UNIVERSIDADE FEDERAL DE OURO PRETO
ESCOLA DE MINAS
COLEGIADO DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO - CECAU

MATEUS NAZÁRIO COELHO

UMA ABORDAGEM COM BLOCKCHAIN PARA O PROBLEMA DE SEGURANÇA EM ROBÓTICA MÓVEL

MONOGRAFIA DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Ouro Preto, 2022

MATEUS NAZÁRIO COELHO

UMA ABORDAGEM COM BLOCKCHAIN PARA O PROBLEMA DE SEGURANÇA EM ROBÓTICA MÓVEL

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Controle e Automação.

Orientador: Prof. Bruno Nazário Coelho, Ph.D.

Ouro Preto
Escola de Minas – UFOP
2022

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

C672a Coelho, Mateus Nazário.
Uma abordagem com blockchain para o problema de segurança em robótica móvel. [manuscrito] / Mateus Nazário Coelho. - 2022.
41 f.: il.: color., tab.. + Código de computador.

Orientador: Prof. Dr. Bruno Nazário Coelho.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Escola de Minas. Graduação em Engenharia de Controle e Automação .

1. Robot Operating System (ROS). 2. Blockchains (Base de dados). 3. Blockchains (Base de dados) - Oráculos. 4. Rede de computador - Protocolos - Contratos Inteligentes. 5. Robótica. 6. Blockchains (Base de dados) - Segurança da Informação. I. Coelho, Bruno Nazário. II. Universidade Federal de Ouro Preto. III. Título.

CDU 681.5

Bibliotecário(a) Responsável: Maristela Sanches Lima Mesquita - CRB-1716

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA CONTROLE E AUTOMACAO

FOLHA DE APROVAÇÃO

Mateus Nazário Coelho

Uma abordagem com blockchain para o problema de segurança em robótica móvel

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Engenheiro de Controle e Automação

Aprovada em 26 de outubro de 2022

Membros da banca

[Doutor] - Bruno Nazário Coelho - Orientador (Universidade Federal de Ouro Preto)
[Doutor] - Agnaldo José da Rocha Reis (Universidade Federal de Ouro Preto)
[Engenheiro] - Jacó Domingues (Instituto Tecnológico Vale)

Bruno Nazário Coelho, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 14/11/2022

Documento assinado eletronicamente por **Bruno Nazário Coelho**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 18/11/2022, às 11:43, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0418402** e o código CRC **4DDAEB4E**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591533 - www.ufop.br

Este trabalho é dedicado aos meus pais, irmãos, sobrinha, família, amigos e à todos que emitem um pouco de luz diariamente para os que estão ao seu redor. Nada disso seria possível sem vocês.

AGRADECIMENTOS

Agradeço meus pais, João e Gessi, por me darem as melhores condições possíveis para que eu pudesse me destacar, além do apoio incondicional. Meus irmãos: Bruno, Igor, Vitor e Vinicius, pelos papos acadêmicos, filosóficos e boêmios, e pela união constante e inabalável que sempre tivemos. Às minhas escolas: Tomás Antônio Gonzaga e IFMG, pelos professores e ambiente de qualidade e referência no âmbito público. À Ufop, pela luta e manutenção de um ensino público superior de qualidade. Aos meus orientadores: Bruno, Igor, Vitor, Marcone e Héctor, obrigado pela paciência, ensinamento diário e por acreditarem no meu trabalho. Aos meus amigos do Wii: pela amizade que transcende fronteiras. Aos grandes amigos e colegas do ITV: Mário, Jacó, Maurício, Fred, Bidas, Alexandre, Nilton e Filipe, por me ensinarem a como montar um robô. Aos meus irmãos do CVR: por todos os momentos de aprendizado e reflexão. À Roberta: por acreditar e me motivar em todos os momentos humanamente possíveis. Aos meus amigos e amigas: Thays, Cris, Kamila, Flávia, Claudiane, Ian, Brendow, Thiago, Felipe, Barbosa, Koda, Arilton, Enrico, Tomás, Cristiano, Nelson, Dauberson, Heitor, Lucas Felipe, Andressa, Davi e Rodrigo, obrigado pela preocupação diária e por ouvirem meus devaneios. Agradeço o apoio do NeoResearch, da FAPEMIG (edital universal - projeto APQ-02463-21), Instituto Tecnológico Vale, CNPq, Fundação Christiano Ottoni, IFMG e UFOP. Para todos aqueles que mencionei e que eu possa ter esquecido no momento, meu muito obrigado. Esta monografia é por vocês e para vocês!

“"Never study to be successful, study for self efficiency. Don’t run behind success. Follow behind excellence, success will come all way behind you.” (Aamir Khan)

RESUMO

A utilização de robôs tem se tornado cada vez mais presente no cotidiano, possuindo perspectivas positivas em seu uso e crescimento. A robótica moderna é composta por uma diversidade de hardware e softwares. Desta maneira, o *Robot Operating System* (ROS) é um framework de desenvolvimento e integração de sensores para robôs. Neste trabalho, propõe-se uma ponte entre o ROS e a blockchain. Em especial, utiliza-se camadas de segurança na disponibilização dos dados para os oráculos de blockchain, os quais fornecem dados externos para a blockchain a partir de um conjunto de regras precisas criadas pelos contratos inteligentes. De maneira inovadora, apresenta-se uma topologia de rede de robôs conectada a internet via satélites através da Starlink, enquanto a rede descentralizada da blockchain utiliza uma internet comercial. A blockchain do NEO foi escolhida para o estudo-de-caso, sendo o contrato inteligente desenvolvido em C#. Este estudo explora os limites de sincronia desta topologia proposta, bem como valida os dados que foram registrados na blockchain através dos oráculos. Os resultados obtidos mostram que foi possível validar boa parte das leituras em cenários locais e remotos de alta latência como a via satélite do Starlink. Esses resultados são frutos da assincronia e complexidade de todos os processos e sistemas envolvidos, podendo ser otimizados a partir de uma seleção mais efetiva de parâmetros da rede e do ROS. Este trabalho apresenta interfaces de interação de forma a prover uma simulação de um sistema de auditoria confiável e descentralizado, de forma inovadora e prática. Por fim, uma série de possíveis utilizações industriais e de uso cotidiano do sistema proposto são discutidas e apresentadas ao longo deste trabalho.

Palavras-chaves: ROS. Blockchain. Oráculos de Blockchain. Contratos Inteligentes. Robótica. Segurança da Informação.

ABSTRACT

The use of robots has become increasingly present nowadays, with positive perspectives in its use and growth. Modern robotics are build up of a diversity of hardware and software. In this way, the Robot Operating System (ROS) is a framework for developing and integrating sensors for robots. In this work, a bridge between ROS and blockchain is proposed. In particular, security layers are used to make data available to blockchain oracles, which provide external data to the blockchain from a set of precise rules created by smart contracts. In an innovative way, a robot network topology connected to the internet via satellites through Starlink is presented, while the decentralized blockchain network uses a commercial internet. The NEO blockchain was chosen for the case study, while the proposed smart contract was developed in C#. This study explores the synchrony limits of the proposed topology, as well as validates the data that was recorded on the blockchain through the use of oracles. The results show that it was possible to validate most of the sensor reading values in local and high latency remote scenarios with the Starlink system. These results come from the effectiveness and complexity of all processes and developed systems, and can be optimized to a more efficient management of all parameters of the network and ROS systems. This work presents interfaces which simulates a reliable and decentralized auditing system, in an innovative and practical way. Finally, a series of possible industrial and everyday uses of the proposed system are discussed and presented throughout this work.

Key-words: ROS. Blockchain. Blockchain Oracles. Smart Contracts. Robotics. Information Security.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ROS	Robot Operating System
DDoS	Denial-of-service attack
API	Application Programming Interface
REST	Representational State Transfer
SLAM	Simultaneous Localization and Mapping
URDF	Unified Robot Description Format
CAD	Computer-Aided Design
JSON	JavaScript Object Notation

SUMÁRIO

1 INTRODUÇÃO

A utilização de robôs para a execução de tarefas diárias e trabalhos complexos tem se tornado cada vez mais presente nas indústrias, podendo contribuir positivamente na produtividade e eliminando a presença de humanos em ambientes potencialmente agressivos (MURPHY, 2014). Robôs têm se tornado mais acessíveis para o uso cotidiano, todavia, devido a complexidade de componentes e ferramentas, em geral, requer-se mão de obra especializada na sua construção e utilização.

Desta forma, o desenvolvimento de uma plataforma robótica de alto nível (AZPURUA et al., 2019), que desempenhe diversas tarefas, requer conhecimentos em diversas áreas, tanto para o desenvolvimento do hardware embarcado (o qual pode possuir diversos sensores) quanto para o software. O último serve como camada de integração de drivers, sensores e também para a disponibilização de uma interface comunicável com o usuário.

O Robot Operating System (ROS) (QUIGLEY et al., 2009) é um meta sistema operacional e um framework de código aberto desenvolvido com o intuito de diminuir o tempo dedicado para reimplementar e integrar as diversas camadas de software presentes em uma plataforma robótica. De forma sucinta, o ROS funciona como um framework que integra diversos processos através da comunicação entre nós. Logo, pode-se dividir as tarefas de criação de um sistema robótico de maneira que o ROS integre as partes em uma linguagem em comum, compatível com distintos hardwares que irão gerir ou controlar o sistema.

Nos últimos anos, um tópico que vem sido bastante discutido entre os pesquisadores que utilizam o ROS é sobre a segurança embutida do sistema, desde a invasão de robôs em redes não privadas (DEMARINIS et al., 2019) e a injeção de robôs maliciosos em enxames de robôs (FERRER et al., 2022). De acordo com Dieber et al. (2017), diversas propostas de implementação e melhoria de segurança já foram feitas, como: a introdução de um servidor de autenticação, para rastrear quais nós se comunicam com quais tópicos e serviços; e mudanças na camada de comunicação de forma a proteger o sistema de ataques de negação de serviço (DDoS). Porém, um tópico surge em comum nos trabalhos mais recentes: a utilização de blockchain para melhoria de segurança e transmissão de dados (FERRER; HARDJONO; PENTLAND, 2019).

Originalmente proposta por Satoshi Nakamoto (NAKAMOTO, 2008), a tecnologia Blockchain surgiu com o intuito de resolver problemas de confiança e centralização das instituições financeiras, viabilizando a criação de moedas puramente eletrônicas e descentralizadas capazes de suportar ataques de gasto duplo (do inglês, *double spending*). No Bitcoin, seu consenso para consolidação de dados é feito por base de um processo de força bruta e prova de trabalho (do inglês, *Proof of Work*), que consiste em encontrar *hashes* criptográficos que satisfaçam certo grau de dificuldade, ficando mais difíceis à medida em que a rede tem mais poder

computacional. A evolução natural da blockchain foi a possibilidade de criar códigos computacionais Turing-completos, conhecidos como Contratos Inteligentes (HONGFEI; ZHANG, 2018), para implantação em redes descentralizadas *trustless* (termo com tradução em português para *confiança sem confiança*). Com o passar dos anos e o desenvolvimento de novas gerações de blockchains, outras tecnologias surgiram dentro do âmbito, como: contratos inteligentes para a criação de código genéricos em distintas linguagens; novos algoritmos de consenso; suporte a armazenamento de dados descentralizados; oráculos para comunicação com a web tradicional (conhecida como *Web 2*); e melhorias de desempenho (GUO; YU, 2022).

Este trabalho propõe um pacote público que oferece uma ponte de comunicação entre o sistema ROS e uma blockchain, de forma a prover um sistema de log seguro no formato *append-only*. Em particular, utiliza-se da tecnologia da blockchain Neo (HONGFEI; ZHANG, 2018) como estudo de caso, a qual contém os componentes necessários para o desenvolvimento de um contrato inteligente em C#, bem como oráculos para integração com a *Application Programming Interface* (API) do robô criado.

1.1 Justificativa do trabalho

O ROS trouxe uma grande inovação para os entusiastas e pesquisadores da área de robótica, mas possui uma falta de design para segurança no sistema. Algumas ferramentas básicas de segurança são encontradas, mas essas focam mais em falhas do sistema, como sincronização de serviços, e não há medidas diretas para ataques à comunicação externa do sistema, como *Man in the Middle*. Desta forma, este trabalho busca trazer uma solução descentralizada através do uso da blockchain para a transmissão e alocação segura de dados, em especial para eventos de logs do tipo *append-only*.

1.2 Objetivos gerais e específicos

1.2.1 Objetivo geral

O objetivo geral deste trabalho é desenvolver um pacote ROS integrado a uma blockchain para resolver problemas de transmissão e alocação segura de dados de uma plataforma robótica.

1.2.2 Objetivos específicos

- Estudo de trabalhos recentes envolvendo tecnologias blockchain e robótica;
- Desenvolvimento do pacote ROS e uma ponte para transmissão de dados para a WEB, em especial, utilizando *Representational State Transfer* (REST).
- Utilização das ferramentas da blockchain do Neo para a criação de contratos inteligentes e comunicação com o robô via oráculos;

- Validação dos resultados a partir de comparações com redes locais e uma topologia de satélites, utilizando a Starlink.

1.3 Estrutura do trabalho

O Capítulo 1 introduz a importância do ROS no contexto do desenvolvimento da robótica, os problemas de segurança encontrados e como eles podem ser solucionados com a blockchain.

O Capítulo 2 enfatiza discussões sobre trabalhos relacionados, estudo do ROS, algumas soluções já propostas para as falhas de segurança e um estudo sobre a blockchain Neo.

O Capítulo 3 descreve o desenvolvimento do pacote ROS criado, os protocolos utilizados para comunicação com a rede NEO e o desenvolvimento do contrato inteligente.

O Capítulo 4 contém os resultado obtidos do pacote através da comparação de resultados com outros trabalhos, além de uma análise da transmissão de dados, packet loss e a utilização de uma rede local ou testnet do Neo.

O Capítulo 5 traz as conclusões, contribuições e sugestões para melhoria dos resultados obtidos com o trabalho.

2 REVISÃO DA LITERATURA

Para este trabalho, buscamos trazer como trabalhos relacionados o estado da arte de sistemas robóticos e também sobre blockchain, bem como seu uso em conjunto.

2.1 ROS

2.1.1 Background

O *Robot Operating System* (ROS) é um meta sistema operacional de código aberto para robôs, possuindo serviços esperados de um sistema operacional comum, como: abstrações de hardware; controle de dispositivos de baixo nível; passagem de mensagens entre processos; e manutenção de pacotes. Ele provê as informações coletadas entre os sensores através do padrão *publish-subscribe*, em tópicos, serviços e ações.

Além disso, o ROS oferece diversas ferramentas relacionadas a visualização de dados, como o *Rviz*, que ajuda o desenvolvedor a visualizar como o robô está interpretando os dados dos sensores, além de ser útil para depurar códigos. Conforme citado por Cid et al. (2020), a utilização de diversas bibliotecas públicas do ROS facilitam algumas tarefas ardilosas como: reconstrução de mapas a partir de dados coletados pelos sensores em uma plataforma robótica (Figura 1); execução prática de fusão sensores através de *Simultaneous Localization and Mapping* (SLAM); e utilização de simulação para facilitar o ciclo de desenvolvimento de uma plataforma robótica. O ROS também possui a capacidade de armazenar e executar dados gravados através da API do *rosbag*, que serializa os dados de interesse com alta performance, permitindo assim a possibilidade de gravar dados em um ambiente real e depois replicar em um ambiente simulado.

Outro conjunto de ferramenta importante que o ROS disponibiliza é o *TF Tool*, que permite o usuário acompanhar e transformar os conjuntos de coordenadas e eixos através de pontos, vetores e planos. Através da *Unified Robot Description Format* (URDF) é possível descrever o conjunto de juntas, propriedades e links de uma plataforma robótica disponibilizadas através do TF (Figura 2). A partir desta descrição URDF do robô, é possível reconstruir a plataforma real através do uso de *Computer-Aided Design* (CAD), como exemplificado na Figura 3.

A estrutura de uma topologia simples do ROS comunicando através do sistema de tópicos pode ser observada na Figura 4. Os Nós[1] são como processos que realizam tarefas dentro do ecossistema ROS, identificados na imagem como o */talker*, *rosbridge_websocket* e *rosapi*. Tópicos[2] transmitem informação entre nós através de um tipo de mensagem, pode ser identificado na Figura 4 como o *txt_msg*.

[1] ROS Nodes: http://wiki.ros.org/Nodes
[2] ROS Topics: http://wiki.ros.org/Topics

Figura 1 – Comparação do robô real dentro de uma mina em Ouro Preto e sua simulação referente (CID et al., 2020).

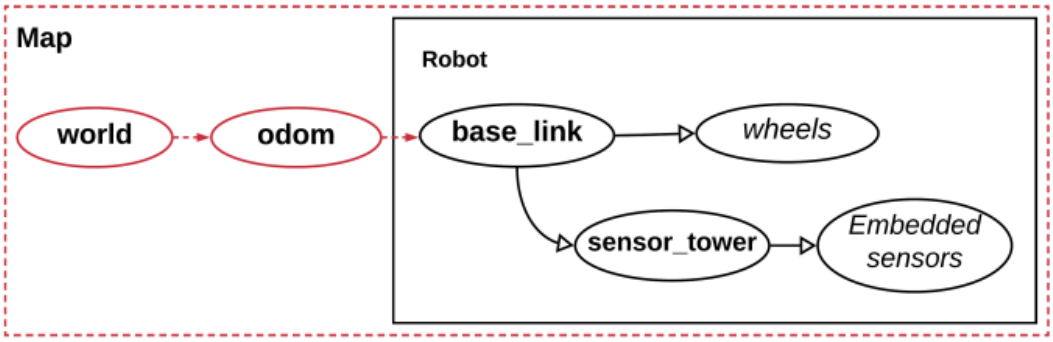

Figura 2 – Árvore TF simplificada de uma plataforma robótica (CID et al., 2020).

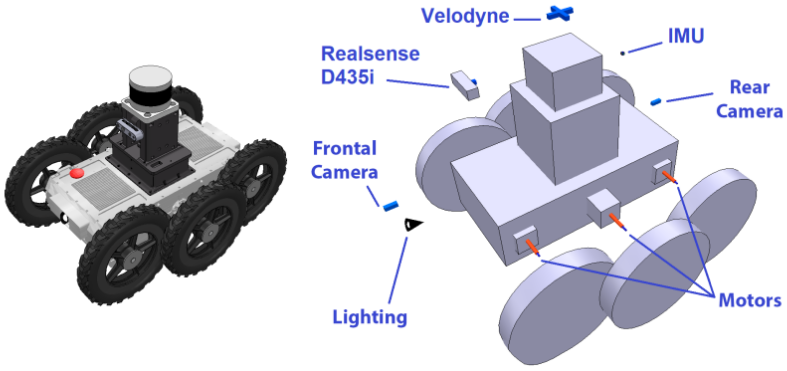

Figura 3 – Comparação do robô modelado a partir da árvore de TF e sua versão final na simulação (CID et al., 2020).

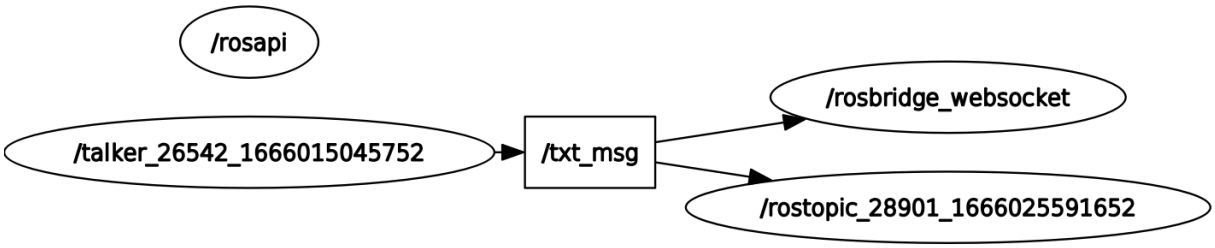

Figura 4 – Comunicação entre nós do ROS através do tópico *txt_msg*.

2.1.2 Segurança no ROS

Conforme demonstrado por Dieber et al. (2017), os nós do ROS não precisam se identificar ou autenticar antes de realizar qualquer ação, como publicar em um tópico ou realizar a chamada de um serviço. As decisões de design encontradas no ROS facilitam a inserção do mesmo em ambientes acadêmicos, bem como facilitam o desenvolvimento, porém, deixam o ROS aberto a vários tipos de ataques. Dentre esses, ressalta-se a facilidade de se interagir com os nós da rede e, também, espionar os valores dos tópicos, já que não existe uma criptografia padrão envolvida.

Existem ferramentas públicas disponíveis, desenvolvidas por roboticistas, que permitem realizar testes de penetração no ROS e exploram as vulnerabilidades que a falta de design de autenticação e validação existente possui (DIEBER et al., 2020).

Apesar do ROS 2.0 implementar sistemas de integração, autenticação e criptografia, ainda não possui um sistema de *logging* e registro, portanto um invasor que obter acesso ao sistema ainda terá controle e poderá manipular os dados sem deixar rastros (ZHANG et al., 2022a). Logo, a utilização da blockchain serve como alternativa para desenvolver um sistema de registros com o intuito de resolver estas desvantagens.

2.2 Blockchain

Blockchain é uma tecnologia de registro descentralizado (do inglês, DLT – *Decentralized Ledger Technology*) inicialmente projetada para que transações financeiras eletrônicas do Bitcoin sejam registradas permanentemente em uma cadeia de blocos (NAKAMOTO, 2008). A evolução da tecnologia trouxe a possibilidade de programação Turing-completa, o que permite maior grau de programabilidade do que originalmente previsto pelo protocolo do Bitcoin. Assim, surge o conceito de Contratos Inteligentes, em blockchains como a Neo Blockchain (HONGFEI; ZHANG, 2018). A diferença principal entre as blockchains consiste geralmente em: mecanis-

mos distintos para consenso; diferentes maneiras de programar contratos inteligentes; recursos especiais como acesso à Web 2 (BERNERS-LEE, 2010) via oráculos (como descrito a seguite na Seção 2.2.3).

2.2.1 Mecanismos de consenso

Atualmente, existem distintos mecanismo de consenso para governar as DLTs. No Bitcoin, o consenso por prova de trabalho (do inglês, *proof of work*) foi o primeiro a ser proposto e colocado em prática, consistindo em uma tarefa árdua de encontrar palavas criptográfica. Encontrar essas palavras torna-se uma tarefa mais difícil a medida que o recurso computacional da rede aumenta, de forma a manter uma estabilidade no tempo de bloco (definido no protocolo da rede).

Por outro lado, a blockchain do Neo (utilizada no sistema proposto neste presente trabalho) se diferencia fortemente do Bitcoin por utilizar um consenso não dependente de prova de trabalho, obtendo maior eficiência energética ao utilizar poucos nós de processamento no consenso, em contraponto às centenas de milhares de nós na rede do Bitcoin. Isso é possível através de avanços tecnológicos em cima do trabalho pioneiro de Bárbara Liskov e Miguel Castro em 1999, chamado de Practical Byzantine Fault Tolerance (PBFT) (CASTRO; LISKOV et al., 1999). Um consenso baseado em "prova de aposta delegada" (do inglês, *delegated proof of stake*) foi implementado no Neo, sendo chamado de dBFT – *delegated Byzantine Fault Tolerance*, sendo capaz de validar blocos corretamente e em tempo finito, dado que 66% dos nós estejam íntegros (ou *não bizantinos*).

2.2.2 Contratos inteligentes

Contratos inteligentes (do inglês, *smart contracts*) permitem a programação de códigos confiáveis *trustless*, sendo implantados em redes descentralizadas, especialmente para situações em que as partes não se confiam. O Neo permite programação de contratos em linguagens de programação populares como C#, Java, Go e Python, sendo compiladas para a plataforma de código de máquina NeoVM[3].

2.2.3 Oráculos

A tecnologia de Oráculos permite o acesso direto à Web 2, mesmo dentro de contratos inteligentes a partir da web descentralizada (também conhecida popularmente como Web 3). Esse recurso é tido como um *building block* fundamental das tecnologias blockchain modernas, permitindo pontes entre diversas tecnologias da web 2, e também blockchains diferentes da web descentralizada. Entretanto, programar com o uso de oráculos exige grande maturidade e conhecimento da arquitetura blockchain, tendo um alto custo computacional (e também monetário), por isso deve ser usado apenas em situações que é fundamental a obtenção de informações

[3] https://github.com/neo-project/neo-vm/

verificáveis do “mundo exterior” para o “mundo da blockchain”. O contrato inteligente com oráculos do Neo opera em duas fases, sendo necessária a invocação de uma função de acesso exterior (incluindo protocolos populares como *https*) e registro de uma função de *callback*, cuja execução apenas acontece quando o dado requisitado pelo oráculo já se encontra verificado. A documentação do Neo Oracle Service pode ser encontrada no website do Neo[4].

2.3 Robótica e Blockchain

Abordaremos nesta seção os principais trabalhos relacionados que concentram em pesquisas que utilizam conceitos de robótico em conexão com a blockchain ou conceitos de descentralização e persistência dos dados.

Em Alsamhi e Lee (2021), um framework conceitual utilizando blockchain para um sistema multi-robôs é proposto, com o intuito de combate a pandemia da COVID-19. A blockchain, nesta abordagem apresentada por eles, permitiria que robôs homogêneos e heterogêneos combatam a COVID-19 de forma colaborativa e eficiente, compartilhando informações de forma autônoma e acessando as informações uns dos outros. Em posse das ferramentas disponíveis na blockchain, seria possível a tomada de decisões colaborativas através dos algoritmos de consenso, além da utilização de dados históricos para inserção de novos robôs no sistema, não havendo assim necessidade de investir tempo nas fases de treinamento e aprendizado. É citado como desafio a complexidade de avaliar os recursos de detecção de robôs e a qualidade dos dados históricos, podendo afetar de forma significativa o desempenho do robô e os dados imprecisos coletados pelos sensores.

Já em Ferrer et al. (2022), é proposto a utilização de blockchain como ferramenta de comunicação de forma a garantir que, se um robô bizantino, que age de forma maliciosa, tentar modificar o conteúdo de um bloco na rede, a consequência será uma mudança na hash de forma que este bloco perderá sua conexão com os anteriores. Desta forma, estas tentativas de alteração serão ignoradas pelos outros robôs na rede. O artigo apresenta um estudo de caso do problema de *Follow the Leader* (FTL), onde cada robô no sistema acessa a blockchain através do seu próprio nó, e robôs bizantinos tentam bloquear ativamente os movimentos de robôs benignos, tentando publicar sinais incorretos coordenados na rede.

Por fim, em Zhang et al. (2022b), é proposto um framework envolvendo o ROS e a rede do Ethereum, onde os usuários não precisam se preocupar com a implementação da transação ou algoritmos de criptografia, focando apenas em inserir o nome dos tópicos que queiram transmitir dados. Os testes foram rodados em uma rede Ethereum privada, com 3 nós e o robô conectado diretamente ao nó mestre do ROS. O trabalho apresenta como proposta principal uma interface amigável ao usuário, podendo se beneficiar da interação com um robô através do ROS.

Conforme descrito por Afanasyev et al. (2019), estudos recentes mostram que a block-

[4] https://docs.neo.org/docs/en-us/advanced/oracle.html

chain desempenha um grande papel no desenvolvimento de sistemas e aplicações robóticas, de forma que possam conduzir de forma efetiva suas tarefas utilizando técnicas de consenso, registro de dados e alocação de tarefas de forma descentralizada.

3 SISTEMA PROPOSTO

Este capítulo contêm a descrição dos componentes utilizados no sistema proposto. Na Seção 3.1 apresenta-se uma visão geral da topologia utilizada. A tecnologia ROS e suas APIs são descritas na Seção 3.2. O smart contract e a descrição da integração com a blockchain do Neo é apresentada na Seção 3.3.

3.1 Topologia da rede

A topologia da rede envolve uma interligação e uso das seguintes tecnologias e componentes (uma uma visão geral dessa topologia é apresentada na Figura 5):

- ROS Core;
- Node.js e Express.js;
- Nós de REST;
- Consenso de blockchain;
- Contratos inteligentes;
- Oráculos de blockchain;
- Interações do usuário via transações criptográficas.

De forma a ressaltar a utilização de robôs em áreas remotas e isoladas, os módulos dos sensores e disponibilização dos dados do robôs são feitos de forma remota, utilizando uma conexão via satélite. O Starlink (FOUST, 2018) é um recente projeto lançado pela SpaceX e disponibiliza satélites de baixa orbita para conexão com a web.

Para tal, o núcleo do ROS cria uma conexão WebSocket (FETTE; MELNIKOV, 2011), da qual um backend em nodejs (CANTELON et al., 2014) é utilizado para criação de APIs REST (BALACHANDAR, 2017) através de um domínio certificado https (requerimento necessário dos oráculos da Neo Blockchain).

A rede do Neo está disponível na web, seja por uma combinação de nós espalhados pelo globo terrestre ou em redes privadas que simulam uma rede descentralizada. Em primeiro lugar, o contrato inteligente (Seção 3.3) armazenado na blockchain é acionado por algum usuário ou entidade validadora (pode ser visto na Figura 5 como a transação tx). Essa transação chega aos nós de consenso (conforme explicado na Seção 2.2.1) que acionam o comitê dos oráculos para que este acesse a url solicitada através do contrato. Ao receber essa informação por uma maioria

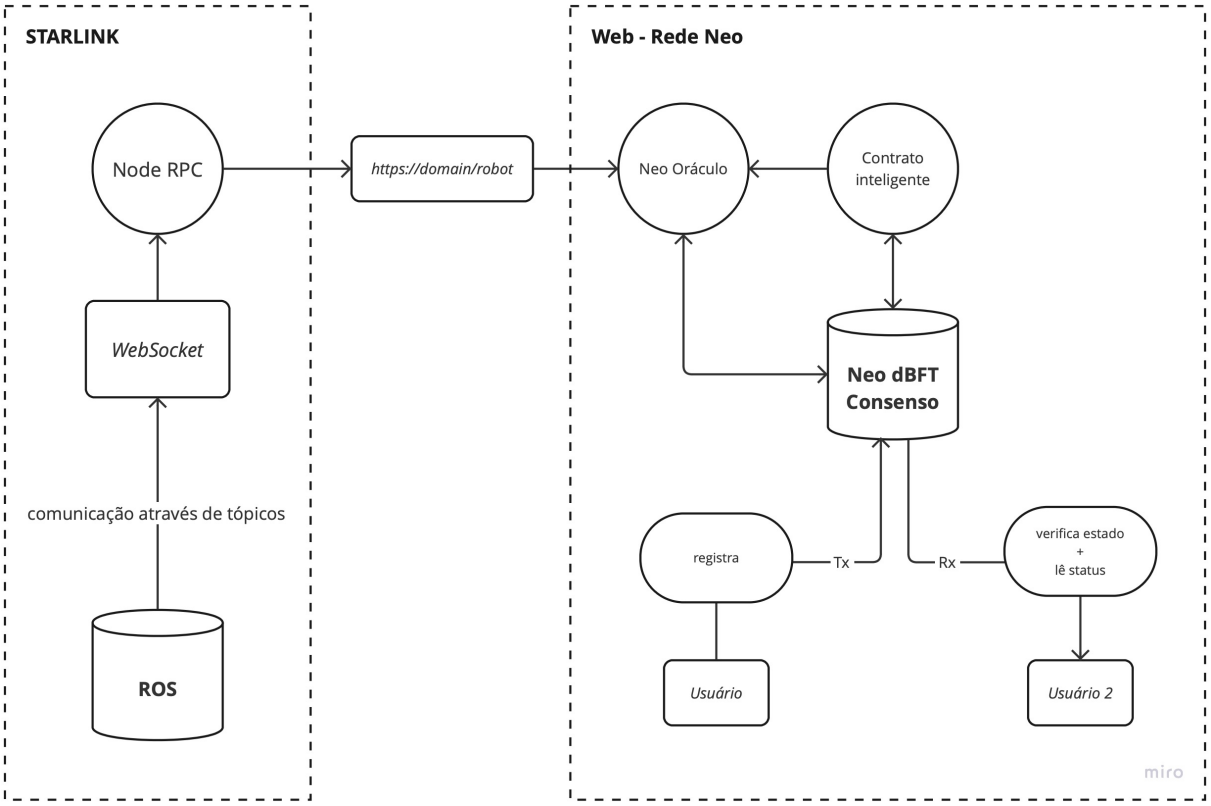

Figura 5 – Fluxograma da topologia de rede proposta no trabalho

dos nós, as informações são consolidadas e salvas na base de dados da blockchain, a qual pode ser acessada a partir de transações de demanda de informação ou analisando, de maneira crua, os dados armazenados (pode ser visto na Figura 5 como a transação rx).

3.2 Pacote ROS

Utilizaremos o ROS para simular dados de sensores, em vista da facilidade de interagir com os tópicos e alterar os seus devidos tipos de mensagem (conforme detalhado na Seção 2.1.1).

O código desenvolvido em Python para o nó publicador é apresentado no Algoritmo 3.1.

```
#!/usr/bin/env python3
# license removed for brevity
import rospy
from std_msgs.msg import String
from json import dumps
def talker():
    pub = rospy.Publisher('txt_msg', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    count = 0
    MAXIMUM_COUNT = 1000000000000000000000
```

```
    while not rospy.is_shutdown():
        if (count < MAXIMUM_COUNT):
            rospy.loginfo("The count is: " + str(count))
            strJSON = dumps({"timestamp":int(rospy.get_time()*1000),
                "value": count})
            pub.publish(strJSON)
            count += 1
        else:
            count = 0
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Algoritmo 3.1 – Código em Python para um Publisher simples.

O código publica no tópico *txt_msg* o formato JSON necessário contendo o *timestamp* e o valor de um contador para a utilização posterior no backend desenvolvido. Para disponibilizar os dados dos tópicos à serviços externos, o ROS possui um pacote (denominado como rosbridge-suite) que provê um servidor WebSocket para navegadores web, facilitando assim a interação. Um exemplo da interação do tópico com o servidor WebSocket pode ser observado na Figura 6.

3.3 Contrato inteligentes desenvolvido

O código desenvolvido para ser utilizado na Blockchain do Neo está disponível no GitHub[1], e suas funções para interação com o código são descritas a seguir.

A função `doRequestWithParameters`, descrita no Código 3.2, apresenta o ponto principal do funcionamento do contrato, acionando um oráculo de leitura *https* para um determinado momento de tempo (chamado de *timestamp*). O oráculo irá visitar o servidor de REST com dados acumulados do ROS (para os últimos 1000 eventos), e irá retornar o dado mais recente, dentro do *timestamp* estipulado, informado na URL de busca (como `https://domain/robot:9092/robot/`).

A função `GetStorage`, descrita no Código 3.3, apresenta o recurso do contrato inteligente para retornar o último dado lido (e armazenado) pelo oráculo. O dado fica armazenado no campo em que foi guardado (nesse exemplo mostra-se o campo denominado *sensor*).

[1] https://github.com/mateusnazarioc/neo-ros-blockchain-bridge/smart-contract

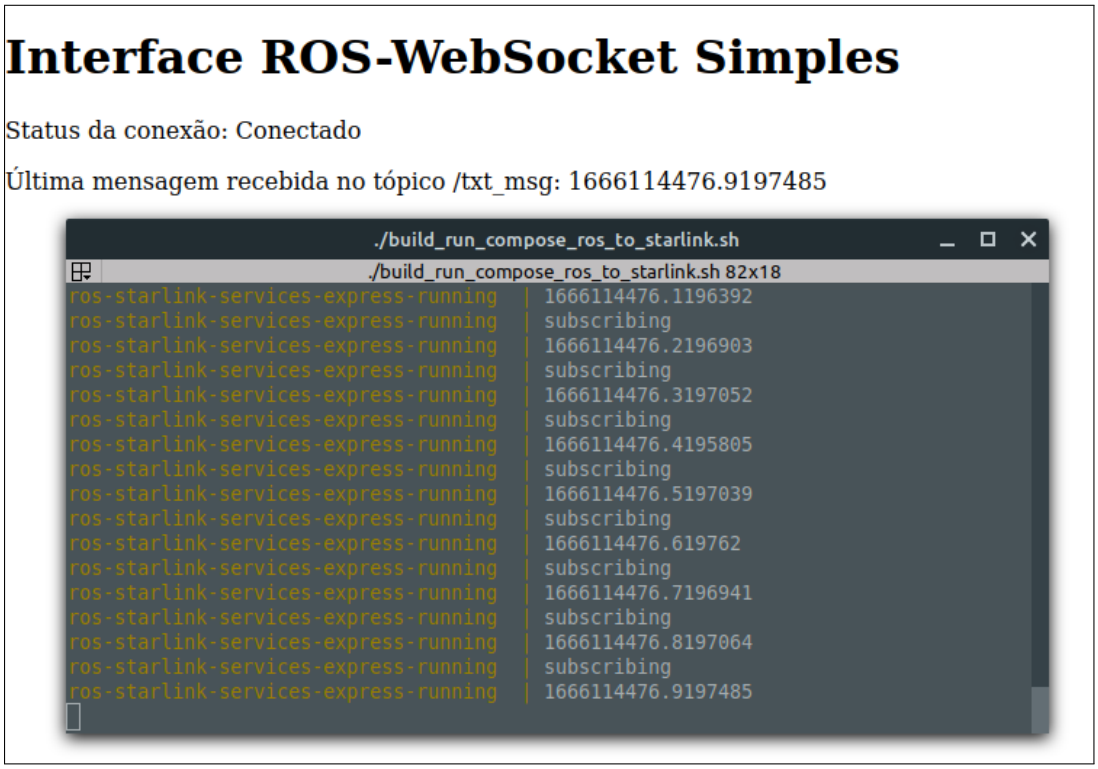

Figura 6 – Interface HTML com o nó publicador do ROS no terminal.

```
public static void DoRequestWithParameters(string filter, string url)
      {
         object userdata = "userdata"; // arbitrary type
         Runtime.Log("Inside DoRequest");
         string callback = "callback"; // callback method
         long gasForResponse = 1000000000;
         Oracle.Request(url, filter, callback, userdata,
            gasForResponse);
      }
```

Algoritmo 3.2 – Código do contrato inteligente em C# utilizado para registrar os valores recebidos do ROS

```
public static string GetStorage()
      {
         string s = Storage.Get(Storage.CurrentContext, "sensor");
         return s;
      }
```

Algoritmo 3.3 – Código do contrato inteligente em C# utilizado para registrar os valores recebidos do ROS

A função `Reset`, descrita no Código 3.4, apresenta o recurso do contrato inteligente de reiniciar o contador para um dado (chamado de *nonce*) específico. Para fins de experimentos, o contrato conta com três espaços de armazenamento, chamados de *val0*, *val1* e *val2*, armazenando os três últimos (e maiores) valores lidos pelo oráculo, em ordem crescente. Essa abordagem evita a redundância de leituras e sobrecarga da capacidade do contrato, que tipicamente é causa fatal quando armazenamentos ilimitados (como em vetores e mapas) causam um aumento de custo de GAS que impossibilita o uso do contrato.

```
public static void Reset(int nonce)
      {
         BigInteger bigzero = BigInteger.Zero;
         Storage.Put(Storage.CurrentContext, "val0", nonce);
         Storage.Put(Storage.CurrentContext, "val1", nonce);
         Storage.Put(Storage.CurrentContext, "val2", nonce);
         Storage.Put(Storage.CurrentContext, "bigdiff", bigzero);
         BigInteger big3 = 3;
         Storage.Put(Storage.CurrentContext, "count", big3);
      }
```

Algoritmo 3.4 – Código do contrato inteligente em C# utilizado para registrar os valores recebidos do ROS

A função `Callback`, descrita no Códigos 3.5–3.6, apresenta o retorno do oráculo, que armazena o dado mais recente em *sensor* e também rotaciona os três últimos valores, em *val0*, *val1* e *val2*. Para fins experimentais, também armazenamos a maior diferença *bigdiff* entre os dados de *nonce*, para monitorar funcionamentos anormais da captura de dados pelo contrato (valores muito altos podem indicar sobrecarga da rede, com “saltos” repentinos nos valores de *nonce*). Finalmente, o *count* permite um processo de “contagem regressiva”, no qual somente após três leituras iniciais (possivelmente com ruído) o contrato efetivamente começa a considerar o dado de leitura. Isso é feito para permitir uma integração mais fácil entre os três sistemas assíncronos envolvidos: (i) produção de dados no ROS; (ii) lançamento de transações de validação por oráculo na blockchain; (iii) monitoramento de logs.

```
public static void Callback(string url, string userdata,
   OracleResponseCode code, string result)
      {
         if (Runtime.CallingScriptHash != Oracle.Hash) throw new
            Exception("Unauthorized!");
         if (code != OracleResponseCode.Success) throw new
            Exception("Oracle response failure with code " +
            (byte)code);

         object ret = StdLib.JsonDeserialize(result);
         object[] arr = (object[])ret;
         BigInteger value = (BigInteger)arr[0];
         value = value + 1;

        Storage.Put(Storage.CurrentContext, "sensor",value);

        // ROTATE latest three data
        // ASSUMING its ZERO for unitialized data
        BigInteger val0 =
           (BigInteger)Storage.Get(Storage.CurrentContext, "val0");
        BigInteger val1 =
           (BigInteger)Storage.Get(Storage.CurrentContext, "val1");
        BigInteger val2 =
           (BigInteger)Storage.Get(Storage.CurrentContext, "val2");

        // CHECK IF STRICTLY INCREASING
        if(val2 > value) {
           object[] arrValue1 = {value};
           Runtime.Notify("IGNORED new value less than
              val2",arrValue1);
           return;
        }

        Storage.Put(Storage.CurrentContext, "val0",val1);
        Storage.Put(Storage.CurrentContext, "val1",val2);
        Storage.Put(Storage.CurrentContext, "val2",value);
```

Algoritmo 3.5 – Código do contrato inteligente em C# utilizado para registrar os valores recebidos do ROS (parte 1/2)

```
        BigInteger count =
            (BigInteger)Storage.Get(Storage.CurrentContext, "count");
        if(count == 0) {
          // CALCULATE and store the biggest difference
          BigInteger diff = value - val2;
          BigInteger bigdiff =
              (BigInteger)Storage.Get(Storage.CurrentContext,
              "bigdiff");
          if(diff > bigdiff) {
            // new big difference
            Storage.Put(Storage.CurrentContext, "bigdiff",diff);
            object[] arrValue0 = {diff};
            Runtime.Notify("UPDATED bigdiff",arrValue0);
          }
        } else {
           Storage.Put(Storage.CurrentContext, "count",count-1);
        }

         object[] arrValue = {value};
         Runtime.Notify("value",arrValue);
         Runtime.Log("response value: " + value);
      }
```

Algoritmo 3.6 – Código do contrato inteligente em C# utilizado para registrar os valores recebidos do ROS (parte 2/2)

3.4 Starlink

A rede de internet via satélites Starlink possui como infraestrutura mais de 4 mil satélites em órbita, com prospecção de lançamento de mais 40 mil nos próximos anos (https://www.starlink.com/technology). O sistema tem como proposta a utilização de internet de alta velocidade e baixa latência em áreas remotas, desde zonas de guerra até áreas de trabalho sem cobertura comercial (Herath, 2021). A compra dos equipamentos é feita de forma única e o serviço é contratado mensalmente, com todos os detalhes e valores do Brasil podendo ser observados no website de aquisição do produto. O kit adquirido é composto de uma antena, um roteador e os cabos de conexão. Todas as especificações técnicas são descritas no arquivo de especificações do produto (https://api.starlink.com/public-files/Starlink%20Product%20Specifications_Standard.pdf).

3.5 Implementação

Essa seção apresenta uma descrição dos componentes desenvolvidos para o funcionamento do sistema proposto, envolvendo uma série de ferramentas e padronizações de uso, como a utilização do Docker (CITO et al., 2017). O último, fornece uma praticidade de extrema necessidade para a aplicação em questão, pois a aplicação lida com uma série de protocolos certificados, os quais precisam de redes precisamente configuradas e comunicações intermitentes. O conjunto de componentes descrito a seguir pode ser encontrado em https://github.com/mateusnazarioc/neo-ros-blockchain-bridge.

3.5.1 Ponte do ROS com o WebSocket

Para a publicação dos dados coletados e enviados através dos tópicos do ROS para o WebSocket, foi desenvolvido um container Docker denominado *ros-bridge-docker*, o qual executa tanto o sistema ROS quanto o servidor WebSocket, de forma a tornar disponível os dados encriptados. Seu código está escrito em Python (conforme anteriormente apresentado).

De forma a tornar a execução mais prática e replicável, um script é disponibilizado (*docker_build.sh*), o qual compila o container Docker com todas as dependências necessárias.

3.5.2 Ponte WebSocket e Backend via REST

Para prover chamadas REST para interação com a blockchain, um outro container Docker (denominado *run-ros-rpc*) foi desenvolvido de forma a integrar a ponte WebSocket com um backend desenvolvido com o framework *express.js*.

A imagem principal de um serviço *node.js* baseado em Linux pode ser encontrado na subpasta *docker-node-ros*.

3.5.3 Conexão com a rede Starlink de satélites

No momento em que esse trabalho foi desenvolvido, a rede Starlink não provia um IP público à clientes residenciais por conta do baixo número de endereços ipv4 disponíveis, sendo necessário então uma forma de contornar o CGNAT aplicado (BOCCHI et al., 2015). A conexão desenvolvida pode ser encontrada na pasta *node-starlink-bridge*, a qual disponibiliza a API REST necessária para os oráculos da blockchain.

3.5.4 Ponte ROS e Starlink

De forma similar à pasta *run-ros-rpc*, as ferramentas encontradas em *run-ros-rpc-to-starlink* disponibilizam uma conexão WebSocket, porém, enviam os dados do ROS para o backend *node express* na rede Starlink (descrito anteriormente), que uma API REST.

3.5.5 Experimentos

A pasta *run-experiments* contém os códigos utilizados para geração dos dados e gráficos de testes apresentados no Capítulo 4.

3.5.6 Contrato inteligente

Por fim, a pasta *smart-contract* contém os exemplos de contratos inteligentes escritos em C# que são executados na blockchain do Neo. Este código é utilizado para interagir com a blockchain, que obtém os dados do ROS a partir do REST utilizando chamadas e callbacks de oráculos.

4 EXPERIMENTOS

Na Seção 4.1, descreve-se a utilização do sistema através dos pacotes desenvolvidos (apresentados na seção 3.5). O estudo de caso criado para esse trabalho é descrito na Seção 4.2. Por fim, a análise dos resultados computacionais gerados através dos scripts de experimentos são descritos na Seção 4.3.

4.1 Utilização do sistema

Para início dos experimentos, é necessário rodar a rede do Neo. Todos os experimentos foram rodados de forma privada, tendo o processo descrito no GitHub do NeoCompiler (neocompiler-neo), que provê uma interface fácil de comunicação com a blockchain do Neo.

Após, é feito a inicialização do container descrito na subseção 3.5.2, que inicializa a comunicação da rede privada do Neo com a ponte do ROS-REST (Figura 7).

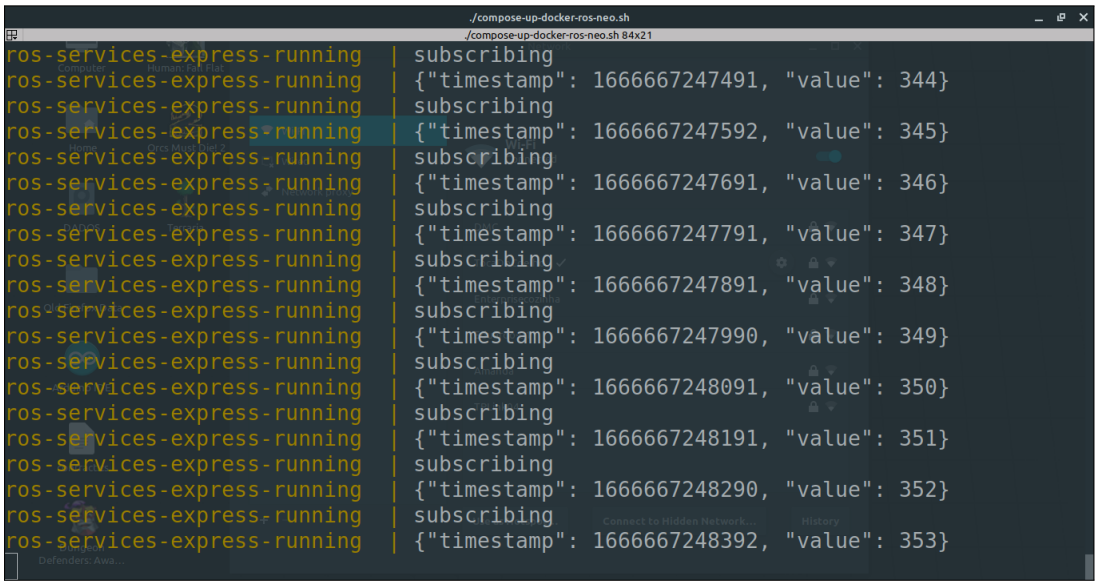

Figura 7 – Inicialização e publicação do sensor através do ROS.

Com o sistema levantado, é possível agora interagir com os oráculos através dos contratos inteligentes fornecidos no repositório, que realizam a interação com a API disponibilizada pelo backend, sendo possível a visualização do REST através da rede local (Figura 8). Para isso, é necessário abrir a interface fornecida pelo NeoCompiler através de uma porta local, compilar o contrato inteligente fornecido (Figura 9), e implantá-lo na rede privada local (Figura 10). Neste momento é necessário também realizar a eleição do oráculo, que será o serviço assíncrono que disponibilizará ao contrato inteligente as informações externas à rede.

Para os experimentos descritos na subseção 3.5.5, é necessário a utilização do script

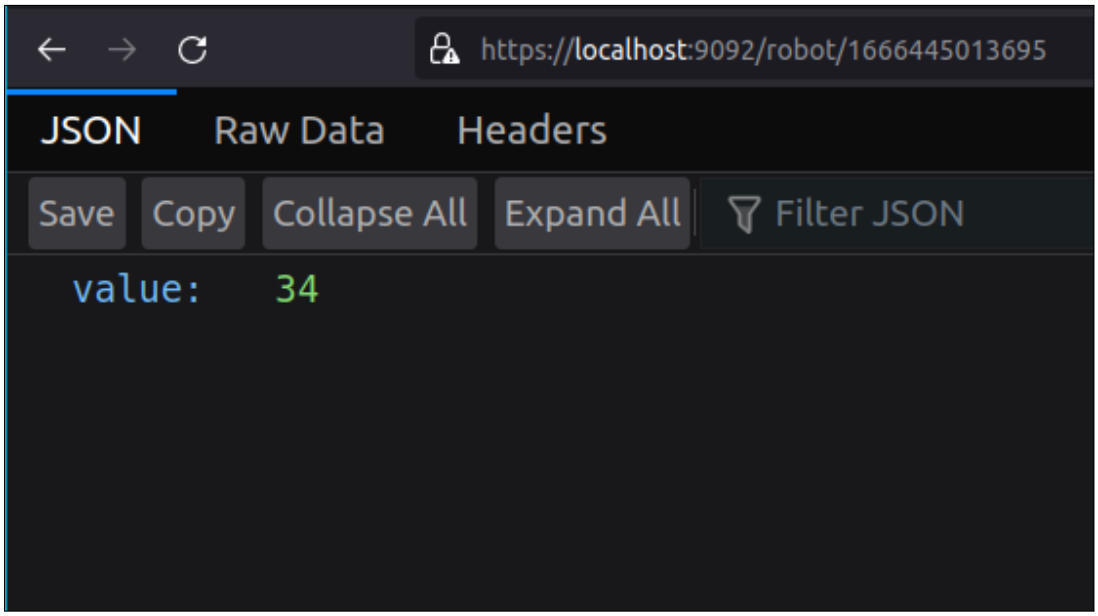

Figura 8 – Visualização do REST que será utilizado para interação na blockchain.

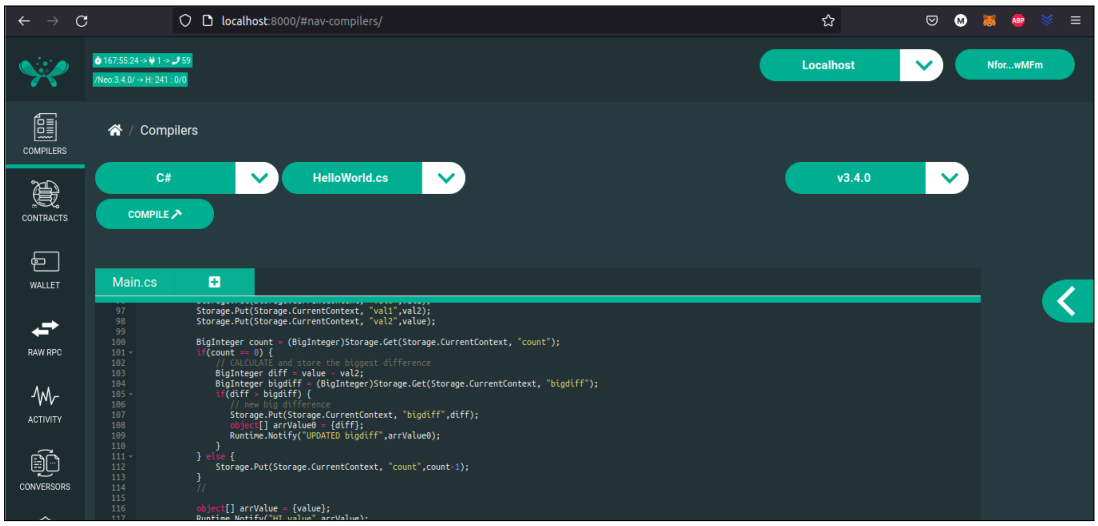

Figura 9 – Inclusão e compilação do contrato inteligente.

que gerará o log (denominado como *run_test.sh*), e a utilização do código javascript dentro do terminal do navegador utilizado (denominado como *run_script.txt.js*).

De forma similar, os experimentos com a utilização da rede Starlink utilizam do pacote descrito na subseção 3.5.3 para a disponibilização do backend e na subseção 3.5.4 para a publicação dos dados do sistema ROS para a rede Starlink. A montagem do computador que publica os dados ROS através da rede Starlink pode ser observado na Figura 11.

Por fim, pode-se observar, na Figura 12, a execução do algoritmo *run_script.txt.js*, que executa um laço de interações com o contrato inteligente. Esse laço pode representar as requisições de registro enviadas por um usuário comum, uma entidade validadora ou o próprio robô, ativando um mecanismo de registros dos valores do ROS pelo smart contract (conforme

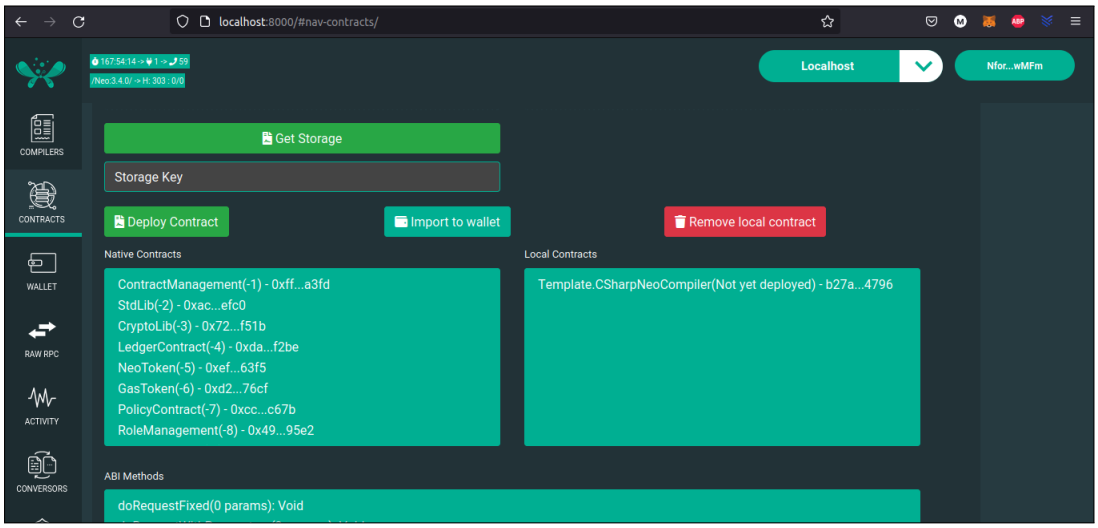

Figura 10 – Implantação do contrato inteligente na rede para interação.

Figura 11 – Publicação dos dados ROS através da rede Starlink, notebook simulando dados do robô ao lado da antena de comunicação.

mostrado na Figura 5 da Seção 3.1). Os logs são obtidos para análise executando, em paralelo, o script *run_test_1sec.sh*, que obtém os dados na base de dados da blockchain, ou seja, aqueles publicados pelo ROS através do backend e o *timestamp* associado.

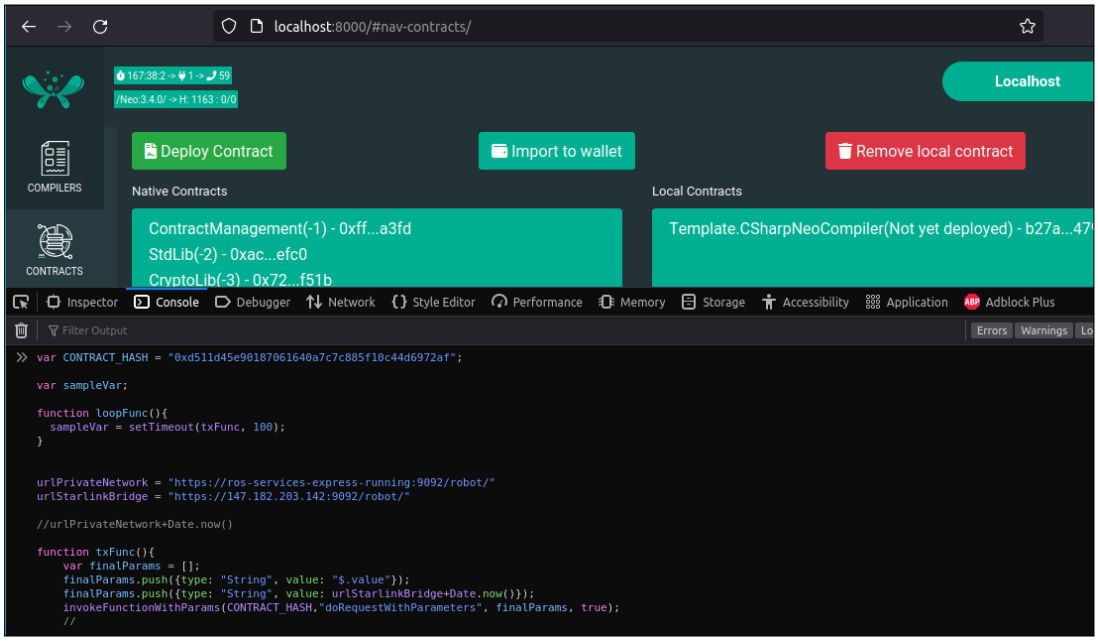

Figura 12 – Execução do algoritmo javascript para log dos dados do ROS.

4.2 Estudo de caso

Como estudo de caso, o trabalho analisa um cenário de uma agência governamental que possui um setor de auditoria, de forma que acesse os dados publicados por uma plataforma robótica através de uma interface com a blockchain, aproveitando da segurança, confiabilidade e sistema descentralizado fornecido. Conforme dito por Elommal e Manita (2022), a tecnologia da blockchain, aliada à outras tecnologias, é capaz de trazer grandes mudanças para processos de auditoria, pois fornece uma nova forma à maneira que um auditor acessa e analisa os dados. Para o cenário de auditoria deste trabalho, serão utilizados e analisados os dados fornecidos pelos experimentos, como: valores do sensor e o *timestamp* associado à medição.

4.3 Resultados computacionais

Foram considerados cenários com geração de blocos local a cada 1000 ms, em rede blockchain privada, com ou sem uso de conexão remota via Starlink. O processo de produção de dados via WebSocket, integrado ao ROS, foi definido como 10 Hz. O processo de log foi definido para o mesmo tempo de produção de blocos, no caso, 1000 ms. Para cada caso, o processo foi monitorado via sistema assíncrono de logs por, aproximadamente, 100 segundos. Finalmente, o processo de emissão de transações de validação via oráculo foi experimentado com três valores distintos: 100 ms, 200 ms e 500 ms. Vale ressaltar que os três processos (WebSocket/ROS,

oráculo e log) são completamente assíncronos, assim como o próprio manuseio de transações na rede P2P interna da blockchain privada, não havendo comunicação direta com os nós de consenso (comportamento típico de blockchains com alto grau de segurança), o que torna o processo de medição bastante desafiador.

Para a blockchain (e também nos experimentos locais sem o Starlink), foi utilizado um computador com processador 11ª geração de Intel(R) Core(TM) i7-11800H (8-core, cache de 24MB, até 4.6GHz), Memória de 16GB (2x8GB), DDR4 3200MHz e SSD de 512GB PCIe NVMe M.2. Para o experimento com a rede Starlink, o ROS e o backend websocket/REST foi hospedado em um computador com processador 8ª geração de Intel(R) Core(TM) i5-8250U (quad-core, até 1.6GHz), Memória de 8GB (2x4GB), DDR4 2133MHz e SSD de 256GB PCIe NVMe M.2. Já o backend de REST (que recebe os dados do robô para disponibilizar de forma pública para a blockchain), foi hospedado no Digital Ocean, em um *cloudlet* de 1 GB RAM / 25 GB Disco / SFO3 - Ubuntu 22.04 (LTS) x64.

As Figuras 13, 14 e 15 exploram cenários com a ponte WebSocket do ROS local, no mesmo dispositivo em que é executada a blockchain (com seus quatro nós de consenso via dBFT, além de um nó de REST adicional que também realiza as chamadas do oráculos). Em vermelho, estão indicadas as diferenças entre os mais recentes valores de *nonce* registrados no contrato inteligente, em dois distintos intervalos de tempo (tipicamente 1000 ms). Já em azul, representa a quantidade de valores de *nonce* **únicos** registrados a cada verificação de estado da blockchain, normalizada pelo limite estipulado de requisições de verificação de armazenamento (feitas a cada 1000ms). Como o valor de *nonce* é incremental, é esperado que o oráculo mantenha no contrato apenas os três maiores valores (naturalmente os mais atuais) obtidos pelas chamadas. Vale ressaltar também que, não há possibilidade de controlar a sequência em que as transações entram nos blocos, sendo possível que uma transação com *timestamp* maior entre na frente de uma transação com *timestamp* menor, acarretando a perda do registro de informações (dado que o *nonce* só é registrado de forma incremental dentro do contrato, novamente por razões de segurança e proteção contra uso excessivo de GAS).

As Figuras 16, 17 e 18 exploram cenários com a ponte WebSocket do ROS através do serviço Starlink, em uma realidade bastante próxima do uso de sensores em áreas remotas (longe da blockchain de registro de validação). Novamente, em vermelho aparecem os intervalos de *nonce* capturados, enquanto em azul estão apenas os valores de *nonce* **únicos**. Assim como nos cenários anteriores (sem Starlink), a média em vermelho deve se aproximar de 10, dado que a frequência do ROS está definida para 10 Hz, enquanto os serviços de logs e de geração de blocos estão definidos para 1000 ms. Porém, é possível perceber um maior grau de perda com o uso de Starlink, possivelmente pela maior latência de rede envolvida no processo, visto a influência de diversos fatores como: obstrução do satélite e intempérie climática.

A Tabela 1 compara os valores percentuais de captura de valores pelo oráculo em cada caso (disparos de 100 ms, 200 ms e 500 ms), com e sem uso de Starlink. Essa taxa de registro

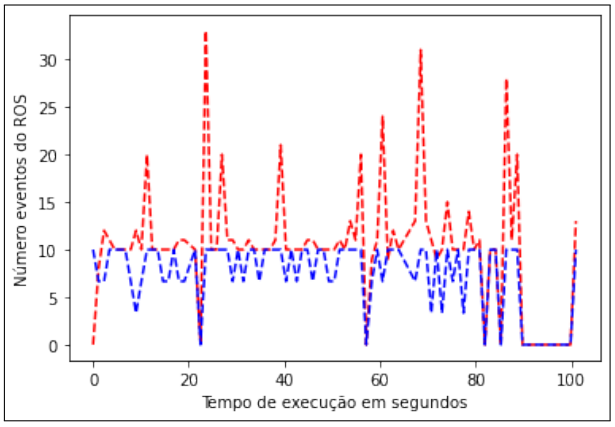

Figura 13 – REST local com ROS 10 Hz e verificação por oráculos a cada 100 ms. Produção de blocos e *logs* a cada 1000ms. Experimento de 100 segundos.

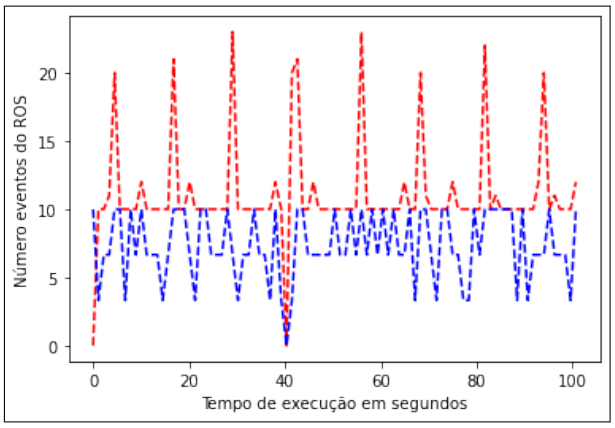

Figura 14 – REST local com ROS 10 Hz e verificação por oráculos a cada 200 ms. Produção de blocos e *logs* a cada 1000ms. Experimento de 100 segundos.

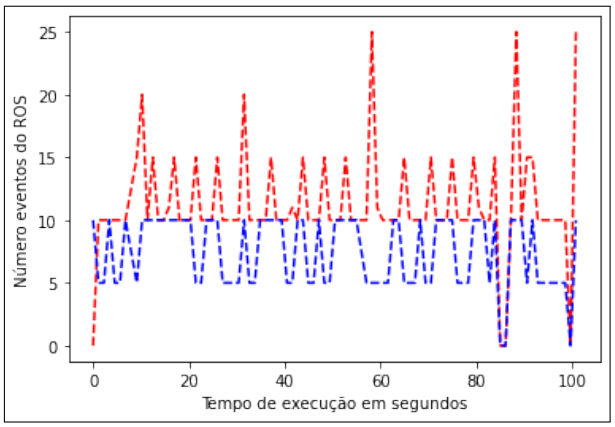

Figura 15 – REST local com ROS 10 Hz e verificação por oráculos a cada 500 ms. Produção de blocos e *logs* a cada 1000ms. Experimento de 100 segundos.

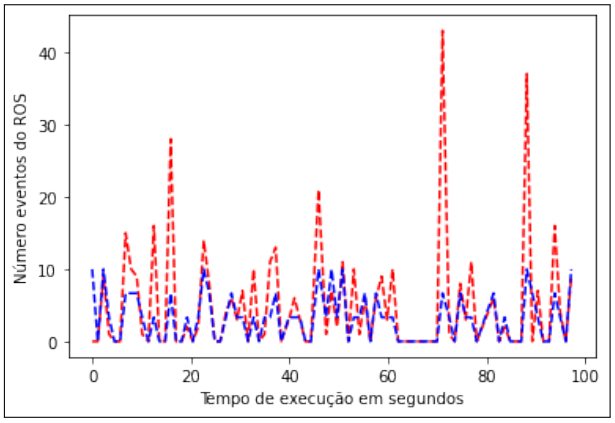

Figura 16 – Starlink REST com ROS 10 Hz e verificação por oráculos a cada 100 ms. Produção de blocos e *logs* a cada 1000ms. Experimento de 100 segundos.

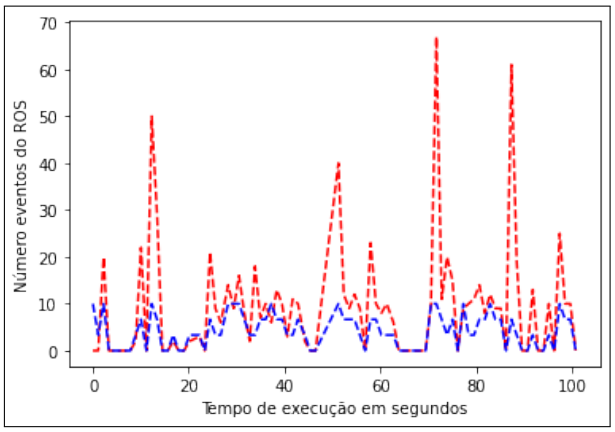

Figura 17 – Starlink REST com ROS 10 Hz e verificação por oráculos a cada 200 ms. Produção de blocos e *logs* a cada 1000ms. Experimento de 100 segundos.

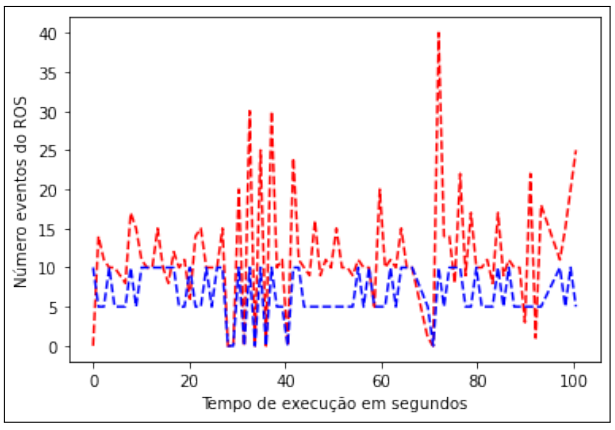

Figura 18 – Starlink REST com ROS 10 Hz e verificação por oráculos a cada 500 ms. Produção de blocos e *logs* a cada 1000ms. Experimento de 100 segundos.

indica a porcentagem de dados aproveitados da blockchain em relação ao registro de dados do ROS, em uma análise temporal.

Freq. Disparo	**Com Starlink**	**Sem Starlink**
100 ms	45,33%	72,26%
200 ms	44,70%	66,27%
500 ms	57,05%	66,06%

Tabela 1 – Comparação do uso de Starlink para disparos de 100 ms, 200 ms e 500 ms

Conforme pode ser observado na Tabela 1, é de se esperar uma diminuição na taxa de captura à medida em que menos transações são enviadas para oráculo. Esse fato se concretiza com maiores taxas de acerto, de 72% em rede local e 45% no Starlink, para um maior número de transações (a cada 100 ms). No mínimo, foram observadas taxas de 44% e 66%, com e sem Starlink respectivamente, na coleta assíncrona via oráculos. Já para 500ms (2 transações por segundo), a rede de blockchain provavelmente conseguiu encadear os blocos de uma maneira mais efetiva, sem gargalo nas transações pendentes, fato que foi verificado com 5 e 10 transações por segundo. Por outro lado, com menos disparos e também com redes mais lentas, podemos esperar uma menor sobrecarga da blockchain (com menor utilização da *mempool*) e maior estabilidade do processo de consenso, tópico a ser explorado em trabalhos futuros. É importante ressaltar que o experimento executado tem como resultado uma porcentagem que valida os dados buscados em uma taxa de tempo definida. Isto não significa uma perda de dados na blockchain, sendo possível verificar os dados restantes diretamente nos blocos da rede.

5 CONCLUSÃO

Neste trabalho, abordamos um processo de validação de dados de sensores do ROS através de uma blockchain privada com consenso bizantino. O processo é altamente complexo e desafiador, dada a pouca quantidade de material online para estudos e o pioneirismo envolvido no trabalho. O uso de oráculos é considerado hoje o estado-da-arte em tecnologias blockchain, permitindo a comunicação entre sistemas *trustless* de contratos inteligentes com o "mundo exterior" da Web 2 via *https*. Dada a natureza verificável da blockchain Neo, consideramos um servidor de agregação de dados via websocket e REST, pelo qual os oráculos podem se comunicar e obter informações precisas e determinísticas, viabilizando um consenso de tempo real e validação em cima dos dados de sensores. Foram feitos experimentos para validar a proposta, utilizando intervalos de transação de validação por oráculo de 100 ms, 200 ms e 500 ms, em uma rede blockchain privada com geração de blocos e logs a cada 1000 ms, enquanto dados eram gerados pelo ROS em 10 Hz. Também foram considerados cenários com e sem acesso via Starlink. Com taxas mais altas de transação (a cada 100 ms) sem Starlink foi possível registrar até 72% das informações (em vista da lógica do contrato inteligente, de registrar apenas os três valores mais novos), enquanto com Starlink e taxa de 100 ms foi possível registrar 45% das informações enviadas pelo ROS.

Dado o alto grau de assincronia e complexidade dos sistemas envolvidos, também considerando cenários práticos em que auditorias de alto grau de confiança são desejáveis, ao mesmo tempo em que se busca minimizar o armazenamento de dados na blockchain, consideramos que o trabalho consegue inovar e demonstrar a forma prática desta possibilidade. Portanto, a utilização da blockchain como ferramenta de melhoria para sistemas de log e validação de dados obtidos em plataformas robóticas se mostra viável em um futuro próximo.

5.1 Trabalhos futuros

Este trabalho pode ser estendido de diversas formas no futuro, seja com maior amplitude nos experimentos, considerando cenários e arquiteturas diversas, bem como buscando maior detalhamento no processo interno da blockchain durante os experimentos, exigindo maior grau de sofisticação dos mecanismos de log e também maiores modificações do contrato inteligente.

Se faz necessário a melhoria na interação do usuário com os pacotes disponibilizados, através de uma abstração genérica de tópicos do ROS com qualquer tipo de mensagem, além de uma generalização da interação com qualquer outra blockchain que possua oráculos. Devido a restrição de tempo, a coleta de um maior número de amostras não foi possível, havendo assim uma possibilidade de uma amostragem mais completa utilizando recursos computacionais mais potentes. É interessante ressaltar também a realização de testes com plataformas robóticas reais

interagindo com as redes públicas do Neo, com e sem a utilização do Starlink, podendo fornecer assim uma interface de auditoria ainda mais próxima de um cenário real de averiguação de dados. Acreditamos que esse trabalho possa abrir portas para diversas outras invenções no futuro, conectando o mundo *trustless* dos contratos inteligentes (chamado de Web 3) ao mundo real (chamado de Web 2), que é atualmente uma das maiores barreiras no ecossistema de blockchain.

REFERÊNCIAS

AFANASYEV, I. et al. *Towards Blockchain-based Multi-Agent Robotic Systems: Analysis, Classification and Applications*. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1907-.07433>. Citado na página 21.

ALSAMHI, S. H.; LEE, B. Blockchain-empowered multi-robot collaboration to fight covid-19 and future pandemics. *IEEE Access*, 2021. v. 9, p. 44173–44197, 2021. Citado na página 21.

AZPURUA, H. et al. Espeleorobô - a robotic device to inspect confined environments. In: *2019 19th International Conference on Advanced Robotics (ICAR)*. [S.l.: s.n.], 2019. p. 17–23. Citado na página 14.

BALACHANDAR, B. M. *RESTful Java Web Services: A pragmatic guide to designing and building RESTful APIs using Java*. [S.l.]: Packt Publishing Ltd, 2017. Citado na página 23.

BERNERS-LEE, T. Long live the web. *Scientific American*, 2010. JSTOR, v. 303, n. 6, p. 80–85, 2010. Citado na página 20.

BOCCHI, E. et al. Impact of carrier-grade nat on web browsing. In: *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*. [S.l.: s.n.], 2015. p. 532–537. Citado na página 31.

CANTELON, M. et al. *Node. js in Action*. [S.l.]: Manning Greenwich, 2014. Citado na página 23.

CASTRO, M.; LISKOV, B. et al. Practical byzantine fault tolerance. In: *OsDI*. [S.l.: s.n.], 1999. v. 99, n. 1999, p. 173–186. Citado na página 20.

CID, A. et al. A simulated environment for the development and validation of an inspection robot for confined spaces. In: *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*. [S.l.: s.n.], 2020. p. 1–6. Citado 3 vezes nas páginas 9, 17 e 18.

CITO, J. et al. An empirical analysis of the docker container ecosystem on github. In: IEEE. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. [S.l.], 2017. p. 323–333. Citado na página 30.

DEMARINIS, N. et al. Scanning the internet for ros: A view of security in robotics research. In: *2019 International Conference on Robotics and Automation (ICRA)*. [S.l.: s.n.], 2019. p. 8514–8521. Citado na página 14.

DIEBER, B. et al. Security for the robot operating system. *Robotics and Autonomous Systems*, 2017. v. 98, p. 192–203, 2017. ISSN 0921-8890. Citado 2 vezes nas páginas 14 e 19.

DIEBER, B. et al. Penetration testing ros. In: ____. *Robot Operating System (ROS): The Complete Reference (Volume 4)*. Cham: Springer International Publishing, 2020. p. 183–225. ISBN 978-3-030-20190-6. Disponível em: <https://doi.org/10.1007/978-3-030-20190-6_8>. Citado na página 19.

ELOMMAL, N.; MANITA, R. How Blockchain Innovation could affect the Audit Profession: A Qualitative Study. *Journal of Innovation Economics*, 2022. v. 0, n. 1, p. 37–63, 2022. Disponível em: <https://ideas.repec.org/a/cai/jiedbu/jie_pr1_0103.html>. Citado na página 35.

FERRER, E. C.; HARDJONO, T.; PENTLAND, A. Editorial: Proceedings of the first symposium on blockchain and robotics, mit media lab, 5 december 2018. *Ledger*, 2019. v. 4, Apr. 2019. Disponível em: <https://ledger.pitt.edu/ojs/ledger/article/view/179>. Citado na página 14.

FERRER, E. C. et al. Following leaders in byzantine multirobot systems by using blockchain technology. *IEEE Transactions on Robotics*, 2022. v. 38, n. 2, p. 1101–1117, 2022. Citado 2 vezes nas páginas 14 e 21.

FETTE, I.; MELNIKOV, A. *The websocket protocol*. [S.l.], 2011. Citado na página 23.

FOUST, J. Spacex's space-internet woes: despite technical glitches, the company plans to launch the first of nearly 12,000 satellites in 2019. *IEEE Spectrum*, 2018. IEEE, v. 56, n. 1, p. 50–51, 2018. Citado na página 23.

GUO, H.; YU, X. A survey on blockchain technology and its security. *Blockchain: Research and Applications*, 2022. v. 3, n. 2, p. 100067, 2022. ISSN 2096-7209. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2096720922000070>. Citado na página 15.

Herath, H. M. V. R. Starlink : A Solution to the Digital Connectivity Divide in Education in the Global South. *arXiv e-prints*, 2021. p. arXiv:2110.09225, out. 2021. Citado na página 30.

HONGFEI, D.; ZHANG, E. Neo white paper. *https://docs.neo.org/docs/en-us/basic/whitepaper.html*, 2018. v. 31, n. 07, p. 2020, 2018. Citado 2 vezes nas páginas 15 e 19.

MURPHY, R. R. *Disaster robotics*. [S.l.]: MIT press, 2014. Citado na página 14.

NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 2008. p. 21260, 2008. Citado 2 vezes nas páginas 14 e 19.

QUIGLEY, M. et al. Ros: an open-source robot operating system. In: KOBE, JAPAN. *ICRA workshop on open source software*. [S.l.], 2009. v. 3, n. 3.2, p. 5. Citado na página 14.

ZHANG, S. et al. Authros: Secure data sharing among robot operating systems based on ethereum. *Research Square*, 2022. p. 1–11, 09 2022. Citado na página 19.

ZHANG, S. et al. ROS-ethereum: A convenient tool to bridge ROS and blockchain (ethereum). *Security and Communication Networks*, 2022. Hindawi Limited, v. 2022, p. 1–14, apr 2022. Disponível em: <https://doi.org/10.1155\%2F2022\%2F7206494>. Citado na página 21.