a comunicação entre os pares da rede acontece através do gRPC, é necessário especificar as plataformas que estão executando o *chaincode* em solicitado. As transações geradas no *fabric* possuem um identificador único, independente de terem sido concluídas ou não, o identificador é gravado no *ledger* e não é mais utilizado pelo sistema. O fonte presente na Figura 15 retrata uma das formas de realizar a configuração para a invocação de um *chaincode* escrito em Java.

O processo de busca de dados tende a ser menos complicado que o processo de gravação de registro. Para realizar uma consulta é necessário informar apenas qual *chaincode* deve ser utilizado para a consulta, a função e os paramentos necessários para a consulta. Como resultado, o processo gera uma serie de retornos com o conteúdo selecionado. Abaixo Figura 16 contendo exemplo de código responsável pela consulta de dados.

Para evitar problemas com quantidade de respostas, é possível selecionar quais pares da rede irão participar da consulta. O fato do *Hyperledger* organizar e distribuir os registros pelos nós Order garante que todos os nós comuns tenham o mesmo conteúdo em seu *ledger*, porém é

```
TransactionProposalRequest requisicao=fabClient.getInstance().newTransactionProposalRequest();

requisicao.setChaincodeID(ChaincodeID.newBuilder().setName(Config.CHAINCODE_1_NAME).build());
requisicao.setFcn("criarOpcao");
OpcaoDeVoto opv = new OpcaoDeVoto();
String[] opcoes = { opv.toString() };
requisicao.setArgs(opcoes);
requisicao.setProposalWaitTime(1000);

Map<String, byte[]> tm2 = new HashMap<String, byte[]>();
tm2.put("HyperLedgerFabric", "TransactionProposalRequest:JavaSDK".getBytes(UTF_8));
tm2.put("method", "TransactionProposalRequest".getBytes(UTF_8));
tm2.put("result", ":)".getBytes(UTF_8));
tm2.put(EXPECTED_EVENT_NAME, EXPECTED_EVENT_DATA);
requisicao.setTransientMap(tm2);
Collection<ProposalResponse> resposta = channelClient.sendTransactionProposal(requisicao);
for (ProposalResponse res : resposta) {
  Status status = res.getStatus();
  Logger.getLogger(Votechain.class.getName()).log(Level.INFO, "Chaincode invocado " + Config.
      CHAINCODE_1_NAME + ". Status da transacao - " + status);
}
```

Figura 15 – Configuração de proposta. Fonte: O próprio autor, 2019

```
private String buscarOpcao(ChaincodeStub stub, List<String> param) {
  if (param.size() != ARGS_BUSCA_VOTO_SIZE) {
    throw new RuntimeException("Argumentos invalidos");
  }
  String value = stub.getStringState(param.get(0));
  if (value == null || value.isEmpty()) {
    throw new RuntimeException("Asset not found: " + param.get(0));
  }
  return value;
}

private String adicionarVoto(ChaincodeStub stub, List<String> param) {
  String value = buscarOpcao(stub, param);
  OpcaoDeVoto opcao = (OpcaoDeVoto) new Util().deserialize(value);
  opcao.addVoto();
  stub.putStringState(opcao.getId(), opcao.serialize());
  return "vote add";
}
```

Figura 16 – Consulta nos pares. Fonte: O próprio autor, 2019

recomendável consultar mais de um para evitar erros de sincronização.

Para que possa ser melhor visualizado, o processo de voto pode ser visualizado na Figura 17. No diagrama presente na figura, é possível perceber que cada etapa é cumprida por um determinado elemento da rede, de forma que um participante do processo não tenha capacidade de interferir ou arbitrar sobre a tarefa do outro.

5 CONCLUSÕES

Analisando os casos de uso e testes feitos utilizando o *blockchain* como solução em sistemas de eleições e o funcionamento da arquitetura, é possível observar que a tecnologia oferece características favoráveis quando aplicado em sistemas de votos. Os testes realizados

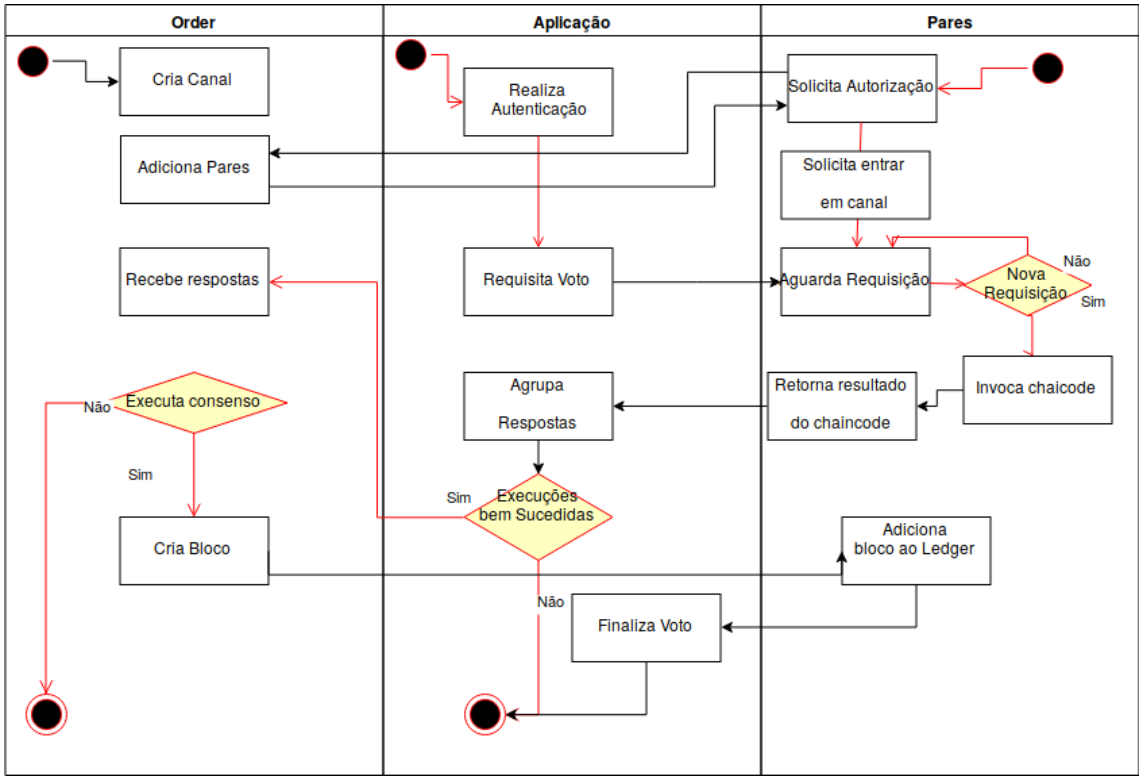

Figura 17 – Diagrama de atividade do processo de voto. Fonte: O próprio autor, 2019

em diversos locais e os relatórios gerados pelos mesmos mostram que é necessário ter cuidados quanto ao uso da ferramenta. Utilizar um sistema *blockchain* para eleições presidenciais ainda não é um cenário aconselhável, tendo em vista que as tecnologias ainda contêm possíveis falhas de segurança e performance que precisam evoluir, bem como o impacto que uma falha no sistema pode gerar ao pais.

Os processos que giram em torno de uma aplicação *blockchain* como infraestrutura e formas de garantir anonimato também devem ser levadas em consideração. Como é possível observar na sessão 4.1, todos os testes apresentaram problemas com a autenticação dos usuários e interação entre humano e aplicação.

Conforme os relatórios analisados, o uso de licenças *open source* são fundamentais para garantia a transparência e auditabilidade do sistema. Através do *open-source*, também é possível que a comunidade participe do processo de desenvolvimento do software, garantindo que testes, melhorias e correção de falhas sejam constantes. Outro fator benéfico que este tipo de software tem, é a confiança da população no sistema.

Os estudos das ferramentas disponíveis e desenvolvimento de uma aplicação mostram que há uma ampla documentação e comunidade engajada na melhoria das ferramentas *blockchain*. O desenvolvimento, apesar de complexo, já conta com ferramentas que auxiliam o trabalho e garantem que as principais características de um *blockchain* sejam atendidas.

Por ser uma tecnologia relativamente nova, estudos que acompanhem sua evolução e iniciativas que fomentem o desenvolvimento de melhorias devem ser realizados. Deixa-se também como sugestão para trabalhos futuros o desenvolvimento de uma aplicação mais robusta

com o *Hyperledger fabric* ou outras ferramentas que estiverem disponíveis e se enquadrem nos requisitos para desenvolvimento de uma boa aplicação.

REFERÊNCIAS BIBLIOGRÁFICAS

ARANHA, D. F. et al. Vulnerabilidades no software da urna eletrônica brasileira. **dos Testes Públicos de Segurança do Sistema Eletrônico de Votação do Tribunal Superior Eleitoral.**, 2013. Disponível em: <https://jornalggn.com.br/sites/default/files/documentos/relatorio-urna.pdf>.

ARAUJO, A. R. C. de. O consenso do blockchain. **TLC-BR**, v. 323, 2018.

BARCELLOS, A. M. P.; GASPARY, L. P. Segurança em redes p2p: Princípios, tecnologias e desafios. In: **Simposio Brasileiro de Redes de Computadores (24.: 2006 maio: Curitiba, PR). Anais dos minicursos. Curitiba:[sn], 2006.** [S.l.: s.n.], 2006.

BOETTIGER, C. An introduction to docker for reproducible research. **SIGOPS Oper. Syst. Rev.**, ACM, New York, NY, USA, v. 49, n. 1, p. 71–79, jan. 2015. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/2723872.2723882>.

CRUZ, B. S.; RIBEIRO, G. F. em 10 brasileiros acreditam que urna eletrônica pode ser violada. UOL São Paulo, 2018. Disponível em: <https://noticias.uol.com.br/tecnologia/noticias/redacao/2018/08/22/brasileiros-nao-confiam-na-urna-eletronica-e-acham-que-ela-pode-ser-violada.htm>. Acesso em: 2 mar. de 2019.

DELAMARO, M.; JINO, M.; MALDONADO, J. **Introdução ao teste de software**. [S.l.]: Elsevier Brasil, 2017.

DESOUZA, K.; SOMVANSHI, K. K. S. How blockchain could improve election transparency. 2018. Disponível em: <https://insights.stackoverflow.com/survey/2019>.

DUNIETZ, J. Are blockchains the answer for secure elections? probably not. **Scientific American**, 2018.

ETHEREUM. **Ethereum Documentation**. [S.l.], 2019. Disponível em: <https://www.ethereum.org/>. Acesso em: 13 jun. de 2019.

GOOGLE. **Documentation**. [S.l.], 2009. Disponível em: <https://tools.ietf.org/html/rfc5531>. Acesso em: 13 jun. de 2019.

GREVE, F. et al. Blockchain e a revolução do consenso sob demanda. **Livro de Minicursos do SBRC**, v. 1, p. 1–52, 2018.

GREVE, F. G. P. Protocolos fundamentais para o desenvolvimento de aplicaç oes robustas. In: **Minicursos SBRC 2005: Brazilian Symposium on Computer Networks**. [S.l.: s.n.], 2005. p. 330–398.

IBM. **A Blockchain Platform for the Enterprise**. [S.l.], 2019. Disponível em: <https://hyperledger-fabric.readthedocs.io/en/release-1.4/>. Acesso em: 13 jun. de 2019.

KUROSE, J.; ROSS, K. **Redes de computadores e a Internet: uma abordagem top-down**. ADDISON WESLEY BRA, 2014. ISBN 9788588639188. Disponível em: <https://books.google.com.br/books?id=i5WwAAAACAAJ>.

MARION, J. C. **Contabilidade básica**. [S.l.]: Saraiva Educação SA, 1985.

MICROSYSTEMS, S. **RPC: Remote Procedure Call Protocol Specification**. rfc5531. [S.l.], 2009. 62 p. Disponível em: <https://tools.ietf.org/html/rfc5531>. Acesso em: 13 jun. de 2019.

NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. Working Paper, 2008. Disponível em: <https://bitcoin.org/bitcoin.pdf>.

OSGOOD, R. The future of democracy: Blockchain voting. **COMP116: Information Security**, 2016.

STACKOVERFLOW. **Developer Survey Results**. 2019. Disponível em: <https://insights.stackoverflow.com/survey/2019>. Acesso em: 13 jun. de 2019.

TSUKUBA first in Japan to deploy online voting system. **The Japan Times**, 2018.

XIE, H. et al. P4p: Provider portal for applications. In: ACM. **ACM SIGCOMM Computer Communication Review**. [S.l.], 2008. v. 38, n. 4, p. 351–362.