Avaliação de Aplicabilidade do Blockchain Ethereum em uma Rede IoT empregando o Blockbench Aprimorado

Maykon Valério da Silva[1], Aldri Luiz dos Santos[2] e Luiz Antonio Rodrigues[1]

[1]Programa de Pós-Graduação em Ciência da Computação (PPGComp)
Universidade Estadual do Oeste do Paraná (Unioeste), Cascavel – PR – Brasil

[2]Departamento de Ciência da Computação, ICEx
Universidade Federal de Minas Gerais (UFMG), Belo Horizonte – MG – Brasil

maykonvs@gmail.com, aldri@dcc.ufmg.br, luiz.rodrigues@unioeste.br

Abstract. *With the spread of the Internet of Things, many devices are being connected, collecting and transmitting data. Blockchain is a secure solution for distributed ledgers, but its high computational and energy costs are limiting for devices with limited resources. In this study, we modified Blockbench to evaluate the performance of Ethereum private networks involving a Raspberry Pi. Scenarios with and without the device and the consensus protocols Ethash and Clique were compared. The results show that the Raspberry Pi cannot be a miner in the Ethash network, but only a lightweight node, and emphasize the superiority of the Clique consensus regarding latency, throughput, and consumption of computing resources.*

Resumo. *Com a expansão da Internet das Coisas, muitos dispositivos estão sendo conectados, coletando e transmitindo dados. O Blockchain é uma solução segura para registro distribuído, mas seu alto custo computacional e energético é restritivo para equipamentos com recursos limitados. Neste estudo, o Blockbench foi aprimorado para avaliar o desempenho de redes Ethereum privadas com a inclusão de um dispositivo Raspberry Pi. Foram comparados cenários com e sem o dispositivo, bem como os protocolos de consenso Ethash e Clique. Os resultados mostram que o Raspberry Pi não pode ser um minerador na rede Ethash, apenas um nó leve, e destacam a superioridade do consenso Clique em termos de latência, vazão e consumo de recursos computacionais.*

1. Introdução

Os dispositivos de Internet das Coisas (IoT), muitas vezes referenciados como sistemas embarcados ou ciber-físicos, formam uma rede distribuída na qual cada objeto possui uma identificação única e trocam informações entre si [Sanju et al. 2018]. As redes IoT têm apoiado novas aplicações em vários domínios, como *Smart Homes*, *Small Office/Home Office* (SOHO), Saúde [Sankaran et al. 2018], IIoT (*Industrial Internet of Things*) para a Automação Industrial, IoV (*Internet of Vehicles*) para a área Automotiva, a IoE (*Internet of Energy*) para a área de Energia, como a geração, distribuição, entre outras. Entretanto, a IoT enfrenta desafios relacionados à segurança, a medida que as suas aplicações estão sujeitas a ataques na interceptação de mensagens e na obtenção de dados armazenados [Vashi et al. 2017]. Além disso, várias das aplicações dependem de sistemas servidores centrais, prejudicando a disponibilidade por terem pontos únicos de falhas.

As segurança em IoT é crucial por lidar com dados sensíveis, e a exposição de dados pode levar à invasões de privacidade em aplicações como *Smart Homes*, segredos industriais em aplicações IIoT e até a riscos físicos à usuários em aplicações IoV e IoE. Além disso, várias aplicações dependem de sistemas servidores centrais, prejudicando a disponibilidade por terem pontos únicos de falhas. Blockchains, teoricamente possuem todas as características para melhorias na segurança da IoT. Eles consistem em bancos de dados imutáveis, distribuídos e auditáveis, também dito como uma tecnologia de livro-razão distribuído (*Distributed Ledger*). Surgida para servir de base para um sistema de moeda, a sua implementação foi proposta para se resolver, sem o uso de um servidor central, o problema de gasto duplo, no qual um recurso seria gasto indevidamente duas vezes em transações diferentes. Além disso, trata-se de uma solução que inclui tolerância a falhas, integridade e privacidade das informações [Zheng et al. 2018].

A imutabilidade do Blockchain exige de seus nós uma abundância de recursos computacionais, porém as aplicações IoT são compostas em sua maioria por dispositivos com pouca memória, pequeno espaço para armazenamento, baixo poder de processamento e baixa disponibilidade energética (quando alimentados por baterias), exigindo Blockchain modificado e refinado especificamente à aplicação em questão [Ghiro et al. 2021]. Em [Raghav et al. 2020], [Fu et al. 2022] e [Chen et al. 2021a], protocolos de consenso adaptados procuram reduzir o poder computacional necessário. [Wang et al. 2021, Wu et al. 2021, Yang et al. 2020] propõem formas de se lançar operações, necessárias para o Blockchain, mas que os dispositivos IoT não conseguem desempenhar, para servidores de processamento na borda da rede, entre outros. No entanto, a maioria das propostas é testada e validada com metodologias próprias, geralmente usando métodos de simulação. Logo, estabelecer uma metodologia baseada em um *benchmark* padronizado favorece a comparação entre diferentes soluções atuais e futuras.

Este trabalho apresenta uma avaliação prática e abrangente da aplicabilidade do Blockchain Ethereum em IoT, formulando uma arquitetura de rede que contenha os dispositivos que podem ser encontrados nesses cenários, desde os mais restritos até os mais eficientes. Especificamente, é analisado o desempenho por meio da latência e da vazão da aplicação no registro de transações Ethereum e nas consultas às informações da rede, e o desempenho de clientes leves (dispositivos de médio/baixo poder computacional) e de clientes com baixo poder computacional, externos à rede blockchain. Para auxiliar a execução dos experimentos, foram realizados diversos aprimoramentos no benchmark Blockbench [Dinh et al. 2017], permitindo a execução facilitada e padronizada dos cenários de teste, incluindo também a incorporação de dispositivos IoT.

No restante do artigo, a Seção 2 discute os trabalhos relacionados. A Seção 3 detalha o ambiente de experimentação e as adequações do Blockbench. A Seção 4 apresenta os resultados alcançados. A Seção 5 inclui a conclusão e os trabalhos futuros.

2. Trabalhos Relacionados

Os recursos computacionais limitados, como pouca memória, pequeno espaço para armazenamento, baixo poder de processamento e baixa disponibilidade energética (quando alimentados por baterias) são discutidos em [Ghiro et al. 2021], no qual é feita uma análise teórica sobre a aplicabilidade do Blockchain em IoT. Os autores destacam esta oposição entre os requisitos de ambos, e chegam a conclusão de que blockchains podem ser utiliza-

dos em IoT se modificados e refinados especificamente para a aplicação em questão. Esta seção apresenta alguns trabalhos de pesquisa na área com ênfase nestes dispositivos.

Em [Sankaran et al. 2018] é avaliado o desempenho e consumo energético do Ethereum em uma rede privada com dois Raspberry Pi e um PC. O BlockBench é utilizado para obter dados de desempenho e o SMU (*Source Measurement Unit*) 2461 da *Keythley* é utilizado para medir o consumo energético dos Raspberry. Em [Sanju et al. 2018] os autores estendem os testes para o Hyperledger Fabric e comparam com os resultados com o Ethereum, mostrando sua superioridade em relação à este no contexto de IoT. Posteriormente, em [Sankaran et al. 2019] dois Raspberry Pi e um Laptop são configurados para fazerem parte de *pools* de mineração de criptomoedas de forma a verificar o consumo energético dos protocolos de consenso utilizados nestes sistemas. Embora o método empregado permita realizar as medições em Blockchains públicas e operacionais, ele está limitado a verificações de protocolos de consenso, sem levar em consideração outros aspectos importantes para uma rede IoT, como latência, vazão e espaço de armazenamento.

Em [Chen et al. 2021b] é apresentado um teste comparativo entre o Ethereum e o IOTA, analisando o desempenho, o consumo energético, o espaço em disco e a quantidade de memória utilizados. A arquitetura montada consiste em dois Raspberry Pi com um deles tendo um sensor de partículas conectado. Os dados deste sensor são armazenados no Blockchain. A forma de medição de consumo energético é semelhante a presente em [Sankaran et al. 2018], com um SMU, mas os dados de desempenho são obtidos através de comandos do linux. Neste caso, o uso de um framework como o BlockBench mostra-se uma opção melhor devido à padronização na obtenção de dados. Além disso, embora um dispositivo de baixíssimo poder computacional esteja inserido na rede, não utilizou-se um de maior capacidade, não contendo uma arquitetura completa para os testes. Em [Yi and Wei 2019], os autores comparam Ethereum, Parity, Quorum e Ethermint usando o Blockbench para avaliar a vazão e a latência destas redes. Os resultados destacam o Ethermint como a melhor opção, devido ao seu consenso baseado no PBFT, levando à uma vazão alta com latência relativamente baixa. Embora este trabalho avalie as redes em um contexto IoT, nenhum dispositivo IoT realmente fez parte dos testes, não levando em consideração, portanto, os custos computacionais de tal aplicação.

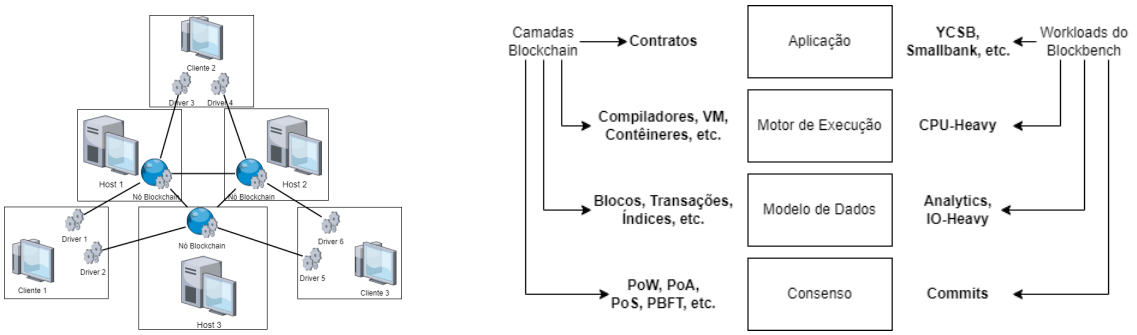
Em [Sun et al. 2018] o Ethereum é utilizado em um sistema de troca de baterias em veículos elétricos. Os sistemas que fazem a troca são representados por Raspberry Pi. Duas configurações de rede são testadas: Uma com os Raspberry Pi fazendo parte da rede como nós leves, e outra com os Raspberry Pi apenas como clientes. O nó minerador fica em um PC. Os testes mostram como a segunda configuração é mais leve que a primeira, com o uso de CPU sendo até 5 vezes menor, o uso da memória em torno de 3 vezes menor e banda de rede utilizada cerca de 10% menor. No entanto, embora o trabalho apresente a comparação da rede com e sem a presença de dispositivos IoT, a vazão e a latência da rede não são medidos, limitando-se a comparação do custo computacional neste nós.

3. Ambiente de Experimentação Aprimorado

A criação e a configuração de redes blockchain envolve a execução de aplicações nos nós que farão parte dela. Cada blockchain possui um ou mais sistemas que são capazes de se integrar nas redes principais e também criar redes privadas. O sistema Geth [Geth 2023], cujo nome é uma abreviação para Go Ethereum, é uma das implementações originais

do Ethereum, ou seja, está presente desde as primeiras versões da rede, tornando-se um dos sistemas mais populares para a mesma[1]. A execução do Geth acontece em duas etapas, uma de preparação e outra de inicialização. A primeira organiza todo o sistema de arquivos do Geth, configurando-o para fazer parte de uma rede. O Geth é usado pelo Blockbench, descrito a seguir, para montar e realizar *benchmarks* na rede Ethereum.

O Blockbench [Dinh et al. 2017] é um *framework* para a realização de *benchmarks* em blockchains privados. Ele possui uma série de *workloads* para avaliar os diversos aspectos da rede. Na raiz da sua arquitetura está a abstração dos sistemas blockchains em quatro camadas: consenso, modelo de dados, motor de execução e aplicação. A camada de consenso tem como papel principal fazer com que os nós concordem com o estado atual da rede, ou em termos práticos, que o conteúdo e a ordem dos blocos seja a mesma em todos os nós, e se um nó adiciona um bloco válido, todos os outros nós também o adicionam nas suas cópias. O modelo de dados define as estruturas de dados e meios utilizados para armazenar as informações da rede. Isto inclui os dados contidos nos blocos e como eles são organizados (Árvore Merkle, Árvore Patricia Merkle, identificadores, etc). O motor de execução representa o mecanismo de execução dos contratos da rede. Neste ponto está a EVM (*Ethereum Virtual Machine*) e o *bytecode* utilizados na Ethereum, juntamente com o conceito de *gas* (monetização). A camada de aplicação representa as diversas aplicações à que o blockchain pode ser empregado.



(a) Arquitetura da rede

(b) Camadas de abstração e *workloads* [Dinh et al. 2017]

Figura 1. Arquitetura e Camadas de abstração na blockchain

Em termos de infraestrutura, o Blockbench possui duas partes principais: 1) a inicialização e preparação das blockchains privadas, e 2) a execução de aplicações nesta rede, que a submetem aos *workloads* configurados no *benchmark*. Ambas são sistematizadas de modo que, após a parametrização do *benchmark*, ele pode ser executado repetidas vezes de forma facilitada. Na inicialização e preparação, têm-se como atores principais múltiplos *hosts* que são inicializados e configurados para conectar-se uns aos outros, formando uma rede *peer-to-peer*, consistindo na blockchain privada. Este processo depende completamente da rede sendo testada, não tendo uma padronização por parte do Blockbench, ou seja, não há mecanismos facilitadores para a inclusão de novos blockchains além dos exemplos das redes já presentes (Ethereum, Hyperledger-Fabric e Quorum). Na segunda parte, de execução das aplicações, tem-se os clientes e os *drivers*. Os clientes

[1]Consulta ao Etherscan (`https://etherscan.io/nodetracker`) em novembro de 2023 mostra um total de aprox. 8,5 milhões de nós Ethereum conectados à rede, dos quais 56,6% utilizam o Geth.

são as máquinas onde os *drivers* são executados, e esses são os responsáveis por submeter as transações na rede. Múltiplos *drivers* podem ser executados num mesmo cliente. Os *drivers* devem possuir capacidade de conexão com a blockchain sendo testada. Para isto o Blockbench inseriu neles camadas de abstração, que podem ser implementadas para o blockchain em questão, reutilizando toda a lógica principal dos mesmos.

A Figura 1(a) mostra o relacionamento entre os atores mencionados, formando a arquitetura de rede do Blockbench. No exemplo, estão presentes três *hosts* e três clientes. Cada cliente possui dois *drivers*, cada um conectado à um *host* diferente. O Blockbench possui um conjunto de *workloads*, cada um voltado ao teste de camadas específicas do blockchain sendo testado [Dinh et al. 2017]. O YCSB (*Yahoo! Cloud Serving Benchmark*) [Dinh et al. 2017], utilizado neste trabalho, é voltado para armazenamento de dados Chave-Valor. O Smallbank pode ser utilizado na simulação de operações básicas em contas bancárias e o CPUHeavy atua na ordenação de vetores, entre outros. A Figura 1(b) mostra para qual camada alguns deles são direcionados. O YCSB, por exemplo, é voltado para análise de aplicações.

A coordenação entre os diversos atores que compõem um *benchmark* do Blockbench acontece através de SSH, utilizado para disparar os comandos necessários para cada *host* e cliente, que, assim, devem possuir acesso aos *scripts* do Blockbench. Logo, remenda-se o uso de diretórios compartilhados na rede. Este trabalho concentrou-se na blockchain Ethereum, com o *driver* e *workload* YCSB, configurado para um cenário com alta quantidade de atualização nos dados. A escolha deste *workload* deve-se a semelhança com as aplicações IoT, onde os dispositivos possuem sensores e enviam aos servidores centrais as atualizações nos valores lidos. A versão original do Blockbench está disponível em `https://github.com/ooibc88/blockbench/`. Entretanto, o código possui características muito focadas nos cenários e infraestruturas utilizados pelos autores da ferramenta, dificultando o uso em outras situações. Assim, várias modificações foram feitas, não apenas para se adaptar ao cenário e infraestrutura usada neste trabalho, como para aumentar a adaptabilidade do *framework*. Para trabalhar nas modificações no Blockbench, criou-se uma derivação do repositório original, disponibilizada no endereço `https://gitlab.com/ppgcomp-blockchain/blockbench`.

Entre as diversas modificações feitas no Blockbench, foi elaborado um conjunto de *Dockerfiles* que geram imagens e contêineres Docker com todas as dependências instaladas, prontos para executar *benchmarks* na rede Ethereum. Entre as dependências, têm-se: a)`restclient-cpp`[2]: deve ser aplicado um *patch* disponível no código-fonte do Blockbench. Posteriormente, deve ser compilado e adicionado ao sistema; b) `libcurl`[3]: também compilada através do código-fonte; c) `Node.js`: necessário para os *workloads* das camadas de Motor de Execução, Modelo de Dados e Consenso (cada *workload* possui suas próprias dependências, que devem ser instalados com `npm install`); e d) Sistemas Blockchain: cada blockchain possui o seu sistema, que deve estar instalado nas máquinas que servirão como *host*. No caso da Ethereum é utilizado o sistema Geth.

Esta quantidade de dependências, com um alto grau de complexidade para a instalação, leva à uma grande barreira inicial para se iniciar no Blockbench. Sendo assim, havia a necessidade de automatizar este processo. A escolha de utilização de imagens

[2]`https://github.com/mrtazz/restclient-cpp`
[3]`https://curl.haxx.se/libcurl/`

Docker se deu pela capacidade de se montar um ambiente virtualizado leve e isolado do sistema hospedeiro. Como vantagem, também há a possibilidade de executar múltiplos contêineres no mesmo sistema hospedeiro, podendo-se criar redes inteiras para teste em uma mesma máquina hospedeira. Foram desenvolvidos *Dockerfiles* para gerar imagens para os dois atores principais da rede do Blockbench, *hosts* e clientes, com o acréscimo do ator coordenador, que é o responsável por coordenar toda sequência de passos do *benchmark*, comunicando-se com os outros nós por meio de `SSH`. Para evitar a repetição de códigos, foram criados *Dockerfiles* auxiliares que servem como base para os *Dockerfiles* principais. O `base.Dockerfile` tem como base a imagem do `ubuntu:18.04`, adicionando apenas o `git` na sua instalação. Isto é recomendado, pois o `git` se torna a forma de replicar as modificações feitas no Blockbench para todos os nós.

O `coordinator.Dockerfile` tem como base o `base.Dockerfile`, adicionando o `curl` e o `jq`. O primeiro é uma ferramenta para se fazer requisições `http`. Já o segundo, permite a leitura e interpretação de arquivos `json` no `bash`. Essas são duas novas dependências para o funcionamento do Blockbench modificado. Este *Dockerfile* também cria um par de chaves criptográficas RSA, sem senha, para serem usadas na comunicação `SSH` do coordenador com os outros nós. O `node.Dockerfile` também tem como base o `base.Dockerfile`. Nele são instalados as ferramentas para a construção e compilação das dependências dos *hosts* e clientes. Também são instaladas as ferramentas para comunicação com o coordenador (`openssh-server`), já expondo a porta padrão do SSH (22), e fazendo as configurações necessárias, que são: senha padrão, permitir acesso `root`, e habilitar ambiente de usuário nas seções (necessário para que os comandos enviados via SSH enxerguem o Geth). O `client.Dockerfile` tem como base o `node.Dockerfile`. Ele instala o `restclient-cpp` e o `node.js`, executa `npm install` nos *drivers* e *workloads* baseados em `node.js` e, por fim, compila os *drivers* dos *workloads* da camada de aplicação. Finalmente, o `host.Dockerfile`, que também tem como base o `node.Dockerfile`, instala as ferramentas da linguagem Go e o sistema de blockchain Ethereum Geth, já expondo à rede as portas necessárias para a comunicação dele com outros sistemas Geth (porta 30303) e a porta utilizada pelo servidor HTTP, criado para servir de interface com a blockchain (8000).

O processo de inicialização da rede, consiste basicamente em inicializar todos os *hosts* e configurá-los para se conectarem entre si através do sistema blockchain utilizado. Este processo teve uma mudança significativa em relação ao Blockbench original. Primeiramente, foi redefinido o arquivo `hosts`, que é uma entrada para o *benchmark*. Ele deve conter uma lista com os endereços IP das máquinas que servirão como *hosts*. No entanto, o comando `ssh` necessita também do nome do usuário na máquina alvo, que estava fixo, sendo colocado explicitamente em todos os *scripts* que utilizavam este comando. Isto dificultava muito o uso do Blockbench em redes customizadas. Portanto, o arquivo `hosts` foi convertido para o arquivo `hosts.csv`. Desta forma, é possível especificar outros parâmetros além do IP para cada *host*, como nome de usuário, portas de comunicação, o tipo do nó na blockchain, e a sua conta principal.

O Geth, no seu processo de preparação, já determina um endereço para o nó, que o identifica na rede e dá o caminho para a comunicação com ele. Desta maneira, o próximo passo de inicialização da rede, é obter este endereço de todos os *hosts* inicializados no passo anterior. Esses endereços são colocados em um arquivo chamado `addPeer.txt`.

Este arquivo é então enviado pelo coordenador à todos os *hosts* para ser utilizado no próximo passo da inicialização.

O último passo é iniciar o Geth em todos os *hosts*. Este procedimento também sofreu uma modificação importante, que agora utiliza o tipo de nó que será instanciado, podendo ser: a) **Nó completo (*full*)**: Este será um nó minerador na rede, ou seja, com capacidade de criação de blocos e com uma cópia completa dos dados da rede; ou b) **Nó leve (*light*)**: Este será um nó leve na rede, ou seja, que não faz a mineração, e possui apenas os cabeçalhos de cada bloco, necessitando da comunicação com um nó completo para submeter transações à rede e também consultá-los.

No Blockbench original, um nó minerador é inicializado com a determinação da criação de oito *threads* específicas para a mineração. No entanto, foram acrescentados até três *threads* para a comunicação com nós *light*, parâmetro necessário no Geth para que os nós *light* consigam fazer parte da rede. Este valor foi determinado empiricamente, para permitir uma boa comunicação. Assim, os endereços contidos no arquivo `addPeer.txt`, gerado no passo anterior, são utilizados para adicionar ao Geth os endereços dos outros nós da rede, formando assim, efetivamente, uma rede *peer-to-peer*. Neste passo também inicializa-se um processo para obtenção de dados de custo computacional da execução do Geth. Este processo pega dados do arquivo `/proc/<pid-geth>/stat` de forma contínua, com intervalos de 1 segundo, de forma a ter a porcentagem de uso da CPU, a quantidade de memória virtual utilizada, e a quantidade de memória física utilizada ao longo do tempo de execução.

Nesta segunda parte do processo de *benchmark* temos a inicialização dos *drivers* dentro de cada cliente. Originalmente, o Blockbench tinha um arquivo `clients`, semelhante ao arquivo `hosts`, que continha uma lista com os endereços IP das máquinas que serão utilizadas como clientes. Este mecanismo apresenta o mesmo problema exposto no arquivo `hosts`, no qual o nome de usuário e a porta para acesso SSH às máquinas estava fixo. Portanto, substitui-se o arquivo `clients` também por uma versão `clients.csv`, que contém os IPs das máquinas, o nome de usuário e a porta de comunicação `SSH`.

Para iniciar os drivers, o Blockbench tinha um comando fixo no script de inicialização, isto fazia com que, para mudar o *driver* a ser utilizado no *benchmark*, fosse necessário mudar o *script* de inicialização, o que não é o ideal, por este arquivo pertencer ao controle de versionamento (`git`) e não ser uma entrada do sistema. Também não havia um controle explícito de qual *host* cada *driver* se conectaria. Assim, foi criado o arquivo `drivers.csv`, que é uma nova entrada dos *benchmarks*. Neste arquivo cada linha representa um *driver* a ser inicializado com os parâmetros determinados nela.

Como é usual na execução de *benchmarks*, por existir um grau de variabilidade nos dados tomados nesses processos, deve-se repeti-los um número de vezes para poder excluir *outliers* e determinar tendências nos dados. Desta forma, o *script* de execução (`run-bench.sh`) foi modificado para receber um parâmetro adicional, que é o número de vezes que se deseja executar o *benchmark*. O script então executa cada uma das repetições na sequência, organizando cada arquivo de *log* em uma pasta com o número da execução, agrupando ao final estas pastas em uma pasta única.

Como visto na descrição de todo o processo, a execução de um *benchmark* é razoavelmente longa, variando ainda com o número de *hosts*. Se incluirmos múltiplas

repetições das execuções, o tempo se escala, tornando oneroso o processo de execução de múltiplos cenários, onde haverá variação dos parâmetros de entrada. Para isto, foi inserido no Blockbench o conceito de cenários. Dentro de uma pasta específica (`/scenarios`) no ambiente do *benchmark* da blockchain Ethereum, é possível criar pastas que representam cenários que se deseja executar. Em cada pasta devem ser criados os arquivos de entradas dos *benchmarks*: `hosts.csv`, `clients.csv`, `drivers.csv`, `contracts.csv` e `genesis.json`. Após isto, pode se rodar o *script* `run-all-scenarios.sh`, que percorre essas pastas, copiando os arquivos de entrada aos seus devidos locais, e executando o *benchmark*. Este processo facilita a execução de múltiplos cenários ao fazê-lo sequencialmente sem exigir ações do usuário.

Todas as modificações feitas no processo de *benchmark* com o Blockbench, resultam em uma contribuição importante deste trabalho, pois a customização e facilidade de uso do *framework* foram aumentadas, facilitando a sua adoção como ferramenta de testes.

4. Avaliação Experimental

Esta seção apresenta os resultados obtidos a partir da execução do experimentos com o Blockbench aprimorado considerando o uso de contêineres Docker e um dispositivo Raspberry Pi 3 em diferentes cenários. Para avaliar a aplicabilidade da blockchain em cenários IoT, e também o impacto do uso de dispositivos IoT em uma rede Blockchain, foram delineados cenários específicos, cada um levando em conta um aspecto do objetivo geral. Os cenários foram executados com dois *hosts* e dois clientes. Cada cliente executa um *driver* e cada *driver* conectando-se à um dos *hosts* realizando processos em um contrato diferente. Implantou-se cada contrato em apenas um *host*, sendo este exatamente ao qual o *driver* que irá manipulá-lo se conectará. Quando integrado à rede, o Raspberry Pi 3 substitui um dos *hosts*. Os demais *hosts* e clientes foram representados por contêineres Docker. Os contêineres, incluindo o contêiner do coordenador, foram executados em uma mesma máquina com Sistema Operacional Ubuntu 22.04 LTS 64 bits, processador Intel i7-12700, 4.9GHz, 64 bits, 12 *cores*, 16GB DDR5 4400MHz de memória, SSD de 512GB PCIe NVM3 M.2 e placa de Rede Gigabit Ethernet. O Raspberry Pi 3 e o PC foram conectados à rede através de cabos Ethernet e um *switch* 100 Mbps.

Considerou-se três variáveis para o delineamento de cenários: i) protocolo de consenso: Ethash ou Clique; ii) taxa de transações: 1, 10 e 100 transações por segundo; e iii) arquitetura dos nós: nós leves e nós completos, contêineres Docker e o Raspberry Pi 3. Inicialmente, foram considerados seis cenários-base, que correspondem aos testes executados com o protocolo Ethash com um nó leve e um nó minerador, ambos executando em contêineres Docker. Em seguida, o nó leve passou a ser executado no Raspberry Pi 3 e, incluindo também os testes com o protocolo de consenso Clique, ainda com a presença do Raspberry Pi 3. Todos foram executados com taxas de 1, 10 e 100 transações por segundo. As demais combinações podem ser encontradas em [Silva 2023].

O primeiro cenário representa o uso do Blockchain em arquiteturas com alto poder de processamento. A presença de um nó leve visa comparar de forma justa o uso de dispositivos IoT na rede, representado por um Raspberry Pi 3, que não possui recursos computacionais suficientes à execução de nós mineradores. Já o último cenário, com a troca do protocolo de consenso, visa mostrar o seu impacto no desempenho da rede e como aplicações Blockchain diante de dispositivos IoT podem se beneficiar de protocolos

alternativos. Neste caso, utilizou-se o Clique pela sua disponibilidade e facilidade de uso no Geth. A variação na taxa de transações objetiva a saturação da rede, onde ela deixa de ser capaz de processar as transações na velocidade necessária.

Na análise e comparação entre os diversos cenários executados no trabalho foram empregadas métricas para medir o desempenho da rede e o custo computacional. As métricas de desempenho são: a) Transações finalizadas (T): total de transações finalizadas no *benchmark*; b) Vazão (V): total de transações executadas por segundo, também sumarizadas pela sua média ($\bar{V}$) e desvio padrão (V_s); c) Latência (L): tempo de execução de uma transação, em segundos, também com sua média ($\bar{L}$) e desvio padrão (L_s); e d) Transações pendentes (P): total de transações pendentes ao final do *benchmark*, dado que também foi tomado em intervalos de um segundo (P_t), juntamente com sua média ($\bar{P}$), seu desvio padrão (P_s) e sua tendência linear (P').

O total de transações finalizadas fornece uma visão geral do desempenho da aplicação no cenário em questão, guiando o restante das análises. Já a métrica de transações pendentes afere questões de saturação na rede. Dentre estas, a tendência linear $P' = \frac{\sum(t-\bar{t})(P_t-\bar{P})}{\sum(t-\bar{t})^2)}$ foi calculada como a inclinação da regressão linear dos números de transações pendentes ao longo do tempo (P_t), sendo t e $\bar{t}$ o instante das amostras e a média de tempo, respectivamente. Este indicador possibilita averiguar o nível de saturação da rede. Valores de P negativos indicam que a rede tem capacidade superior ao que está sendo exigido. Os valores próximos de zero indicam que a rede tem capacidade para processar o volume de requisições sendo submetidas. Os valores positivos indicam um nível de saturação na rede, no qual as transações pendentes estão se acumulando ao longo do tempo. Neste trabalho, considera-se $P > 0,005$ como indicação de saturação na rede.

A fim de diminuir a influência de fatores externos, executou-se cada cenário cinco vezes. Os valores de média, desvio padrão e tendências, citados acima, foram calculados para cada execução, sendo considerada a média desses valores. Quanto a análise de custo computacional, têm-se: a) porcentagem de uso da CPU, levando em consideração todos os *cores* ($\%CPU$); b) ocupação da memória virtual em MB ($VSIZE$); e c) ocupação da memória física em MB (RSS). Todos estes são tomados ao longo do tempo com um intervalo de um segundo entre cada medida. Os resultados obtidos na execução dos cenários, e as planilhas completas para análise deles estão disponíveis de forma pública[4].

4.1. Raspberry Pi 3 como nó leve

O Etash é um protocolo de consenso PoW, ou seja, sua execução consome grandes quantidades de recursos computacionais, neste caso, memória. Porém, ele aceita a categoria de nós leves que, mesmo com recursos restritos, ainda podem participar da rede desempenhando um papel limitado. Isto permitiria o uso de dispositivos IoT na rede.

Para verificar o impacto do uso destes dispositivos, elaborou-se dois cenários, ambos com um nó minerador, um nó leve e dois clientes, cada um conectado à um dos nós. No primeiro cenário, os dois nós são executados em contêineres Docker, com o Host 2 sendo um nó leve. No segundo cenário, o contêiner Docker do Host 2 foi substituído por um Raspberry Pi 3. É possível perceber, no gráfico comparativo de transações finalizadas

[4]https://unioestebr-my.sharepoint.com/:f:/g/personal/maykon_silva_unioeste_br/EtSqVAdg_DBJrJhhee3fzfwBfPbRZjlqwNgOgjFbmRa4VQ?e=3bsJtO

da Figura 2, que ocorre a saturação do nó minerador, no qual as transações do nó leve são ignoradas nos cenários com taxa de 10 transações por segundo. Entretanto, o nó minerador tem um desempenho melhor no cenário com o Raspberry Pi 3. Isto deve-se ao fato destes cenários possuírem apenas um contêiner sendo executado no computador, liberando recursos computacionais para a sua execução.

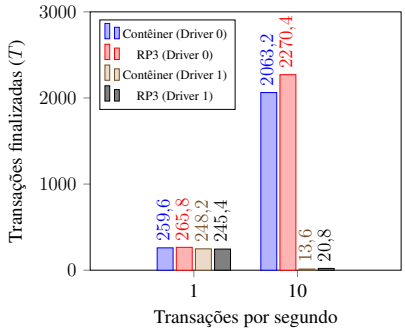


Figura 2. Número de transações finalizadas (T) entre os cenários com nó leve em contêiner Docker e Raspberry Pi 3 (RP3)

Já nos cenários completamente válidos (uma transação por segundo), o número de transações finalizadas nos *drivers* conectados aos nós leves (*Driver* 1) é bem próximo, permitindo análises mais profundas. Para isto, a Tabela 1 mostra as estatísticas de transações pendentes. Pode-se observar que houve uma melhora em relação aos cenários com o nó leve sendo executado no contêiner. Porém, esta melhora é justificada pelo maior poder do nó minerador, não estando relacionada ao uso do Raspberry Pi.

Tabela 1. Número (P) e tendência (P') das transações pendentes nos cenários com contêiner Docker e RP3 com 1 transação por segundo

	Driver 0		**Driver 1**	
Cenário	P	P'	P	P'
Contêiner	11,4	0,010	11,4	0,010
RP3	8,2	0,007	9,8	0,003

Observando a vazão e latência apresentados na Tabela 2, nota-se que o comportamento se manteve semelhante em ambos os cenários, tendo uma latência alta e com alta variabilidade, indicada pelos desvios-padrões (V_s e L_s). [Perez et al. 2019] cita estas características como indesejadas para o IoT, que nos sistemas de tempo real precisam de uma alta previsibilidade da rede (desvios-padrões baixos) e que suas transações sejam finalizadas poucos milissegundos após o envio (baixa latência).

Tabela 2. Dados de vazão e latência dos cenários com contêiner Docker e RP3 com uma transação por segundo

	Driver 0				**Driver 1**			
Cenário	V	V_s	L	L_s	V	V_s	L	L_s
Contêiner	0,953	1,711	10,384	3,629	0,952	1,711	10,347	3,623
RP3	0,967	1,928	11,655	4,339	0,940	1,830	11,955	4,350

Seguindo com a análise, examina-se o desempenho individual dos dispositivos na execução de suas tarefas. Para isto, são utilizadas as métricas de custo computacional,

utilizando-as para comparar o nó leve executando no Raspberry Pi com a execução dele no contêiner Docker. Os gráficos da Figura 3 apresentam essa comparação. Os dados são referente ao cenário com taxa de uma transação por segundo, por ser o único sem saturação significativa nestas configurações. Observa-se durante a sincronização como o processamento é mais pesado para o Raspberry Pi, mantendo o processamento em 100% durante quase todo o tempo de execução. Percebe-se este mesmo comportamento nas operações de desbloqueio das contas. Os picos nestes momentos são maiores, atingindo valores acima dos 100% e ainda durando mais tempo. O processo no contêiner dura em torno de três segundos, enquanto que no Raspberry Pi fica ao redor de 15 segundos.

Os dados de consumo de memória foram omitidos do gráfico. No entanto, o Raspberry Pi teve menor consumo de memória virtual, sendo 40% menor na fase de sincronização. Nas fases de *deploy* de contratos e *workloads*, o consumo é 60% menor. Em relação à memória física, o comportamento é semelhante.

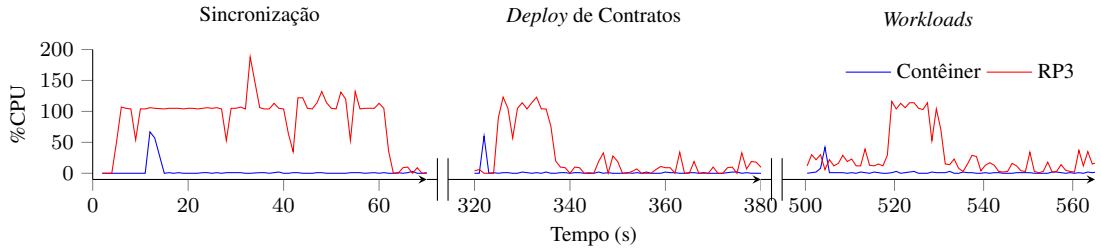


Figura 3. Custo computacional para nós leves com RP3 e contêiner Docker

Com estes cenários, conclui-se que o Raspberry Pi 3 pode ser usado como um nó leve integrante da rede. Porém, o mecanismo de autenticação deve ser levado em conta. Em arquiteturas como a utilizada neste trabalho, nas quais a autenticação (desbloqueio) acontece apenas uma vez por conta, este quesito diminui em importância. Entretanto, nas arquiteturas que exijam autenticação a cada operação, o uso do Raspberry Pi deve ser melhor avaliado. Além disso, embora o Raspberry demonstre grande esforço para desempenhar parte das tarefas que um cliente leve pode exercer, isto não é refletido no desempenho geral da rede. A dificuldade nestes cenários não foi causado por sua presença, mas sim pelas características do protocolo de consenso Ethash, que leva à uma alta latência, baixa previsibilidade no comportamento, e fácil saturação da rede. Logo, fez-se uma comparação com outros protocolos de consenso, como o Clique, apresentando a seguir.

4.2. Protocolo de consenso Clique

Nos cenários a seguir, substituiu-se o protocolo Ethash dos cenários da seção anterior pelo Clique, que é um protocolo de consenso desenvolvido para ser utilizado nas redes de testes da Ethereum que não possuem dispositivos com alto poder computacional. Ele é um protocolo da categoria *proof-of-authority* (PoA), onde apenas um conjunto de nós são autorizados a anexarem blocos na rede, portanto, não tendo a competição e alto gasto computacional dos protocolos PoW. Desta forma, ele se apresenta como uma possível alternativa para uso em aplicações Blockchain com a presença de dispositivos IoT.

O gráfico da Figura 4 exibe o número de transações finalizadas em cada nó dos cenários com o nó leve sendo desempenhado pelo Contêiner e pelo Raspbery Pi 3. Comparando com o Ethash (Figura 2) observa-se que a rede continua com capacidade su-

ficiente para processar uma transação por segundo. Com 10 transações por segundo, observa-se uma melhora significativa, indicando que a saturação da rede não foi atingida. Já com 100 transações por segundo, nota-se que as transações dos nós leves não foram processadas pelos nós com autoridade, apontando a saturação.

Comparando o desempenho da rede entre os cenário com Contêiner e Raspberry Pi 3, nota-se que o Raspberry Pi como nó leve tem um desempenho levemente inferior, tendo 12% menos transações finalizadas (em média) nos cenários com 10 transações por segundo. Como o esperado, a saturação no cenário com 100 transações por segundo permaneceu, já que é uma limitação de processamento da autoridade.

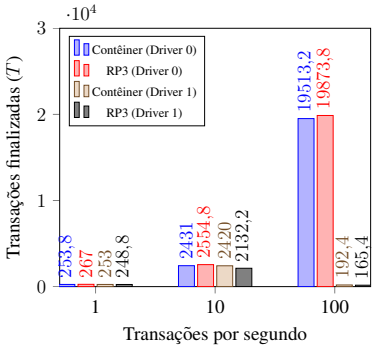


Figura 4. Número de transações finalizadas nos cenários nó leve em contêiner Docker e em Raspberry Pi 3 (RP3)

A Tabela 3 mostra as estatísticas de transações pendentes dos cenários considerando 1 e 10 transações por segundo. Observa-se que os valores não mostram impacto perceptível do uso do Raspberry Pi 3. A Tabela 4 apresenta os dados da latência e vazão. Nota-se, na comparação entre os dados de desvio-padrão dos cenários com uma transação por segundo com os obtidos nos cenários com protocolo Ethash (Tabela 2), que o Clique proporciona uma estabilidade maior, contando ainda com uma latência média menor, tanto nas transações enviadas ao nó autoridade, comparadas com o nó minerador, quanto nas enviadas ao nó leve.

Tabela 3. Número (P) e tendência (P') das transações pendentes nos cenários com protocolo de consenso Clique e com 1 e 10 tx/s

		Driver 0		Driver 1	
Tx	**Cenário**	P	P'	P	P'
1	**Contêiner**	6,2	0,0005	6,0	0,0010
	RP3	6,0	0,0004	6,2	0,0001
10	**Contêiner**	64,4	-0,003	63,8	0,0010
	RP3	61,0	0,0011	57,2	0,0004

Para a análise de custo computacional, comparou-se o custo entre o nó leve executando no contêiner Docker e o nó leve executando no Raspberry Pi. Os gráficos da Figura 5 mostram essa comparação. Os dados são referente aos cenários com taxa de 10 e 100 transações por segundo. Na sincronização e *deploy* de contratos, o processamento é baixo em todos os cenários, mostrando apenas a característica, já percebida em outros cenários, do processamento pesado para desbloqueio de contas pelo Raspberry Pi 3. É

Tabela 4. Vazão e latência com o protocolo de consenso Clique com 1 e 10 tx/s

		Driver 0				Driver 1			
Tx	**Cenário**	V	V_s	L	L_s	V	V_s	L	L_s
1	**Contêiner**	0,969	0,174	7,020	0,277	0,969	0,172	6,820	0,255
	RP3	0,971	0,168	6,909	0,272	0,953	0,211	6,890	0,285
10	**Contêiner**	9,279	2,034	7,244	0,320	9,272	1,679	7,144	0,289
	RP3	9,290	1,621	7,124	0,289	8,169	2,346	7,420	0,344

interessante observar o processamento na fase de *workloads*, que mostra como o custo de processamento é maior no Raspberry Pi. No cenário com 100 Transações por segundo, ele já atinge picos perto dos 400% de processamento. No uso de memória, o padrão se assemelha aos cenários anteriores, com o Raspberry ocupando menos memória que o contêiner, tanto virtual quanto física.

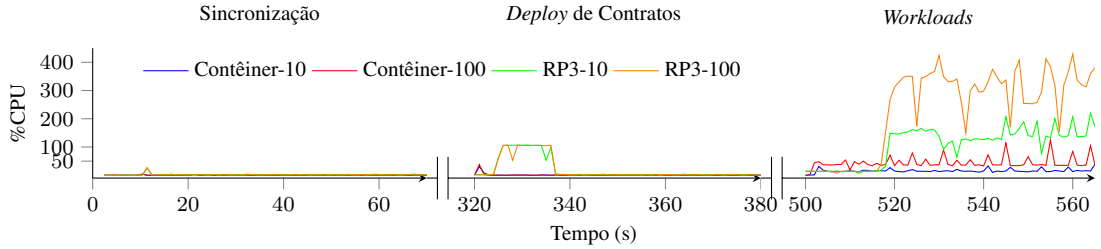


Figura 5. Custo computacional com protocolo de consenso Clique

Neste último cenário, fica evidente que o Raspberry Pi 3 pode participar nas redes com protocolo Clique em um determinado nível de carga de transações. Além disso, há uma faixa maior de trabalho, podendo ultrapassar as 10 transações por segundo.

5. Conclusão

Este trabalho explorou a aplicabilidade do Blockchain Ethereum em dispositivos IoT, integrando-os à rede. Os testes revelaram a necessidade de uma ferramenta configurável para facilitar a coleta de dados, destacando o Blockbench como uma opção amplamente utilizada. Para maior flexibilidade e facilidade de uso, o benchmark foi amplamente aprimorado. Os resultados dos testes mostraram os impactos de nós leves e dispositivos IoT nas redes Ethereum, revelando considerações cruciais para sua viabilidade. A comparação dos protocolos Ethash e Clique indicou benefícios de desempenho do Clique, especialmente para aplicações com taxas de transação menores.

Como trabalhos futuros, pretende-se realizar testes em dispositivos IoT com menor poder computacional, como Raspberry Pi Pico, Arduino Nano e ESP32. Além disso, realizar comparações com outros Blockchains, como Hyperledger Fabric, para identificar características positivas e/ou negativas em aplicações IoT.

Agradecimentos

Este trabalho foi parcialmente financiado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Ao LAPP Multi (Laboratório de Processamento e Prototipação Multiusuário) da Unioeste pela disponibilidade dos recursos utilizados nas simulações.

Referências

Chen, P., Han, D., Weng, T.-H., Li, K.-C., and Castiglione, A. (2021a). A novel byzantine fault tolerance consensus for green iot with intelligence based on reinforcement. *Journal of Information Security and Applications*, 59:102821.

Chen, X., Nakada, R., Nguyen, K., and Sekiya, H. (2021b). A comparison of distributed ledger technologies in iot: Iota versus ethereum. In *20th Int'l Symp. Commun. Information Tech. (ISCIT)*, pages 182–187.

Dinh, T. T. A., Wang, J., Chen, G., Liu, R., Ooi, B. C., and Tan, K.-L. (2017). Blockbench: A framework for analyzing private blockchains. In *Proc. ACM Int'l Conf. Management of Data*, SIGMOD '17, page 1085–1100, New York, NY, USA. ACM.

Fu, X., Wang, H., Shi, P., and Zhang, X. (2022). Teegraph: A blockchain consensus algorithm based on tee and dag for data sharing in iot. *J. Syst. Archit.*, 122:102344.

Geth (2023). go-ethereum: Official go implementation of the ethereum protocol. Disponível em: `https://geth.ethereum.org`. Acesso em: 04/Nov/2023.

Ghiro, L. et al. (2021). A blockchain definition to clarify its role for the internet of things. In *2021 19th MedComNet*, pages 1–8.

Perez, M. R. L., Lagman, A. C., Legaspi, J. B. C., De Angel, R. D. M., and Awat, K. A. S. (2019). Suitability of iot to blockchain network based on consensus algorithm. In *2019 IEEE 11th HNICEM*, pages 1–5.

Raghav, Andola, N., Venkatesan, S., and Verma, S. (2020). Poewal: A lightweight consensus mechanism for blockchain in iot. *Pervasive and Mobile Computing*, 69:101291.

Sanju, S., Sankaran, S., and Achuthan, K. (2018). Energy comparison of blockchain platforms for internet of things. In *IEEE iSES*, pages 235–238.

Sankaran, S., Pramod, N., and Achuthan, K. (2019). Energy and performance comparison of cryptocurrency mining for embedded devices. In *9th Int'l Symp. Embedded Comp. and Syst. Design (ISED)*, pages 1–5.

Sankaran, S., Sanju, S., and Achuthan, K. (2018). Towards realistic energy profiling of blockchains for securing internet of things. In *IEEE 38th ICDCS*, pages 1454–1459.

Silva, M. V. (2023). *Avaliação de Desempenho e Custo Computacional na Utilização da Blockchain Ethereum em Dispositivos de Internet das Coisas*. Mestrado em ciência da computação, Unioeste, Cascavel, PR. `https://tede.unioeste.br/handle/tede/6734`.

Sun, H., Hua, S., Zhou, E., Pi, B., Sun, J., and Yamashita, K. (2018). *Using Ethereum Blockchain in Internet of Things: A Solution for Electric Vehicle Battery Refueling*, pages 3–17. Springer, Cham.

Vashi, S., Ram, J., Modi, J., Verma, S., and Prakash, C. (2017). Internet of things (iot): A vision, architectural elements, and security issues. pages 492–496.

Wang, L., Sun, X., Jiang, R., Jiang, W., Zhong, Z., and Kwan Ng, D. W. (2021). Optimal energy efficiency for multi-mec and blockchain empowered iot: a deep learning approach. In *IEEE Int. Conf. Commun. (ICC)*, pages 1–6.

Wu, H., Wolter, K., Jiao, P., Deng, Y., Zhao, Y., and Xu, M. (2021). Eedto: An energy-efficient dynamic task offloading algorithm for blockchain-enabled iot-edge-cloud orchestrated computing. *IEEE Internet of Things Journal*, 8(4):2163–2176.

Yang, L., Li, M., Zhang, Y., Si, P., Wang, Z., and Yang, R. (2020). Resource management for energy-efficient and blockchain-enabled industrial iot: A drl approach. In *2020 IEEE 6th Int. Conf. Comput. and Commun. (ICCC)*, pages 910–915.

Yi, H. and Wei, F. (2019). Research on a suitable blockchain for iot platform. In Patnaik, S. and Jain, V., editors, *Recent Developments in Intelligent Computing, Communication and Devices*, pages 1063–1072, Singapore. Springer Singapore.

Zheng, Z., Xie, S., Dai, H.-N., Chen, X., and Wang, H. (2018). Blockchain challenges and opportunities: a survey. *Int. J. of Web and Grid Services*, 14(4):352–375.