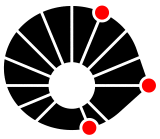

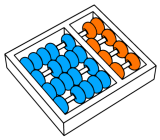

Avaliação do mecanismo de consenso para blockchain Committeeless Proof-of-Stake

G. Gigilas Junior *R. Dahab* *M. A. A. Henriques*

Relatório Técnico - IC-PFG-23-43
Projeto Final de Graduação
2023 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Avaliação do mecanismo de consenso para blockchain Committeeless Proof-of-Stake

George Gigilas Junior* Ricardo Dahab† Marco Aurélio Amaral Henriques‡

Resumo

Este trabalho é o relatório final de um projeto final de graduação, cujo objetivo é implementar novas funcionalidades, realizar testes e validar o mecanismo de consenso para blockchain Committeeless Proof-of-Stake (CPoS). Mais especificamente, este trabalho descreve o desenvolvimento que permite a execução do mecanismo de forma distribuída, além da inserção de transações nos blocos da blockchain, e avalia o impacto dessas adições.

A partir de diversos experimentos executados, encontramos um conjunto de parâmetros que viabiliza a execução distribuída do CPoS, assim como a inserção de transações nos blocos. Os resultados utilizando esses parâmetros se mostraram satisfatórios, mas indicaram pontos a serem trabalhados para alcançarmos um melhor desempenho. Em especial, destacamos o grande volume de dados trafegados pela rede, que posa como um grande desafio a ser enfrentado para a utilização do CPoS em cenários mais realistas.

1 Introdução

As criptomoedas, como Bitcoin [1] e Ethereum [2], hoje ocupam parte significativa do mercado de investimentos. Com a popularização muitas vezes abrupta desses ativos, seus valores crescem e baixam subitamente. Por trás das criptomoedas, existe um mecanismo computacional complexo chamado blockchain, que viabiliza esse tipo de moeda disruptiva e outras diversas aplicações importantes.

Em linhas gerais, a blockchain pode ser interpretada como um livro-razão replicado entre os nós participantes. Em outras palavras, trata-se de um banco de dados distribuído. Sua estrutura é composta por uma sequência de blocos encadeados, por isso o nome blockchain. O que torna esse mecanismo especial são suas propriedades listadas a seguir [3]:

- imutabilidade: é extremamente difícil alterar o conteúdo das transações inseridas em blocos confirmados sem que os demais participantes percebam;
- transparência: tudo que é escrito nos blocos é visível para todos os membros do esquema;

*Instituto de Computação, Universidade Estadual de Campinas
†Instituto de Computação, Universidade Estadual de Campinas
‡Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

- descentralização: não há necessidade de um controlador central. Sendo assim, para determinar os blocos a serem inseridos na blockchain, existe um mecanismo de consenso;
- privacidade: costuma ser difícil associar as transações aos seus autores, apesar de serem públicas;
- segurança: existe um mecanismo de assinatura, facilmente verificável e difícil de ser forjado, que garante a autenticidade das transações.

Diante das propriedades que foram expostas, além das criptomoedas, a blockchain pode ser usada, por exemplo, para armazenar dados de pacientes de um hospital de maneira anônima. Dessa forma, pode-se realizar análises e estudos a partir dos dados coletados, evitando problemas de direitos sobre eles, graças à propriedade de privacidade. Outra aplicação interessante é o armazenamento de dados estratégicos, aproveitando as características de segurança e de descentralização. Assim, seria possível manter uma base de dados consistente entre diversos pontos, por exemplo, com sua integridade garantida.

Conforme já mencionado na característica de descentralização, a blockchain possui um mecanismo de consenso para que os nós participantes possam entrar em acordo sobre qual será o próximo bloco inserido nela. Existem diversos tipos de mecanismo de consenso, sendo os mais populares o Proof-of-Work, utilizado no Bitcoin, e o Proof-of-Stake, utilizado no Ethereum. Cada um deles adota uma estratégia para garantir que apenas um bloco seja inserido na blockchain.

1.1 Proof-of-Work

No mecanismo Proof-of-Work (PoW), o consenso é obtido a partir de um desafio [1]. Os nós participantes procuram nonces (números aleatórios) repetidamente até que o hash do nonce, combinado com vários outros dados de um bloco, atenda o requisito do desafio: ser menor do que um certo valor pré-determinado. Dessa forma, os nós conseguem buscar o consenso de maneira independente e descentralizada, além de ser simples a verificação por outros nós de que o nonce atendeu o desafio (basta calcular o hash dos dados recebido do bloco produzido pelo nó vencedor do desafio e comprovar que atende o que foi especificado).

Este mecanismo funciona bem e costuma atender os requisitos das blockchains, especialmente por ser difícil atender o desafio (pela propriedade de espalhamento das funções hash) e por ser fácil verificar o resultado. No entanto, vários nós gastam inúmeras operações para tentar resolver o desafio ininterruptamente, sendo que apenas uma delas acaba sendo aproveitada (a que vence o desafio). Sendo assim, o número de operações computacionais desperdiçadas aumenta consideravelmente com o passar do tempo. Por esse motivo, quando aplicado em grande escala, este mecanismo desperdiça uma grande quantidade de energia, o que pode causar sérios impactos ambientais [4].

Além dos impactos ambientais, a forma como o Proof-of-Work é estruturado faz com que detentores de grandes centros computacionais tenham maior chance de conseguir publicar um bloco, por conta do alto poder computacional. Essa característica fez com que fossem formados centros mineradores de criptomoedas que possuem grande influência sobre

as decisões das blockchains. Por isso, a característica de descentralização é afetada, pois as decisões sobre os próximos blocos tendem a ser altamente influenciadas pelos grandes centros.

1.2 Proof-of-Stake

Com o objetivo de melhorar os aspectos negativos do PoW, foi proposto o mecanismo de consenso Proof-of-Stake (PoS) [5]. Esse mecanismo se assemelha ao funcionamento de uma loteria, em que cada participante possui um ou mais tickets, e um desses tickets é sorteado para poder criar um novo bloco. No mecanismo, o recurso equivalente ao ticket geralmente é a quantidade de moedas (chamadas de stake) que cada nó possui, sendo limitado o valor total de stakes. Assim, a probabilidade de cada nó vencer o sorteio é diretamente proporcional à quantidade de stake que ele possui. O sorteio, em si, é o que confere ao nó o direito de inserir seu bloco à blockchain, e ocorre apenas uma vez a cada intervalo de tempo pré-definido (rodada).

De maneira geral, esses mecanismos requerem algum tipo de sincronismo na rede, para coordenar as rodadas de sorteios para todos os participantes. Ainda, na maioria dos mecanismos baseados em PoS, existe um comitê de validação cujo objetivo é confirmar os blocos produzidos, ou seja, determinar se o bloco produzido por um nó sorteado pode ser inserido na blockchain de forma definitiva. Entretanto, apesar de os membros dos comitês poderem ser rotativos, a ideia de haver um comitê de validação restringe a característica de descentralização (desejada em blockchains), pelo fato de um grupo menor de nós determinar os próximos blocos a serem inseridos. Além disso, foi comprovado que o comitê de validação pode ser alvo de ataque [6], algo prejudicial para a segurança e confiabilidade do sistema como um todo.

Com relação ao consumo de energia, o PoS cumpre o objetivo de ser mais sustentável. Uma vez que os nós não precisam desperdiçar inúmeras operações feitas de forma contínua, e que não há vantagem em ter um poder computacional maior, esse mecanismo apresenta uma redução de consumo de energia considerável, comparado com o PoW. No entanto, apesar de não beneficiar os grandes centros mineradores, pelo fato de o stake poder ser comprado com dinheiro, ainda há uma questão de vantagens para os nós que possuírem mais dinheiro e, consequentemente, mais stake.

1.3 Committeeless Proof-of-Stake

Em uma tentativa de desenvolver um mecanismo baseado em PoS sem um comitê de validação, D. F. G. Martins criou o Committeeless Proof-of-Stake (CPoS) [7]. Nesse mecanismo, ao invés de contar com um grupo menor de nós validadores para verificar os novos nós a serem confirmados, todos os nós participantes podem verificar o sorteio localmente. Assim, a partir de um consenso probabilístico atrasado, é possível inserir novos blocos na blockchain de forma totalmente distribuída, de maneira similar ao que é feito em mecanismos do tipo PoW.

No CPoS, pode haver mais de um nó sorteado por rodada, mas apenas um dos blocos produzidos será confirmado na blockchain. Sendo w_i o stake do nó i, $W = \sum w_i$ o valor total

de stake em uma rodada, p a probabilidade de um ticket ser sorteado e τ o número esperado de blocos gerados por rodada, tem-se que $\tau = W \times p$. Já que τ é um parâmetro configurável no esquema e W é conhecido, pode-se configurar, consequentemente, a probabilidade de sorteio de um ticket.

O sorteio, em si, consiste em utilizar o hash do bloco anterior da blockchain, juntamente com a identidade e outros dados do nó, para obter um número verificável e pseudoaleatório. Esse número, baseado na probabilidade p, vai conferir, a cada nó, um número de tickets sorteados (podendo ser 0 ou mais). Após esse processo, que ocorre a cada rodada, os nós divulgam um único bloco, dentre os que foram produzidos, sendo este aquele que tiver o menor hash de prova - valor pseudoaleatório produzido a partir do conteúdo do bloco. Então, dentre os blocos divulgados de todos os participantes, o que tiver o menor hash de prova é o escolhido, em consenso, para ser inserido na blockchain.

Vale ressaltar que, por ser um mecanismo de consenso atrasado, os blocos vencedores demoram algumas rodadas para serem confirmados (inseridos de forma irreversível na blockchain). Durante essas rodadas, o nó pode tomar conhecimento de um outro bloco, com hash de prova menor, pela comunicação baseado em protocolo *gossip*. Nesse caso, o nó substitui seu bloco pelo bloco com menor hash de prova, na sua visão local. Pelo fato de o critério de seleção não ser ambíguo, com o passar das rodadas, o mecanismo converge probabilisticamente.

2 Objetivos

Diante do que foi exposto, o CPoS se mostra como um mecanismo de consenso favorável, por aderir às vantagens do PoS e extendê-las com a ausência de um comitê de validação. No entanto, apesar de promissor, ele ainda é um mecanismo recente que necessita ser validado e testado exaustivamente. Em sua primeira versão [7], implementada em Python 2.7, o código passou por vários testes, mas possuía diversos pontos a melhorar, incluindo o uso de blocos cheios de transações. Posteriormente, o código foi reestruturado em Python 3.11, de forma a torná-lo mais legível, com o objetivo de facilitar a realização de novos testes [8].

Nessa nova implementação, também foram feitas melhorias no esquema como um todo. A principal delas foi a adição de beacons, com os quais os nós se comunicam e obtêm os endereços IP dos demais nós do esquema. Na versão original, os endereços IP dos nós vizinhos deveriam ser passados manualmente para cada nó.

O CPoS possui diversos parâmetros ajustáveis que podem impactar diretamente o seu desempenho. Os principais deles são: o parâmetro τ (número esperado de blocos gerados por rodada), o intervalo da rodada (que dita quanto tempo os nós esperam para um novo sorteio) e a tolerância (que indica o atraso - em número de rodadas - que o esquema tolera na chegada de um bloco). É importante avaliar o impacto de cada um desses parâmetros para determinar o conjunto que traz o melhor desempenho para o esquema como um todo.

Validar o funcionamento do código executando apenas em uma máquina não condiz com o que ocorre em um cenário prático de blockchain. Na realidade, os nós podem estar espalhados por qualquer lugar do planeta, o que introduz uma latência de rede oriunda da distância geográfica entre os nós. Caso a latência seja muito grande, seria necessário,

por exemplo, aumentar o intervalo da rodada, de forma que os nós possam participar das rodadas de sorteio sem que estejam fora de sincronia.

Com relação aos blocos com cabeçalhos, mas sem transações, utilizados nos testes anteriores para validar o CPoS, é possível perceber que eles também não condizem com o que é esperado de um cenário prático de uso de uma blockchain. Por ser caracterizada como um banco de dados distribuído, a blockchain só faz sentido a partir do momento que ela armazena informações na sua estrutura. Em um contexto envolvendo cripomoedas, por exemplo, as informações se dão na forma de transações, que são inseridas nos blocos. Uma vez que o código estava em fase inicial de desenvolvimento, as versões que foram testadas estavam sem as transações, com o objetivo apenas de validar o funcionamento do mecanismo em si. A adição de transações nos blocos, no entanto, pode causar um grande impacto no volume de dados na rede, já que cada bloco conterá mais dados e, portanto, a cada divulgação de blocos, mais dados circularão pela rede.

Nesse contexto, os objetivos deste projeto são:

- implementar a funcionalidade de executar os nós de forma distribuída (em várias máquinas), com o uso de beacons;
- produzir e inserir transações nos blocos;
- realizar testes, avaliando o impacto das novas adições e buscando um conjunto de parâmetros que se traduzam em um melhor desempenho do mecanismo.

3 Metodologia

Para realizar as implementações discutidas na seção anterior, tomamos como base a versão do código reescrita em Python 3.11 e desenvolvemos as funcionalidades a partir dela. As subseções a seguir descrevem como foram implementadas a execução dos nós de forma distribuída e a inserção de transações nos blocos, ambas previstas nos objetivos do projeto. Além disso, foi feita a implementação de um banco de dados para a blockchain local, pelo bolsista de iniciação científica Filipe Franco Ferreira, que será brevemente discutida em uma subseção mais adiante.

3.1 Nós Distribuídos

A execução do CPoS de forma local, com os nós executados em uma única máquina, foi feita com a ferramenta Docker, que permite a criação de ambientes virtuais isolados. Dessa forma, cada um dos nós e o beacon eram executados em um desses ambientes isolados, e atuavam como se fossem máquinas distintas. Dando sequência ao que foi feito, optamos por utilizar o Docker Swarm [9] para executar os contêineres de forma distribuída. Essa ferramenta é capaz de orquestrar a execução dos contêineres em diferentes máquinas (workers), a partir de uma máquina central (manager). Para isso, criamos um arquivo Docker Compose, a partir do arquivo já existente para a versão local, para configurar os serviços de nó e de beacon, além da rede, a fim de possibilitar a execução distribuída.

O Docker Compose permite a configuração da inicialização dos contêineres. Para o serviço de beacon, especificamos sua execução na máquina central, configuramos a porta em que o beacon ouvirá os nós e determinamos a execução de um script bash que inicializa o código do beacon. Para o serviço dos nós, configuramos o número de réplicas (número de nós totais no esquema), o número máximo de réplicas por nó (para garantir um espalhamento dos nós), a execução somente em máquinas do tipo worker e a inicialização do código do nó a partir de um script bash. Ainda, especificamos a porta e o IP do beacon, além da porta do próprio nó (para se comunicar com os demais). O Docker Swarm permite uma abstração do IP do beacon, já que, nos testes feitos nesta oportunidade, existe apenas uma instância do serviço beacon sendo executada; logo, podemos especificar o IP como sendo o próprio nome do serviço.

Além dessas configurações específicas de cada serviço, habilitamos os logs (para acompanharmos a execução dos contêineres), especificamos a imagem a ser utilizada (publicada no Dockerhub), e criamos uma rede overlay. A rede é importante para garantir que os contêineres consigam se comunicar através do próprio Swarm, possibilitando a troca de mensagens. Vale ressaltar que, para o funcionamento correto do sistema, cada uma das máquinas precisa estar em uma rede com as seguintes portas abertas para o mundo exterior: 2377 TCP, 7946 TCP e UDP e 4789 UDP.

Devido à dificuldade de acesso a máquinas com essas configurações de rede, as máquinas utilizadas para execução do CPoS de forma distribuída foram máquinas da Faculdade de Engenharia Elétrica e de Computação (FEEC), da Unicamp. A topologia utilizada pode ser observada na Figura 1. O beacon é executado na máquina join, denotada pela cor laranja, que corresponde à máquina central (manager) do Swarm. Em verde, temos os nós (workers), cada um correspondente a uma das máquinas da sala LE-27 da FEEC. Apesar de, em um cenário prático, haver um número de nós bem maior e mais geograficamente distantes, concluímos que essa topologia já seria suficiente para realizarmos testes preliminares.

3.2 Inserção de Transações nos Blocos

Como foi dito anteriormente, é importante adicionarmos transações aos blocos da blockchain. Em um cenário real, as wallets (carteiras) dos usuários produziriam transações que seriam enviadas ao mempool (nome dado ao banco de dados de transações) dos nós que participam do mecanismo de consenso. Por estarmos em fases iniciais, apenas testando o mecanismo do CPoS, optamos por emular o funcionamento desse sistema de forma simplificada. Dessa forma, cada nó possui um mempool e um gerador de transações aleatórias, pois o conteúdo das transações não é importante neste momento e queremos apenas validar o funcionamento do CPoS com transações para avaliar o impacto dessa adição. Assim, o gerador popula o mempool, e, ao instanciar um bloco, o nó do CPoS busca transações do seu mempool para adicioná-las ao bloco.

O script gerador de transações é inicializado de forma paralela, em uma nova thread, assim que o nó inicia sua execução. No início do script, é feita uma conexão com o banco de dados (mempool). Tomando o Ethereum como base, por ser o mecanismo de consenso baseado em PoS mais popular, estipulamos o tamanho médio de cerca de 1,2 kB para cada transação, podendo variar em torno desse valor seguindo uma distribuição normal,

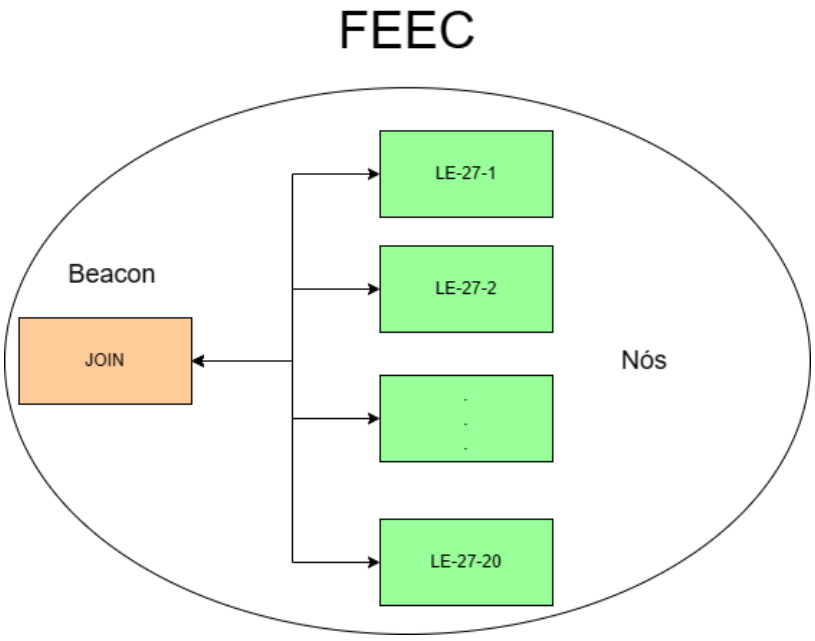

Figura 1: Topologia obtida a partir das máquinas da FEEC. Em laranja, temos a máquina central (manager) do Swarm. Em verde, são os nós do tipo worker.

com um desvio padrão de 200 (já que o tamanho de cada transação depende muito da aplicação). Com relação à taxa recolhida em cada transação, atribuímos um valor médio de 100, seguindo uma distribuição normal com desvio padrão 20.

Para criação do mempool, por sua vez, utilizamos o banco de dados MariaDB [10], um fork open-source do banco de dados MySQL. A chave primária do banco é o identificador único da transação. Além desse campo, as transações possuem os campos: hash da transação (atualmente, é o mesmo valor do identificador da transação), valor da taxa associada, endereços de input (de onde veio a transação) e de output (para onde foi a transação), dados (que correspondem ao conteúdo da transação) e timestamp (horário em que a transação foi criada). Além disso, adicionamos os campos booleanos *chosen* (que assume valor 1 quando a transação é escolhida para ser inserida em um bloco) e *committed* (que assume valor 1 quando um bloco que contém a transação foi confirmado).

Quanto ao código referente ao nó do CPoS, adicionamos uma flag que, quando assume o valor booleano *True*, passa a inserir as transações aleatórias do mempool nos blocos, ao instanciá-los. Determinamos o tamanho do bloco como sendo 200 kB, também baseado nos valores da blockchain do Ethereum. Descontamos 1 kB referente ao cabeçalho do bloco e, então, temos 199 kB para inserirmos transações. As transações são selecionadas em ordem decrescente, de acordo com o valor da taxa, desde que elas possuam os campos *committed* e *chosen* com valor 0, para evitar que as mesmas transações sejam inseridas em mais de um bloco. Então, após inseri-la no bloco, o valor do campo *chosen* é atualizado para 1. As transações são buscadas uma por vez no mempool, sendo esse processo repetido até que não

caibam mais transações no bloco. Existem formas mais otimizadas de fazer essa busca que podem ser implementadas em versões futuras do código.

3.3 Banco de Dados para Blockchain Local

Na implementação em Python 3.11 do CPoS, houve uma grande refatoração do código, mas deixando ainda algumas pendências. Uma delas foi a blockchain local de cada nó ficar armazenada em memória RAM. Além de consumir muito recurso do computador, essa abordagem não é escalável. Com o passar o tempo, a blockchain cresce, até chegar um ponto em que a memória RAM não a comporta por inteiro. Assim, a memória virtual passa a ser utilizada, trazendo um desempenho consideravelmente inferior. Por esses motivos, os testes que foram feitos na nova implementação tiveram um limite de 30 rodadas. Esse número é muito pequeno, se comparado com uma aplicação real, que não deveria ter um limite de rodadas.

Para contornar esse problema, o nosso grupo de estudos começou a desenvolver um banco de dados dedicado para a blockchain local dos nós, também utilizando o MariaDB. Assim, ela ficaria armazenada em disco, sendo movido para memória apenas um bloco para realizar operações com ele. É claro que, em algumas situações, alguns blocos poderiam continuar na memória, para otimizar o desempenho de determinadas operações. Este desenvolvimento ainda está em fases iniciais de teste: o código ainda mantém a blockchain em memória mas, de forma paralela, ele insere os blocos em um banco de dados e os recupera. Apesar de ainda termos o mesmo problema, estamos dando passos em direção à sua solução. É importante realizarmos testes com essa versão para analisarmos se há algum impacto que afete a produção de blocos, por questões de processamento envolvendo o banco de dados.

4 Resultados e Discussões

A partir da implementação do CPoS em Python 3.11, juntamente com as adições comentadas nas seções anteriores, realizamos diversos testes com o CPoS. O código está disponível no GitHub no link `https://github.com/regras/cpos_v2/`. Pelo fato de termos implementado diversas funcionalidades, optamos por realizar os seguintes testes, todos eles executando por um total de 30 rodadas e com 25 nós:

- nós em uma única máquina, com transações e sem banco de dados para a blockchain;
- nós distribuídos, sem transações e sem banco de dados para a blockchain;
- nós distribuídos, com transações e sem banco de dados para a blockchain;
- nós distribuídos, com transações e com o banco de dados para blockchain local.

Vale ressaltar que, para todas as execuções com nós distribuídos, estabelecemos um limite de no máximo 2 nós sendo executados por máquina, a fim de espalhar os nós pelas máquinas. O objetivo de realizar todos esses testes foi de avaliar o impacto de cada uma

dessas adições. Assim, seria possível identificar qual delas teve maior influência no desempenho do CPoS, a fim de propor otimizações. Os dados foram obtidos a partir de um script que extrai informações de arquivos de logs, que são produzidos toda vez que um nó está em execução. Na versão local, salvamos esses logs utilizando o recurso de volume do Docker, que copia os arquivos dos contêineres Docker para uma pasta da máquina que os estava executando. Para a versão distribuída, utilizamos o comando scp (secure copy) para copiar os logs dos contêineres para uma máquina central, ao final de sua execução.

Os resultados extraídos foram: número de blocos confirmados por minuto, atraso de confirmação dos blocos (em número de rodadas), número total de mensagens (circuladas pela rede) e o volume de dados dessas mensagens. Em especial, o número de mensagens é medido pela soma do número de vezes que cada nó recebeu uma mensagem pela rede, e o volume de dados corresponde ao número de bytes dessas mensagens. Em outras palavras, quando o nó recebe uma mensagem em seu socket, o seu número de mensagens é incrementado em 1 e o seu volume de dados é incrementado de acordo com o tamanho, em bytes, da mensagem. Os valores finais correspondem à soma desses valores para todos os nós da blockchain. Já as informações sobre os blocos são extraídas a partir do registro, nos logs, da instância do objeto de blockchain de cada nó. Essa instância contém campos que auxiliam essa análise, como índice do último bloco confirmado, intervalo da rodada, rodada atual, último atraso de confirmação e os próprios blocos.

Para todos os testes, experimentamos valores de τ e de intervalo de rodadas diferentes. Tomando como base os últimos testes que foram feitos [8], variamos τ com os valores 3, 5, 7 e 10, e iniciamos os testes com intervalo de rodada de 5 segundos. No entanto, como esperávamos que as novas adições trouxessem um grande impacto no tempo de processamento de cada rodada, também experimentamos valores de intervalo de rodada maiores, sendo eles 15, 25 e 35 segundos. A maioria dos testes foram feitos com o parâmetro de tolerância configurado para 2 rodadas, mas também fizemos alguns testes pontuais variando a tolerância para 3 ou 4 rodadas. Por uma questão de tempo, não conseguimos repetir todos os testes várias vezes, a fim de tirar uma média e obter um valor mais estável para cada conjunto de parâmetros. No entanto, para os conjuntos que julgamos mais interessantes, executamos os testes 10 vezes, calculando a média e o desvio padrão.

Na Tabela 1, temos os resultados obtidos para os testes realizados localmente, com a inserção de transações nos blocos. De imediato, notamos que praticamente nenhum bloco foi confirmado, com exceção de um dos testes (que provavelmente foi um outlier). Podemos observar que, mesmo variando o valor de tolerância, não foi possível corrigir esse problema. Alguns desses testes foram refeitos com 15, 10 e 5 nós, na tentativa de obter algum resultado melhor, mas não foi possível.

Analisando os logs, notamos que muitos blocos (possivelmente todos) eram descartados por estarem fora do intervalo de tolerância. Esse tipo de erro indica que os nós possivelmente estavam fora de sincronia. Levando em consideração que cada nó precisa inicializar e operar em seus bancos de dados, juntamente com o gerador de transações e com o próprio CPoS, e tudo isso sendo executado em uma mesma máquina, acreditamos que a causa disso seja o consumo intensivo de recursos. Assim, os nós precisam disputar para utilizar o processador e a memória do computador, o que faz com que eles fiquem fora de sincronia. Esse argumento ganha força ao compararmos os resultados desta tabela com as demais, que representam

execuções de forma distribuída.

Tabela 1: Resultados obtidos para execução local, com adição de transações e sem banco de dados para a blockchain.

Tau	Intervalo da rodada (s)	Tolerância	Blocos/min	Atraso de confirmação	Total de mensagens	Total de dados (MiB)
3	5	2	0,00	0 rodadas	109	1,2
5	5	2	0,00	0 rodadas	109	1,1
7	5	2	0,00	0 rodadas	107	1,9
10	5	2	0,00	0 rodadas	106	2,5
3	15	2	0,00	0 rodadas	247	8,2
5	15	2	0,00	0 rodadas	231	21,5
3	25	2	0,00	0 rodadas	302	30,6
3	35	2	0,00	0 rodadas	237	22,9
5	15	3	0,05	1,9 rodadas	303	32,7
5	15	4	0,00	0 rodadas	248	23,6

Já na Tabela 2, podemos observar os resultados obtidos para a execução distribuída, em 25 nós, mas sem a inserção de transações. Vale ressaltar que o gerador de transações ainda está populando o mempool, então o processamento dos nós está sendo dividido com essas demais atividades. De qualquer forma, é interessante observarmos o sucesso na confirmação de blocos, uma vez que conseguimos obter vários blocos confirmados por minuto, com um atraso de confirmação baixo (de maneira geral). Podemos notar alguns possíveis outliers, como nos casos em que o atraso foi de 8,12 rodadas e 7,68 rodadas, que provavelmente são resultado de termos executado os testes apenas uma vez.

Em todo caso, o número de mensagens divulgadas foi muito maior do que na Tabela 1, o que indica, realmente, um problema na execução de forma local. Já o volume de dados, em alguns casos, chegou a ser menor em alguns testes, o que é um reflexo da ausência de transações. A partir de uma análise mais profunda, nota-se que, de maneira geral, quanto maior o valor de τ, menor o atraso de confirmação e maior o número de blocos confirmados por minuto. Esse resultado é esperado pois a confirmação do bloco é diretamente proporcional ao número de sorteios bem sucedidos. Aumentando o valor de τ, o número esperado de nós sorteados é automaticamente aumentado.

Dando continuidade aos testes, a Tabela 3 corresponde aos resultados da execução distribuída e com transações nos blocos. Já em uma primeira análise, é possível observar que os resultados para intervalo de rodada de 5s se distinguem bastante dos demais, especialmente com relação ao total de mensagens e ao volume de mensagens. O baixo número de blocos confirmados por minuto indica que o intervalo da rodada está muito baixo, provavelmente por conta dos esforços para buscar transações no mempool e inseri-las no bloco. Com exceção na primeira linha da tabela, nota-se que, conforme o aumento do valor de τ, esse efeito piorou. Esse comportamento sugere que o aumento de nós sorteados por rodada faz com que fique inviável o intervalo de rodada de 5s, possivelmente porque mais blocos buscam por mais transações na mempool.

Já para os demais conjuntos de parâmetros, é possível observar que blocos foram con-

Tabela 2: Resultados obtidos para execução distribuída (25 nós), sem transações e sem banco de dados para a blockchain.

Tau	Intervalo da rodada (s)	Tolerância	Blocos/min	Atraso de confirmação	Total de mensagens	Total de dados (MiB)
3	5	2	5,25	2,72 rodadas	18.728	10,1
5	5	2	8,99	2,92 rodadas	19.134	10,2
7	5	2	9,10	3,48 rodadas	21.129	11,4
10	5	2	10,58	1,40 rodadas	25.452	13,7
3	15	2	2,65	8,12 rodadas	18.147	9,8
5	15	2	2,93	7,68 rodadas	20.421	11,0
7	15	2	2,89	1,40 rodadas	25.286	13,6
10	15	2	3,81	1,40 rodadas	25.200	13,6
3	25	2	2,14	2,36 rodadas	22.222	11,9
5	25	2	2,26	1,79 rodadas	22.967	12,4
7	25	2	2,27	1,58 rodadas	23.462	12,6
10	25	2	2,28	1,30 rodadas	26.965	14,6
3	35	2	1,56	1,76 rodadas	20.592	11,1
5	35	2	1,61	1,80 rodadas	21.076	11,3
7	35	2	1,62	1,64 rodadas	23.292	12,5
10	35	2	1,64	1,40 rodadas	27.488	14,8

firmados normalmente. No entanto, há diferenças nítidas entre esta tabela e a Tabela 2. Começando pela análise sobre o número de blocos confirmados por minuto e sobre o atraso de confirmação, houve um impacto negativo na confirmação dos blocos. Uma possível causa para esse comportamento, além de possíveis variações devido a ter sido feita apenas uma execução para cada conjunto de parâmetros, é o aumento de processamento envolvendo a busca por transações no mempool e a inserção delas no bloco. Esse processo traz uma complexidade que aumenta as operações necessárias a cada vez que um nó produz um bloco, podendo resultar em blocos fora do intervalo de tolerância.

Além disso, analisando o número de mensagens e, principalmente, o volume de dados trafegados, há uma diferença gritante. A inserção de transações fez com que o volume de dados, que era da ordem de dezenas de MiB, fosse para a ordem de GiB. Já com relação ao número total de mensagens, com base nas próximas tabelas, acreditamos que existe uma oscilação nesses resultados. Para tirar melhores conclusões, seria necessário repetir os testes e comparar os valores médios de número de mensagens.

A Tabela 4 ilustra um comportamento bastante parecido com o da Tabela 3, com a diferença mais impactante sendo na primeira linha em que, dessa vez, nenhum bloco foi confirmado. Isso pode ser, novamente, por conta de ter sido feito apenas uma execução para cada conjunto de parâmetros. No entanto, outra possibilidade é de que o processamento oriundo da interação com o banco de dados local trouxe um novo impacto negativo no tempo de produção dos blocos. Assim, os blocos acabam ficando fora do intervalo de tolerância. Quanto aos demais resultados, acreditamos que a adição do banco de dados não os influenciou.

Intrigados com os resultados das Tabelas 3 e 4 para intervalo de rodada de 5s, optamos

Tabela 3: Resultados obtidos para execução distribuída (25 nós), com transações e sem banco de dados para a blockchain.

Tau	Intervalo da rodada (s)	Tolerância	Blocos/min	Atraso de confirmação	Total de mensagens	Total de dados (MiB)
3	5	2	5,33	4,60 rodadas	28.077	351,4
5	5	2	0,02	0,40 rodadas	2.609	352,5
7	5	2	0,29	1,24 rodadas	3.632	503,8
10	5	2	0,02	0,50 rodadas	3.581	531,3
3	15	2	2,17	9,60 rodadas	28.609	4.377,9
5	15	2	3,51	2,00 rodadas	29.929	4.524,6
7	15	2	3,08	2,56 rodadas	34.402	5.226,6
10	15	2	3,53	3,40 rodadas	43.177	6.577,0
3	25	2	1,18	14,32 rodadas	28.645	4.378,2
5	25	2	1,79	1,96 rodadas	32.413	4.914,5
7	25	2	2,05	4,04 rodadas	42.436	6.469,3
10	25	2	2,08	3,00 rodadas	45.490	6.929,3
3	35	2	1,44	3,36 rodadas	31.414	4.767,2
5	35	2	0,83	13,88 rodadas	35.607	5.457,8
7	35	2	0,90	4,96 rodadas	36.693	5.614,1
10	35	2	1,26	5,00 rodadas	43.163	6.610,6

por executar novos testes variando o intervalo de tolerância, que podem ser vistos na Tabela 5. No entanto, os resultados ainda foram bem ruins, indicando que o intervalo de rodada está muito curto. O grande problema é que, se um nó demora um tempo maior que o intervalo da rodada para produzir um bloco, o seu atraso aumenta a cada rodada. Dessa forma, mesmo aumentando o intervalo de tolerância, o nó não consegue ficar sincronizado com os demais.

Finalmente, na Tabela 6, realizamos 10 execuções, para cada um dos conjuntos de parâmetros que julgamos mais interessantes, e tiramos a média. Vale notar que essas execuções foram feitas com a implementação mais completa (distribuída, com transações e com banco de dados para blockchain local). Como o intervalo de rodada de 15s não pareceu apresentar problemas para confirmação de blocos, optamos por utilizá-lo, já que qualquer intervalo maior que ele poderia implicar em uma espera desnecessária para começar outra rodada. O valor de tolerância foi mantido em 2 e os mesmos valores de τ foram testados.

Conforme foi dito anteriormente, o aumento do valor de τ tende a aumentar o número de blocos confirmados por minuto. No entanto, aqui notamos que, na verdade, os melhores valores de confirmação de blocos foram obtidos para τ com valor 7. Acreditamos que esse resultado seja um impacto direto da sobrecarga da rede, por conta do grande volume de dados trafegados. Pela tabela, é possível concluir que o valor de τ é diretamente responsável pelo aumento de mensagens (e do volume de dados como um todo), o que traz sérios impactos para o desempenho do CPoS.

Além disso, também trouxemos o número de blocos produzidos e blocos confirmados. Podemos observar que o número de blocos produzidos não aumenta muito depois de τ

Tabela 4: Resultados obtidos para execução distribuída (25 nós), com transações e com banco de dados para blockchain local.

Tau	Intervalo da rodada (s)	Tolerância	Blocos/min	Atraso de confirmação	Total de mensagens	Total de dados (MiB)
3	5	2	0,00	0,00 rodadas	1.566	165,4
5	5	2	0,06	0,68 rodadas	1.720	195,6
7	5	2	0,00	0,32 rodadas	2.622	347,0
10	5	2	0,85	1,60 rodadas	3.780	542,0
3	15	2	3,17	3,80 rodadas	28.346	4.294,0
5	15	2	3,22	1,88 rodadas	30.827	4.666,6
7	15	2	3,02	3,08 rodadas	34.993	5.322,9
10	15	2	3,01	3,40 rodadas	41.817	6.374,4
3	25	2	1,19	11,20 rodadas	24.327	3.700,4
5	25	2	2,03	3,52 rodadas	34.452	5.235,3
7	25	2	1,83	2,56 rodadas	43.279	6.605,5
10	25	2	2,00	2,20 rodadas	46.686	7.124,5
3	35	2	1,09	4,48 rodadas	28.230	4.285,5
5	35	2	1,26	3,52 rodadas	36.251	5.545,2
7	35	2	1,22	4,08 rodadas	36.339	5.551,2
10	35	2	1,41	4,70 rodadas	41.243	6.293,5

assumir valor 5. Isso fortalece o argumento anterior, visto que, mesmo aumentando o número de sorteios esperados por rodada, o número de blocos produzidos ficou bem próximo para esses valores de τ. No entanto, houve um impacto um pouco maior sobre o número de blocos confirmados para valores de τ maiores.

Na Tabela 7, podemos observar os dados extraídos nos últimos testes feitos com o CPoS [8], na implementação em Python 3.11. Os valores da tabela são resultados da média de 10 execuções, com 25 nós, 30 rodadas, tolerância de 2 rodadas e intervalo de rodada de 5s. Vale ressaltar que essa versão não usava base de dados e que os 25 nós estavam em uma só máquina. Comparando com os nossos melhores resultados (Tabela 6), notamos que o número de blocos por minuto na Tabela 6 é menor, o que faz sentido, pelo fato de o intervalo da rodada ser maior. O atraso de confirmação, no entanto, não depende do intervalo da rodada e apresentou melhores resultados na versão anterior, o que indica o impacto negativo das novas funcionalidades, adicionadas por este trabalho, no desempenho do CPoS. O número de mensagens e o volume de dados eram bem menores do que os resultados deste trabalho. O volume era esperado ser maior após a adição de transações nos blocos, mas o número de mensagens é algo que precisa ser melhor investigado. Possivelmente, mensagens de controle do Docker Swarm podem estar distorcendo esses valores.

5 Dificuldades

Ao longo da execução dos experimentos, deparamo-nos com diversas dificuldades que serão listadas nesta seção. A primeira delas corresponde à falta de sincronismo entre alguns

Tabela 5: Resultados obtidos para execução distribuída (25 nós), com transações e com banco de dados para blockchain local, variando o parâmetro de tolerância.

Tau	Intervalo da rodada (s)	Tolerância	Blocos/min	Atraso de confirmação	Total de mensagens	Total de dados (MiB)
3	5	3	0,00	0,00 rodadas	2.110	238,3
5	5	3	0,00	0,00 rodadas	2.495	307,8
7	5	3	0,00	0,00 rodadas	2.220	277,6
10	5	3	0,35	0,80 rodadas	2.919	389,8
3	5	4	0,05	0,44 rodadas	1.730	185,4
5	5	4	0,08	0,60 rodadas	2.525	331,4
7	5	4	0,03	0,80 rodadas	2.808	358,2
10	5	4	0,00	0,00 rodadas	2.988	406,7

Tabela 6: Resultados obtidos a partir da média de 10 execuções, para os conjuntos de parâmetros mais interessantes da Tabela 4: rodada de 15s e tolerância de 2 rodadas.

Tau	Blocos/min	Atraso de confirmação	Total de mensagens	Total de dados (MiB)	Blocos	Blocos confirmados
3	1,10	9,79 rodadas	17.428,2	2.691,3	474,3	232,1
5	2,41	5,74 rodadas	24.712,5	3.757,6	630,1	476,9
7	2,72	3,29 rodadas	26.917,1	4.084,6	633,2	535,6
10	2,66	4,04 rodadas	31.257,2	4.765,3	652,7	524,8

nós, mesmo em casos de teste em que isso não era esperado (como em casos em que o intervalo da rodada era de 35s). Isso foi observado de duas formas: alguns nós iniciavam sua execução algum tempo (não desprezível) depois dos demais e/ou terminavam sua execução depois dos demais; várias vezes apareceram mensagens nos logs de blocos que estavam fora do intervalo de tolerância, mesmo nas primeiras rodadas de execução.

Com relação ao início e a finalização da execução de alguns nós de forma não sincronizada, ainda foi possível observar alguns casos em que eles ultrapassavam a rodada 30. Isso, provavelmente, ocorreu pelo fato de eles terem iniciado mais tarde e terem tido que ressincronizar com os demais nós, o que faz com que eles saltem algumas rodadas. Assim, como eles estavam programados para executar por 30 rodadas, eles ultrapassaram a 30ª

Tabela 7: Resultados obtidos na versão anterior do CPoS [8]: 30 rodadas, 25 nós, intervalo de rodada de 5s e tolerância de 2 rodadas.

Tau	Blocos/min	Atraso de confirmação (rodadas)	Total de mensagens	Total de dados
3	2,41	5,1	2,8 x 10^3	1,5 MiB
5	3,78	3,2	5,8 x 10^3	3,1 MiB
7	4,61	2,7	8,0 x 10^3	4,3 MiB
10	5,87	2,0	9,1 x 10^3	5,1 MiB

rodada. Já quanto aos blocos fora do intervalo de tolerância, achamos bastante estranho isso ter ocorrido logo nas primeiras rodadas, pois os nós deveriam iniciar em instantes muito próximos.

Acreditamos que esses problemas se devam ao fato do Docker Swarm inicializar os contêineres de forma não sincronizada, ou por conta de algum problema na rede que atrasou os comandos do Swarm enviados às máquinas. Para contornar isso, seria necessário implementar algum mecanismo de sincronização na própria aplicação. Uma ideia simples seria o script de inicialização dos nós receber um parâmetro correspondente ao horário em que ele deve começar a executar. Assim, o Swarm poderia passar o horário como sendo 1 minuto (por exemplo) depois do início de sua execução, aumentando as chances de todas as máquinas iniciarem ao mesmo tempo. No entanto, seria necessário garantir que as máquinas estejam com seus relógios sincronizados, o que pode ser um problema de difícil solução.

Outra dificuldade encontrada, que pode ter impactado alguns testes, foi que algumas máquinas foram desligadas ao longo do período de coleta dos dados. Houve, também, casos em que algumas das máquinas foram religadas posteriormente. Por estarmos executando 25 nós com, no máximo, 2 nós por máquina, precisávamos de apenas 13 máquinas para conseguir executar os testes. Em todos os casos, tivemos pelo menos 13 máquinas ligadas executando os nós, mas também tivemos casos com até 17 máquinas ligadas. Não sabemos o quanto essas diferenças impactaram o desempenho do CPoS, mas nos certificamos de que nenhuma máquina foi desligada ou religada durante a execução de algum dos testes. Infelizmente, pelo fato de as máquinas serem públicas (e pelos testes serem demorados), é difícil remover este problema.

Uma característica do código atual, que pode ter trazido impactos negativos aos testes, é a configuração do beacon. Atualmente, quando o nó se apresenta ao beacon, ele recebe uma lista dos peers que o beacon conhece. Assim, o primeiro nó não recebe nenhum peer, o segundo nó recebe 1 peer e o n-ésimo nó recebe $n - 1$ peers. Essa abordagem se mostra bem desequilibrada, pelo menos no início, antes de os nós adicionarem os peers que os enviaram mensagens à lista de peers conhecidos. Mesmo assim, ao decorrer do consenso, todos os peers tendem a enviar mensagens para todos os outros, o que pode ter sido um dos motivos pelo grande número de mensagens circuladas pela rede. Esse problema tende a se tornar maior conforme aumentamos o número de nós no CPoS. Para melhorar esse comportamento, pode ser interessante limitar o tamanho da lista de peers conhecidos para um valor pequeno, como 5 ou 10. Assim, seria possível diminuir o número de mensagens na rede e tornar a rede mais equilibrada.

Com relação ao Docker Swarm, tivemos bastante dificuldade em fazê-lo funcionar, especialmente pelas suas abstrações internas de rede. Após atingirmos uma versão estável, nos restaram dois problemas: a ordem em que os serviços (nó e beacon) e a rede eram inicializados; e, em alguns casos, a impossibilidade de determinados nós conseguirem se comunicar.

O primeiro desses problemas ocorria quando a rede não era inicializada primeiro, o que causava erro ao tentar inicializar os serviços que dependem dela, ou quando o serviço do nó era inicializado antes do serviço do beacon. Neste último caso, o nó não conseguia se comunicar com o beacon pois não conhecia seu serviço e, portanto, não conhecia seu IP. Para contornar esse problema, optamos por inicializar os serviços e, imediatamente,

remover o serviço de nó para, então, reinicializar o serviço de nó. Dessa forma, garantimos, manualmente, que o nó era inicializado por último. Quanto ao problema da rede não inicializando primeiro, foi necessário tentar a inicialização do sistema novamente, até que ela inicializasse antes dos serviços. Esses problemas ocorrem porque o modo Swarm não aceita ordenamento dos serviços. Por isso, para resolvê-los, seria necessário implementar uma lógica na própria aplicação para lidar com isso.

Com relação ao segundo problema, notamos algumas mensagens que um peer falhou em se comunicar com outro peer, seguido pelo erro "Host Unreachable". Esse erro geralmente ocorre quando não há rota entre dois nós, o que não deveria acontecer, pois todos os nós estavam no Swarm. Não sabemos a causa deste problema, podendo ser algo do próprio Swarm ou talvez da rede da FEEC. Ainda, pode ser que alguma máquina tenha sido desligada ou religada durante a execução de algum desses testes, mas achamos improvável, porque não observamos problemas desse tipo. Em todo caso, para coleta dos dados, descartamos todos os testes que apresentaram esse erro. Acreditamos que seja importante investigar esse erro mais a fundo.

Por fim, tivemos algumas dificuldades com a coleta de dados por si só, que foi um processo bastante manual e tedioso. Pela forma como as execuções dos nós produzem os logs, não temos como distinguir à qual execução cada log pertence. Por esse motivo, os testes precisam ser executados manualmente e os arquivos de log precisam ser movidos para outro diretório, a fim de não atrapalhar os experimentos. Achamos importante desenvolver uma forma melhor de coletar esses dados, possivelmente através do desenvolvimento de um script que cria uma fila para inicialização de vários testes em sequência, de forma automática. Vale destacar, no entanto, a importância de separar os resultados de cada teste em diretórios diferentes. Com isso, acreditamos que será possível executar os testes de forma mais eficiente.

6 Conclusão e Trabalhos Futuros

Com relação aos mecanismos de consenso para blockchain mais populares, o Proof-of-Work e o Proof-of-Stake, o Committeeless Proof-of-Stake se mostra como um forte concorrente. A ausência de comitê de validação, juntamente com as vantagens do PoS sobre o PoW, fazem com que o CPoS seja uma alternativa promissora. No entanto, seu desenvolvimento recente se mostra como um grande desafio a ser superado e aponta necessidades de realização de testes para sua validação.

Diante desse contexto, este trabalho descreveu o desenvolvimento que possibilitou a execução do CPoS de forma distribuída, importante para qualquer tipo de uso prático de blockchains, o que introduz a latência de rede nos novos testes realizados. Além disso, implementamos a inserção de transações nos blocos, que amplificou o volume de dados trafegados na rede e trouxe um grande impacto nos resultados alcançados. O trabalho também avalia a adição da implementação, em andamento, do banco de dados para blockchain local, que não pareceu impactar tanto os resultados, pelo menos nessa versão inicial.

A partir dos testes realizados, encontramos um conjunto de parâmetros que viabilizou todas essas novas funcionalidades, trazendo resultados satisfatórios. No entanto, não po-

demos deixar de citar pontos de melhoria a serem trabalhados e otimizados. Em especial, o grande volume de dados (que aumentou muito com a adição de transações) e o grande número de mensagens circuladas se mostraram como os principais problemas a serem trabalhados.

Como trabalhos futuros, gostaríamos de ressaltar a importância de atuar nos pontos citados na seção Dificuldades. Mais especificamente, acreditamos que a topologia da rede precisa ser aprimorada o quanto antes. Ainda, aspectos como os problemas de sincronismo, os problemas com o Swarm e a ineficiência da coleta dos dados são pontos que trarão grandes benefícios ao CPoS quando forem resolvidos.

Além disso, ainda há uma grande lista de melhorias e novas funcionalidades que podem ser adicionados ao CPoS. Em primeiro lugar, é importante adicionar algum tipo de heartbeat entre os nós e o beacon, para que, caso um nó termine sua execução, o beacon perceba e pare de divulgar o seu IP para os demais peers. Esse comportamento é importante para que possamos deixar o CPoS em execução por mais rodadas e com peers entrando e saindo da rede. A escolha do banco de dados a ser utilizado também pode ser repensado, de forma a otimizar consultas para melhorar o desempenho do CPoS como um todo.

Com relação à execução distribuída, gostaríamos de poder trabalhar com nós com uma distância geográfica maior. Para isso, poderíamos adicionar, ao Swarm, nós de outros países, ou até de outras regiões do Brasil. Com esse objetivo, podemos utilizar serviços que disponibilizam a utilização de máquinas remotas. No entanto, pode haver um certo grau de dificuldade em configurar os nós corretamente, quanto a aspectos de rede, para que eles possam fazer parte ativa do Swarm.

Quanto ao grande volume de dados circulado pela rede, pode ser interessante avaliar uma forma alternativa de divulgação dos blocos. Atualmente, todos os nós sorteados são divulgados por inteiro, incluindo as transações. No processo de desenvolvimento da primeira versão do CPoS [7], foi proposta uma versão alternativa que consiste em divulgar apenas o cabeçalho dos blocos sorteados. Então, apenas depois de decidir o bloco vencedor, seu conteúdo completo seria divulgado. Essa proposta tem grande potencial para reduzir o volume de dados trafegados, mas ainda precisa ser testada e avaliada.

Por fim, para uma simulação mais realista da geração de transações, poderíamos ter um gerador global que populasse o mempool de todos os nós. Com isso, estaríamos emulando o funcionamento das wallets, que muitas vezes divulgam suas transações para mais de um nó ao mesmo tempo. Entretanto, seria necessário um maior controle das transações, que precisaria ser desenvolvido. Como mais de um nó poderia ter a mesma transação, ao aceitar um novo bloco em sua visão local da blockchain, o nó deveria conferir se as transações desse bloco estão contidas em seu mempool. Em caso positivo, essa transação deverá ser sinalizada como *chosen* ou *commited* (caso o bloco já esteja confirmado). Além disso, caso o bloco produzido por um nó seja rejeitado, as transações que estavam presentes nele deverão voltar a possuir o campo *chosen* como 0, caso elas não estejam contidas no bloco que substituiu o rejeitado. Esse controle das transações é importante para garantir que não haverá transações duplicadas ou descartadas do esquema.

Referências

[1] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*. [Online] Whitepaper (2009). Disponível em: `https://bitcoin.org/bitcoin.pdf` (acessado em 12/10/2023).

[2] G. Wood, *Ethereum: A secure decentralised generalised transaction ledger*. Ethereum Project Yellow Paper (2014)

[3] F. Greve, et al., *Blockchain e a Revolução do Consenso sob Demanda*. Minicursos do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (2018). Campos de Jordão, SP, Brasil: Sociedade Brasileira de Computação. Disponível em: `http://143.54.25.88/index.php/sbrcminicursos/article/view/1770` (acessado em 12/10/2023).

[4] M. Wendl, M. H. Doan, R. Sassen, *The environmental impact of cryptocurrencies using proof of work and proof of stake consensus algorithms: A systematic review*. Journal of Environmental Management (2023), 326:116530.

[5] C. T. Nguyen, et al., *Proof-of-Stake Consensus Mechanisms for FutureBlockchain Networks: Fundamentals, Applications and Opportunities*. IEEE Access, v. 7, p. 85727–85745 (2019). ISSN 21693536. Disponível em: `https://ieeexplore.ieee.org/document/8746079/` (acessado em 08/12/2023).

[6] M. Neuder, D. J. Moroz, R. Rao, D. Parkes, *Low-cost attacks on Ethereum 2.0 by sub-1/3 stakeholders*. (2021). Disponível em: `https://arxiv.org/pdf/2102.02247.pdf` (acessado em 08/12/2023).

[7] D. F. G. Martins, *Um novo mecanismo de consenso probabilístico para blockchains públicas*. (2021). Disponível em: `https://repositorio.unicamp.br/Busca/Download?codigoArquivo=507683` (acessado em 12/10/2023).

[8] V. Peixoto, M. A. A. Henriques, *Analysis of Committeeless Proof-of-Stake protocol: Searching for a better point of operation*. XXII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (2023). Juiz de Fora, MG, Brasil: Sociedade Brasileira de Computação.

[9] *Swarm mode overview*. Docker Inc.. Disponível em: `https://docs.docker.com/engine/swarm/` (acessado em 09/12/2023).

[10] *MariaDB Server: The open source relational database*. MariaDB Foundation. Disponível em: `https://mariadb.org/` (acessado em 09/12/2023).