UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA
PROGRAMA DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Gustavo Inácio Raimundo

DAGORA MARKET:
um *marketplace* descentralizado para *blockchain*

Florianópolis, Santa Catarina – Brasil
2022

Gustavo Inácio Raimundo

DAGORA MARKET:
um *marketplace* descentralizado para *blockchain*

Trabalho de Conclusão de Curso submetido ao Programa de Graduação em Ciências da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador(a): Ray Willy Neiheiser, MSc.
Coorientador(a): Luciana de Oliveira Rech, Dr.

Florianópolis, Santa Catarina – Brasil
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Inácio Raimundo, Gustavo
DAGORA MARKET : um marketplace descentralizado para blockchain / Gustavo Inácio Raimundo ; orientador, Ray Neiheiser, coorientadora, Luciana de Oliveira Rech, 2022.
87 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2022.

Inclui referências.

1. Ciências da Computação. 2. blockchain. 3. smart contract. 4. layer-2 solution. 5. marketplace. I. Neiheiser, Ray. II. de Oliveira Rech, Luciana. III. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. IV. Título.

Gustavo Inácio Raimundo

DAGORA MARKET: um *marketplace* descentralizado para *blockchain*

Este(a) Trabalho de Conclusão de Curso foi julgado adequado(a) para obtenção do Título de Bacharel em Ciências da Computação, na área de concentração de Sistemas Distribuídos, e foi aprovado em sua forma final pelo Programa de Graduação em Ciências da Computação do INE – Departamento de Informática e Estatística, CTC – Centro Tecnológico da Universidade Federal de Santa Catarina.

Florianópolis, Santa Catarina – Brasil, 16 de março de 2022.

Renato Cislagh, Dr.
Coordenador(a) do Programa de
Graduação em Ciências da Computação

Banca Examinadora:

Ray Willy Neiheiser, MSc.
Orientador(a)
Universidade Federal de Santa
Catarina – UFSC

Luciana de Oliveira Rech, Dr.
Coorientador(a)
Universidade Federal de Santa
Catarina – UFSC

Jean Everson Martina, Dr.
Universidade Federal de Santa
Catarina – UFSC

Ricardo Felipe Custódio, Dr.
Universidade Federal de Santa
Catarina – UFSC

Dedico este trabalho a meus pais, Valedir e Daniela, por todo apoio e carinho que me levaram até onde estou hoje.

AGRADECIMENTOS

Agradeço a todos que acreditaram em mim que esse dia se tornaria possível, em especial meus pais que me deram a base necessária para poder me desenvolver sem preocupações, meus avós e familiares que sempre me apoiaram e minha namorada que me apoiou nos melhores momentos e nos momentos mais difíceis. Agradeço também a quem não acreditou pois me deu forças para provar que estavam errados. Agradeço meu orientador e professora responsável pelo trabalho excepcional e pela atenção e carinho que tornaram esse trabalho possível. Por ultimo, agradeço a mim mesmo por não ter desistido e por ter escolhido se dedicar a essa área que pode abrir um caminho promissor pela frente.

“If you don’t believe it or don’t get it, I don’t have the time to try to convince you, sorry.”
Satoshi Nakamoto

RESUMO

Desde a invenção da internet, a negociação de mercadorias através de *marketplaces* vêm crescendo consideravelmente. Entretanto, estes servem como intermediadores e cobram uma taxa sobre o preço total do produto. O uso de *blockchains* em aplicações descentralizadas pode mudar isso. Através de *smart contracts*, é viável desenvolver um *marketplace* descentralizado sem intermediários. Na literatura, alguns trabalhos foram realizados a fim de exibir o que é necessário para um mercado descentralizado e alguns protótipos foram apresentados. Todavia, caso fossem implantados nos dias de hoje, os custos de executá-los seria proibitivo para adoção em massa. Fora da literatura, alguns projetos também foram concebidos, porém poucos permitem a comercialização de bens físicos, e estes que permitem possuem suas operações inviabilizadas devido ao custos da *blockchain* onde executam. Neste trabalho, é apresentado o *Dagora Market*, um mercado decentralizado de compra e venda de mercadorias construído a partir de contratos inteligentes. Seu objetivo principal é remover o intermediário de *marketplaces* centralizados, e assim permitir produtos a preços mais acessíveis para compradores e faixas de lucro maiores para vendedores. Para tal, contratos foram concebidos com compatibilidade com as mais diversas *blockchains* e *layer-2 solutions* existentes. São avaliados os custos de executá-los e é feita uma comparação entre as soluções de escalabilidade para atingir o objetivo.

Palavras-chaves: *blockchain*. *smart contract*. *layer-2 solution*. *marketplace*.

ABSTRACT

Since the invention of the internet, the negotiation of goods through marketplaces is growing considerably. However, these marketplaces serve as a middleman and charge fees depending on the total price of the product. The usage of blockchains in decentralized applications can change that. Through smart contracts, it is viable to develop a decentralized marketplace without middlemen. In the literature, some works had been presented in order to show what is necessary for a decentralized market and some prototypes have been presented. Nevertheless, in case they were deployed nowadays, the cost of running them would be prohibitive for mass adoption. Outside the literature, some projects have been designed as well, although few allow the comercialization of physical goods and, the ones that allow, have their operations inviabilized due to the blockchain costs where they run. In the present work, we present Dagora Market, a decentralized markplace for buying and selling goods built on smart contracts. It's main goal is to remove the middleman from markeplaces and therefore allow affordable product prices for buyers and greater profits for sellers. In order to do that, the contracts were built to maintain compatibility with various blockchains and existing layer-2 solutions. The costs of execution are evaluated and scalability solutions are compared to achieve the goal.

Keywords: blockchain. smart contract. layer-2 solution. marketplace.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

EVM	Ethereum Virtual Machine
IPFS	Interplanetary File System
ERC-20	Ethereum Request for Comments nº 20
EIP	Ethereum Improvement Proposal
ZK	Zero Knowledge
P2P	Peer-to-Peer
ICO	Initial Coin Offering
PoW	Proof of Stake
PoS	Proof of Stake
NFT	Non Fungible Token
DAO	Decentralized Autonomous Organization
API	Application Programming Interface

SUMÁRIO

1 INTRODUÇÃO

Desde os primórdios da internet comercial, onde cidadãos comuns começaram a usufruí-la, diversas empresas perceberam o potencial de utilizar essa nova tecnologia como um canal alternativo para adquirir novos clientes e anunciar seus produtos e serviços. Dessa forma, houve a capacidade de consumidores comprarem produtos pela internet, entretanto o pagamento através desta ainda não era possível. Inúmeros *websites* compreenderam a possibilidade de venda de produtos através de cartões de crédito porém diversos deles eram inseguros e a desconfiança reinava sobre os usuários.

Com o objetivo de inovar os meios de pagamento, foi então criado o *Paypal*, uma carteira virtual, segura e confiável (PAYPAL. . . , s.d.). Seu principal objetivo era servir como um intermediário, onde usuários não precisariam prover dados sigilosos de pagamento em todas as páginas da internet que gostariam de comprar bens, apenas no *Paypal*, e em caso de fraude, o mesmo serviria como intermediador de disputas e facilmente devolveria o dinheiro ao usuário sem a necessidade de contato com bancos. Gradativamente, usuários começaram a depositar confiança no serviço e com isso, *websites* começaram a aderir ao uso do *Paypal* como meio de pagamento.

Inúmeras empresas almejavam ter seu próprio *e-commerce* para negociar seus produtos, e isso resultou em uma infinidade de problemas, pois há custos envolvidos ao criar páginas da *web* como contratar desenvolvedores, pagar hospedagem, manter o serviço atualizado, etc. Desse modo, muitas páginas se tornaram desatualizadas, com falhas de segurança e, do ponto de vista do usuário, era muito difícil encontrar produtos dentre os diversos portais de compras. Assim, viu-se uma oportunidade de desenvolver mercados virtuais, lugares centrais onde compradores e vendedores pudessem negociar produtos variados, e assim foram surgindo ao redor do mundo *marketplaces* centralizados que solucionavam esses problemas como *Ebay*, *Mercado Livre*, e, hoje em dia, *Amazon*, *Magazine Luiza*, *Aliexpress* entre tantos outros.

Até então, todas as transações pela internet necessitavam de numerosos intermediadores como serviços de pagamento, operadores de cartão de crédito, bancos, governos, etc, tornando-se um gargalo, pois havia necessidade de confiabilidade em todas as entidades para o sistema operar. Então, em 2008 foi publicado o artigo *"Bitcoin: A Peer-to-Peer Electronic Cash System"* por Satoshi Nakamoto, na qual descrevia um sistema descentralizado *peer-to-peer* capaz de prover transações entre duas partes sem a necessidade de um intermediário. Esse sistema criou uma criptomoeda chamada *Bitcoin* (NAKAMOTO, 2008), que funciona com base na tecnologia de *blockchain* cuja principal finalidade é resolver o problema de gasto duplo, isto é, o problema de garantir que um bem digital é enviando somente para uma outra pessoa. Essa tecnologia consiste em uma cadeia de blocos ligados por criptografia na qual cada bloco

possui um ponteiro ao bloco anterior e ao adicionar novos blocos, os anteriores se tornam imutáveis, tornando a quebra dessa cadeia extremamente difícil. Isso forma um livro-caixa (*ledger*) inquebrável cujas transações anotadas não podem ser apagadas ou modificadas.

A *blockchain* permitiu um horizonte de oportunidades, dentre elas, a possibilidade de desenvolver programas simples que podem ser persistidos e executados na *ledger*. Em vista disso, foram projetadas novos *blockchains*, como o *Ethereum* (BUTERIN, 2013), capazes de executar e gravar esses códigos, os chamados contratos inteligentes (*smart contracts*) (SZABO, 1997). Inúmeras aplicações descentralizadas foram criadas utilizando *smart contracts*, desde *tokens* (ERC..., 2020) e *exchanges* (ADAMS *et al.*, 2021) até sistemas de justiça (LESAEGE; GEORGE; AST, 2020; DECENTRALIZED..., s.d.) capazes de resolver disputas de forma descentralizadas.

Com a ajuda dessa tecnologia, diversos sistemas e serviços que antes dependiam de um intermediário têm potencial de se tornarem descentralizados e serem executados de forma autônoma. Dentre eles, o *e-commerce* torna-se possível sem a necessidade de serviços de pagamentos ou bancos, assim como *marketplaces* seguros e confiáveis. Anunciar produtos, intermediar disputas e transacionar dinheiro dispensam a necessidade de um intermediário, através da *blockchain* e de *smart contracts*.

1.1 MOTIVAÇÃO

Mercados centralizados são hoje a principal forma de negociar mercadorias. Geralmente, são uma das primeiras opções de busca por parte dos compradores e com isso, garantem maior visibilidade a produtos de vendedores da plataforma. Para manter suas operações, esses *marketplaces* cobram taxas que variam de 5% a 15% (UNDERSTANDING..., s.d.; QUANTO..., s.d.; CONDITIONS..., s.d.; CATEGORY..., s.d.) do valor total do produto. Como resultado, consumidores têm que pagar um preço maior, e vendedores recebem uma faixa de lucro menor. Todos esses custos poderiam ser reduzidos ao custo de processamento de pagamentos, através de uma plataforma descentralizada, onde vendedores e compradores negociam sem a necessidade de uma empresa intermediária.

1.2 OBJETIVOS

O principal objetivo deste trabalho é propor e desenvolver um protótipo de um *marketplace* descentralizado que irá executar através de *smart contracts* sobre *blockchain*, sendo capaz de anunciar e moderar produtos, realizar ordens e resolver disputas. Além disso, os objetivos específicos são listados a seguir:

- Utilizar criptomoedas como pagamento de produtos;

- Promover resolução de disputas justa para todas as partes;
- Evitar venda de produtos considerados impróprios[1];
- Reduzir custos em comparação a *marketplaces* centralizados.

1.3 METODOLOGIA

1.3.1 Classificação de pesquisa

A pesquisa pode ser classificada da seguinte forma:

- Quanto à Natureza da Pesquisa: Aplicada, pois gera conhecimentos para aplicação prática que podem ser dirigidos à solução de problemas específicos.
- Quanto à Forma de Abordagem do Problema: Quantitativa. Será feita uma análise de custos dentro dos *smart contracts* a fim de reduzi-los em comparação a mercados centralizados.
- Quanto aos Objetivos da Pesquisa: Exploratória. Pois busca maneiras de desenvolver os *smart contracts* diferentes de abordagens comuns visando o menor custo.
- Quanto aos Procedimentos Técnicos: parte Bibliográfica e Documental porém principalmente Experimental. Em um primeiro momento, será feita uma revisão bibliográfica buscando soluções para esse problema na literatura e fora dela como projetos já existentes. Em um segundo momento, será feita uma busca por tecnologias de *blockchain* que permitem a execução de contratos a custos menores.

1.3.2 Roteiro de pesquisa

Para atingir os objetivos deste trabalho, é organizado o método de pesquisa nos seguintes passos:

1. Aplicar uma revisão sistemática, procurando por propostas utilizando *blockchain* de mercados descentralizados. Nessa etapa, é buscado por soluções propostas na literatura e projetos existentes.
2. Classificação das propostas, verificando quais objetivos foram alcançados por elas. Ao final, traçar um caminho de inovação e aprimoramento.
3. Desenvolver uma solução de mercado descentralizado utilizando contratos inteligentes.

[1] A ser definido pela DAO

4. Analisar custos de execução dos contratos.

5. Verificar e discutir se a solução e o custo atingem os objetivos iniciais.

1.4 CONTRIBUIÇÕES CIENTÍFICAS

No contexto desse TCC, foi elaborado um artigo científico publicado no Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (INÁCIO *et al.*, 2021). O seguinte artigo encontra-se integralmente no Apêndice A:

- INÁCIO, Gustavo *et al.* Dagora: Um Mercado Virtual Baseado em Side-Chains. *In:* SBC. ANAIS do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos. [*S.l.*: *s.n.*], 2021. P. 155–168

1.5 ORGANIZAÇÃO DO TEXTO

Neste capítulo são descritas as motivações para a realização do trabalho e seus objetivos. O restante do documento está organizado da seguinte maneira:

1. No capítulo 2, são apresentados os conceitos sobre criptografia, *blockchain*, *smart contracts* e *layer-2 solutions* com o objetivo de sustentar o trabalho proposto;

2. No capítulo 3, são listados os trabalhos correlatos ao tema de mercados descentralizados destacando a implementação e análise. É apresentada uma revisão da literatura e o estado-da-arte no qual a proposta está inserida;

3. No capítulo 4, é apresentada a proposta deste trabalho, um mercado virtual descentralizado, explicitando suas características funcionais;

4. No capítulo 5, é desenvolvido um protótipo e discorrido sobre detalhes de implementação;

5. No capítulo 6, é feita uma análise completa dos custos do protótipo em diferentes *blockchains* e soluções de escalabilidade;

6. Ao final, no capítulo 7, é apresentada a conclusão trazendo possibilidades de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, é apresentada a base teórica da proposta. Na seção 2.1, é introduzida a criptografia utilizada pelas tecnologias presentes neste trabalho. Na seção 2.2, é definido *blockchain* e seu funcionamento. Na seção 2.3, são apresentados conceitos diretamente ligados a rede do *Ethereum* como *EVM*, *ERC-20* e *Smart Contracts*. Finalmente, na seção 2.4, são descritas soluções para tratar a escalabilidade de um *blockchain*.

2.1 CRIPTOGRAFIA

2.1.1 Funções de Hash Criptográficas

Uma função *hash* é uma função matemática $H(M) = m$, onde M é uma mensagem original e m é a mensagem digerida, também chamado de *hash*. A função segue as seguintes três propriedades (NARAYANAN *et al.*, 2016):

- Sua entrada pode ser qualquer *string* de qualquer tamanho.
- Produz uma saída de tamanho fixo.
- É eficientemente computável. Isso significa que para uma dada entrada, é possível conseguir o retorno de uma saída da função *hash* em uma quantidade de tempo razoável.

Essas propriedades definem funções de *hash* em geral, que podem ser utilizadas em estruturas de dados como *hash tables*. Porém, para esse tipo de função ser considerada criptograficamente segura, são requeridas três propriedades adicionais (NARAYANAN *et al.*, 2016):

- Resistência à colisão.

  Uma função *hash* H é dita ser resistente a colisão se for inviável encontrar dois valores, x e y, sendo $x \neq y$, e $H(x) = H(y)$

- Ocultação

  Dada saída da função *hash* $y = H(x)$, não há uma maneira viável de descobrir qual a entrada x.

- *Puzzle Friendliness*

  Se alguém quiser que uma *função hash* tenha um determinado valor de saída y, e parte da entrada foi escolhida de uma maneira aleatória, então é muito difícil encontrar outro valor que acerte exatamente a saída já encontrada.

Apesar desta última propriedade não ser comumente utilizada em livros de criptografia, é uma propriedade importante para a aplicação em *blockchain*.

Um exemplo de função de *hash* criptográfica é o SHA-256 (NIST, 2002), na qual é amplamente utilizada na área de *blockchains* (WANG *et al.*, 2019).

2.1.2 Hash Pointer

Um *Hash Pointer* é um ponteiro para onde alguma informação está armazenada junto com um *hash* criptográfico do valor desta informação. Enquanto um ponteiro regular provê uma maneira de recuperar a informação, um ponteiro *hash* também permite verificar que a informação não foi alterada.

Ele é usado para fazer estruturas de dados mais complexas. Intuitivamente, pode-se utilizar estruturas de dados familiares que usam ponteiros, como uma lista ligada ou uma árvore de busca binária e implementá-las com ponteiros *hash* ao invés de ponteiros normais. Como exemplo, na Figura 1 é mostrada uma *linked-tree*, que também é uma representação de uma cadeia de blocos.

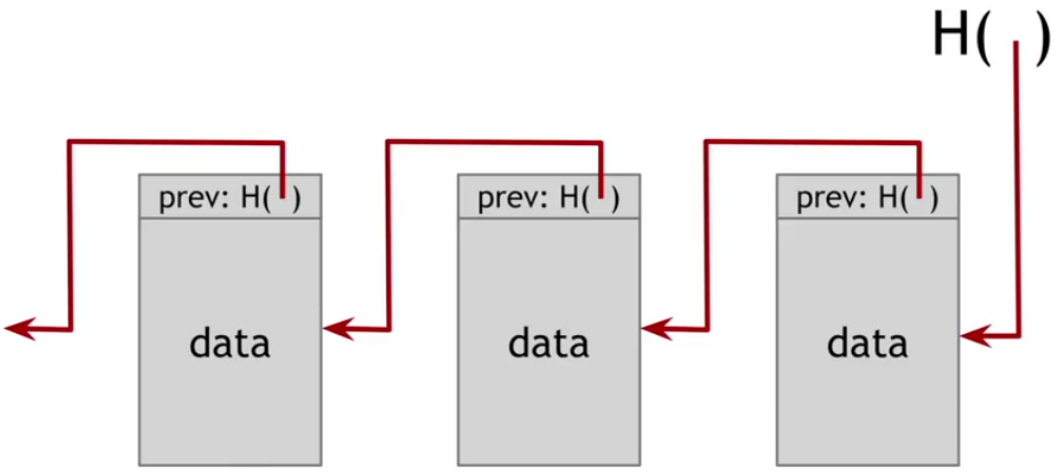


Figura 1 – Representação de cadeia de blocos (NARAYANAN *et al.*, 2016)

2.1.3 Merkle Tree

Merkle Tree, ou *Hash Tree*, é uma estrutura de dados na forma de árvore onde o valor de um nó folha é a função de *hash* de um determinado conteúdo, e os nós intermediários são constituídos pela função *hash* da concatenação de seus nós filhos. Essa estrutura de dados permite a validação segura e eficiente de grandes conjuntos de dados. Cabe ressaltar que além do conteúdo, a ordem das folhas é determinante dado que ao alterar suas posições, a função de *hash* retornará um valor totalmente diferente.

Geralmente, uma função de *hash* como o SHA-2 é utilizada. A escolha da função irá depender das necessidades da aplicação que está utilizando. Se for necessário

apenas proteger contra dano não intencional, outras funções de *hash* podem ser utilizadas como *checksums* inseguros.

No topo da árvore está o *Hash Root* ou *Merkle Root*, que é o *hash* principal da árvore. É ele quem garante que dados podem ser transmitidos através de uma fonte não confiável e assegurar que os mesmos estão corretos. O *Hash Root* é utilizado em *BitTorrent* para assegurar que as partes do arquivo recebido estão corretos, e também é utilizado em *blockchain* para representar as transações (Figura 2) em um determinado bloco.

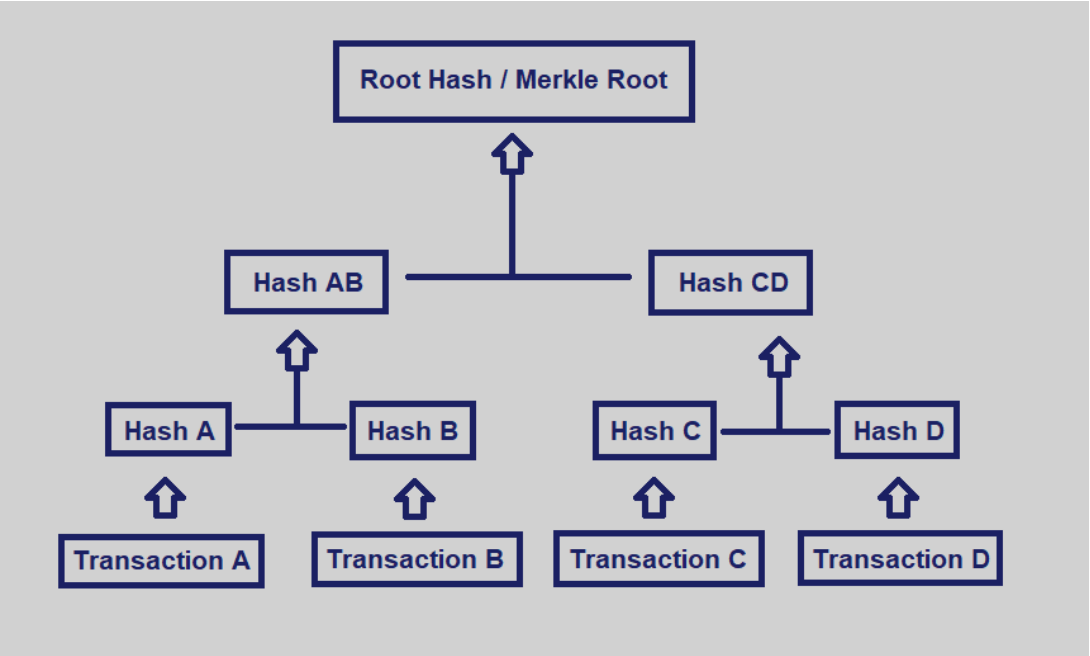


Figura 2 – Representação de *Merkle Tree*

A complexidade de se verificar que um determinado nó folha pertence a uma dada *hash tree* binária é de $O(log(n))$ (SZYDLO, 2004), proporcional ao logaritmo da quantidade de nós folhas, que contrasta com *hashlists* que é proporcional a quantidade de nós-folha ($O(n)$).

2.1.4 Assinaturas Digitais

Assinaturas Digitais são uma versão digital de assinaturas do mundo real. Apenas o detentor da assinatura pode utilizá-la, entretanto qualquer pessoa pode verificar que a assinatura é do mesmo. Além disso, uma assinatura que foi utilizada em um documento em particular não pode ser usada em outro documento.

Para isso, geralmente é utilizada criptografia assimétrica. Diferentemente de criptografia simétrica, que utiliza uma chave para criptografar e descriptografar uma mensagem, a criptografia assimétrica faz uso de um par de chaves: pública e privada. A chave pública é uma chave que todos os participantes têm conhecimento enquanto a chave privada é de conhecimento apenas de seu detentor. Na Figura 3, é mostrado um

exemplo utilizando duas entidades X e Y que serão chamadas de Alice e Bob. Alice pode utilizar sua chave privada para assinar uma mensagem. O resultado é então concatenado à mensagem original e enviado a Bob. Bob então pode utilizar a chave pública de Alice para verificar que foi realmente ela quem escreveu a mensagem. Apenas o detentor da chave privada pode assinar a mensagem, contudo todos podem verifica-lá com a chave pública. Note que, caso a mensagem seja alterada, haverá falha na verificação da assinatura, pois esta só é válida para a mensagem correspondente (RIVEST; SHAMIR; ADLEMAN, 1978a).

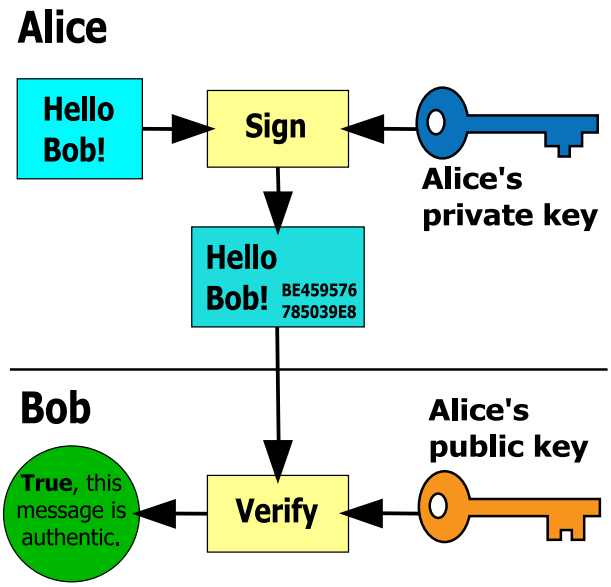


Figura 3 – Alice e Bob em troca de mensagens, de (COMMONS, 2020)

Uma assinatura digital garante integridade (verificação de que não foi alterada), autenticação (quem é o autor da mensagem) e não-repúdio (previne a negação do usuário sobre o envio da mensagem).

Alguns exemplos de algoritmos de certificados de digitais incluem RSA (RIVEST; SHAMIR; ADLEMAN, 1978b), que utiliza números primos para a criação de chaves públicas e privadas, e ECDSA (JOHNSON; MENEZES; VANSTONE, 2001), que utiliza curvas elípticas.

2.2 BLOCKCHAIN

Tecnicamente, uma *blockchain* nada mais é que uma *linked-list*, onde cada elemento na lista possui o *hash* do elemento anterior. Em *blockchain*, esses elementos são chamados de blocos. Com isso, foi proposto por Satoshi Nakamoto o *Bitcoin* (NAKAMOTO, 2008) com intenção de criar um dinheiro eletrônico *peer-to-peer* sem passar pela coordenação ou centralização de uma instituição financeira. Para atingir este objetivo toda a aplicação deve rodar distribuidamente.

Sendo um sistema distribuído, é necessário um protocolo de consenso que garanta a consistência e sincronização das réplicas (ou seja, réplicas com o mesmo estado).

No caso do *Bitcoin*, esta sincronização é feita pelo algoritmo *Proof of Work* (PoW). Para entender o funcionamento do algoritmo, é necessário explicar o conteúdo de um bloco no *bitcoin*. Além do *hash* anterior (requisito de *linked-lists*), todas as transações válidas são organizadas em uma *Merkle Tree*, e o *Root Hash* é anexado ao bloco. Ademais, existe um campo para *timestamp* e um para *nonce* como mostrado na Figura 4.

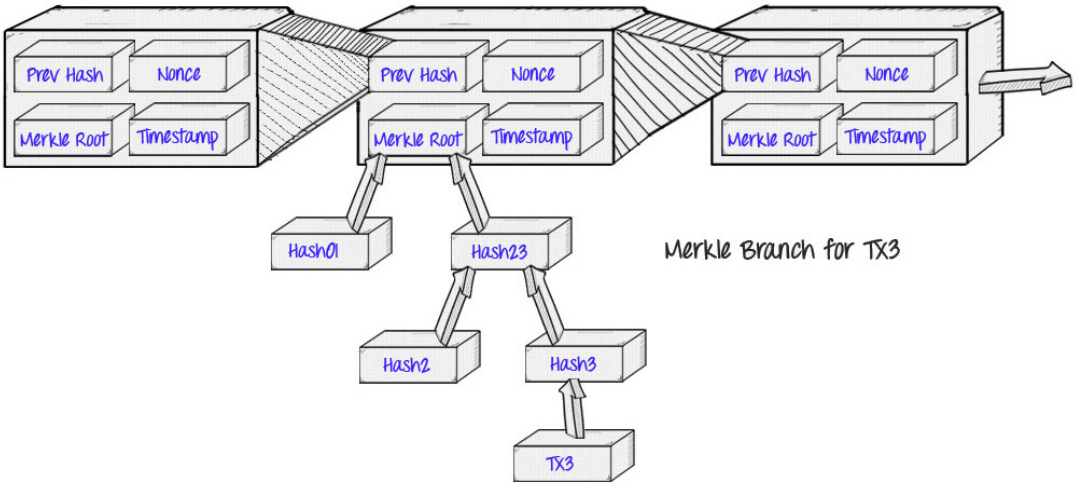


Figura 4 – Representação do *blockchain* Bitcoin (BUTERIN, 2015)

Após organizar as transações, *timestamp* e *hash* anterior dentro do bloco, o protocolo de PoW opera verificando se o *hash* SHA-256 do bloco possui uma certa quantidade de *bytes* com o valor zero[1]. Como essa função tem propriedades criptográficas (mostrado em 2.1.1), a única forma de descobrir um *nonce* que satisfaça essa condição é por força-bruta, incrementando toda vez que o número desejado de zeros não for encontrado e gastando energia no processo para assegurar o protocolo. O primeiro participante que conseguir acha-lo pode propor o seu bloco como o seguinte na *blockchain*. Esse processo é chamado de mineração (*mining*) e seus envolvidos são chamados de mineradores (*miners*).

Quando outros participantes recebem um bloco recentemente minerado, é verificado se o bloco minerado é válido, e ao se confirmar, adicionam em sua própria *ledger*, assim mantendo a *blockchain* replicada em todos os pares da rede (NAKAMOTO, 2008). Ao se criar e adicionar um novo bloco, torna-se persistente o bloco anterior, e quando mais blocos são persistidos, mais difícil se tornam ataques tentando manipular o conteúdo dos blocos mais internos da cadeia (NAKAMOTO, 2008). Para um novo bloco ser considerado válido por um nodo, ele deve ter o *hash* anterior igual ao *hash* do bloco mais recente, um *Merkle Root* de transações válidas e o *hash* do bloco deve ter a quantidade de zeros necessária pela dificuldade da rede. Uma transação para ser considerada válida deve possuir a assinatura digital do dono e o mesmo deve possuir

[1] A quantidade de zeros é definida pela dificuldade da rede, que é revisada a cada 2016 blocos, e se adapta buscando um tempo médio entre blocos de aproximadamente 10 minutos.

fundos suficientes. Tudo isso torna o *blockchain* um armazenamento imutável, já que ao alterar uma transação sequer, seu *hash* será alterado e, como consequência, o *Merkle Root*, o *hash* do bloco e dos blocos seguintes também seriam alterados.

A segurança do *Proof of Work* se dá pelo custo de ataque na rede. Para haver um ataque na *blockchain*, é necessário que um atacante tenha 51% do poder computacional investido na rede. Isso concede ao atacante poder suficiente para evitar que transações ganhem confirmações de blocos e até reverter transações que já ocorreram, praticamente controlando a rede. Porém o custo de realizar um ataque na rede é muito alto pois exige um alto custo computacional, e caso ocorra um ataque bem sucedido, a tendência é que os participantes realizem um *fork* (ignorem os blocos minerados pelo atacante criando duas redes) e deixem de dar valor aos *tokens* da rede antiga que já não são mais seguros.

Além do *Proof of Work*, o *Proof of Stake* é outro importante protocolo de consenso. Ao invés de fazer uso intensivo de energia para achar o *hash* via força bruta, a escolha de quem é o próximo a validar o bloco se dá pela quantidade de *tokens* em *stake*, ou seja, bloqueados como depósito de segurança. Assim caso haja uma proposta de bloco irregular, parte desse *stake* é perdido. A segurança desse protocolo se dá também pelo custo de ataque na rede. Para haver um ataque na *blockchain*, é necessário que um atacante tenha 51% dos *tokens* em *stake* assim o tornando o maior participante da rede. Caso isso ocorra, ele seria o maior prejudicado no caso de um ataque pois a perca de valor e de confiança na *blockchain* é uma consequência direta de uma rede insegura, tornando o ataque não recompensador.

A combinação do *blockchain* com o protocolo de consenso é chamado de *ledger* (ou *ledger* distribuído). Além do *Bitcoin*, foram criados centenas de outros *ledgers* com protocolos de consenso diversos e até com abordagens diferentes de teste de armazenamento persistente (BACH; MIHALJEVIC; ZAGAR, 2018).

2.3 ETHEREUM

Além do uso financeiro, foi proposto por *Vitalik Buterin* (BUTERIN, 2013) a utilização de *ledgers* distribuídos como plataforma para aplicações descentralizadas permitindo a execução de códigos, dando origem à consolidação dos chamados *smart contracts*. Seu *token* principal é chamado de *Ether* e sua unidade básica é chamada de wei[2]. Assim como o *Bitcoin*, o Ether é minerado pelo algoritmo de *PoW* e pode ser usado como moeda digital. Contudo, sua principal finalidade é ser utilizado para pagamento de taxas na execução de aplicativos descentralizados.

No ambiente do *Etheurem*, tanto contas de usuários quanto contratos possuem endereços. A representação de endereços é feita co-

[2] 1 wei equivale a 10^{-18} Ether, a menor unidade de divisão.

meçando por '0x' com mais 20 *bytes* em hexadecimal, por exemplo '0xab5801a7d398351b8be11c439e05c5b3259aec9b'. No caso de contas, o endereço se dá pelos 20 últimos *bytes* da função hash $keccak256()$ da chave pública. Já para contratos, o endereço é formado pelo mesma função só que do endereço de quem está implantando o contrato junto com o *nonce* de criação da conta. Isso torna os contratos um recipiente de *tokens* da mesma forma que contas de usuários, permitindo que funções dentro do contrato dispendam seus próprios fundos.

Toda transação na *Ethereum Virtual Machine* (EVM - subseção 2.3.2) utiliza *gas* ao ser executada, por exemplo, uma simples transferência de um recipiente a outro utiliza 21000 *gas*. Mineradores possuem uma quantidade limitada de *gas* disponível por bloco e assim escolhem quais transações serão incluídas com base na *priority fee* que é um valor pago aos mineradores para darem prioridade para suas transações. Além da *priority fee*, há também a *base fee* que é definida pela própria rede. Ao ser criado um bloco, se o total de *gas* dentro de um bloco for acima de 50% a *base fee* do próximo bloco aumenta, caso contrário diminui (BUTERIN *et al.*, 2019). O conjunto de *base fee* com *priority fee* é o que chamamos de *gas price*. Logo, esse parâmetro pode servir como uma métrica de carga da rede, tendo em vista que um *price* maior significa um maior custo para utilizar a rede.

O preço normalmente é medido em *Gwei* (*gigawei*), ou seja, para cada unidade de *gas*, $gasprice * 10^{-9}$ *ethers* (ou $gasprice * 10^{9}$ *weis*) são dispendidos como taxa.

2.3.1 Smart Contract

Um *smart contract* (SZABO, 1997) nada mais é do que a formação de contratos digitais firmados por partes que expressam suas manifestações de vontade de realizar uma determinada transação. Este contrato é similar a todos os outros do mundo real, com a diferença de ser digital e inteligente. Isto representa que o mesmo não pode sofrer perdas ou violações na sua forma.

Desde o *Bitcoin*, tipos primitivos de *smart contracts* já poderiam ser confeccionados através do *bitcoin script* (SCRIPT. . . , s.d.). Parecido com *Forth* (WHAT. . . , s.d.), *Script* é simples, baseado em pilhas e intencionalmente Turing-incompleto[3] (OKUPSKI, 2014). Em sua origem, é essencialmente uma lista de instruções que descreve como alguém pode receber acesso a uma quantidade de *bitcoins* bloqueados. Por exemplo, é possível definir a quantidade mínima de um conjunto de chaves privadas pré-definidas necessárias na assinatura de uma transação, ou então permitir que uma transação só possa ocorrer depois de uma data limite.

Por ser uma linguagem de pilha, ou seja baixo nível, foram criadas algumas linguagens de mais alto nível (*subset*) para facilitar o desenvolvimento e análise de

[3] O *Bitcoin script* é considerado Turing-incompleto devido a falta de laços de repetição em seus OP_-CODES.

código como *Miniscript* e *Minsc* (um aprimoramento do *Miniscript*). Todavia, sua Turing-incompletude impossibilitava a criação de aplicações mais complexas em cima da *blockchain*. Visando esse problema, a *Ethereum* (BUTERIN, 2013) foi criada, permitindo a execução de *bytecodes* Turing-completos. Junto a ela, foi concebido pela fundação *Ethereum*, o *Solidity*, uma linguagem de alto nível, que através de seu compilador *solc* poderia gerar *bytecode* compatível com a EVM (subseção 2.3.2).

Solidity é uma linguagem orientada a objetos usada para implementação de *smart contracts*. É uma linguagem que utiliza colchetes, inspirada em *C++*, *Javascript* e *Python*. É estaticamente tipada, suporta herança, bibliotecas e tipos complexos definidos pelo desenvolvedor. Além dela, pode-se destacar a linguagem *Vyper* que também é compatível com EVM. Esta é uma linguagem *pythonica* com forte tipagem, e possui deliberadamente menos funcionalidades que *Solidity*, já que visa a criação de contratos mais seguros e fáceis de auditar. Embora as duas linguagens sejam atualizadas frequentemente, *Solidity* tem mais ferramentas disponíveis e uma comunidade maior, sendo um dos principais motivos para diversos projetos utilizarem ela.

O ciclo de vida de contratos no *Ethereum* se dá da seguinte forma:

1. Um *smart contract* é criado ao fazer um transação de criação enviando o *bytecode* do contrato. O resultado da interação é o endereço do contrato como visto anteriormente.

2. Pode-se invocar funções públicas do contrato através de transações de interação. Nelas, o endereço do contrato é o destinatário e junto são enviadas informações adicionais como: qual função deve ser executada e quais os argumentos[4].

    - *Ether* pode ser enviado em interações caso a função seja do tipo *payable*;
    - Dentro de funções existe a possibilidade de emitir *events* que são mensagens indexadas na *Ethereum* e servem para acessar informações mais facilmente.

3. Outros *smart contracts*, ao serem executados, também podem invocar funções disponíveis nesse contrato.

4. Se programada no *bytecode*, pode-se executar uma função de destruição quando não houver mais a necessidade de interação com a aplicação.

Sendo assim, um contrato só é executado caso haja interações com ele. Ademais, todas as transações de criação, execução e destruição necessitam de *gas* e estão sujeitas à carga da rede (*gas price*).

[4] Normalmente, os argumentos são preenchidos automaticamente pelo *frontend* da aplicação descentralizada.

2.3.2 EVM

A execução de *smart contracts* em *blockchain* só é possível quando a mesma possui uma maneira de executar código determinístico, necessário para todos os participantes chegar ao mesmo resultado (consenso). Para tal, a rede do *Ethereum*, além de *ledger* distribuído, é considerado uma máquina de estados distribuído devido sua capacidade de executar esse tipo de código (ETHEREUM. . . , 2021). Para permitir a execução de código arbitrário, todos os nodos possuem a chamada *Ethereum Virtual Machine*, ou EVM. Isto é necessário, pois é ela quem limita quais operações podem ser realizadas e como isto é salvo no *ledger*.

Faz-se o uso de uma linguagem de programação (por exemplo, *Solidity*) que, ao ser compilada, retorna um conjunto de OPCODES compatíveis com a máquina de estados. Durante a execução, a EVM mantém tanto uma memória efêmera, que não persiste entre as transações, quanto um espaço de armazenamento persistente, que fica disponível em todos os nodos da rede após executado.

Um *smart contract* compilado, isto é, seu conjunto de OPCODES, pode realizar diversas operações padrão de pilhas como XOR, AND, ADD, SUB, etc. A EVM também implementa diversas operações de pilhas específicas de *blockchain*, como ADDRESS, BALANCE, KECCAK256, BLOCKHASH, etc. Esses OPCODES, além de serem executados, servem para estimar quanta computação foi realizada, e assim determinar quanto de taxa será exigida pela execução. Certas operações são mais custosas que outras como por exemplo operações de armazenamento, já que persistem informação em todos os nós da rede. Por outro lado, ao excluir informação do *ledger*, há uma restituição que serve como incentivo à limpeza.

O custo dos OPCODES em EVM é avaliado em *gas*. Cada uma das operações gasta uma quantidade de *gas* pré-definida e, ao executar o conjunto delas numa transação, tem-se o custo total da mesma, que é utilizada para pagar pelos recursos utilizados e avaliar sua eficiência.

Para evitar esgotamento de fundos em uma execução de contrato (como em *loops* infinidos), cada transação possui junto a ela um limite de *gas*. Isto também é importante para a própria EVM, pois podem existir *bugs* nesses códigos arbitrários e assim gastando recursos com uma transação que não retornará um resultado. Ao executar esses contratos faltosos, é consumido todo o *gas* disponível e a execução é finalizada com falha. Ao ocorrer erros, a EVM tem a capacidade de reverter toda a transação para o estado anterior, entretanto, a execução fica registrada no *blockchain* (BEREGSZASZI; MUSHEGIAN, 2017).

2.3.3 ERC-20

Com ajuda de contratos inteligentes foram criadas também moedas adicionais na rede *Ethereum*, dando início a muitos projetos, com muitas *startups* usando a rede

para obter financiamento e mesmo lançando suas próprias redes, participando então de ICOs (*Initial Coin Offerings*) (FENU *et al.*, 2018).

Uma funcionalidade, e também um problema, na criação de *smart contracts* é a possibilidade de escrever código da maneira que o desenvolvedor achar melhor, e nisso acaba-se gerando soluções diferentes para um mesmo problema. Dessa forma, viu-se a necessidade de se criar padrões para problemas gerais no *Ethereum*. Por exemplo: muitos projetos queriam ter seus próprios *tokens* fungíveis, da mesma forma que *Bitcoin* e *Ether*, sem a necessidade de criar uma própria *blockchain*. Assim foram criadas *Ethereum Improvement Proposals* (EIP) (BECZE; JAMESON, 2015), que dentre outras propostas, gerou padrões através de *Ethereum Request for Comments* (ERC).

Uma das propostas foi o *EIP-20* (VOGELSTELLER; BUTERIN, 2015) que gerou o padrão para *tokens* fungíveis, ou *ERC-20 tokens*. Esse padrão define um conjunto de funções e eventos necessários para os participantes do *Ethereum* identificar esses *tokens*. A geração de um padrão também permitiu uma maior composibilidade entre os contratos já que agora seria possível interagir com qualquer *token* da mesma forma.

O padrão *ERC-20* cria as seguintes funções:

- `name()`: Retorna o nome.
- `symbol()`: Retorna o símbolo.
- `decimals()`: Retorna quantas casas decimais o *token* possui.
- `totalSupply()`: Retorna a fornecimento total.
- `balanceOf(address _owner)`: Retorna o balanço de um endereço.
- `transfer(address _to, uint _value)`: Transfere seus fundos para outro endereço.
- `approve(address _spender, uint _value)`: Permite outro endereço gastar seus fundos.
- `allowance(address _owner, address _spender)`: Retorna a quantidade de fundos que um endereço pode gastar de um dono.
- `transferFrom(address _from, address _to, uint256 _value)`: Transfere uma quantidade de fundos de um endereço a outro.

Dentre as funções, `transfer()` e `approve()` são utilizadas por usuários para transferir e permitir o dispêndio de fundos. Já para *smart contracts*, o `transferFrom()` é mais utilizado pois são dependentes da interação entre os contratos. Sempre que um contrato for interagir com os fundos de um usuário, deve-se pedir a aprovação do usuário. Se este apenas enviasse fundos ao endereço do contrato (`transfer()`), o código não identificaria de quem recebeu pois não foi executado.

2.4 LAYER-2 SCALING

Executar aplicações no *Ethereum* tem vantagens como um considerável ecossistema de aplicações com segurança e um grande potencial de usuários. Mas, com o crescente uso do mesmo e devido aos limites de escalabilidade do *Ethereum*, o desenvolvimento e manutenção de muitas das possíveis aplicações se torna muito mais caro(ROZEN, 2021).

Soluções de *layer-2* foram propostas para contornar o problema destes custos de escalabilidade principalmente para aplicações com contratos inteligentes. Muitas destas soluções são baseadas em servidores externos que executam as transações de clientes e depois publicam os respectivos códigos e resultados destas transações na *blockchain*. Desta maneira, qualquer interessado pode verificar a validade dos resultados (LAYER. . . , 2021).

Um requisito para estas soluções é a existência de um depósito de segurança por parte dos operadores do servidor, de modo a ter incentivos corretos e evitar alguma ação errônea ou desonesta do serviço prestado. Esta abordagem é usada quando os agentes do serviço envolvem grandes empresas ou órgãos públicos (NEIHEISER *et al.*, 2020).

Entidades com recursos não tão significativos (pequenos empreendedores ou clientes comuns) normalmente fazem uso de plataformas mais distribuídas para execuções de nível 2. No entanto, isso ainda requer um certo nível de confiança na disponibilidade distribuída destes contratos inteligentes de nível 2.

Como isto, o uso de *side-chains* na sincronização em *main-chain* (*Ethereum*, por exemplo) vem se tornando uma alternativa muito consistente em relação a servidores centralizados. *Side-chains* começaram a ser usadas em 2015 (SINGH *et al.*, 2020) com a intenção de executar smart contracts em nível 2, deixando a *main-chain* para a persistências dos resultados. O motivo deste uso foi a forte aceitação de *blockchains* como o *Ethereum* e sua dificuldade em escabalidade diante do crescente uso. As *side-chains* podem apresentar seus próprios modelos de consenso e, portanto, serem independentes da *main-chain* (o *Ethereum*, por exemplo) (LAYER. . . , 2021). A *Polygon* (KANANI; NAILWAL; ARJUN, 2019) tem sido uma destas *side-chains*, usando a mesma maquina virtual como o *Ethereum* (EVM) e desta forma oferecendo suporte para contratos inteligentes. Ou seja, a priori, pode-se mover facilmente uma aplicação do *Ethereum* para a *side-chain*.

2.4.1 Rollups

Além deste aspecto da forma de disposição de soluções de *layer-2* (seja em servidores ou em outros *ledgers*), surgiram também algumas abordagens de como executar estas funções de camada 2. Muitas das execuções de transações na camada 2 adotam as abordagens identificadas como *rollups*. Neste caso, as execuções de

transações podem ser apresentadas em duas categorias: os *rollups* otimistas e os *zero knowledge (zkrollups)*.

Os *rollups* otimistas correspondem a serviços de segundo nível (*layer-2*) que possuem contratos inteligentes na *main-chain* (*blockchain* de nível 1) cujas funções são validar transações e servirem como ponte de criptomoedas e *tokens* (LAYER. . . , 2021). Estes serviços, possuem "agregadores" que são responsáveis pelo gerenciamento das transações enviadas pelos seus usuários e pela publicação de *batches* de transações em um *smart contract* na *layer-1* (publicação na *blockchain* em forma "agregada"). Esses *batches* são mais baratos de executar pois é feito uso do *"calldata"*, a parte dos parâmetros da função, que tem o custo muito menor do que armazenar dados dentro da *blockchain*.

São chamados de otimistas pois é feita a publicação de todas transações sem fazer a computação da validade dela nem armazenando informações de cada uma das transações. A verificação envolve a reexecução das transações do *batch*, respeitando suas ordens relativas no mesmo e comparando os resultados com os que persistiram na *layer-2*. Caso haja alguma inconsistência na informação, outros participantes podem enviar provas de fraude ao contrato na *main-chain* e assim aplicar uma penalidade ao agregador responsável revertendo ao estado anterior.

Como dependem de provas de fraude, existe um *trade-off* entre a latência de finalidade pelo custo de transações nos *optimistic rollups*. Para uma transação ser considerada final, há um período de normalmente 7 dias e por isso qualquer saque do *layer-2* não pode ser realizado antes desse tempo. São exemplos dessa abordagem os projetos *Arbitrum* (ARBITRUM. . . , s.d.) e o *Optimism* (OPTIMISM. . . , s.d.).

Os *rollups* identificados como *Zero knowledge (zkrollups)* são similares ao caso anterior, funcionando em *layer-2*. Os *rollups* ZK, agrupam ou acumulam transações executadas fora da *main-chain* e publicam na camada 1 (na *blockchain*) uma prova criptográfica de zero conhecimento, que podem ser *SNARKs*(BOWE; GABIZON; GREEN, 2018), *STARKs*(BEN-SASSON *et al.*, 2018) ou *SNORKs*(GABIZON; WILLIAMSON; CIOBOTARU, 2019), e funcionam na verificação da validade das execuções. Estas *ZK proofs* comparam o estado (*snapshot*) da *blockchain* antes das execuções das transações com o estado da *blockchain* após as execuções (ou seja, os valores antes e depois das execuções) e relata apenas as mudanças para a *main-chain* em um *hash* verificável. Os *zkrollups* cumprem a finalidade de rápido desempenho nas verificações, porém necessitam de poder de processamento maior. O *Loop-Ring* (LOOPRING. . . , s.d.) implementa em servidor centralizado a abordagem *rollups* ZK enquanto o *zkSync* utiliza uma *side-chain* para isso.

2.4.2 Plasma

A abordagem *Plasma* é também identificada nas execuções da camada 2, onde uma *side-chain*, além dos *checkpoints* regulares, também oferece suporte para provas de fraude como *rollups* para melhorar as garantias de segurança. A rede *Polygon* é um *side-chain* que implementa este tipo de modelo.

2.4.3 Sidechains

Cadeias laterais completamente independentes do *Ethereum* que são conectados com o *Ethereum* por *smart-contracts* são chamados *side-chains*. Estes *smart-contracts* determinam as trocas fáceis de ativos entre a *side-chain* e o *Ethereum* e vice-versa. Uma forma de oferecer isto, são *checkpoints* regulares do estado da *side-chain*, publicando regularmente o *hash* do *Merkel Tree Root* na *main-chain*. Essas cadeias laterais (*side-chains*) devem ter o seu próprio modelo de confiança, pois os fundos transferidos para a cadeia lateral usualmente não são protegidos pelo modelo de confiança do *Ethereum*. A plataforma *Polygon* (KANANI; NAILWAL; ARJUN, 2019) segue estes princípios.

Muitas propostas envolvendo a disposição de *side-chains* oferecem ainda um *mix* das diferentes abordagens citadas (uma mistura de *rollups*, *plasma* e *state channels*) nas execuções de camada 2. Estas proposições são conhecidas como *hybrid side-chains* (HYBRID. . . , s.d.). Estas combinações de abordagens procuram sempre alto desempenho, baixa latência, baixo custo em comparação com o uso só do *Ethereum*.

Em *main-chains* onde não existe contratos inteligentes, o processamento pode ser feito em um servidor externo, porém os resultados são publicados no *blockchain* com o código. Desta maneira, qualquer terceiro pode verificar se a execução é válida (NEIHEISER *et al.*, 2020). Porém, pela ausência de um depósito de segurança em contrato inteligente, é necessário estabelecer um modelo de confiança entre os recursos que executam a *layer-2* e a *main-chain*, de modo a se processar por malversação a entidade que mantém estes recursos de camada 2.

3 TRABALHOS RELACIONADOS

No contexto de mercados virtuais, existem diversos trabalhos focando a troca de bens (digitais). São exemplos de mercados de *Non Fungible Tokens* (NFT) (NON-FUNGIBLE. . . , s.d.) o *OpenSea* (OPENSEA. . . , s.d.), KnownOrigin (KNOWNORIGIN. . . , s.d.), Axie Marketplace (AXIE. . . , s.d.), entre outros.

Todos os citados fazem uso da descentralização, baseados em contratos inteligentes do *Ethereum*. Uma alternativa para produtos da vida real é o *OriginProtocol* (ORIGIN. . . , s.d.) que segue a mesma linha dos demais (usa smart contracts somente do *Ethereum*) e funciona como um *backend* descentralizado para a criação de *webshops* pessoais.

Em relação a trabalhos acadêmicos, recentemente surgiram diversos *marketplaces* fazendo uso de *blockchains*. O DESEMA (KLEMS *et al.*, 2017) é um destes trabalhos onde um protótipo de uma plataforma de mercado descentralizado é desenvolvido. Como um *marketplace*, o DESEMA apresenta: um "serviço de produtos" que fornece uma listagem de produtos; "registro de transações" indicando as transações entre cliente e vendedor; e "serviço de entrega" assinalando a entrega de produto. Todas esses serviços (ou conceitos) são baseados em *smart contracts*. Apesar de que algumas soluções de disputas terem sido propostas, não foram implementados. Portanto, o custo destes serviços em uma *blockchain* não têm como ser avaliado.

O trabalho em (RANGANTHAN *et al.*, 2018) propõe um aplicativo descentralizado para mercado virtual construído sobre a *blockchain Ethereum*. Este mercado é baseado somente em contratos inteligentes do *Ethereum* (ou seja, não usa a definição de *layer-2*). Este trabalho faz um estudo comparativo de suas proposições com mercados estabelecidos e centralizados em servidores que não usam *blockchains* e *smart contracts*. A base deste estudo são os custos da aplicação proposta na *blockchain* usada, porém a análise quantitativa apresentada é muito limitada. Primeiramente, foi feita uma análise das diferenças de executar em rede principal e em rede de teste do *Ethereum*. Porém, o custo de *gas* depende do código que está sendo executado e não da rede onde são executados os contratos. O custo de *gas* de redes de teste do *Ethereum* não podem ser levadas em consideração para análise de custo pois não condiz com a rede em produção. Além disto, este trabalho descreve um estudo mostrando as margens de lucro de um vendedor que tem seus produtos no *eBay* e o *Sotheby's*, comparando estas margens com as obtidas quando o mesmo vendedor apresenta seus produtos no mercado proposto sobre o *Ethereum*. O *eBay* e o *Sotheby's* são sites que não fazem uso de *blockchains* e cobram dos vendedores taxas sobre os produtos vendidos que se aproximam de 10% do valor obtido. O texto mostra que se usado mercado baseado no *Ethereum*, o vendedor teria mais receita em produtos mais caros e o custo para o comprador aumentaria em alguns centavos (devido a execução em *smart contracts*).

Com isto, os autores reiteram que por ter uma margem maior de lucro, produtos com valores altos poderiam ser vendidos mais baratos no *blockchain*. Entretanto, não é falado no texto que tudo isso depende do preço de *gas* e do valor do *token* (considerando o *Ethereum*). Comparado com a época em que o artigo foi escrito, o custo de execução hoje em dia seria 90 vezes maior (de US$3.73 dólares no artigo para US$336.79 nos dias de hoje) tornando a proposta, independente do produto, não competitiva.

Como apresentado, existem poucos trabalhos sobre mercados descentralizados baseados em *blockchains*. Dos citados, todos fazem uso do *Ethereum* e usam seu suporte para *smart contract* diretamente, sem o uso de *layer-2*. Nenhum destes trabalhos acima apresentam análises dos custos de uso da *blockchain Ethereum* que são reputadas como satisfatórias. Alguns trabalhos acadêmicos que foram citados apresentam algum tipo de avaliação, porém é importante ressaltar que foram implementados somente parcialmente e as medições apresentadas estão fundamentadas em avaliações já defasadas no tempo, considerando os custos atual do *Ethereum*. No próximo capítulo, é apresentada a proposta deste trabalho, um mercado virtual descentralizado, explicitando suas características funcionais e arquiteturais.

4 PROPOSTA

Neste capítulo, é proposto a aplicação de um *marketplace* descentralizado denominado Dagora Market, apresentando seus aspectos arquiteturais e funcionais. O Dagora conecta diretamente de forma descentralizada vendedores a compradores, utilizando criptomoedas como meio de pagamento e contratos inteligentes nestas conexões. Desta forma é possível anunciar produtos, assim como administrar anúncios existentes através de contratos inteligentes compatíveis com EVM. Informações sobre anúncios existentes e seus estados podem ser obtidos com *queries* a ferramentas de indexação da *blockchain*.

4.1 FERRAMENTAS UTILIZADAS

4.1.1 IPFS

O IPFS (BENET, 2014) ou *InterPlanetary File System* é um sistema de arquivos descentralizado. Ele funciona da seguinte maneira: cada arquivo é considerado único, ao invés de buscar o arquivo pelo nome dele, é utilizado o *Content ID* (CID) que nada mais é que o resultado de uma função *hash*. É possível enviar arquivos únicos ou até pastas.

Ao solicitar um arquivo para um nodo IPFS, é verificado no armazenamento próprio se existe o arquivo com CID correspondente. Caso não possua, é enviada uma mensagem para os nodos da rede e, caso algum nó responda, o arquivo é baixado para ser disponibilizado como resposta.

Diversos projetos em *blockchain* utilizam o IPFS como maneira de armazenar dados *off-chain* e assim otimizar os custos. Por exemplo, DAOs que possuem sistema de votação *on-chain* cuidam apenas da contagem de votos enquanto disponibilizam a proposta *off-chain* via IPFS.

4.1.2 The Graph

Indexar informação na *blockchain* é uma tarefa difícil, levaria horas para um cliente buscar todos os eventos que ocorreram e agregar esses valores a fim de representar informações úteis. Uma maneira de resolver isso é ter um servidor escutando todos os eventos e agregando esses valores para poder disponibilizar uma API para consultas. Essa abordagem, no entanto, adiciona um ponto único de falha e os clientes dependem que o servidor agregue de forma correta.

Para solucionar esse problema, surge *The Graph* (THEGRAPH. . . , s.d.) que é uma ferramenta descentralizada para indexar informações na *blockchain* de maneira eficiente. Ele funciona escutando eventos que ocorrem em um *smart contract* e mapeando para seu banco de dados. Esses dados ficam disponíveis através de APIs de

GraphQL, chamadas `subgraphs`, que permitem buscas muito mais complexas do que as disponíveis nativamente.

Uma API de *GraphQL*, ou *Graph Query Language*, é um servidor *backend* que possui linguagem própria para fazer consultas. Dentro do *subgraph* pode-se definir funções que serão executadas quando o nodo do *The Graph* detectar um evento. Essas funções permitem lógicas de programação e também permitem buscar dados JSON dentro do IPFS, assim, agregando também as informações dentro do IPFS e disponibilizando facilmente para a interface do usuário.

4.2 ARQUITETURA

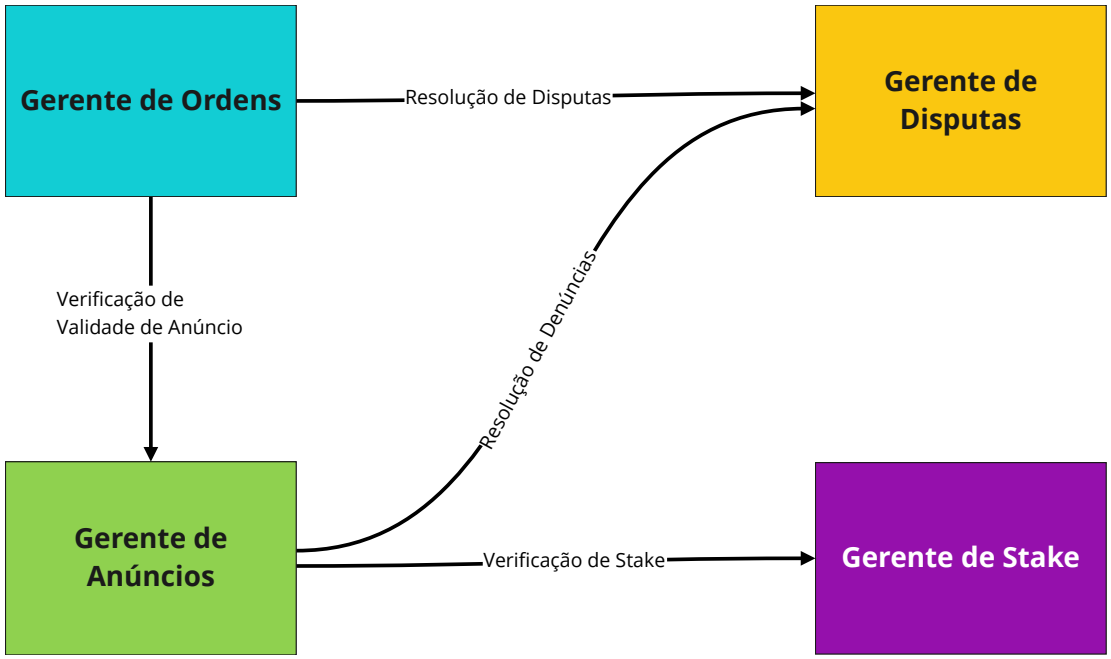


Figura 5 – Comunicação entre os contratos

O comportamento da proposta foi dividido em 4 contratos onde cada parte cuida de uma função do mercado e se comunicam conforme a Figura 5. São eles:

- Gerente de Stake
- Gerente de Anúncios
- Gerente de Ordens
- Gerente de Disputas

4.2.1 Gerente de Stake

O *marketplace* gira em torno de um *token* ERC-20 próprio denominado de DGR. Para manter a honestidade dos anunciantes, os mesmos devem realizar um depósito de segurança (*stake*) de uma quantia mínima de DGRs[1]. Essa quantidade deve ser significativa porém não proibitiva e exerce a função de evitar spam de anúncios falsos e anúncios que não seguem as diretrizes do mercado (por exemplo produtos ilegais).

A diferença principal dessa abordagem para *marketplaces* centralizados, é que o anunciante adquire *tokens* e de certa maneira se torna um investidor do mercado. Contudo esse não é um dinheiro gasto, já que, caso ele não queira continuar anunciando, ele pode vender seus *tokens* e obter seu investimento de volta.

4.2.2 Gerente de Anúncios

No Gerente de Anúncios, vendedores podem publicar seus produtos. As responsabilidades do Gerente são disponibilizar e atestar a validade dos anúncios para outros usuários na *blockchain*. Ademais, caso haja algum problema com certos anúncios, são disponibilizas funções para denúncias que são resolvidas se comunicando com o Gerente de Stake (4.2.1) e o Gerente de Disputas (4.2.4).

Os anúncios devem ser disponibilizados aos compradores utilizando a quantidade em *stake* do vendedor como critério para ordem de exibição. Ou seja, quanto mais *tokens* investidos na plataforma, mais visibilidade para os produtos e mais confiança para os compradores. Todavia, há mais risco por parte do anunciante devido à maior exposição de *tokens* no caso de uma denúncia.

4.2.3 Gerente de Ordens

Caso um comprador se interesse por algum produto, ele deve enviar sua proposta para o Gerente de Ordens. A proposta possui diversas informações necessárias para a execução, dentre elas o valor total e o endereço do ERC-20. Esses fundos devem ser transferidos para o contrato no ato da proposta enquanto aguardam a resposta do vendedor. Este pode aceitar dependendo das condições propostas pelo comprador, como valor de frete, local de frete, entre outros. Enquanto não for aceita pelo vendedor, ambas as partes podem cancelar a qualquer momento.

O Gerente de Ordens fica responsável por verificar com o Gerente de Anúncios (4.2.2) se o anúncio é válido e guardar os fundos da em segurança além de disponibilizar toda a lógica de liberação de fundos, confirmação de entrega, garantia e reembolso. Caso as duas partes não entrem em um acordo, o Gerente de Disputas (4.2.4) poderá

[1] Esta quantidade mínima poderia ser fixa, definida anteriormente, ou pode ser ajustada por uma organização autônoma descentralizada (*DAO*) no Gerente de Stake. Porém este processo está fora do escopo deste trabalho

ser acionado e assim levar o caso a um juri que irá decidir qual parte tem o direito de ficar com os fundos.

4.2.4 Gerente de Disputas

O Gerente de Disputas é uma camada abstrata para comunicação com os Gerentes de Anúncios e Ordens. É então utilizada sistemas de arbitração como serviço (*Arbitration as Service*) já disponíveis na *blockchain* como Kleros (LESAEGE; GEORGE; AST, 2020) ou Aragon Court (DECENTRALIZED. . . , s.d.). O Gerente de Disputas poderá implementar qualquer lógica ligada a essas cortes descentralizadas servindo como um mecanismo de *upgrade*, tendo em vista que contratos são imutáveis.

O Gerente de Disputas fica responsável por receber as taxas de arbitragem das partes, enviar a disputa para a corte descentralizada e, ao receber a resposta, devolver a taxa de arbitragem à parte ganhadora e comunicar o resultado para os Gerentes de Anúncios ou Ordens.

4.3 FUNCIONAMENTO

4.3.1 Anúncio de produtos

Com o DGR em *stake* no Gerente de Stake, um anunciante é capaz de publicar seu anúncio. Entretanto, publicar informação na *blockchain* é algo extremamente custoso em termos de uso de *gas* visto que a informação publicada nela fica eternamente gravada.

Uma abordagem primordial seria armazenar no estado da *blockchain* todas as informações necessárias de um anúncio e, ao ser realizada uma ordem, utilizar as informações já salvas. Isso foi implementado pelo *OriginProtocol* (ORIGIN. . . , s.d.), contudo, conforme a rede do *Ethereum* foi ficando cada vez mais congestionada, a publicação de um simples anúncio chegava a custar dezenas de dólares.

Por isso, foi utilizada na proposta uma abordagem diferente, que reduz custos com *gas* de armazenamento. Sua principal diferença é passar todos os argumentos de uma estrutura de `Listing` (5.2.1) pelo *calldata* e validar apenas o *hash* dela. Assim, essa estrutura é passada em todas as funções envolvendo o anúncio, apenas verificando a validade do *hash*. Essa abordagem é ordens de magnitude mais barata pois cada byte passado como parâmetro custa entre 4 e 16 *gas*[2] enquanto cada operação de escrita chega a custar 20000 *gas* (DYNAMIC. . . , s.d.).

Um anúncio tem validade até a data de expiração definida. O vendedor também pode oferecer alguns benefícios como *cashback* para os compradores e comissão para incentivar outras pessoas a divulgar o produto. Também é possível dar um período de garantia onde o comprador pode pedir reembolso total do produto.

[2] Cada byte zero custa 4 *gas* e cada byte não-zero custa 16 *gas*

Essas são informações necessárias dentro da *blockchain* para realizar a validação. Entretanto diversas informações como título, descrição, imagem, quais *tokens* são aceitos e qual o preço, etc não devem estar na *blockchain* pois tornaria muito mais caro. Poderia ser utilizado um servidor centralizado para publicar essas informações ou então utilizar uma abordagem descentralizada. Para a agregação de informações do anúncio é publicado um arquivo JSON no IPFS (4.1.1), cujo *hash* fica disponível no contrato.

4.3.2 Denúncia de anúncio

Ao ser denunciado, um anúncio deixa de ser válido até que um resultado seja retornado. Se o resultado for contra, além de ter seus *tokens* queimados, o anúncio é desativado. Como o *stake* das duas partes fica bloqueado, caso o *stake* liberado ficar abaixo do valor mínimo em *stake*, os anúncios ficam desativados temporariamente até que sejam liberados novamente ou que seja adicionado mais *stake* novamente.

Um ponto importante a ser destacado é: para uma pessoa denunciar o vendedor, ele precisa ter no mínimo 20%[3] do *stake* total do vendedor. Ao utilizar porcentagem, uma denúncia afeta diretamente a visibilidade do vendedor tornando um custo gigante anunciar de forma errônea para quem possui muitos *tokens* em *stake*. Se, após a queima de *stake* devido a um anúncio inválido, o vendedor possuir menos que a quantidade mínima de *stake*, todos os seus anúncios são desabilitados até que seja corrigido.

Normalmente, quem terá *tokens* em *stake* são vendedores querendo anunciar, e ter um ambiente agradável para os clientes é crucial para manter os clientes na plataforma. Um vendedor com muitos *tokens* não se importa tanto com outros vendedores menores pois seus anúncios sempre aparecem na frente. Enquanto isso vendedores menores ficarão fiscalizando quem possui mais *stake* pois, além de ser benéfico para os detentores menos *tokens* em circulação, também significa que seus anúncios serão disponibilizados mais acima.

4.3.3 Ordem de produtos

Após um comprador se interessar por um produto, ele vai interagir com o Gerente de Ordens. Nesse contrato, um comprador faz uma proposta para um vendedor. Como um anuncio permite múltiplos *tokens*, e cada *token* teria sua própria quantidade equivalente, não faz sentido fazer essa verificação dentro do contrato e então fica a critério do vendedor escolher qual proposta melhor lhe agrada.

Assim que uma proposta é feita, os *tokens* já são transferidos para o contrato como maneira de garantir que os fundos estarão disponíveis no momento que o vendedor

[3] A ser definido pela DAO

aceitar. Caso o comprador ou o vendedor queiram cancelar a ordem, os fundos são transferidos novamente para o comprador. Vendedores tem o incentivo de responder todas as propostas pois essas informações de propostas não respondidas ficam disponíveis na *blockchain* podendo desencorajar outros compradores de fazerem novas propostas.

Como foi falado anteriormente, dados que são passados como argumentos pelo *calldata* são mais baratos do que salvar na *blockchain*. Por isso, ao invés de realizar a computação múltiplas vezes nos contratos inteligentes, calcula-se tudo anteriormente e é passado dentro da estrutura da `Order` (5.3.1) como valores, comissões, *cashback* e taxa do protocolo. Isso então é validado no momento que é criado a `Order` na primeira vez, e depois é utilizado como verdade, tirando a computação e o armazenamento do contrato. O anúncio também deve ser passado dentro da `Order`, pelos mesmos motivos anteriormente citados, que então é validado com o Gerente de Anúncios e assim permite a criação de uma proposta.

O identificador usado em uma `Order` é o *hash* dos elementos na mesma ordem da estrutura. Esse identificador é então usado para verificar o estado atual de uma `Order` e por isso é necessário um `nonce`, caso um comprador deseje comprar um mesmo produto duas vezes. Caso fosse usado um mesmo anuncio com o mesmo *token* e valores, o contrato identificaria como o mesmo *hash*, e não permitiria a criação.

Após ter sido aceita, uma ordem se torna uma `Transação` e entra para estado de `Aguardando Confirmação`. Esse é um período definido na proposta onde caso o comprador não dê nenhuma resposta, os fundos são liberados para o vendedor. Isso é necessário pois a informação de entrega não fica facilmente disponível na *blockchain* sem o auxílio de serviço de Oráculos, servidores que publicam a informação na rede.

Um incentivo para o comprador confirmar que chegou os produtos liberando os fundos para o vendedor é o uso de *cashback*. Esse *cashback* fica disponível apenas caso o comprador libere os fundos antes do período de confirmação, assim permitindo que o vendedor receba seus *tokens* mais cedo. Outro incentivo disponível nos contratos é o uso de garantia. Esse período de garantia é definida no próprio anúncio e após o período de garantia expirar, o vendedor pode sacar seus *tokens*. Caso o comprador não fique satisfeito com um produto que tenha garantia, ele pode pedir seu dinheiro de volta. Os papeis então se invertem e o vendedor tem um o mesmo período de confirmação que serve como mecanismo até receber o produto novamente. Após o tempo de confirmação, o comprador pode sacar seus *tokens* integralmente.

Há situações que devolver um produto é inviável, seja pelo valor do frete, ou por que foi danificado durante a entrega. Nesses casos, o comprador pode tentar chegar em um acordo com o vendedor e este pode dar um reembolso parcial. Se o comprador não se sentir satisfeito ou não chegar um acordo, durante o período de confirmação é possível abrir uma disputa que irá levar para o Gerente de Disputas. No lado do vendedor, se houver algum problema na devolução de um produto com garantia, o

vendedor pode abrir uma disputa durante o período de confirmação de volta.

Os possíveis estados que uma transação pode ter são os seguintes:

```
enum Status {
    NoTransaction,
    WaitingSeller,
    WaitingConfirmation,
    Warranty,
    WarrantyConfirmation,
    InDispute,
    Finalized
}
```

O *status* inicialmente é `NoTransaction` para todas as propostas e vai sendo alterados conforme os métodos são chamados. Determinados métodos só podem ser chamados caso o *status* esteja em um estado específico. A Figura 6 mostra os possíveis fluxos que podem ocorrer, dependendo de há ou não garantia, se a garantia foi reivindicada e se houve um desacordo entre as partes.

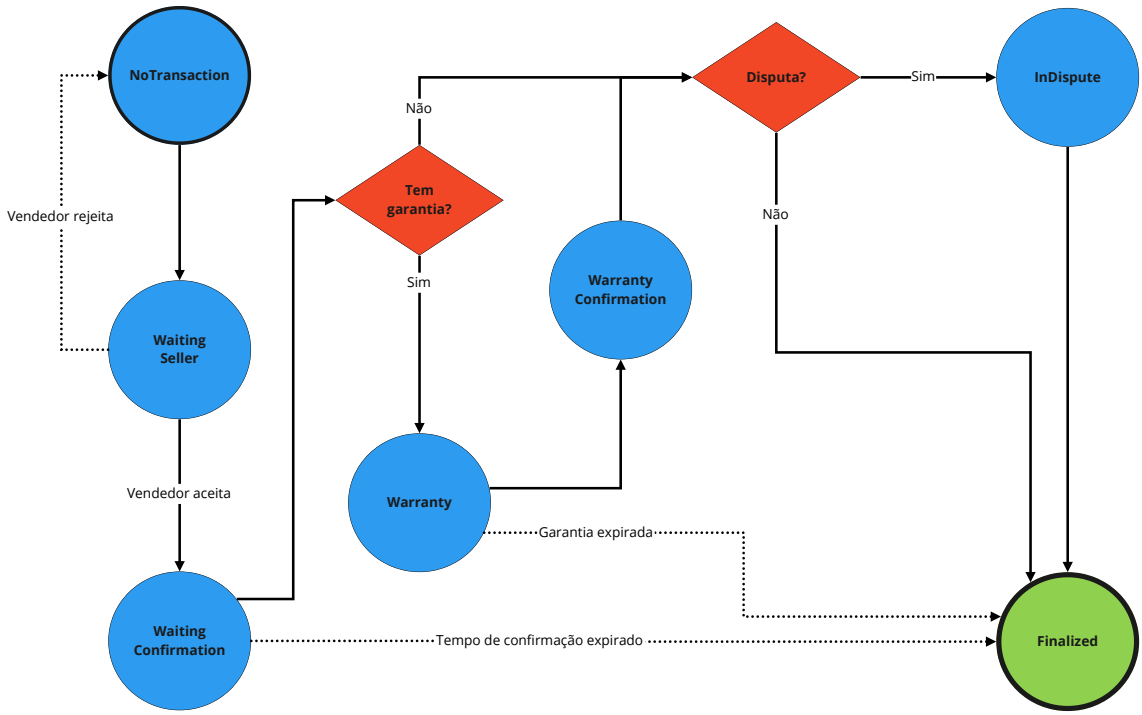


Figura 6 – Máquina de Estados de uma Transação

4.3.4 Disputas

Sempre que houver um desacordo entre as partes uma disputa pode ser criada e levada a um juri. O Gerente de Disputas serve como uma camada abstrata para interagir com o Juri verdadeiro. Existem alguns serviços de Arbitragem como Serviço na *blockchain* como por exemplo *Kleros* (LESAEGE; GEORGE; AST, 2020) e *Ara-*

gon Court (DECENTRALIZED. . . , s.d.), entretanto cada uma delas tem sua própria implementação, e caso fosse necessário alterar qual serviço usar, um sistema modular facilita muito.

Parte do que torna um *marketplace* rodando em *blockchain* mais eficiente, é o fato de não haver necessidade de interação de terceiros. Contudo, justiça necessariamente precisa de um terceiro que irá julgar. Devido a isso, essa é a parte mais cara do processo, pois para levar a disputa a um juri, é necessário pagar pelo julgamento. O valor depende da plataforma que está sendo usada (*Kleros*, *Aragon Court*, etc), as duas partes devem pagar essa taxa de arbitragem totalizando duas vezes o valor da taxa. Ao final, o lado vencedor recebe sua taxa de arbitragem enquanto o lado perdedor arca com todos os custos. Isso incentiva os participantes a abrirem disputas apenas caso tenham a absoluta certeza de que estão certos pois caso não estejam os custos são muito altos.

4.3.4.1 Kleros

Inicialmente, a plataforma de justiça utilizada foi o *Kleros* e por isso, é apresentado seu funcionamento. No *Kleros*, um juri é composto de 3 pessoas, escolhido com base no *stake* de PNK, o *token* ERC-20 da plataforma. O objetivo dos participantes da plataforma é votar conforme a maioria. Na ocorrência de uma votação contra a maioria, parte do *stake* do participante é perdido.

Mesmo havendo uma possibilidade de haver um resultado incorreto, todos na corte tem um objetivo em comum: julgar o mais justo possível e assim votar com a maioria o mais justo possível.

Ainda sim haverá casos que a parte perdedora se sentirá injustiçada e para isso a mesma pode apelar. Neste caso, o lado injustiçado pode pagar por mais 2 pessoas para refazer o julgamento. Isso incentiva ao lado injustiçado a pensar se irá ganhar mais apelando ou aceitando o resultado da disputa. O processo de apelação pode ocorrer até 3 vezes, aumentando o tamanho do juri e ficando mais caro a cada iteração.

Ao final, o resultado é devolvido para o Gerente de Disputas que devolve a taxa de arbitragem inicial e envia 100% dos fundos ao lado vendedor.

4.3.4.2 Funcionamento

Em uma disputa sempre haverá duas partes: a acusação (*prosecution*) e a defesa (*defendant*). As duas partes sempre vão estar disputando uma quantidade de *token* ERC-20. Além disso, deve-se ter registrado a quantidade de taxa de arbitragem paga por cada uma das partes para poder devolver o excedente ao final do processo. Também é guardado o endereço do contrato que criou a disputa, o *disputable*. Por último, tem-se um sistema de *timeout* e `DisputeStatus` para identificar o estado da disputa.

O `DisputeStatus` possui os seguintes valores:

```
enum DisputeStatus {
    NoDispute,
    WaitingProsecution,
    WaitingDefendant,
    DisputeCreated,
    Resolved
}
```

A Figura 7 mostra a máquina de estados da disputa. Inicialmente todas as disputas começam com o estado `NoDispute`. Para realizar a abertura, a acusação deve pagar sua taxa de arbitragem e alterando o estado para `WaitingDefendant`. Quando a defesa paga sua parte, é feita uma verificação se a outra parte também realizou o pagamento integral da taxa de arbitragem. Se não houver taxa suficiente, o estado é alterado para `WaitingProsecution` e assim por diante. Esse funcionamento ocorre pois a taxa de arbitragem está sujeita a alterações a qualquer momento e há a possibilidade de ocorrer entre o período de criar a disputa e pagar pela taxa de arbitragem. Quando as duas partes tem fundos suficientes, uma disputa é aberta e o estado é alterado para `DisputeCreated`. Assim que o resultado é retornado, o altera-se o estado para `Resolved` e a disputa é finalizada.

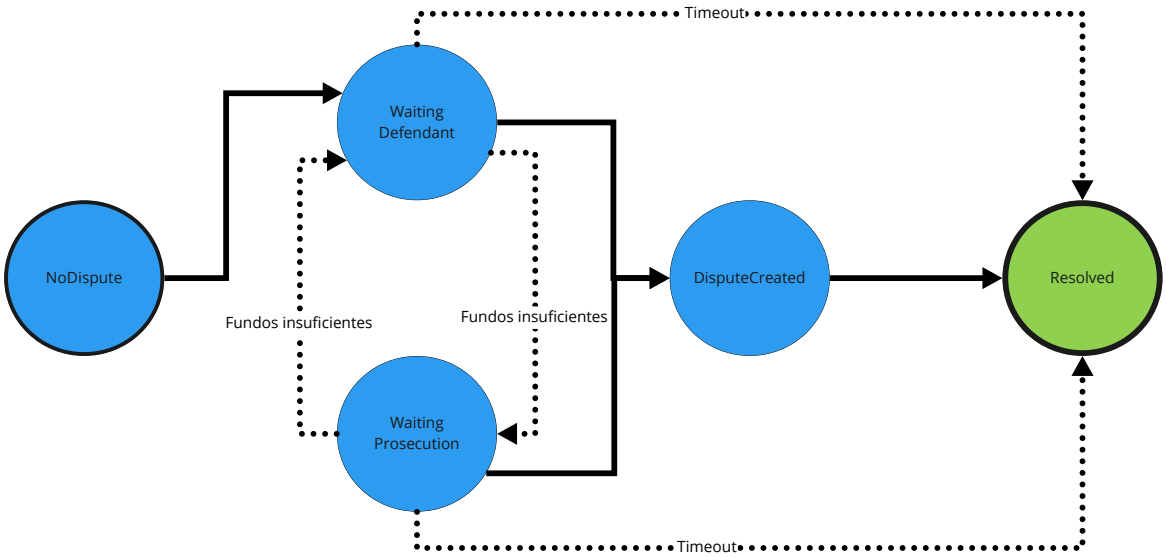


Figura 7 – Máquina de Estados de uma Disputa

5 PROTÓTIPO

Um protótipo do Dagora Market foi construído utilizando a linguagem de programação *Solidity* em conjunto com o *framework* Hardhat (HARDHAT..., s.d.) que facilita os testes dos contratos inteligentes.

Inicialmente, apenas um contrato contendo todas as funções necessárias para o funcionamento foi criado. Contudo, conforme o projeto foi evoluindo, viu-se a necessidade da divisão em múltiplos contratos. Isso reduziu a complexidade dos testes e ajudou na modularidade. A divisão dos contratos foi feita seguindo a proposta:

- StakeManager
- ListingManager
- OrderManager
- DisputeManager

5.1 STAKE MANAGER

Para permitir o *stake* do DGR, foram implementadas as seguinte funções para vendedores:

- `stakeToken(uint amount)`: Deposita recursos que ficam no contrato como garantia e mantém a validade dos anúncios.
- `unstakeToken(uint amount)`: Saca os recursos que não estiverem bloqueados.

Além disso, outras funções são necessárias para a comunicação entre o StakeManager e o ListingManager. As seguinte funções podem ser chamadas apenas pelo ListingManager (5.2):

- `lockStake(address staker, uint amount)`: Bloqueia parte dos fundos em *stake* e os impede de serem sacados.
- `unlockStake(address staker, uint amount)`: Libera os fundos bloqueados para serem sacados novamente.
- `burnLockedStake(address staker, uint amount)`: Queima *tokens* bloqueados, tirando-os de circulação.

5.2 LISTING MANAGER

5.2.1 Listing

A estrutura de `Listing` é definida da seguinte maneira:

```
struct Listing {
    string ipfsHash;
    address payable seller;
    uint256 expiration;
    uint256 commissionPercentage; /* two decimal places */
    uint256 warranty; /* In days */
    uint256 cashbackPercentage; /* two decimal places */
}
```

O identificador de um `Listing` é o *hash* dos elementos na mesma ordem da estrutura. Esse identificador é então usado para salvar na *blockchain* um *bool* dizendo que o anúncio está habilitado, assim qualquer pessoa pode saber a validade do mesmo *on-chain* apenas usando o *hash* do anúncio:

```
mapping(bytes32 => bool) public approvedListings;
```

5.2.2 IPFS

No protótipo, o seguinte JSON foi utilizado como uma versão inicial do que poderia ser necessário para um anúncio. Entretanto, como este sistema não depende da *blockchain* ele pode ser facilmente modificado e estendido.

Segue o exemplo de um anúncio de um videogame:

```
{
    "version": 1,
    "title": "Playstation 4 completo",
    "description": "Playstation 4 com 5 jogos, 2 controles, e brinde
       Giftcard PSN",
    "categories": [
        "Games"
    ],
    "tags": [
        "Playstation",
        "4",
        "completo",
        "jogos",
        "controles",
        "brinde",
        "giftcard",
        "psn"
    ],
    "images": [
```

```
        "ipfs://QmVVrGY9b2HXnMmmadxdVuVbf5eHX8C8yRqgjktkkPbGnL"
    ],
    "location": {
        "lat": -23.54788,
        "lng": -46.63818
    },
    "allowed_tokens": [
        "0xe6b8a5cf854791412c1f6efc7caf629f5df1c747",
        "0x1d4e6027a943fffbbd9f53999d67dad4a3ab1c0c",
        "0x9c3C9283D3e44854697Cd22D3Faa240Cfb032889"
    ],
    "price": "100000000"
}
```

Como curiosidade, caso deseja-se visualizar o arquivo acima no IPFS, o *Content ID* do arquivo acima é `QmbnnUD6LUfup58kux5ZfuviKUSfNUC1jJJW7YmmoHhFfZ`.

5.2.3 The Graph

Para a indexação dos anúncios, é necessário definir o `schema` do *subgraph*, que possui as entidades disponíveis para as buscas através de *GraphQL*:

```
type Listing @entity {
  id: ID!
  seller: Seller!

  commissionPercentage: BigInt!
  cashbackPercentage: BigInt!
  warranty: BigInt!

  expiration: BigInt!

  ipfsHash: String! # hash of the ipfs content
  quantity: BigInt!

  price: BigInt!
  title: String
  description: String

  categories: [String!]!
  tags: [String!]!
  images: [String!]!


  tagSearch: String
  sellerStake: BigInt!

  allowed_tokens: [Bytes!]!
```

```
}

type Seller @entity {
  id: ID!
  balance: BigInt!
  lockedTokens: BigInt!
  listings: [Listing!]!
}
```

Uma consulta para buscar os anúncios da categoria `Games` ordenado por stake fica da seguinte forma:

```
{
  listings(first: 1, orderBy: sellerStake, where: {
      categories_contains:["Games"]}) {
    title
    description
    price
    allowed_tokens
    seller
  }
}
```

E assim pode-se obter a seguinte resposta:

```
{
  "data": {
    "listings": [
      {
        "allowed_tokens": [
          "0xe6b8a5cf854791412c1f6efc7caf629f5df1c747",
          "0x1d4e6027a943fffbbd9f53999d67dad4a3ab1c0c",
          "0x9c3c9283d3e44854697cd22d3faa240cfb032889"
        ],
        "description": "Playstation 4 com 5 jogos, 2 controles, e brinde
           Giftcard PSN",
        "price": "100000000",
        "title": "Playstation 4 completo",
        "seller": {
          "id": "0x1d4e6027a943fffbbd9f53999d67dad4a3ab1c0c"
        }
      }
    ]
  }
}
```

5.2.4 Métodos

Um anunciante pode gerenciar seus anúncios através das seguintes funções disponíveis no ListingManager:

- `createListing(Listing calldata _listing, uint256 _quantity)`: Comunica com o StakeManager para saber a quantidade de *tokens* em *stake*, verifica a validade do anúncio e, caso tudo esteja em correto, habilita o anúncio publicando as informações através de um evento.
- `updateListing(Listing calldata _listing,uint256 _quantity)`: Publica um evento informando a nova quantidade de produtos.
- `cancelListing(Listing calldata _listing)`: Cancela o anúncio publicando um evento.

Caso haja algum problema com o anúncio criado, qualquer pessoa com *stake* suficiente é capaz de denunciar utilizando a seguinte função:

- `report(Listing memory _listing)`: Denuncia um anúncio criando uma disputa contra o vendedor no DisputeManager.

Além dessas funções, o DisputeManager se comunica com o ListingManager usando o conceito de `Disputable` que implementa duas funções:

- `onDispute(bytes32 _hash)`: Ao criar uma disputa, é chamado o StakeManager para bloquear os fundos em *stake* de ambos os lados até que seja resolvida essa disputa.
- `rulingCallback(bytes32 _hash, uint256 _ruling)`: Ao receber o resultado da disputa, os *tokens* em *stake* do lado vencedor são liberados enquanto o *stake* do perdedor é queimado.

5.3 ORDER MANAGER

5.3.1 Order

A estrutura de uma `Order` é definida da seguinte maneira:

```
1 struct Order {
2     Listing listing;
3     address payable buyer;
4     ERC20 token;
5     uint256 total;
6     uint256 protocolFee;
7     uint256 confirmationTimeout; /* In days */
8     uint256 nonce; /* A buyer may want to buy the same product twice */
```

```
9       address payable commissioner;
10      uint256 cashback;
11      uint256 commission;
12  }
```

Mesmo utilizando a maior quantidade possível de informação fora da *blockchain*, algumas informações necessariamente precisam ser armazenadas dentro e pra isso, quando uma `Order` é criada, existe também o conceito de `Transaction`, que são as informações salvas dentro do contrato:

```
1   struct Transaction {
2       /* Keep track of status update */
3       uint256 lastStatusUpdate;
4       /* Refund the seller can give */
5       uint256 refund;
6       /* Current status */
7       Status status;
8   }
9   /* mapping of order hash to transaction */
10  mapping(bytes32 => Transaction) public transactions;
```

5.3.2 Métodos

O OrderManager é de longe o contrato mais complexo do DagoraMarket. Ele deve guardar os fundos com segurança e ter todas a lógica de estados necessária para o funcionamento de cada `Transaction`. As seguintes funções estão disponíveis:

- `createOrder(Order calldata _order)`: Enviada pelo comprador, transfere os tokens para o contrato e muda o estado da `Transaction` para `WaitingSeller`.
- `acceptOrder(Order calldata _order)`: Enviada pelo vendedor, confirma uma ordem mudando o estado para `WaitingConfirmation`.
- `cancelOrder(Order calldata _order)`: Comprador ou vendedor podem cancelar a ordem, devolvendo os fundos para o comprador e voltando o estado para `NoTransaction`.
- `confirmReceipt(Order calldata _order)`: Caso o anúncio tenha garantia, o comprador pode confirmar o recebimento e alterando o estado para `Warranty`. A ordem só é elegível para garantia caso o vendedor não tenha dado reembolso parcial.
- `claimWarranty(Order calldata _order)`: Caso o anúncio tenha garantia, o comprador pode reivindica-la. Nesse caso, os papeis se invertem e quem aguarda pelo recebimento do produto é o vendedor. O estado é alterado para `WarrantyConfirmation`.

- `confirmWarrantyReceipt(Order calldata _order)`: O vendedor pode confirmar a chegada do produto e liberar o dinheiro para o comprador. O estado é alterado para `Finalized`.

- `executeOrder(Order calldata _order)`: Essa função fica disponível para finalizar a transação sempre que as condições sejam atendidas. Ou seja, em casos que o anúncio não há garantia, o comprador pode liberar e confirmar antes do período. Caso haja um *timeout* nos estados `WaitingConfirmation`, `Warranty` ou `WarrantyConfirmation`, essa função pode ser usada por qualquer entidade para finalizar a transação. O estado é alterado para `Finalized`.

- `updateRefund(Order calldata _order, uint256 _refund)`: O vendedor durante o período de `WaitingConfirmation` pode permitir o reembolso parcial para o comprador. Esse reembolso deve ser maior que o valor do *cashback*, e menor do que o valor total menos a taxa do protocolo e comissão.

- `disputeOrder(Order calldata _order)`: Caso o comprador não entre em acordo com o vendedor, é criada uma disputa.

- `disputeWarranty(Order calldata _order, uint256 _refund)`: Caso o vendedor não receba seu produto de forma correta na garantia, é criada uma disputa.

Da mesma forma que o ListingManager, em caso de disputas é implementado o `Disputable` que implementa as seguintes funções para ser chamado pelo DisputeManager:

- `onDispute(bytes32 _hash)`: Ao criar uma disputa, o estado é alterado para `InDispute`.

- `rulingCallback(bytes32 _hash, uint256 _ruling)`: Ao receber o resultado da disputa, os *tokens* são enviados ao lado vencedor ou divididos meio a meio no caso de indecisão do juri. O estado é alterado para `Finalized`.

O diagrama na Figura 8 mostra todas as possíveis tomadas de decisões e resultados.

5.4 DISPUTE MANAGER

5.4.1 Dispute

Como o Dispute Manager é uma camada a mais para interagir com o sistema de justiça, todos os dados que antes podiam ser passados como argumento agora devem estar armazenadas na *blockchain*. Uma disputa é representada da seguinte maneira no contrato:

```
1  struct Dispute {
2      address payable prosecution;
3      address payable defendant;
4      ERC20 token;
5      uint256 amount;
6      uint256 prosecutionFee;
7      uint256 defendantFee;
8      IDisputable disputable;
9      uint256 lastInteraction;
10     DisputeStatus status;
11 }
```

5.4.2 Métodos

Para interagir com o Dispute Manager, os seguintes métodos ficam disponíveis:

- `createDispute(bytes32 _hash, address payable _prosecution, address payable _defendant, ERC20 _token, uint256 _amount )`: Essa função deve ser chamada pelo ListingManager em caso de denúncia ou pelo OrderManager em caso de desacordo. Inicia o processo de disputa levando todos os dados à *blockchain*. É utilizado o *hash* como ID da disputa.
- `payArbitrationFee(bytes32 _hash)`: Deve ser chamado pela acusação ou defesa para pagar sua taxa de arbitragem. Caso as duas partes tenham pago o valor integral da taxa de arbitragem, o processo é mandando para a plataforma de justiça.
- `disputeTimeout(bytes32 _hash)`: Se um dos lados não pagar a taxa de arbitragem em um período pre-determinado, o outro lado pode chamar esta função para finalizar a disputa a seu favor
- `appeal(bytes32 _hash)`: Apela à corte, solicitando um julgamento com mais pessoas.
- `submitEvidence(bytes32 _hash, string calldata _evidence)`: Envia um *hash* de IPFS como evidencia.
- `rule(bytes32 _hash, uint256 _ruling)`: Apenas o contrato da corte pode enviar a resposta final do julgamento.

O diagrama da Figura 9 mostra todos os fluxos de chamada pelo DisputeManager.

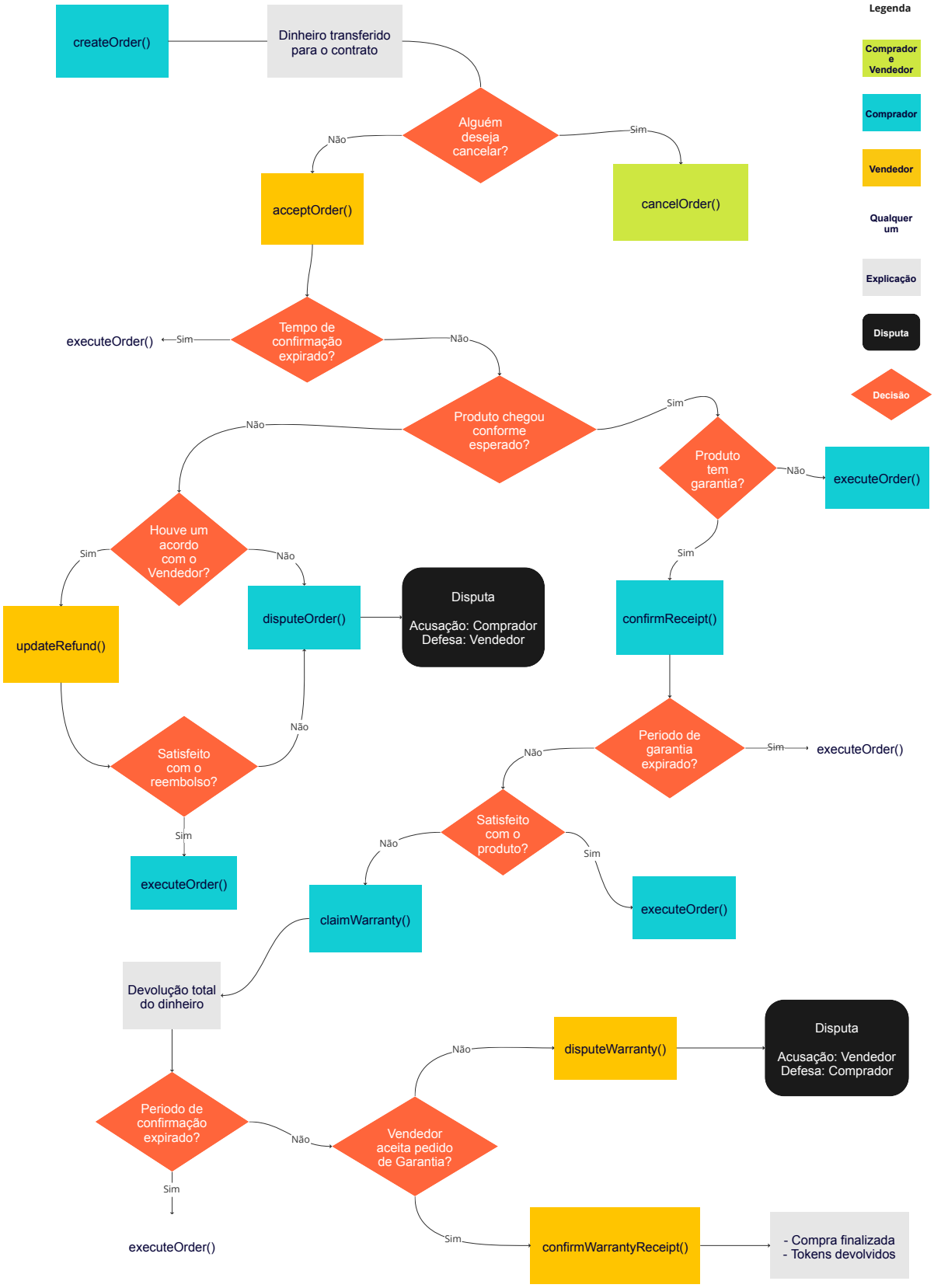


Figura 8 – Fluxos disponíveis em uma Transação

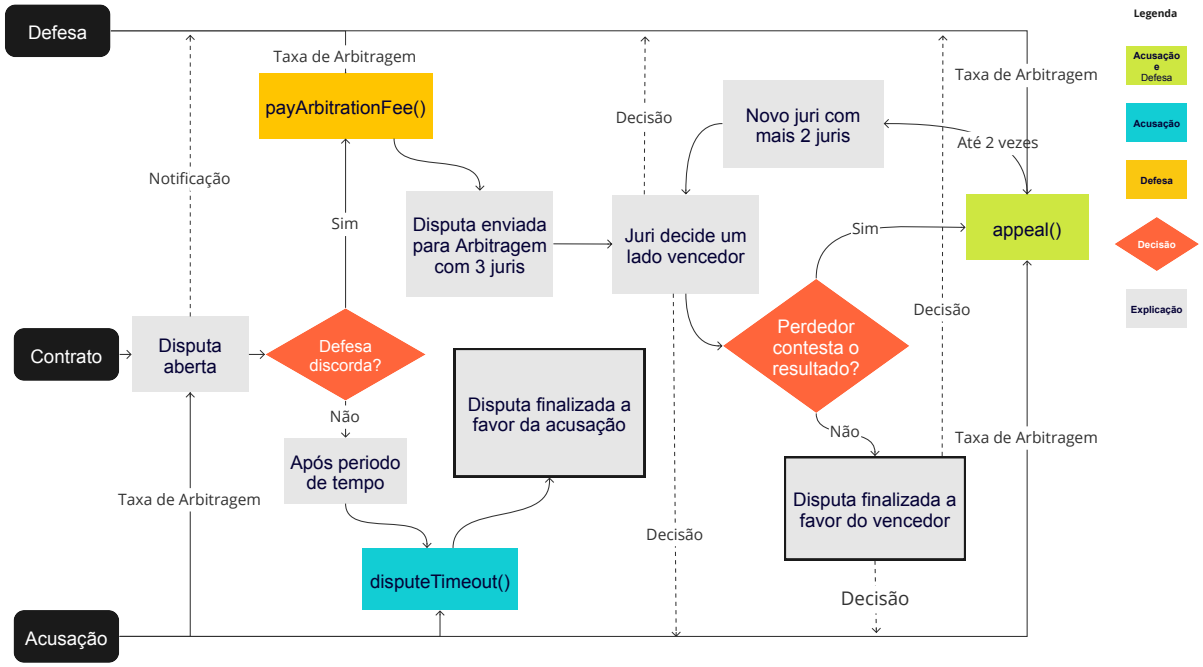


Figura 9 – Fluxo de Disputa

6 ANÁLISE

Neste capítulo, análises serão feitas sobre o uso de *gas* pelos contratos e serão comparados o custos nas mais diversas *blockchains* e *layer-2 solutions*. Porém, antes de analisar os resultados, foram realizados testes nos contratos para garantir a corretude de toda a implementação chegando a 100% de cobertura nos testes. Ao mesmo tempo, foi utilizada a ferramenta `Gas Reporter` do *Hardhat*(HARDHAT..., s.d.) que possibilita a exportação de uma tabela com o uso de *gas* de cada função. Na Tabela 1 é descrito o contrato, o método testado, o mínimo, máximo e média de uso de *gas*.

6.1 AVALIAÇÃO DE CUSTOS EM GAS

Para melhor representar a realidade, foram criados diversos fluxos que representam os casos de uso mais comuns. O objetivo é a comparação de custos com *marketplaces* centralizados para verificar a viabilidade da abordagem descentralizada. Inicialmente, os contratos são avaliados em termos de custo de *gas* pois o valor final depende da taxa da rede e do preço to *token*, como falado anteriormente.

Os seguintes casos foram avaliados:

- Anúncio de produto
- Denuncia de anúncio
- Ordem bem sucedida
- Ordem mal sucedida
- Disputa de ordem

6.1.1 Anúncio de produto

Neste fluxo é avaliado o custo de anunciar na plataforma. É levado em consideração que o vendedor já tem *tokens* DGR e que o StakeManager está aprovado a transferir os fundos dele. Usando a Tabela 1 para a situação representada na Figura 10, o custo médio de anunciar o produto é de 141489 *gas*.

6.1.2 Denuncia de anúncio

Neste fluxo é avaliado o custo de denunciar um anúncio já ativo. É levado em consideração que o vendedor e o denunciante já possuem *tokens* DGR em Stake. Usando a Tabela 1 para a situação representada na Figura 11, sem levar em consideração o custo do contrato da corte, o custo médio de denunciar e finalizar a disputa é de 395833 *gas*.

Tabela 1 – Custo em *gas* por método

Contract	Method	Min	Max	Avg
ListingManager	cancelListing	31589	33469	32341
ListingManager	createListing	76330	76507	76414
ListingManager	report	252038	286307	257806
ListingManager	updateListing	-	-	52532
OrderManager	acceptOrder	45368	45749	45512
OrderManager	cancelOrder	52154	52413	52284
OrderManager	claimWarranty	67616	67757	67691
OrderManager	confirmReceipt	47491	47632	47572
OrderManager	confirmWarrantyReceipt	62524	62614	62587
OrderManager	createOrder	143768	160982	144637
OrderManager	disputeOrder	214790	214901	214830
OrderManager	disputeWarranty	-	-	214897
OrderManager	executeOrder	83123	111318	89199
OrderManager	updateRefund	62457	62760	62547
StakeManager	burnLockedStake	-	-	53853
StakeManager	lockStake	32932	50032	37596
StakeManager	stakeTokens	53673	87885	65075
StakeManager	unlockStake	-	-	30995
StakeManager	unstakeTokens	-	-	48672
DisputeManager	appeal	-	-	25685
DisputeManager	disputeTimeout	74424	76678	75551
DisputeManager	payArbitrationFee	40205	80065	54780
DisputeManager	rule	68608	95246	83247
DisputeManager	submitEvidence	-	-	27407

6.1.3 Ordem bem sucedida

Neste fluxo é avaliado o custo de criar uma ordem de um anúncio ativo e finaliza-la com sucesso. É levado em consideração que o comprador já possui seus *tokens* aprovados para transferir para o OrderManager. Usando a Tabela 1, e usando o caminho mais longo na situação representada na Figura 12, o custo médio de concluir uma ordem é de 326920 *gas*.

6.1.4 Ordem mal sucedida

Neste fluxo é avaliado o custo de criar uma ordem de um anúncio ativo e finaliza-la sem sucesso. É levado em consideração que o comprador já possui seus *tokens* aprovados para transferir para o OrderManager. Usando a Tabela 1, e usando o caminho mais longo na situação representada na Figura 13, o custo médio de concluir uma ordem é de 367999 *gas*.

Figura 10 – Fluxo de Anúncio

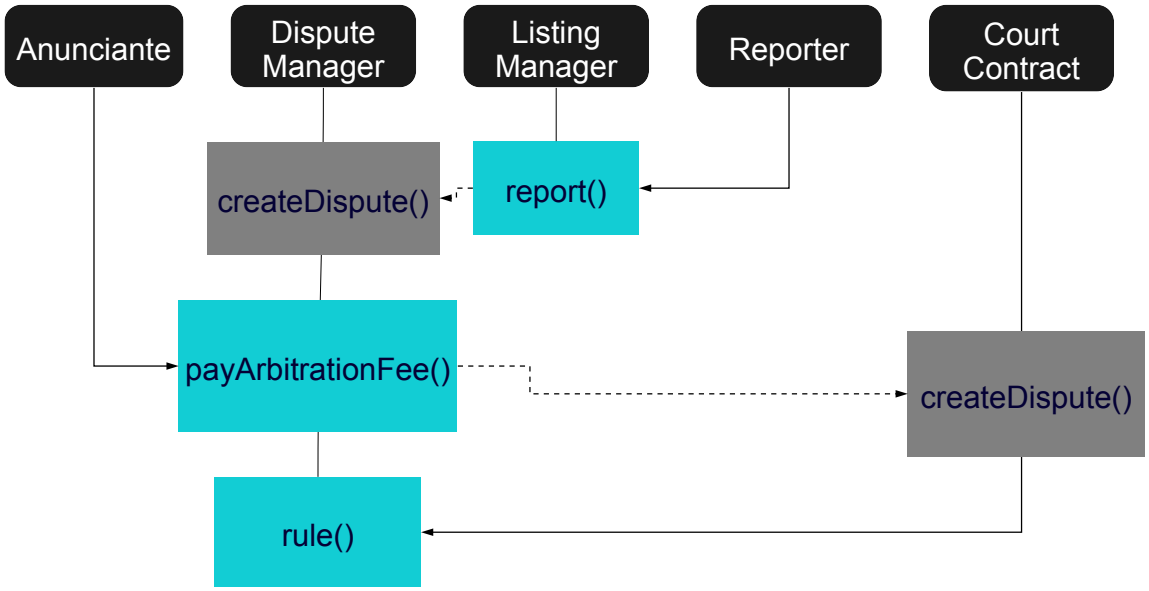


Figura 11 – Fluxo de Denúncia

6.1.5 Disputa de ordem

Neste fluxo é avaliado o custo de criar uma ordem, criar uma disputa e resolve-la. É levado em consideração que o comprador já possui seus *tokens* aprovados para transferir para o OrderManager. Usando a Tabela 1, e usando o caminho mais longo na situação representada na Figura 14, o custo médio de concluir uma ordem com disputa é de 658336 *gas*.

6.1.6 Resultado final

Como pode-se ver no gráfico da Figura 15, o fluxo mais barato em termos de uso de *gas* é anunciar um produto que pode ficar ainda mais barato no caso já haja

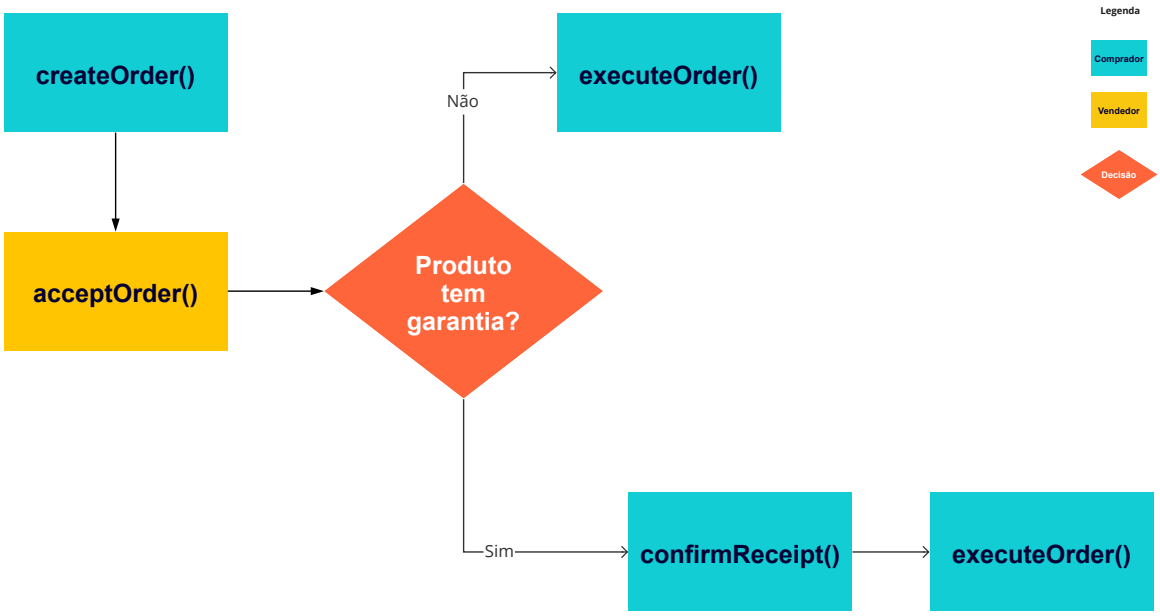


Figura 12 – Fluxo de Ordem bem sucedida

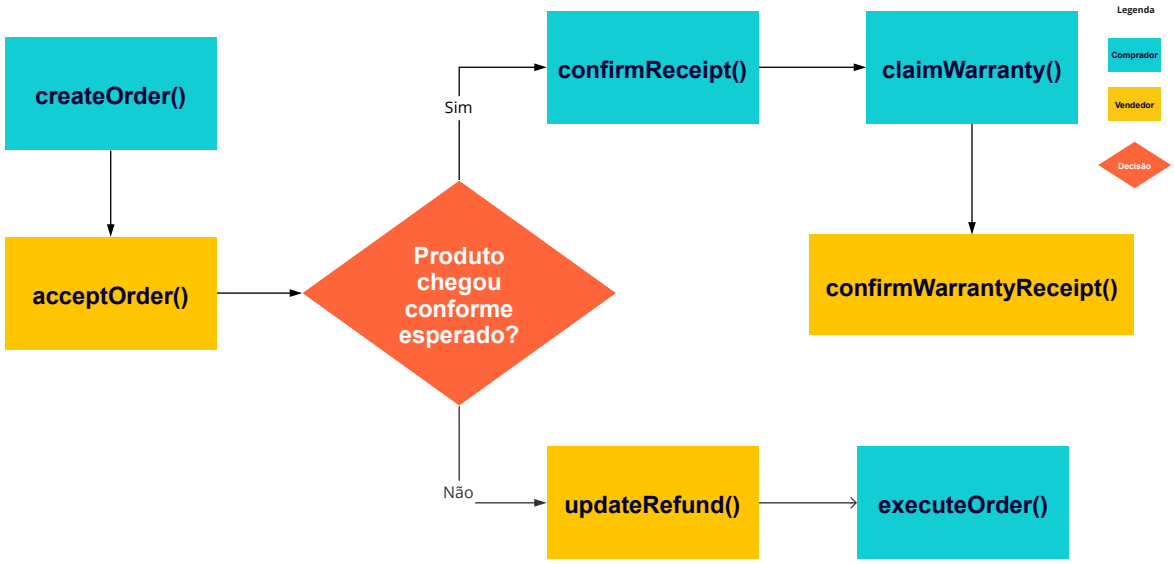


Figura 13 – Fluxo de Ordem mal sucedida

tokens em *stake*. Os custos dos fluxos de ordem bem e mal sucedida são muito próximos e a diferença deles se dá principalmente pela quantidade de interações com a *blockchain*. Os custos com *DisputeManager* é o que torna o fluxo de denúncia e disputa as interações mais caras, pois há a necessidade do armazenamento de informações dentro da *blockchain*.

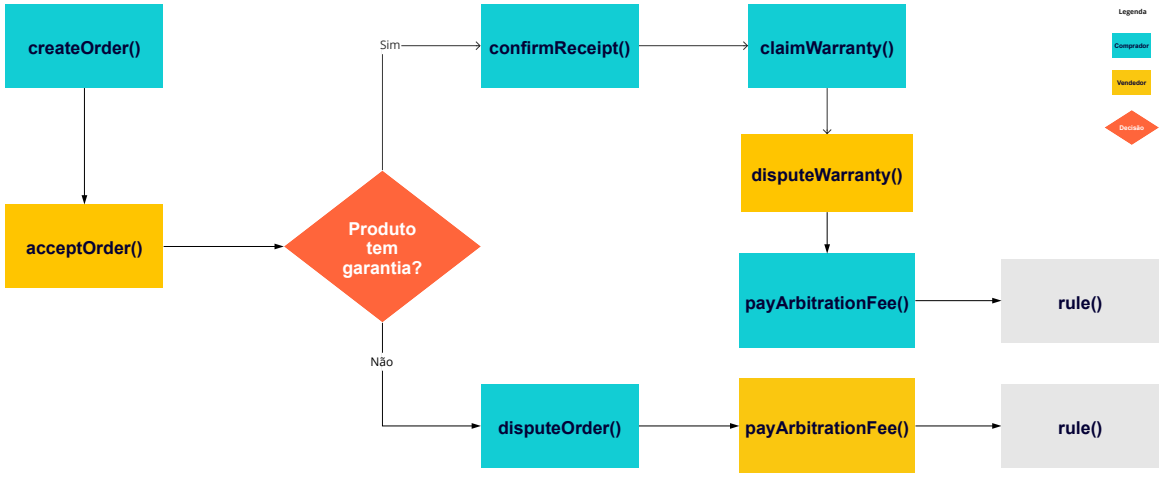


Figura 14 – Fluxo de Ordem com Disputa

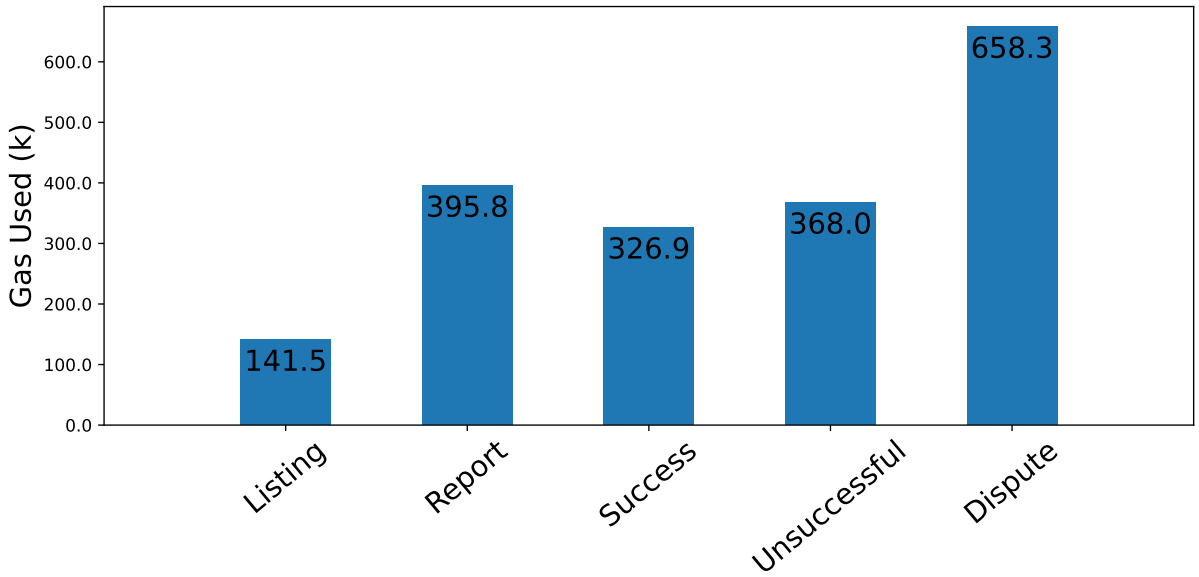


Figura 15 – Uso de *gas* por fluxo

6.2 CUSTOS EM DIFERENTES BLOCKCHAINS

Hoje, projetos descentralizados famosos como Uniswap (ADAMS *et al.*, 2021), AAVE (WOW@AAVE.COM, 2020), etc, rodam seus contratos não apenas em uma rede de *blockchain*, mas sim em diversas *blockchains* que possuem interoperabilidade. Isso tornou-se necessário pois conforme a rede do *Ethereum* foi ficando cada vez mais requisitada, também começou a ocorrer congestionamento na rede e suas taxas começaram a subir para manter a rede funcionando. Assim, interações mais baratas não valem a pena serem executadas na rede do *Ethereum* e por isso troca-se a segurança de rodar na rede principal por acessibilidade de pagar taxas menores.

Nesta secção, são mostrados os custos de executar os fluxos da secção anterior nas mais diversas *blockchains* que possuem algum tipo de *bridge* (mecanismo para

transferir fundos entre *blockchains*) com o *Ethereum*. Para chegar ao valor do *token* nativo é utilizado a mediana dos últimos 200 dias do *gas price* da rede e para converter esse valor para dólares é utilizado a média do preço do *token* dos últimos 200 dias. A data de referência para essa busca é dia 06/02/2021.

A escolha de quais redes foram avaliadas foi de certa forma arbitrária porém levando em consideração a quantidade e qualidade dos projetos que estão rodando nelas e o nível de segurança da rede. As *blockchains* que serão avaliadas são as seguintes:

- *Ethereum* (BUTERIN, 2013): rede principal de referencia
- *Optimism* (OPTIMISM. . . , s.d.): L2 usando *Optimistic Rollup*
- *Polygon* (KANANI; NAILWAL; ARJUN, 2019): *side-chain* usando *Proof of Stake* com *commits*
- *Fantom* (FANTOM. . . , s.d.): rede principal com bridge para *Ethereum*
- *BSC* (BNB. . . , s.d.): rede principal com bridge para *Ethereum*
- *Avalanche* (AVALANCHE:. . . , s.d.): rede principal com bridge para *Ethereum*

6.2.1 Ethereum

Usando dados do Etherscan (ETHEREUM. . . , s.d.) foi apurado o preço de *gas* mediano dos últimos 200 dias que trouxe o resultado de 107 gwei. O *Ethereum* tem o ETH como *token* principal. Seu preço usando a média móvel dos últimos 200 dias é de $3538.02. Para executar o Dagora Market no *Ethereum*, seriam gastos em dólares os valores do gráfico da Figura 16.

6.2.2 Optimism

Os *Optimistic Rollups* possuem uma forma diferente de calcular o custo das transações. De maneira geral existem 2 custos envolvidos: o custo da transação no L2 (armazenamento e computação) e o custo de *calldata* em L1. O primeiro é muito pequeno pois não necessita que todos os nodos do *Ethereum* processem a transação ficando o segundo como a maior parte do custo da transação. Como foi falado na secção 2.4.1, *rollups* otimistas não executam nada em L1 mas todas as transações são publicadas através do *calldata*, ou seja, esse custo depende diretamente do *gas price* do *Ethereum*.

A Tabela 2 possui os custos de *calldata* para comparação nas subsecções seguintes:

As taxas do *Optimism* em L1 seguem a seguinte formula (TRANSACTION. . . , s.d.):

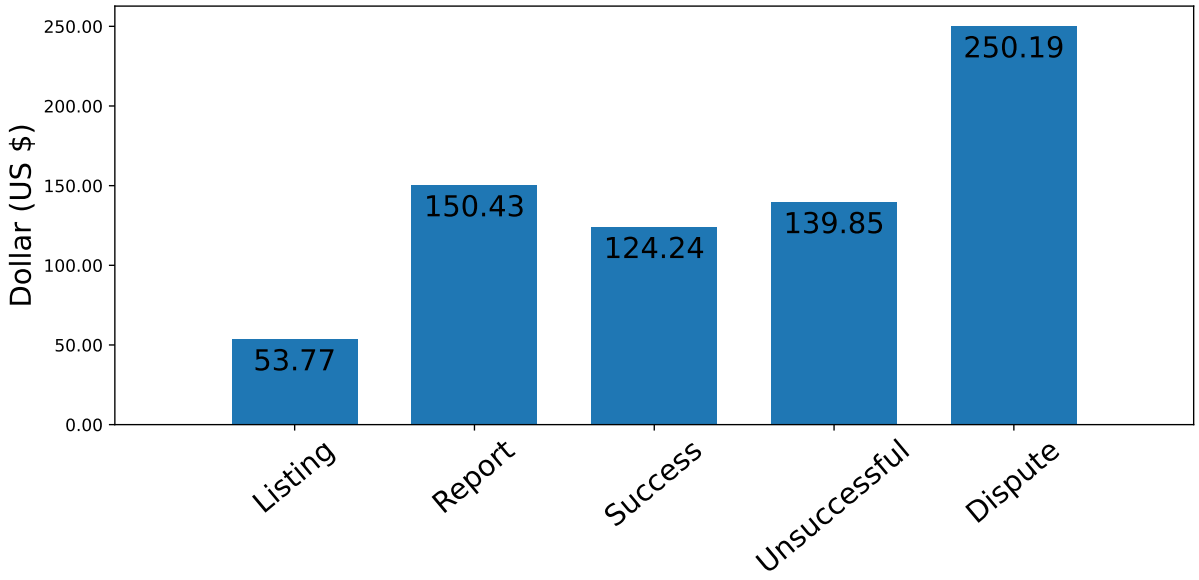


Figura 16 – Custo por fluxo no *Ethereum*

```
l1_data_fee = l1_gas_price * (tx_data_gas + fixed_overhead) *
    dynamic_overhead
```

Esse custo de *gas* passa por um multiplicador de bonus (*dynamic_overhead*) e o *gas price* para todo o custo de *gas* é o próprio da rede *Ethereum*. O *fixed_overhead* hoje está em 2100 e o dynamic_overhead em 1.24x[1].

Além do custo do L1, existe o custo de L2 que é o valor usado em *gas* (Tabela 1) multiplicado pelo *gas price* da rede Optimism que possui uma mediana de 0.001 *gwei* dos últimos 200 dias(PUBLIC..., s.d.). Para executar o contrato do Dagora no *Optimism*, seriam gastos em dólares os valores do gráfico da Figura 17.

A abordagem utilizada pelo Dagora Market não se beneficia muito de *optimistic rollups* pois todas as transações possuem a estrutura `Order` e `Listing` passada como parâmetro no *calldata*, contudo, pode-se ver uma grande redução nos custos de disputas, que utilizam armazenamento na *blockchain*. Uma solução para reduzir os custos dentro de *Optimistic Rollups* seria reescrever os contratos para salvar o *calldata* inicial no armazenamento e utilizar apenas o *hash* como id, assim teria-se menos informações no *calldata* e menos custos de *gas* em L1.

6.2.3 Polygon

Usando dados do Polygonscan(POLYGON..., s.d.) foi apurado o preço de *gas* mediano dos últimos 200 dias que trouxe o resultado de 94 gwei. O *Polygon* tem o MATIC como *token* principal. Seu preço usando a média móvel dos últimos 200 dias é de $1.66. Para executar o contrato do Dagora no *Polygon*, seriam gastos em dólares os valores do gráfico da Figura 18.

[1] Os valores foram buscados no contrato utilizado pelo Optimism em 06/02/2021 (OPTIMISM..., s.d.)

Tabela 2 – Custo em *gas* por método em Optimistic Rollups

Contract	Method	Calldata Gas Usage
ListingManager	cancelListing	8269
ListingManager	createListing	8570
ListingManager	report	8300
ListingManager	updateListing	8592
OrderManager	acceptOrder	11960
OrderManager	cancelOrder	11670
OrderManager	claimWarranty	11875
OrderManager	confirmReceipt	11855
OrderManager	confirmWarrantyReceipt	11857
OrderManager	createOrder	11932
OrderManager	disputeOrder	11795
OrderManager	disputeWarranty	11820
OrderManager	executeOrder	12158
OrderManager	updateRefund	12201
StakeManager	burnLockedStake	2208
StakeManager	lockStake	2200
StakeManager	stakeTokens	1471
StakeManager	unlockStake	2196
StakeManager	unstakeTokens	1440
DisputeManager	appeal	2184
DisputeManager	disputeTimeout	2148
DisputeManager	payArbitrationFee	2222
DisputeManager	rule	2436
DisputeManager	submitEvidence	3152

6.2.4 Fantom

Usando dados do Ftmscan(FANTOM. . . , s.d.) foi apurado o preço de *gas* mediano dos últimos 200 dias que trouxe o resultado de 281 gwei. A *Fantom* tem o FTM como *token* principal. Seu preço usando a média móvel dos últimos 200 dias é de $1.70. Para executar o contrato do Dagora na *Fantom*, seriam gastos em dólares os valores do gráfico da Figura 19.

6.2.5 BSC

Usando dados do Bscscan(BINANCE. . . , s.d.) foi apurado o preço de *gas* mediano dos últimos 200 dias que trouxe o resultado de 6 gwei. A *BSC* tem o BNB como *token* principal. Seu preço usando a média móvel dos últimos 200 dias é de $468.90. Para executar o contrato do Dagora na *BSC*, seriam gastos em dólares os valores do gráfico da Figura 20.

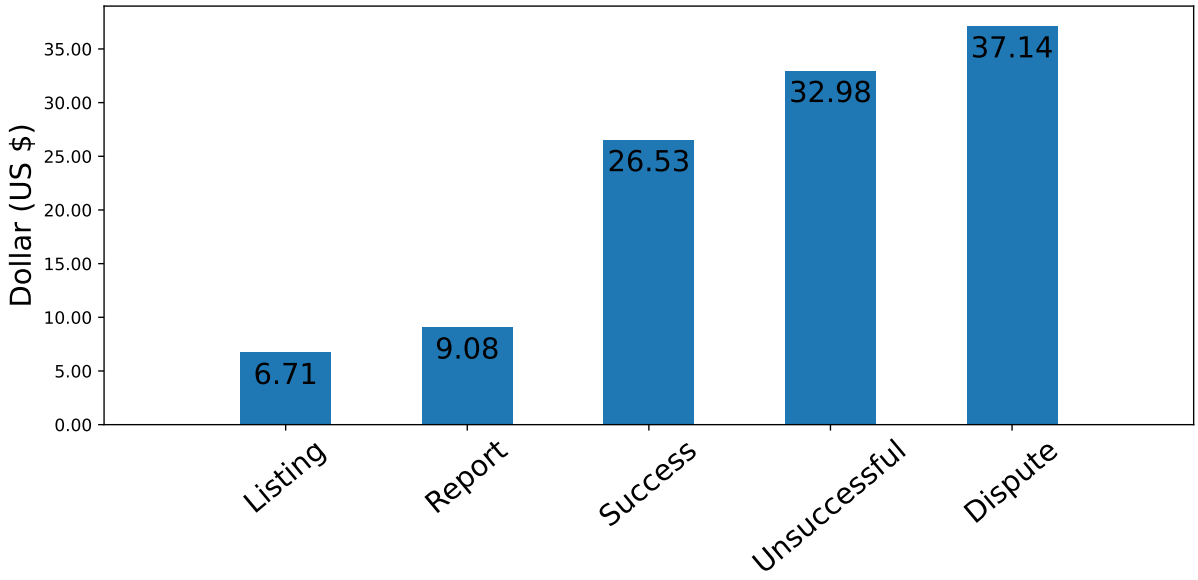


Figura 17 – Custo por fluxo no Optimism

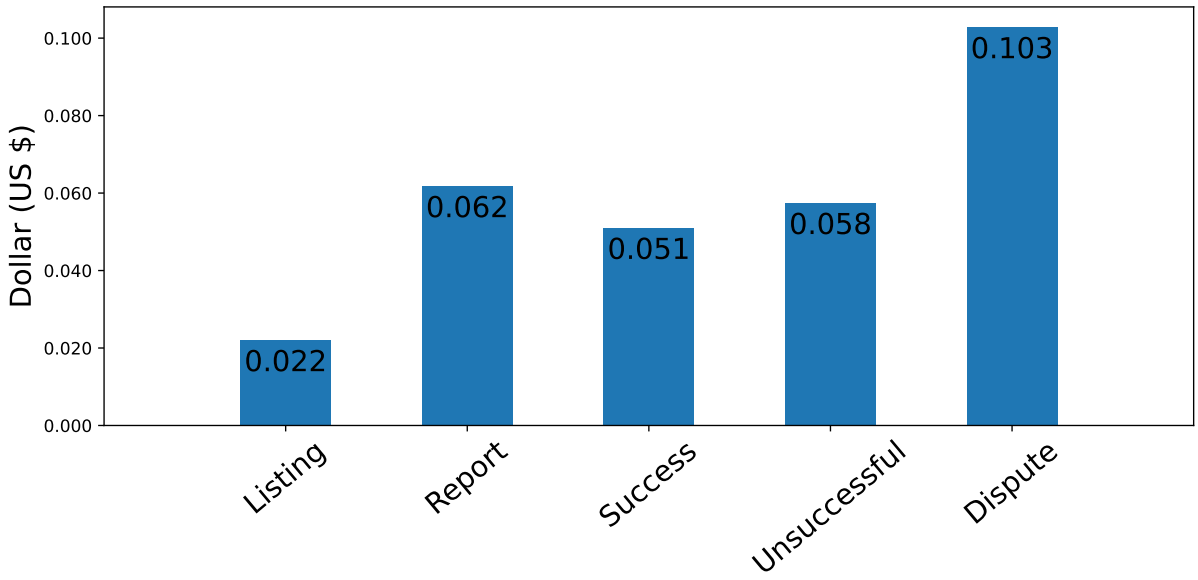


Figura 18 – Custo por fluxo no Polygon

6.2.6 Avalanche

Usando dados do Snowtrace(SNOWTRACE:. . . , s.d.) foi apurado o preço de *gas* mediano dos últimos 200 dias que trouxe o resultado de 54 gwei. A *Avalanche* tem o AVAX como *token* principal. Seu preço usando a média móvel dos últimos 200 dias é de $69.23. Para executar o contrato do Dagora na *Avalanche*, seriam gastos em dólares os valores do gráfico da Figura 21.

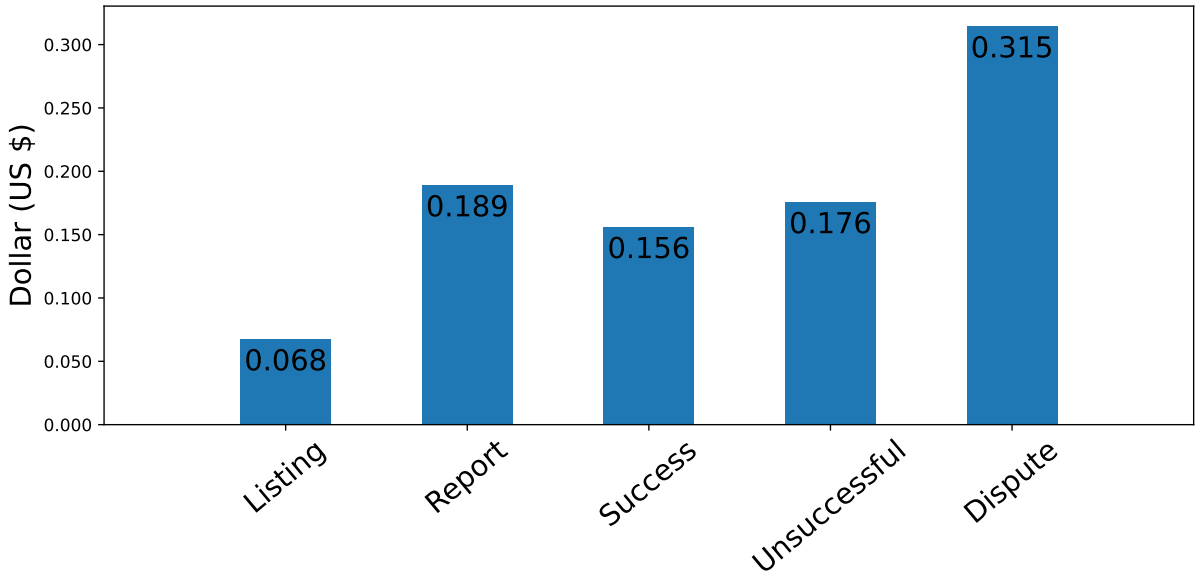


Figura 19 – Custo por fluxo no Fantom

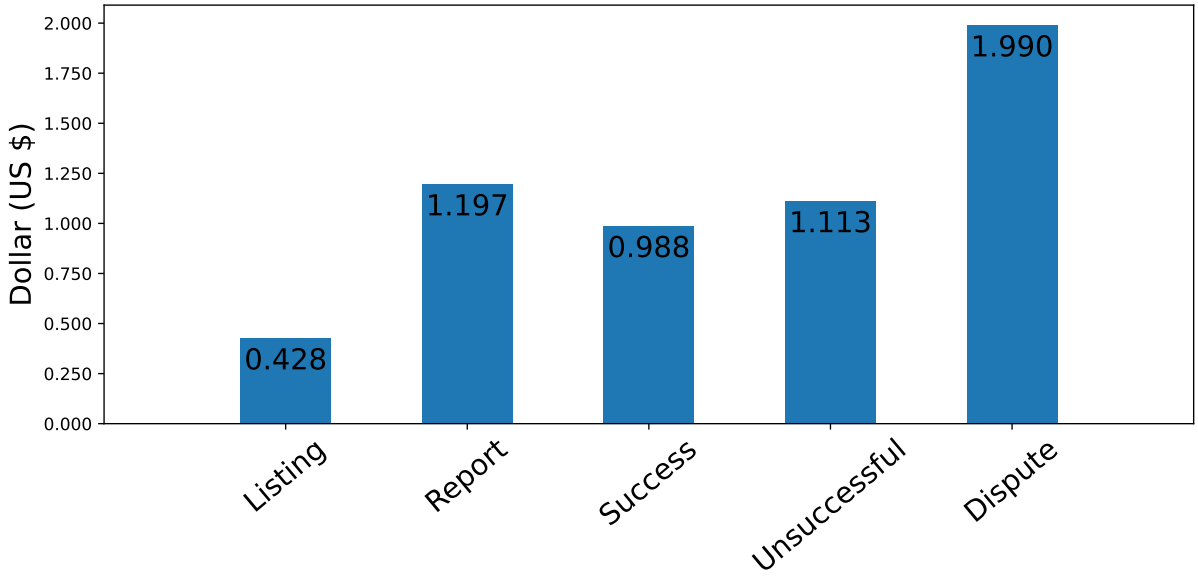


Figura 20 – Custo por fluxo na BSC

6.3 RESULTADOS

Com os dados obtidos na secção anterior, tem-se o gráfico para o fluxo de ordem bem sucedida na Figura 22. Apesar de ser notável a diferença de custo entre *Ethereum* e Optimism e as outras *blockchains*, não significa que não seja interessante rodar uma instância do contrato nessas redes.

A diferença entre as redes, além das taxas, se dá pela segurança que elas oferecem, seja pela quantidade de força computacional investida na rede, seja pela quantidade de *tokens* em *stake* por validadores. Ademais, certas *blockchains* são consideradas mais centralizadas enquanto outras são mais suscetíveis ao ataque de 51%. Por exemplo, a *Fantom* possui 58 validadores ativos com 1,373,966,099

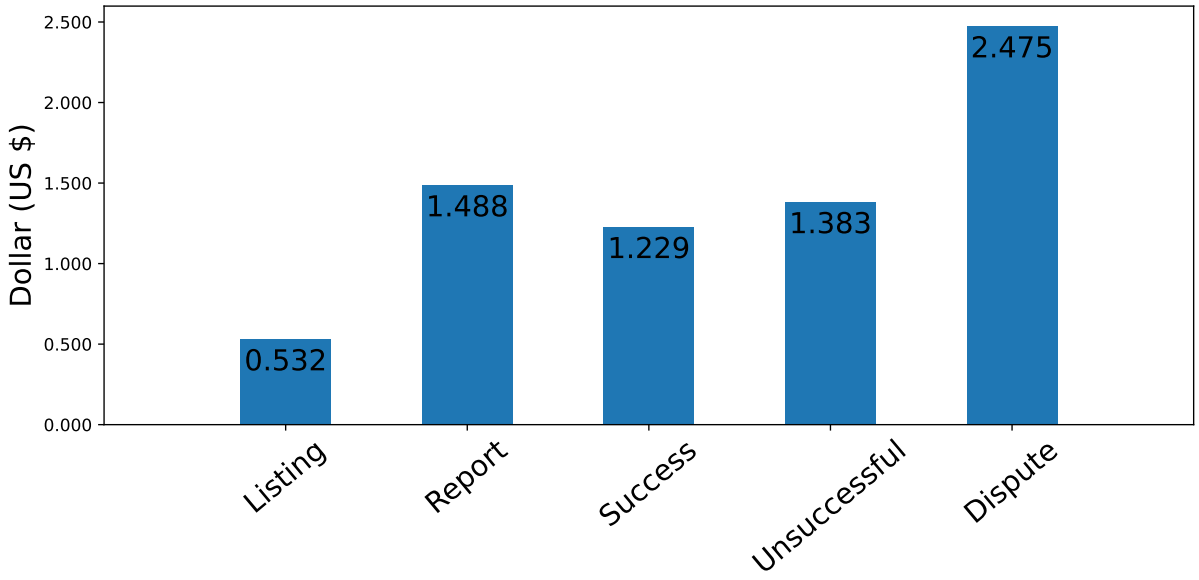


Figura 21 – Custo por fluxo na Avalanche

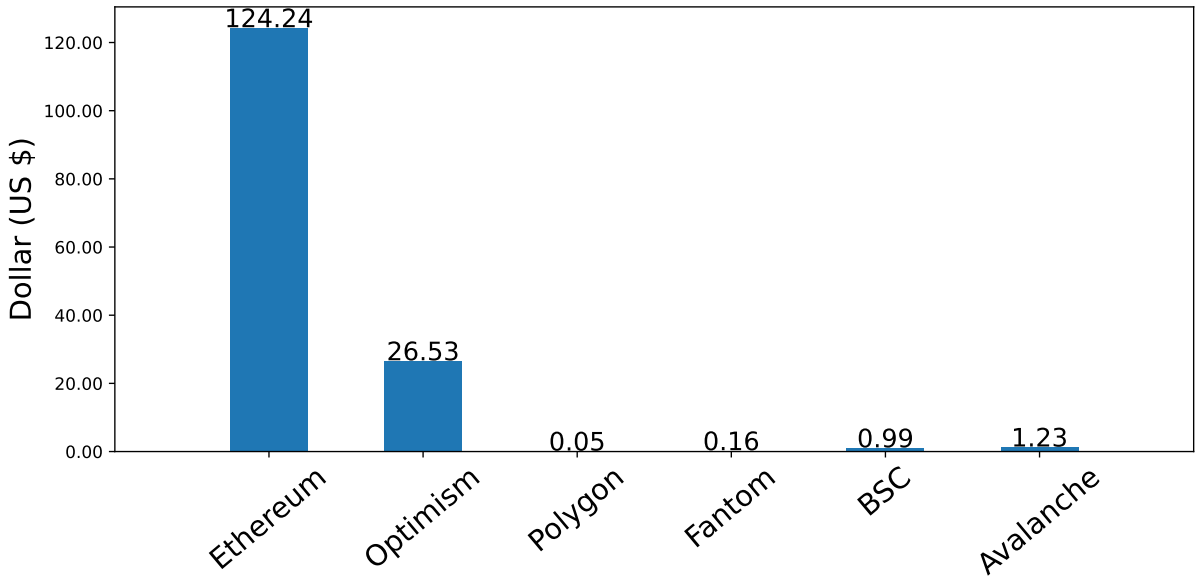


Figura 22 – Comparação de custos nas diferentes *blockchains*

FTM (US$ 2,335,742,368.30) em *stake* enquanto a *BSC* possui 21 validadores com 17,643,929.23 BNB (US$ 8,273,238,415.95).

A *Polygon*, apesar de ser considerada uma *Layer 2* do *Ethereum*, depende muito do *Proof of Stake* de seus validadores enquanto a *Optimism* usa *Optimistic Rollups* e herda mais a segurança do *Ethereum*. Sendo assim, diferentes *blockchains* podem ser utilizadas para diferentes valores de produtos no Dagora Market e tudo depende da segurança que o vendedor deseja.

Se for levado em consideração um mínimo de 5% do valor do produto em taxas cobradas por mercados centralizados, produtos com os valores apresentados na Figura 23 começam a compensar mais transacionar em cada uma das redes do que utilizar

marketplaces tradicionais.

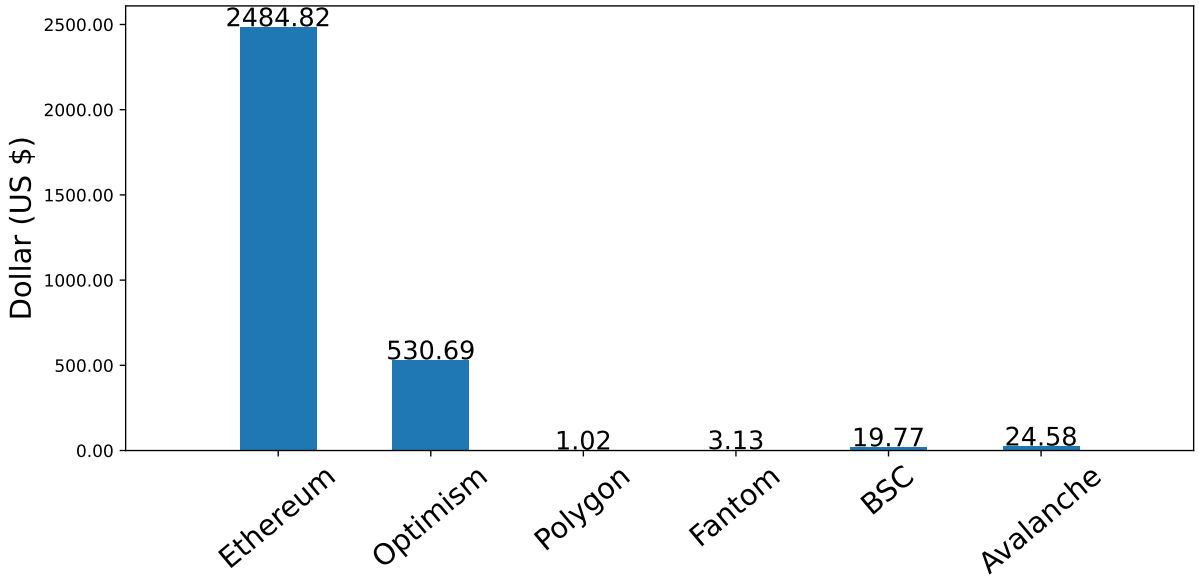


Figura 23 – Preços de produtos para tornar o Dagora viável nas diferentes *blockchains*

Assim sendo, pode-se definir classes de produtos a serem vendidos em cada uma das *blockchains* levando em consideração o nível de segurança. Produtos como casas, terrenos, carros, de maior valor requerem uma segurança maior e assim podem ser vendidos utilizando a rede do *Ethereum*. Produtos como computadores, celulares, bicicletas, de valor médio podem ser vendidos utilizando a rede da *Optimism* já que possuem uma certa segurança herdada do *Ethereum*.

Produtos como roupas, acessórios eletrônicos, produtos de beleza poderiam ser transacionados pelas redes da *BSC* e Avalanche que oferecem relativa segurança e uma baixa taxa. Por último, produtos como peças, papelaria, brinquedos pequenos podem ser negociados através das redes da *Polygon* e *Fantom* já que ofecerem as taxas mais baratas assim viabilizando todos os tipos de produtos. Vale evidenciar que nada impede produtos mais caros de serem negociados em redes mais baratas pois a utilização da rede é uma questão de escolha do vendedor.

7 CONSIDERAÇÕES FINAIS

No contexto desse trabalho, foi desenvolvido um protótipo de um *marketplace* descentralizado rodando em *smart contracts* de *blockchains* compatíveis com EVM. Esses *smart contracts* são capazes de publicar anúncios, transacionar ordens de compradores e se conectam com sistema de arbitragem para resolução de disputas de maneira descentralizada.

Com o protótipo, foram feitas análises de custos em *gas* de fluxos corriqueiros em um *marketplace* desde anunciar um produto até resolução de disputas. Ao final das análises, foi realizada uma comparação de custos em dólares através de diversas *blockchains* compatíveis com EVM e também foi feita uma comparação entre quais situações são mais vantajosas transacionar utilizando *blockchains* em relação à mercados centralizados.

Como resultado final, foram divididas classes de produtos nas quais são mais vantajosas transacionar em *blockchain*, já que possuem custos fixos, levando em consideração o nível de segurança necessário para cada classe.

Para trabalhos futuros, serviços de Oráculos, como Chainlink (BREIDENBACH *et al.*, 2021), poderiam ser implementados para liberar os fundos no momento que for a entrega for confirmada. A implementação de meta-transações, que não utilizam *gas*, proporcionaria uma melhor experiência do usuário, já que ao invés de gastar o *token* nativo, esse custo poderia ser descontado diretamente do preço do produto.

Pensando na otimização de recursos, *tokens* parados no Gerente de Ordens poderiam render juros caso fosse utilizada alguma estratégia DeFi (*Decentralized Finance*) como prover liquidez para plataformas de empréstimo, por exemplo o AAVE (WOW@AAVE.COM, 2020). Os juros poderiam ser utilizados para pagar as taxas de *gas* ou serem usados como parte dos lucros.

O Gerente de Ordens poderia ser mais modular pois hoje existem diversos fluxos opcionais como a garantia, o *cashback*, a comissão e no futuro poderiam haver mais opções como meta-transações, confirmação via *Chainlink*, leilões, etc. Cada um desses fluxos poderia ser um contrato diferente e otimizado de forma diferente podendo economizar recursos e estender ainda mais as possibilidades sem a necessidade de um upgrade que pode implementar riscos.

Por último, a implementação de uma interface gráfica para interação com o usuário seria um trabalho futuro fundamental já que o *backend* está implementado aguardando clientes. Inicialmente poderia ser feito uma aplicação *Web* e ser facilmente estendida para um aplicativo *mobile*. Nesses *frontends*, deveriam ser realizadas integrações com serviços como a Ramp Network (RAMP. . . , s.d.), Moonpay (MOONPAY. . . , s.d.), Wyre (WYRE. . . , s.d.), Transak (TRANSAK. . . , s.d.), etc, que permitem uma melhor experiencia do usuário ao possibilitar a compra de criptomoedas usando

cartão de crédito/débito.

REFERÊNCIAS

ADAMS, Hayden *et al.* **Uniswap v3 Core**. [*S.l.*: *s.n.*], 2021. Acessado em 2021-03-24. Disponível em: <https://uniswap.org/whitepaper-v3.pdf>. Citado nas pp. 16, 57.

ARBITRUM. [*S.l.*: *s.n.*]. Acessado em 2021-04-12. Disponível em: <https://offchainlabs.com/>. Citado na p. 30.

AVALANCHE: Blazingly Fast, Low Cost, Eco-Friendly | Dapps Platform. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://www.avax.network/>. Citado na p. 58.

AXIE Infinity. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://axieinfinity.com/>. Citado na p. 32.

BACH, L. M.; MIHALJEVIC, B.; ZAGAR, M. Comparative analysis of blockchain consensus algorithms. *In:* 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). [*S.l.*: *s.n.*], 2018. P. 1545–1550. DOI: 10.23919/MIPRO.2018.8400278. Citado na p. 24.

BECZE, Martin; JAMESON, Hudson. **EIP-1: EIP Purpose and Guidelines**. [*S.l.*: *s.n.*], out. 2015. Acessado em 2021-05-07. Disponível em: <https://eips.ethereum.org/EIPS/eip-1>. Citado na p. 28.

BEN-SASSON, Eli *et al.* **Scalable, transparent, and post-quantum secure computational integrity**. [*S.l.*: *s.n.*], 2018. Cryptology ePrint Archive, Report 2018/046. https://eprint.iacr.org/2018/046. Citado na p. 30.

BENET, Juan. Ipfs-content addressed, versioned, p2p file system, 2014. Disponível em: <https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf>. Citado na p. 34.

BEREGSZASZI, Alex; MUSHEGIAN, Nikolai. **EIP-140: REVERT instruction**. [*S.l.*: *s.n.*], fev. 2017. Acessado em 2022-02-14. Disponível em: <https://eips.ethereum.org/EIPS/eip-140>. Citado na p. 27.

BINANCE Smart Chain Average Gas Price Chart. [*S.l.*: *s.n.*]. Acessado em 2022-02-10. Disponível em: <https://bscscan.com/chart/gasprice>. Citado na p. 60.

BNB Smart Chain. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://www.bnbchain.world/>. Citado na p. 58.

BOWE, Sean; GABIZON, Ariel; GREEN, Matthew D. A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK. *In:* SPRINGER. INTERNATIONAL Conference on Financial Cryptography and Data Security. [*S.l.*: *s.n.*], 2018. P. 64–77. Citado na p. 30.

BREIDENBACH, Lorenz *et al.* **AAVE Protocol Whitepaper**. [*S.l.*: *s.n.*], 2021. Disponível em: <https://research.chain.link/whitepaper-v2.pdf>. Citado na p. 65.

BUTERIN, Vitalik. **Ethereum Whitepaper**. [*S.l.*: *s.n.*], 2013. Disponível em: <https://ethereum.org/en/whitepaper/>. Citado nas pp. 16, 24, 26, 58.

BUTERIN, Vitalik. **Merkling in Ethereum | Ethereum Foundation Blog**. [*S.l.*: *s.n.*], nov. 2015. Acessado em 2021-05-01. Disponível em: <https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/>. Citado na p. 23.

BUTERIN, Vitalik *et al.* **EIP-1559: Fee market change for ETH 1.0 chain**. [*S.l.*: *s.n.*], abr. 2019. Acessado em 2022-02-09. Disponível em: <https://eips.ethereum.org/EIPS/eip-1559>. Citado na p. 25.

CATEGORY Referral Fees. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://sell.amazon.com/pricing>. Citado na p. 16.

COMMONS, Wikimedia. **File:Illustration of digital signature.svg — Wikimedia Commons, the free media repository**. [*S.l.*: *s.n.*], 2020. Acessado em 2021-05-02. Disponível em: <https://commons.wikimedia.org/w/index.php?title=File:Illustration_of_digital_signature.svg&oldid=478145952>. Citado na p. 22.

CONDITIONS for sale in AliExpress. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://sell.aliexpress.com/en/__pc/condizioni.htm>. Citado na p. 16.

DECENTRALIZED Dispute Resolution Protocol. [*S.l.*: *s.n.*]. Acessado em 2022-02-09. Disponível em: <https://aragon.org/aragon-court>. Citado nas pp. 16, 37, 41.

DYNAMIC Gas Costs. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://github.com/wolflo/evm-opcodes/blob/main/gas.md>. Citado na p. 37.

ERC 20 Token Standard. [*S.l.*: *s.n.*], 2020. Acessado em 2021-03-24. Disponível em: <https://ethereum.org/en/developers/docs/standards/tokens/erc-20/>. Citado na p. 16.

ETHEREUM Average Gas Price Chart. [*S.l.*: *s.n.*]. Acessado em 2022-02-10. Disponível em: <https://etherscan.io/chart/gasprice>. Citado na p. 58.

ETHEREUM Virtual Machine. [*S.l.*: *s.n.*], 2021. Acessado em 2021-03-24. Disponível em: <https://ethereum.org/en/developers/docs/evm/>. Citado na p. 27.

FANTOM. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://fantom.foundation/>. Citado na p. 58.

FANTOM Average Gas Price Chart. [*S.l.*: *s.n.*]. Acessado em 2022-02-10. Disponível em: <https://ftmscan.com/chart/gasprice>. Citado na p. 60.

FENU, G. *et al.* The ICO phenomenon and its relationships with ethereum smart contract environment. *In:* 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE). [*S.l.*: *s.n.*], 2018. P. 26–32. DOI: 10.1109/IWBOSE.2018.8327568. Citado na p. 28.

GABIZON, Ariel; WILLIAMSON, Zachary J.; CIOBOTARU, Oana. **PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge**. [*S.l.*: *s.n.*], 2019. Cryptology ePrint Archive, Report 2019/953. https://eprint.iacr.org/2019/953. Citado na p. 30.

HARDHAT. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://hardhat.org/>. Citado nas pp. 43, 53.

HYBRID Solutions. [*S.l.*: *s.n.*]. Acessado em 2021-04-12. Disponível em: <https://ethereum.org/en/developers/docs/layer-2-scaling/%5C#hybrid-solutions>. Citado na p. 31.

INÁCIO, Gustavo *et al.* Dagora: Um Mercado Virtual Baseado em Side-Chains. *In:* SBC. ANAIS do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos. [*S.l.*: *s.n.*], 2021. P. 155–168. Citado na p. 18.

JOHNSON, Don; MENEZES, Alfred; VANSTONE, Scott. The elliptic curve digital signature algorithm (ECDSA). **International journal of information security**, Springer, v. 1, n. 1, p. 36–63, 2001. Citado na p. 22.

KANANI, Jaynti; NAILWAL, Sandeep; ARJUN, Anurag. **Matic Whitepaper**. [*S.l.*: *s.n.*], 2019. Acessado em 2021-03-24. Disponível em: <https://github.com/maticnetwork/whitepaper>. Citado nas pp. 29, 31, 58.

KLEMS, Markus *et al.* Trustless Intermediation in Blockchain-Based Decentralized Service Marketplaces. *In:* MAXIMILIEN, Michael *et al.* (Ed.). **Service-Oriented Computing**. Cham: Springer International Publishing, 2017. P. 731–739. Citado na p. 32.

KNOWNORIGIN. [*S.l.*: *s.n.*]. Acessado em 2021-03-24. Disponível em: <https://knownorigin.io/>. Citado na p. 32.

LAYER 2 Scaling. [*S.l.*: *s.n.*], 2021. Acessado em 2021-03-20. Disponível em: <https://ethereum.org/en/developers/docs/layer-2-scaling/>. Citado nas pp. 29, 30.

LESAEGE, Cl´ement; GEORGE, William; AST, Federico. **Kleros**. [*S.l.*: *s.n.*], 2020. Disponível em: <https://kleros.io/static/whitepaper%5C_en-8bd3a0480b45c39899787e17049ded26.pdf>. Citado nas pp. 16, 37, 40.

LOOPRING. [*S.l.*: *s.n.*]. Acessado em 2021-04-12. Disponível em: <https://loopring.org/>. Citado na p. 30.

MOONPAY. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://www.moonpay.com/>. Citado na p. 65.

NAKAMOTO, Satoshi. **Bitcoin: A peer-to-peer electronic cash system**. [*S.l.*: *s.n.*], 2008. Disponível em: <http://www.bitcoin.org/bitcoin.pdf>. Citado nas pp. 15, 22, 23.

NARAYANAN, Arvind *et al.* **Bitcoin and cryptocurrency technologies: a comprehensive introduction**. [*S.l.*]: Princeton University Press, 2016. Citado nas pp. 19, 20.

NEIHEISER, Ray *et al.* HRM smart contracts on the blockchain: emulated vs native. **Cluster Computing**, Springer, p. 1–18, 2020. Citado nas pp. 29, 31.

NON-FUNGIBLE tokens. [*S.l.*: *s.n.*]. Acessado em 2021-03-24. Disponível em: <https://ethereum.org/en/nft/>. Citado na p. 32.

OKUPSKI, Krzysztof. Bitcoin developer reference. *In:* EINDHOVEN. [*S.l.*: *s.n.*], 2014. Citado na p. 25.

OPENSEA. [*S.l.*: *s.n.*]. Acessado em 2021-03-24. Disponível em: <https://opensea.io/>. Citado na p. 32.

OPTIMISM. [*S.l.*: *s.n.*]. Acessado em 2021-04-12. Disponível em: <https://optimism.io/>. Citado nas pp. 30, 58.

OPTIMISM Gas Oracle Contract Code. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://optimistic.etherscan.io/address/0x420000000000000000000000000000000000000F#code>. Citado na p. 59.

ORIGIN Protocol. [*S.l.*: *s.n.*]. Acessado em 2021-03-24. Disponível em: <https://www.originprotocol.com/>. Citado nas pp. 32, 37.

PAYPAL - About Us. [*S.l.*: *s.n.*]. Acessado em 2021-05-13. Disponível em: <https://www.paypal.com/gi/webapps/mpp/about>. Citado na p. 15.

POLYGON PoS Chain Average Gas Price Chart. [*S.l.*: *s.n.*]. Acessado em 2022-02-10. Disponível em: <https://polygonscan.com/chart/gasprice>. Citado na p. 59.

PUBLIC dashboard. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://public-grafana.optimism.io/d/9hkhMxn7z/public-dashboard>. Citado na p. 59.

QUANTO custa vender um produto. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://www.mercadolivre.com.br/ajuda/quanto-custa-vender-um-produto_1338>. Citado na p. 16.

RAMP. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://ramp.network/>. Citado na p. 65.

RANGANTHAN, V. P. *et al.* A Decentralized Marketplace Application on the Ethereum Blockchain. *In:* 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC). [*S.l.*: *s.n.*], 2018. P. 90–97. DOI: 10.1109/CIC.2018.00023. Citado na p. 32.

RIVEST, Ronald L; SHAMIR, Adi; ADLEMAN, Leonard. A method for obtaining digital signatures and public-key cryptosystems. **Communications of the ACM**, ACM New York, NY, USA, v. 21, n. 2, p. 120–126, 1978. Citado na p. 22.

RIVEST, Ronald L; SHAMIR, Adi; ADLEMAN, Leonard. A method for obtaining digital signatures and public-key cryptosystems. **Communications of the ACM**, ACM New York, NY, USA, v. 21, n. 2, p. 120–126, 1978. Citado na p. 22.

ROZEN, Jacob. **The ridiculously high cost of Gas on Ethereum**. [*S.l.*: *s.n.*], 2021. Acessado em 2021-04-12. Disponível em: <https://coingeek.com/the-ridiculously-high-cost-of-gas-on-ethereum>. Citado na p. 29.

SCRIPT - Bitcoin Wiki. [*S.l.*: *s.n.*]. Acessado em 2021-05-04. Disponível em: <https://en.bitcoin.it/wiki/Script>. Citado na p. 25.

SINGH, Amritraj *et al.* Sidechain technologies in blockchain networks: An examination and state-of-the-art review. **Journal of Network and Computer Applications**, v. 149, p. 102471, 2020. ISSN 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2019.102471. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804519303315>. Citado na p. 29.

SNOWTRACE: Avalanche C-Chain Blockchain Average Gas Price Chart. [*S.l.*: *s.n.*]. Acessado em 2022-02-10. Disponível em: <https://snowtrace.io.com/chart/gasprice>. Citado na p. 61.

SZABO, Nick. Formalizing and securing relationships on public networks. **First monday**, 1997. Citado nas pp. 16, 25.

SZYDLO, Michael. Merkle tree traversal in log space and time. *In:* SPRINGER. INTERNATIONAL Conference on the Theory and Applications of Cryptographic Techniques. [*S.l.*: *s.n.*], 2004. P. 541–554. Citado na p. 21.

THEGRAPH. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://thegraph.com/en/>. Citado na p. 34.

TRANSACTION fees on L2. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://community.optimism.io/docs/developers/build/transaction-fees/>. Citado na p. 58.

TRANSAK. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://transak.com/>. Citado na p. 65.

UNDERSTANDING selling fees. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://pages.ebay.com/seller-center/get-started/seller-fees.html>. Citado na p. 16.

VOGELSTELLER, Fabian; BUTERIN, Vitalik. **EIP-20: ERC-20 Token Standard**. [*S.l.*: *s.n.*], nov. 2015. Acessado em 2021-05-07. Disponível em: <https://eips.ethereum.org/EIPS/eip-20>. Citado na p. 28.

WANG, Licheng *et al.* Cryptographic primitives in blockchains. **Journal of Network and Computer Applications**, Elsevier, v. 127, p. 43–58, 2019. Citado na p. 20.

WHAT is the Forth programming language? [*S.l.*: *s.n.*]. Acessado em 2021-05-04. Disponível em: <https://www.forth.com/forth/>. Citado na p. 25.

WOW@AAVE.COM. **AAVE Protocol Whitepaper**. [*S.l.*: *s.n.*], 2020. Disponível em: <https://github.com/aave/protocol-v2/blob/master/aave-v2-whitepaper.pdf>. Citado nas pp. 57, 65.

WYRE. [*S.l.*: *s.n.*]. Acessado em 2022-02-23. Disponível em: <https://www.sendwyre.com/>. Citado na p. 65.

Apêndices

APÊNDICE A – ARTIGO PUBLICADO NO SBRC

A.1 DAGORA: UM MERCADO VIRTUAL BASEADO EM *SIDE-CHAINS*

Resumo: Com o recente crescimento dos preços das criptomoedas, o uso de *block-chains* em diversas aplicações com contratos inteligentes tem se tornado impeditivo devido aos custos dos serviços. Uma das soluções que vem sendo muito aceita é a do uso de *side-chains* para a execução dos *smart contracts*, ficando uma *blockchain* principal (*main-chain*) para a sincronização dos resultados. Os benefícios apontados na literatura indicam enormes vantagens. Porém, a complexidade de mover uma aplicação para um *side-chain* e o ganho em termos de custos não são examinados com o devido rigor na literatura. Neste artigo, apresentamos o *Dagora Market*, um mercado decentralizado de compra e venda de mercadorias construído a partir de contratos inteligentes em *side-chain*. Além de apresentar os aspectos arquiteturais, demonstramos os ganhos nesta abordagem de *side-chain*, principalmente no referente ao custo econômico se comparado com execuções de *smart-contracts* em *main-chain*.

Dagora: Um Mercado Virtual Baseado em *Side-Chains*

Gustavo Inácio[1], Luciana Rech[1], Ray Neiheiser[2], Joni Fraga[2]

[1] Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)

[2]Departamento de Automação e Sistemas – Universidade Federal de Santa Catarina (UFSC)

Abstract. *The price increase resulting from the growing popularity of the blockchain technology, created numerous issues for decentralized applications. An increasing popular solution is running side-chains next to a popular blockchain, offering vast cost advantages. While the security impacts of these solutions are well understood, the complexity of moving an application to a side-chain and the cost advantages have received little attention in the literature. In this paper we analyze both the complexity as well as the cost advantages using a decentralized market application called Dagora Market as an example. We show that it is relatively simple to move an application to a side-chain resulting in a cost advantage of up to 6 orders of magnitude compared to running it on the main-chain.*

Resumo. *Com o recente crescimento dos preços das criptomoedas, o uso de blockchains em diversas aplicações com contratos inteligentes tem se tornado impeditivo devido aos custos dos serviços. Uma das soluções que vem sendo muito aceita é o uso de side-chains para a execução dos smart contracts, ficando uma blockchain principal (main-chain) para a sincronização dos resultados. Os benefícios apontados na literatura indicam grandes vantagens. Porém, a complexidade em mover uma aplicação para um side-chain e o ganho em termos de custos não são examinados com o devido rigor na literatura. Neste artigo, apresentamos o Dagora Market, um mercado descentralizado de compra e venda de mercadorias construído a partir de contratos inteligentes em side-chain. Além de apresentar os aspectos arquiteturais, demonstramos os ganhos nesta abordagem de side-chain, principalmente no referente ao custo econômico se comparado com execuções de smart-contracts em main-chain.*

1. Introdução

A grande aceitação de *ICOs* (*Initial Coin Offerings*) [Fenu et al. 2018] nos últimos anos, deu origem a uma quantidade surpreendente de aplicações baseadas em *blockchains* ao redor do mundo. Estes projetos estiveram sempre na promoção da descentralização, com o objetivo de colocar o controle das operações bem próximo aos usuários. Deste modo, afastando sempre que possível o controle das *Big Techs* (como *Amazon*, *Microsoft*, etc.).

O uso do conceito de contratos inteligentes (*smart contract*) contribui muito para o controle descentralizado (*smart contracts*[1] [Szabo 1997]) disponíveis em *blockchains*

[1]Um *smart contract* [Szabo 1997] nada mais é do que a formação de contratos digitais firmados por partes que expressam a vontade de realizar um determinado negócio. Este contrato é igual a contratos do mundo real, com a diferença de ser digital. Os *smart contracts* são auto-executáveis por fazerem uso de *blockchain*. Um contrato inteligente pode ser descrito como uma máquina de estado replicada e disposta em todos os nós de um *ledger* distribuído, podendo ser invocada por clientes e mesmo outros *smart contracts*.

e usados em muitas aplicações. São exemplos: o *ERC-20 token* fornece uma espécie de criptomoeda, permitindo qualquer pessoa criar a sua criptomoeda ao lançar seu *token* em uma *blockchain* [eth 2021a]; o *Uniswap* é uma bolsa de criptomoedas descentralizada que permite a troca de diferentes tipos de *tokens* (criptomoedas) [Adams et al. 2021] e o *Kleros* é um sistema para resolver disputas de forma descentralizada [Lesaege et al. 2020].

O *ledger Ethereum*, é um dos ecossistemas de *blockchain* mais valiosos [coi 2021]. Entretanto, o crescimento de interesse nestes *ledgers* distribuídos gerou como consequência um forte aumento no preço das criptomoedas dos mesmos, tornando seu uso mais restritivo. No *Ethereum* [Buterin 2013], cada operação gera um custo em *"gas"*. Neste caso, cada transação tem que conter *gas* suficiente para cobrir o gasto de sua execução. O preço do *gas* depende da carga da rede (*ledger*) e é medido em *Gwei* [eth 2021a]. Altos custos provocam muitas vezes a inviabilidade de muitas aplicações, principalmente as menores e menos lucrativas [Foxley 2021].

Para resolver o problema na execução de contratos inteligentes, muitas soluções foram propostas na literatura. Estas soluções têm sido identificadas como *layer-2-solutions*, ficando o *ledger* principal (o *Ethereum*, por exemplo) denotado como *layer-1*. Em síntese, as soluções de nível 2 são construídas em um servidor centralizado ou em outra *blockchain* (*side-chain*) para a execução de contratos inteligentes. Neste caso, uma transação é recebida em nível 2, sendo então validada e executada no contrato inteligente. Depois, o *layer-2* pode fazer persistir os resultados de um *batch* de transações no *layer-1* podendo oferecer as mesmas garantias que a *blockchain* principal [eth 2021a].

A execução dos códigos de contratos em nível 2 usando servidores centralizados quase sempre diminui o custo significantemente quando comparado com execuções em qualquer *blockchain* de *layer-1*. Porém, para isso funcionar, é necessário um certo nível de responsabilidade do desenvolvedor de contratos. Um exemplo de contratos inteligentes em nível 2 baseado em servidor é apresentado em [Neiheiser et al. 2020] descentralizando concursos públicos. Neste caso, todas as informações são publicadas pelo servidor em *blockchain* (nível 1) para que os envolvidos possam confirmar a regularidade e a transparência do processo. Na ocorrência de irregularidades, a instituição pode ser processada. Porém, este tipo de abordagem não é adequado para todos os casos possíveis de aplicações. Em alguns casos, como o de um mercado fornecedor de bens onde as compras podem estar espalhadas pelo mundo inteiro, a centralização da camada de nível 2 em um servidor se torna inviável. Em qualquer malversação do vendedor fora da jurisdição do país do comprador impossibilitaria possíveis ações jurídicas.

Por outro lado, as soluções de *side-chains* colocam as execuções de nível 2 dos contratos inteligentes em uma outra *blockchain* (a *side-chain*). Os resultados são geralmente sincronizados, e tornam-se persistentes na *blockchain* principal (*main-chain*). Este tipo de solução, comparada a do servidor, apresenta transparência e maior segurança, podendo ser aplicada em vários tipos de aplicação [eth 2021a].

A escolha de uma *blockchain* como *main-chain* deve ser guiada por objetivos bem definidos. Como existem muitas *blockchains* diferentes, numa aplicação de mercado, o desenvolvedor da aplicação se apoiará na vantagem de usar *blockchains* consagradas e bem difundidas. Pois, terá disponível a comunidade (possivelmente grande) já existente na plataforma, além de ter garantias que a *blockchain* continuará efetiva muitos anos.

Neste artigo é apresentado um mercado descentralizado, o *Dagora Market*, onde suas funções são implementadas a partir de contratos inteligentes de *side-chain.* Esta aplicação foi também objeto de estudo com o objetivo de verificar a complexidade em mover contratos para uma *side-chain* e analisar o ganho com relação aos custos neste tipo de solução. O *Ethereum* foi escolhido como *main-chain* para o *Dagora Market*. Os contratos inteligentes foram executados em uma *side-chain* (*Polygon* [Kanani et al. 2019]).

Na quantificação de nossos resultados, analisamos os custos de execução dos contratos inteligentes tanto no uso do *side-chain Polygon*, como a execução direta no *Ethereum*. Seguindo isso, discutimos a complexidade de se aplicar estes contratos no *Polygon* em relação à implementação destes diretamente no *Ethereum*. Ou seja, tratamos de dar uma medida de impacto da aplicação ao utilizar *side-chains*.

O artigo apresenta uma revisão de trabalhos relacionados na seção 2, apontando uma classificação que considera a distribuição da *layer-2* e as abordagens usadas na implementação desta camada. Na seção 3, a funcionalidade do *Dagora Market* é descrita. Na seção 4, são mostradas as análises dos resultados dos contratos *Dagora* sendo executados em *side-chain* e em *main-chain*. Por fim, a seção 5 apresenta as considerações gerais dos resultados e comparações com a literatura.

2. Trabalhos relacionados

O *Bitcoin* [Nakamoto 2008] foi introduzido com a intenção de criar uma moeda eletrônica *peer-to-peer* sem passar pela coordenação ou centralização de uma instituição financeira. No núcleo desta rede *P2P*, está uma *blockchain* que corresponde a um armazenamento imutável. A *blockchain* é uma cadeia de blocos replicada em diferentes nós da rede, onde cada bloco contém o *hash* do bloco anterior. Desta forma, um novo bloco ao ser inserido torna persistente o bloco anterior [Nakamoto 2008].

Para a consistência desta rede, considerando a replicação da cadeia de blocos, é necessário um protocolo de consenso que garanta a sincronização das réplicas (ou seja, réplicas com o mesmo estado). No caso do *Bitcoin*, esta sincronização é feita pelo algoritmo *Proof of Work* (PoW) onde cada nó para colocar um bloco na rede tem que resolver um problema matemático bem definido. O primeiro nó que consegue pode propor o seu bloco como o seguinte na *blockchain*. Os outros participantes verificam se este bloco é válido, e só então o mesmo é tornado persistente [Nakamoto 2008]. A combinação da *blockchain* com o protocolo de consenso é chamado de *ledger* (ou *ledger* distribuído). Além do *Bitcoin*, foram criados centenas de diversos *ledgers* [Bach et al. 2018].

Com essa tecnologia em mente, foi proposto por *Vitalik Buterin* [Buterin 2013], a utilização de *ledgers* distribuídos como plataforma para aplicações descentralizadas permitindo a execução de códigos, dando origem ao conceito dos contratos inteligentes. Este conceito também norteou o projeto do *Ethereum*. No caso deste *ledger*, para garantir que todos os nós na rede obtenham o mesmo resultado de códigos replicados, é usada a *Ethereum Virtual Machine* (EVM) [eth 2021a] e a linguagem *Solidity* [sol 2021]. Com ajuda de contratos inteligentes foram criadas também moedas adicionais na rede *Ethereum*, dando início a muitos projetos. Executar aplicações no *Ethereum* tem vantagens como a de ser um considerável ecossistema com um grande potencial de usuários. Mas, com o crescente uso do *Ethereum*, torna-se muito dispendioso [Rozen 2021].

2.1. Disposição e Abordagens para Execuções de Camada-2.

Soluções de *layer-2* foram propostas para contornar o problema destes custos de escalabilidade. Muitas destas soluções são baseadas em servidores externos que executam as transações de clientes e depois publicam os respectivos resultados na *blockchain*. Desta maneira, qualquer interessado pode verificar a validade dos resultados [eth 2021a]. Um requisito para estas soluções é a existência de um depósito de segurança por parte dos operadores do servidor de modo a ressarcir clientes que tenham sido prejudicados por alguma ação errônea ou desonesta do serviço prestado. Esta abordagem é usada quando os agentes do serviço envolvem grandes empresas ou órgãos públicos. [Neiheiser et al. 2020].

Entidades com recursos não tão significativos (pequenos empreendedores ou clientes comuns) normalmente fazem uso de plataformas distribuídas para execuções de nível 2. Como isto, o uso de *side-chains* na sincronização em *main-chains* vem se tornando uma alternativa muito consistente em relação a servidores centralizados. *Side-chains* começaram a ser usadas com a intenção de executar *smart contracts* em nível 2, deixando a *main-chain* para a persistências dos resultados [Singh et al. 2020]. As *side-chains* podem apresentar seus próprios modelos de consenso e, portanto, serem independentes da *main-chain* (o *Ethereum*, por exemplo) [eth 2021a]. A *Polygon* tem sido uma destas *side-chains*, usando a mesma maquina virtual como o *Ethereum* (EVM) e desta forma oferecendo suporte para contratos inteligentes.

Além deste aspecto da forma de disposição de soluções de *layer-2* (seja em servidores ou em outros *ledgers*), surgiram abordagens de como executar estas funções de camada-2. Muitas das execuções adotam as abordagens identificadas como *rollups* que se apresentam em duas categorias: os *rollups* otimistas e os *zero knowledge (zkrollups)*.

Os *rollups* otimistas correspondem a serviços de segundo nível (*layer-2*) com uma certa quantia de criptomoedas em um contrato inteligente na *main-chain* (*layer-1*) [eth 2021a]. Estes serviços, possuem "agregadores" que executem as transações de seus usuários e publicam *batches* dos resultados compactados em *smart contract* na *layer-1*. Um *batch* inserido na *main-chain* pode ser verificado por um cliente ou verificador externo. A verificação envolve a reexecução das transações do *batch* comparando os resultados com os que persistiram na *main-chain*. Se não houver um *match*, o verificador correspondente denuncia o agregador via o contrato inteligente na *layer-1*. Se o *batch* se confirmar errôneo, o agregador perde o depósito de segurança e o verificador recebe uma parte pela detecção da malversação. Em resumo, estes *optimistic rollups* tornam, no melhor caso, as transações menos custosas para serem executadas na *layer-2*. Porém, existe um *trade-off* entre a latência na finalização e o custo de transações.

Os *rollups* identificados como *Zero knowledge (zkrollups)* são similares ao caso anterior, funcionando em *layer-2*, acumulando transações executadas fora da *main-chain* e publicando na camada 1 a correspondente prova criptográfica (*SNARKs* [Bowe et al. 2018]). Estas *ZK proofs* comparam o estado (*snapshot*) da *blockchain* antes das execuções das transações com o estado da *blockchain* após as execuções (ou seja, os valores antes e depois das execuções) e relata apenas as mudanças para a *main-chain* em um *hash* verificável. Os *zkrollups* cumprem a finalidade de rápido desempenho nas verificações, porém necessitam de poder de processamento maior.

Uma outra abordagem presente na literatura, identificada como *State-Channels*,

faz uso de recursos de clientes na execução de camada-2 [sta 2021]. Neste caso, dois clientes, ou um cliente e uma aplicação se comunicam por fora da *blockchain*. Com as trocas finalizadas, ambos mandam a história das trocas (os resultados e transações) para um *smart contract* na camada 1 para a persistência dos mesmos. Desta forma, a rede do *Ethereum* fica reduzida a um mínimo de operações e o cliente só paga taxas de transferência.

Side-chains conectadas a uma *main-chain*, constituem outro tipo de abordagem para a execução da *layer-2*. Contratos inteligentes determinam as trocas fáceis de ativos entre das duas *blockchains*. *Checkpoints* regulares das execuções de nível 2 são publicados regularmente com o *hash* do *Merkel Tree Root* de *Batches* destas transações na *main-chain*. Estas cadeias laterais devem ter o seu próprio modelo de confiança, pois os fundos transferidos para a *side-chain* usualmente não são protegidos pelo modelo de confiança do *Ethereum*. *Side-chains* que usam provas de fraudes, são identificados como *Plasma*. A rede *Polygon* é um *side-chain* que implementa este tipo de modelo.

A tabela 1 resume as discussões sobre as abordagens e como são dispostas as funcionalidades com contratos em nível 2, citando nas entradas projetos exemplos.

Abordagens de *Layer-2*		Posicionamento da *Layer-2*		
		Servidor centralizado	Decentralizados	Recurso de Cliente
Rollups	Optimistic	Arbitrum [arb 2021]	Optimism [opt 2021]	
	Zero Knowledge	Loopring [loo 2021]	zkSync	
Side-chain	Plasma		Polygon	
	Puro		Skale	
State-channels				kChannels [kch 2021]

Tabela 1. Abordagens e Disposições de Contratos em Nível 2

2.2. Mercados Virtuais de Bens

No contexto de mercados virtuais, existem diversos trabalhos focando a troca de bens (digitais). São exemplos de mercados *Non Fungible Tokens* (NFT) [wha 2021] o *OpenSea* [ope 2021]. Todos os citados fazem uso da descentralização baseados em contratos do *Ethereum* e sem adotar a *layer-2*. Uma alternativa para produtos da vida real é o *Origin-Protocol* [ori 2021] que tambem usa *smart contracts* do *Ethereum* e funciona como um *backend* descentralizado para a criação de *webshops* pessoais.

Em relação a trabalhos acadêmicos, recentemente surgiram diversos *market places* fazendo uso de *blockchains*. O DESEMA [Klems et al. 2017] é um destes trabalhos onde um protótipo de um mercado descentralizado é parcialmente desenvolvido. No DESEMA, todos os serviços são baseados em *smart contracts*, porém não foram implementados por completo. Portanto, seus custos em uma *blockchain* não têm como serem avaliados.

O trabalho em [Ranganthan et al. 2018] propõe um aplicativo descentralizado de mercado virtual construído sobre o *Ethereum*. Este mercado é baseado somente em contratos inteligentes do *Ethereum* (não usa a definição de *layer-2*). Este trabalho faz um estudo comparativo de suas proposições com mercados estabelecidos que não usam *blockchains*. A base deste estudo são os custos da aplicação proposta na *blockchain* usada. Além disto, este trabalho descreve as margens de lucro de um vendedor que tem seus produtos no *eBay* e o *Sotheby's*, comparando estas margens com as obtidas quando o mesmo vendedor apresenta seus produtos no mercado proposto sobre o *Ethereum*. O *eBay* e o

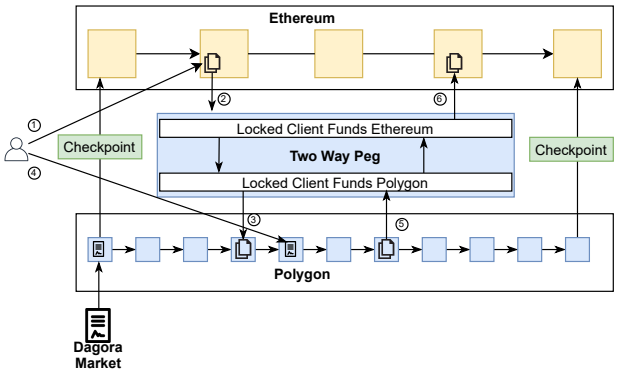
Sotheby's são sites que não fazem uso de *blockchains* e cobram dos vendedores taxas sobre os produtos vendidos que se aproximam de 10% do valor obtido. O texto mostra que se usado mercado baseado no *Ethereum*, o vendedor teria mais receita em produtos mais caros e o custo para o comprador aumentaria em alguns centavos. Com isto, os autores reiteram que por ter uma margem maior de lucro, produtos com valores altos poderiam ser vendidos mais baratos no *blockchain*. Entretanto, não é falado no texto que tudo isso depende do preço de *gas* e do valor do *token* (considerando o *Ethereum*). Comparado com a época em que o artigo foi escrito, o custo de execução hoje em dia seria 75 vezes maior (de 0.86 centavos no artigo para 64.50$ nos dias de hoje) tornando a proposta, independente do produto, não competitiva.

3. Dagora Market

Neste trabalho, usamos a aplicação de um *marketplace* descentralizado, denominado *Dagora Market* para explorar os custos de uso de uma composição *side-chain/main-chain*. O *Dagora* conecta diretamente de forma descentralizado vendedores a compradores, utilizando criptomoedas como meio de pagamento e contratos inteligentes nestas conexões. Desta forma é possível anunciar produtos assim como administrar anúncios existentes através de contrato inteligente no *Polygon* referente ao *Dagora Market*. Informações sobre anúncios existentes podem ser obtidas com *queries* ao histórico da *blockchain Polygon*. O *Dagora Market* é apresentado nos seus aspectos arquiteturais e funcionais nesta seção.

3.1. Arquitetura

Figura 1. System Archicecture



A arquitetura do sistema proposto é mostrada na Figura 1. O *Polygon* é apresentado como cadeia lateral e o *Ethereum* como a cadeia principal. Portanto, a aplicação de mercado é implementada como *layer-2* no *Polygon*, na forma de contratos inteligentes. Neste arranjo *side-chain/main-chain*, o *Polygon* e o *Ethereum* apresentam um procedimento identificado como *two-way peg* (Figura 1) para as trocas seguras de fundos (criptomoedas) e pontos de verificação regulares (*checkpointing*). O *two-way peg* envolve as seguintes trocas:

1. Um cliente envia fundos para um endereço especifico no *Ethereum*;
2. Os fundos ficam então bloqueados neste contrato do *Ethereum*;

3. Os fundos são criados (*minted*) na cadeia lateral *Polygon*;
4. O cliente interage com o mercado no *Polygon* despendendo numa compra, além do custo do produto, um valor pela transação significativamente reduzido. E, uma vez terminada a transação, os fundos restantes podem ser transferidos de volta para a cadeia principal, bloqueando os mesmos em um contrato inteligente no *Polygon*;
5. Os fundos restantes são transferidos e desbloqueados no Ethereum. Estes fundos são destruídos na *side-chain Polygon*.

A entidade do *Dagora* que recebe fundos do *Ethereum* via o *Polygon* é um contrato inteligente e foi nomeada Gerente de Fundos. No passo 4 do procedimento acima, a interação do cliente acontece com o contrato inteligente que chamamos de Gerente de Compras.

O *Polygon* oferece duas maneiras de manter o *two-way peg* confiável que diferem significativamente ao retirar *tokens* para o *Ethereum*. A primeira é através de um quórum *PoS* de validadores do *Polygon*, permitindo então que o contrato inteligente no *Ethereum* possa liberar os *tokens* (os fundos) ao cliente em aproximadamente 10-30 minutos. A segunda maneira de retirar *tokens* do *Polygon* e desbloqueá-los no *Ethereum* é através de uma prova de fraude (*Plasma*) que indica que os fundos foram enviados. Esta prova é enviada para um contrato no *Ethereum*, a ser verificado em até 7 dias. Após este período os *tokens* estarão desbloqueados no *Ethereum*. Embora a solução de quórum *PoS* seja mais rápida, ela é menos segura uma vez que requer confiança nos validadores do *Polygon*.

Os *checkpoints* mostrados na Figura 1 contém *Merkel Root Hash* de todos os blocos desde o último *checkpoint* bem como *hash* de um *batch* de transações. As transferências destes *checkpoints* são dirigidas a um contrato no *Ethereum* que garantem a imutabilidade do estado do *Polygon*. Estes *checkpoints*, portanto, são garantias do aplicativo de mercado aos vendedores e compradores contra os possíveis riscos em suas transações.

3.2. Funcionamento

A honestidade dos participantes é garantida por incentivos baseados na teoria de jogos[2] [Liu et al. 2019]. Para esse fim, utilizamos *tokens ERC-20* [eth 2021a] que denotamos no texto como *DGR*. Um vendedor para anunciar um produto necessita adquirir *tokens DGRs* e fazer um depósito de segurança (*stake*) de uma quantidade mínima[3] no contrato do mercado (Gerente de Compras). Esta quantia pode ser considerada um depósito de segurança para garantir a honestidade do mesmo.

Com o *stake* realizado, os vendedores podem anunciar um produto. Compradores poderão buscar produtos na plataforma usando palavras-chave, categorias, etc. A ordem em que os anúncios são mostrados é proporcional à quantidade que possuem de *tokens* que estão em seus *stakes*, quanto mais *DGRs*, maior a chance do anúncio aparecer no topo da página de busca. Desta forma, vendedores que investem em um maior depósito de segurança, podem ser considerados mais confiáveis. O Gerente de Fundos emite *tokens DGR* que são usados como principal meio de troca entre compradores e vendedores. Uma característica importante desses *tokens* (ERC-20) é que, além de permitir a realização de

[2]A teoria de jogos, no contexto de *blockchain*, trata de incentivos monetárias com o fim de criar um ambiente onde a participação honesta de um usuário seja a mais vantajosa.

[3]Esta quantitade mínima poderia ser fixa, definida anteriormente, ou pode ser ajustada por uma organização autônoma descentralizada (*DAO*). Porém este processo está fora do escopo deste trabalho

transferências de fundos entre participantes, estes também autorizam contratos inteligentes a realizarem transferências de uma quantidade pré-definida de seus fundos.

3.2.1. Interações entre os Participantes de uma transação comercial

A Figura 2 mostra o exemplo de uma compra. Dois tipos de interações são mostrados na figura. No primeiro, temos as setas contínuas que são usadas nas identificações de chamadas de operações nos contratos gerentes. Estas chamadas são interceptadas pelo quórum de validadores do *Polygon* para as verificações de procedência e de validade, sendo depois executadas nos contratos de destino. As setas tracejadas na figura 2, correspondem a obtenção de notificações via detecção na blockchain *Polygon*.

Figura 2. Fluxo otimista

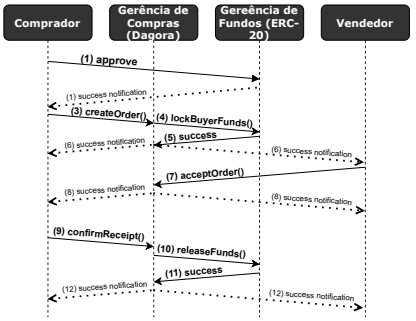
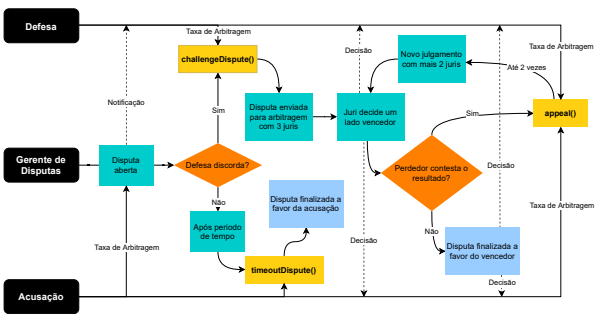


Figura 3. Fluxo de Disputa



O fluxo de interações mostrado na Figura 2 começa inicialmente com um comprador autorizando (através de chamada) o Gerente de Fundos a usar quantia em *tokens DGR* (passo 1). O Gerente de fundos então insere a operação executada (*approve()*) e informações do resultado na *blockchain* (passo 2). Ao detectar (na *blockchain*) a disposição de fundos, o comprador executa a chamada *createOrder()*, mas desta vez dirigida ao contrato Gerente de Compras (passo 3). Na execução desta operação, este contrato faz a chamada *lockBuyerFunds()* no Gerente de Fundos para bloquear fundos do comprador (passo 4). A resposta positiva ou negativa do Gerente de Fundos aparece imediatamente no Gerente de Compras por ser uma chamada entre contratos (passo 5).

Tendo fundos suficientes, o Gerente de Compras coloca a operação *createOrder()* com suas informações na *blockchain* (passo 6). O comprador e o vendedor detectam esta ordem de compra na *blockchain*. O vendedor estando de acordo com a venda faz a chamada *acceptOrder()* no Gerente de Compras (passo 7) que, por sua vez, publica na *blockchain* a operação (*acceptOrder()*) e informações correspondentes (passo 8). Ambos, comprador e vendedor detectam na *blockchain* a ordem aceita e então devem trocar detalhes da entrega do produto por fora do *Dagora* no sentido de manter suas privacidades. O comprador ao receber o produto deve na sequência executar a chamada *confirmReceipt()* no Gerente de Compras (passo 9). Como consequência este contrato invoca *releaseFunds()* no Gerente de Fundos (passo 10). Este último gerente então publica a operação de transferência de fundos para a conta do vendedor na *blockchain*. A partir deste momento os participantes sabem que a transação está concluída.

3.2.2. Resolução de Disputas

Disputas são uma parte importante da plataforma, neste sentido, são definidos meios para resolver conflitos. No *Dagora Market* existem dois tipos de conflitos: denúncia de anúncios irregulares e disputas entre clientes e vendedores. Se um produto anunciado violar as regras, o seu proponente pode ser denunciado por qualquer participante do *Dagora* desde que este último possua uma quantidade mínima de *stake* na plataforma. O anúncio correspondente é então pausado e uma disputa é criada. No segundo tipo de conflito, caso o comprador não receba o produto em período de tempo pré-acordado ou entenda que o produto é não satisfatório, ele pode negociar um reembolso com o vendedor. Se não for atendido, o comprador pode abrir uma disputa. Se a devolução for solicitada, o vendedor tem o mesmo período de confirmação para aceitar a devolução ou, sentindo-se prejudicado, pode abrir uma disputa. Todas as situações de disputas citadas acima são levadas a um júri que irá julgar baseando-se nas regras pré-definidas no *Dagora*, na descrição do produto e nas evidências colocadas pelas duas partes (a acusação e a defesa).

Uma disputa é iniciada por um dos participantes (comprador, vendedor ou terceiros) através do envio da taxa de arbitragem para o contrato inteligente do *Dagora* que chamamos de Gerente de Disputa. O fluxo de disputas é mostrado na Figura 3. Este diagrama de fluxo é autoexplicativo, diante disto nos limitamos a pequenos comentários. No teste "Defesa discorda?", é oferecida a possibilidade da defesa assumir a culpa pela saída do arco "Não", ao esperar o *time-out*. Como resultado a acusação ganha a causa e recebe a devolução da sua taxa[4]. Se a defesa discorda então ela paga uma taxa de arbitragem e um júri de três participantes é montado. Este júri decide e no teste "Perdedor contesta o resultado?" no arco "Sim" existe a possibilidade de apelo sobre a decisão onde são requisitados mais dois juízes para arbitrar a disputa. A seleção do júri é feita de forma aleatória sobre participantes da plataforma de disputas, porém a chance de escolha é fortemente influenciada pela quantidade de *tokens* em *stake* dos participantes. O Gerente de Disputa oferece certos incentivos aos jurados de modo que estes façam um julgamento honesto. Um jurado que decida diferente da maioria pode, por exemplo, perder *tokens* na plataforma. Esta seleção não precisa de um júri altamente qualificado, pois existem *guidelines* previamente definidos. Do mesmo modo, verificar o rastreamento de um produto também pode ser considerado uma tarefa simples. Para Gerente de Disputa do *Dagora* foi usada a plataforma Kleros [Lesaege et al. 2020].

4. Avaliação dos Resultados

Nós criamos os *smart contract* da aplicação, em *Solidity*, com aproximadamente 1100 linhas de código, utilizando *Truffle Suite* [tru 2021], onde implementamos funções necessárias para executar todos os possíveis fluxos de interação entre compradores e vendedores. Entre as funções estão: *stake* e *unstake* de *tokens*, *create, read, update, delete* (CRUD) de anúncios, a criação, execução e finalização de ordens, levantamento e resolução de disputas e denúncia de anúncios.

Estes fluxos, subdividimos em 4 categorias: Fluxo "Bem-Sucedido" onde compras são finalizadas e o dinheiro é enviado para o vendedor; Fluxo "Mal Sucedido" onde o

[4]Uma taxa de arbitragem é cobrada para pagar o processo de disputa, a taxa do lado vencedor é devolvida no fim do processo (resultando no pagamento do processo por parte do perdedor)

dinheiro é devolvido ao comprador; Fluxo "Disputa" onde ocorre algum problema e uma das partes sinaliza uma discordância e o Fluxo de "Denúncia" onde alguém emite um anúncio de comportamento ilegal segundo as regras definidas.

Para nossa avaliação, a *main-chain* é a *Ethereum* que utiliza como *token* principal o *Ether* (ETH) e executa *smart-contracts* através da *Ethereum Virtual Machine* (EVM). A blockchain *Polygon*, considerada como a maior *side-chain* do Ethereum, é capaz de executar contratos inteligentes. Além disso, também usa a *EVM* nas execuções de seus contratos, ou seja, é fácil fazer a portabilidade de código do *Ethereum* para o *Polygon* e vice-versa. Portanto, fazer uma comparação entre estas blockchains nas execuções é praticamente imediata. A *Polygon* utiliza *MATIC* como *token* principal para pagar taxas na rede. Devemos ressaltar que podem existir possíveis custos ao depositar *tokens* do *Ethereum* para a *Polygon*, porém esses custos não foram mapeados, pois, como descrito na seção anterior, podem existir facilidades de venda desses *tokens* por cartão de crédito, bem como o depósito de corretoras de criptomoedas diretamente em *side-chains*.

Figura 4. Uso de *gas* médio por subgrupo de fluxos

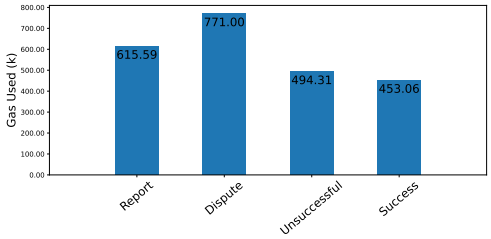
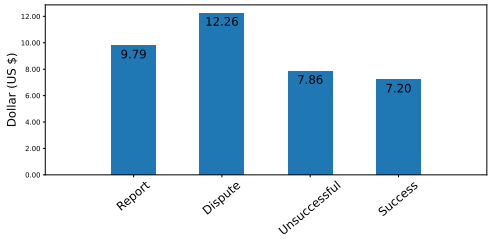


Figura 5. Custo em dólares no *Ethereum*



O custo de execução em *blockchain* depende do total de informação que foi enviada e salva na *blockchain*. Tendo isso em vista, para avaliação de custos foram executados todos os fluxos de interação 10 vezes no *Ganache*[tru 2021] (uma instância de *blockchain* local do *Ethereum*) com diferentes parâmetros de anúncios para calcular o custo médio de execução em *gas* de cada fluxo. Resultando no custo médio de execução para cada subcategoria como representado na Figura 4.

Considerando que ambos, *Ethereum* e o *Polygon* utilizam a *EVM* para execução dos contratos, o custo de *gas* é igual. Devido a isso, o custo final depende de dois fatores: o preço do *gas*, para saber o custo em *tokens*, e o preço do *token*, para saber o custo em moeda fiduciária. Utilizamos a seguinte fórmula para conseguir o preço em dólares:

(custo em *gas* * preço do *gas* em *tokens* * preço do *token* em dólares) / 10^{18}.[5]

Para obter o preço de execução no *Ethereum* em dólares, buscamos o preço do *gas* mediano dos últimos 200 dias[6] resultando em 94.27 GWEI [eth 2021b] ($94.27 * 10^9 wei$). Para buscar o preço em *Ether*, a fim de evitar flutuações, utilizamos a média dos últimos 25 dias chegando ao valor US$1686.31[7]. Obtemos com isso os resultados na Figura 5.

Similarmente, para calcular o preço de execução na *Polygon* em dólares, utilizamos o preço do *gas* como 1 *GWEI* ($1 * 10^9 wei$) que é considerado pela própria rede um

[5]Convertido para a unidade básica do *token* (1 Ether = 10^{18} *wei*)
[6]Preço de *gas* obtido dia 05/03/2021
[7]Preços de *tokens* obtidos dia 17/03/2021

Figura 6. Custo em dolares por subgrupo de fluxos na *Polygon*

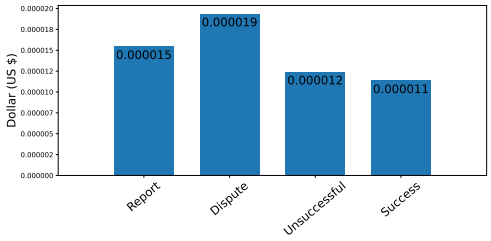
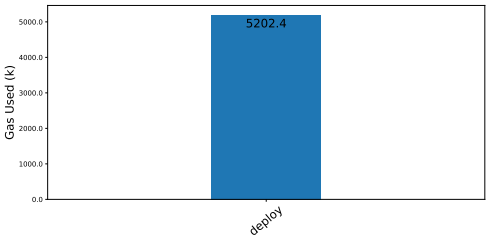


Figura 7. Custo de implantação. *Ethereum* - US$827.02 — *Polygon* - US$0.001308



preço padrão [mat 2021] levando em consideração as últimas 500 transações. Para buscar o preço em *MATIC*, a fim de evitar flutuações, utilizamos a média dos últimos 25 dias que resultou em US$0.25149. A Figura 6 demonstra os valores finais obtidos.

Com os resultados finais em dólares das duas abordagens, pode-se perceber que o custo de executar contratos inteligentes na *Polygon* é cerca de 6 ordens de magnitude menor. Também foram realizados testes para obter o custo da implantação do contrato (Figura 7). Considerando o custo de implantação no *Ethereum* e *Polygon*, o custo de um centavo de dólares no *Polygon* é ínfimo comparado ao custo de US$827 no *Ethereum*.

5. Considerações Gerais

Como mostramos na seção 2.2, existem poucos trabalhos sobre mercados decentralizados baseados em *blockchains*. Dos citados, todos executam diretamente contratos inteligentes no *Ethereum*, sem o uso de camada-2. Poucos destes trabalhos apresentam análise dos custos de uso da *blockchain Ethereum* que consideramos insipientes. Alguns trabalhos acadêmicos [Klems et al. 2017, Ranganthan et al. 2018] apresentam alguma forma de avaliação, porém é importante salientar que estes não foram implementados por completo e as medições apresentadas estão fundamentadas em avaliações já defasadas no tempo, considerando-se os custos do *Ethereum* atualmente.

Neste artigo, apresentamos nosso trabalho de mercado virtual baseado em *side-chain*, um dos objetivos de nossas análises é verificar execuções em *side-chain* com comparações das mesmas execuções no *Ethereum*, pois com isto estaremos relacionando o *Dagora* com os trabalhos de mercado apresentados na literatura que desconsideram processamentos de camada-2).

Existem diversos trabalhos que não tratam mercados virtuais, mas apresentam análise comparativa entre execuções de contratos da aplicação em *side-chains* com execuções realizadas diretamente em *main-chains*, misturando as camadas 1 e 2. Porém estes estudos se limitam em análises qualitativas como o apresentado em [Singh et al. 2020]. Na verdade, uma análise quantitativa e de complexidade de portabilidade da aplicação entre as blockchains é necessária para assegurar os possíveis ganhos nas escolhas de abordagens.

A aplicação de mercado descentralizado apresenta um caso de uso interessante, atualmente grandes mercados online como *Amazon*, *eBay*, MercadoLivre são sites que

centralizam as buscas de compradores e vendedores anunciam seus produtos. Para os vendedores, manter seus anúncios por sites próprios não dá a mesma visibilidade, por esta razão, vendedores profissionais são coagidos a anunciar seus produtos nestas plataformas, sendo que estes *marketplaces* centralizados cobram taxas entre 10% e 15%.

Além disso, em muitas das plataformas, para poder competir com outros vendedores profissionais, os mesmos devem pagar taxas adicionais para ganhar visibilidade através de planos *premium*, e/ou pagar por pacotes para impulsionar seus anúncios. Também existem taxas cobradas pelos sistemas de pagamento, normalmente mediado pela própria plataforma, estes valores variam entre 1% e 5%, dependendo do prazo para o vendedor receber o dinheiro e a forma de pagamento. A união destes fatores citados resulta em quase 20% do preço do produto, sem considerar os impostos. Dessa maneira, perdem ambos, consumidores pagando um preço maior, e vendedores com reduzida faixa de lucro. Todos esses custos poderiam ser reduzidos ao custo de processamento de pagamentos, através de uma plataforma descentralizada, onde vendedores e compradores podem se encontrar sem precisar de uma empresa intermediária.

Com uma possível adoção de mercados descentralizados, alguns problemas são evidentes. Por exemplo, o uso de *tokens* para pagamentos substituindo moeda fiduciária. Hoje, o processo de conversão acontece através de *exchanges* e funciona da seguinte forma: a moeda é depositada na conta da corretora, convertida, e por fim transferida para uma carteira própria na rede escolhida. Além disso, existem alguns serviços, como o *Moonpay*[moo 2021] e *Wyre*[wyr 2021], que permitem o pagamento por cartão de crédito para depósito direto em carteiras de criptomoedas, porém resultando em taxas maiores.

Atualmente não existem corretoras e/ou serviços que depositam diretamente em *side-chains* como a *Polygon*. Resultando em um passo adicional para se obter *tokens* na carteira na *side-chain*. Sendo necessário primeiramente depositar no *Ethereum* para depois transferir para o *Polygon*. Como já mencionado, executar uma transferência no *Ethereum* exige um custo alto, apesar de ser significantemente menor do que realizar a execução do contrato completo no mesmo. Este custo adicional não foi considerado na avaliação, visto que, como mencionado anteriormente, o criador do *marketplace* pode oferecer um *gateway* de transferência de dinheiro (*paypal*, cartão de crédito, etc) para *tokens* DGR no *Polygon*.

Com os dados obtidos, é possível comparar o uso em mercados centralizados e descentralizados. Considerando as taxas cobradas (10% a 20% do preço do produto) nos mercados centralizados executando a compra diretamente no *Ethereum*, percebe-se que qualquer produto acima de US$72.85 resulta em taxas menores do que se estivesse utilizando mercados centralizados. Executando o mesmo processo na *Polygon*, o custo se torna insignificante. Analisando todos os testes realizados, comprovou-se uma grande vantagem ao executar transações na *Polygon*. Para produtos com valores elevados, pode ser vantajoso executar essas transações em uma instância do *Dagora Market* no *Ethereum*, devido ao maior grau de segurança.

6. Conclusões

Este artigo apresenta uma solução prática e completa para a criação de mercados descentralizados na rede *Ethereum* e também na *side-chain* como o *Polygon* com o *Ethereum* como *main-chain*. A partir destes exemplos mostramos que existe uma grande oportu-

nidade de redução de custos (de até 6 ordens de magnitude) em se executar contratos inteligentes de aplicativos em *side-chain*.

As complexidades do desenvolvimento e *deployment* do mercado descentralizado em esquemas de *side-chain* ou diretamente em *main-chain* são idênticas no nosso caso, pois as duas *blockchains* usadas possuem a mesma máquina virtual para execuções de contratos. Com relação a complexidade adicional para o usuário (vendedores e compradores) que não podem depositar e sacar diretamente na *side-chain*, é possível ocultá-la oferecendo um serviço de pagamentos.

Portanto, conforme é possível verificar em nossas avaliações e é importante enfatizar que independente do preço do produto, sempre existe uma grande vantagem tanto para vendedores como para compradores na utilização de um mercado descentralizado.

A complexidade das chamadas dos contratos inteligentes, assim como a interação entre cliente e vendedor podem ser facilmente escondidas nas interfaces de usuários como discutido em [Neiheiser et al. 2020, Neiheiser et al. 2019]).

Agradecimentos Este trabalho teve auxílio financeiro da CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior).

Referências

(2021). Arbitrum. https://offchainlabs.com/. Acesso em 2021-04.

(2021). Ethereum - #2 market cap. https://coinmarketcap.com/pt-br/currencies/ethereum/. Acesso em 2021-03.

(2021a). Ethereum doc. https://ethereum.org/en/developers/docs. Acesso em 2021-03.

(2021b). Ethereum gas price chart. https://etherscan.io/chart/gasprice. Acesso em 2021-03.

(2021). kchannels. https://www.kchannels.io/. Acesso em 2021-04.

(2021). Loopring. https://loopring.org/. Acesso em 2021-04.

(2021). Matic gas station. https://docs.matic.network/docs/develop/tools/matic-gas-station. Acesso em 2021-03.

(2021). Moonpay. https://www.moonpay.com/. Acesso em 2021-04.

(2021). Non-fungible tokens. https://ethereum.org/en/nft/. Acesso em 2021-03.

(2021). Opensea. https://opensea.io/. Acesso em 2021-03.

(2021). Optimism. https://optimism.io/. Acesso em 2021-04.

(2021). Origin protocol. https://www.originprotocol.com/. Acesso em 2021-03.

(2021). Solidity programming language. https://soliditylang.org/. Acesso em 2021-03.

(2021). State channels. https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/state-channels/. Acesso em 2021-03.

(2021). Truffle suite. https://www.trufflesuite.com/docs. Acesso em 2021-03.

(2021). Wyre. https://www.sendwyre.com/. Acesso em 2021-04.

Adams, H., Zinsmeister, N., Salem, M., Keefer, R., and Robinson, D. (2021). Uniswap v3 core. https://uniswap.org/whitepaper-v3.pdf. Acesso em 2021-03.

Bach, L. M., Mihaljevic, B., and Zagar, M. (2018). Comparative analysis of blockchain consensus algorithms. In *2018 MIPRO*.

Bowe, S., Gabizon, A., and Green, M. D. (2018). A multi-party protocol for constructing the public parameters of the pinocchio zk-snark. In *International Conference on Financial Cryptography and Data Security*, pages 64–77. Springer.

Buterin, V. (2013). Ethereum whitepaper. https://ethereum.org/en/whitepaper/.

Fenu, G., Marchesi, L., Marchesi, M., and Tonelli, R. (2018). The ico phenomenon and its relationships with ethereum smart contract environment. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 26–32.

Foxley, W. (2021). Ethereum's top dapps are increasingly turning to 'rollups': Here's why. https://www.coindesk.com/ethereum-dapps-rollups-heres-why. Acesso em 2021-04.

Kanani, J., Nailwal, S., and Arjun, A. (2019). Matic whitepaper. https://github.com/maticnetwork/whitepaper. Acesso em 2021-03.

Klems, M., Eberhardt, J., Tai, S., Härtlein, S., Buchholz, S., Tidjani, A., Vallecillo, A., Wang, J., and Oriol, M. (2017). Trustless intermediation in blockchain-based decentralized service marketplaces. In *Service-Oriented Computing*.

Lesaege, C., George, W., and Ast, F. (2020). Kleros. https://kleros.io/static/whitepaper_en-8bd3a0480b45c39899787e17049ded26.pdf.

Liu, Z., Luong, N. C., Wang, W., Niyato, D., Wang, P., Liang, Y., and Kim, D. I. (2019). A survey on applications of game theory in blockchain. *CoRR*, abs/1902.10865.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf.

Neiheiser, R., Inácio, G., Rech, L., and Fraga, J. (2020). Hrm smart contracts on the blockchain: emulated vs native. *Cluster Computing*.

Neiheiser, R., Inácio, G., Rech, L., and Fraga, J. (2019). Hrm smart contracts on the blockchain. In *2019 IEEE Symposium on Computers and Communications (ISCC)*.

Ranganthan, V. P., Dantu, R., Paul, A., Mears, P., and Morozov, K. (2018). A decentralized marketplace application on the ethereum blockchain. In *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, pages 90–97.

Rozen, J. (2021). The ridiculously high cost of gas on ethereum. https://coingeek.com/the-ridiculously-high-cost-of-gas-on-ethereum/. Acesso em 2021-04.

Singh, A., Click, K., Parizi, R. M., Zhang, Q., Dehghantanha, A., and Choo, K.-K. R. (2020). Sidechain technologies in blockchain networks: An examination and state-of-the-art review. *Journal of Network and Computer Applications*, 149:102471.

Szabo, N. (1997). Formalizing and securing relationships on public networks. *First Monday Journal*.