UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE TELEINFORMÁTICA

MESTRADO ACADÊMICO EM ENGENHARIA DE TELEINFORMÁTICA

JOSÉ EDILSON SILVA FILHO

BLOCKCHAIN APLICADO EM ATUALIZAÇÃO DE FIRMWARE DE NANOSATÉLITES EDUCACIONAIS

FORTALEZA

2022

JOSÉ EDILSON SILVA FILHO

BLOCKCHAIN APLICADO EM ATUALIZAÇÃO DE FIRMWARE DE NANOSATÉLITES EDUCACIONAIS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Engenharia de Teleinformática do Programa de Pós-Graduação em Engenharia de Teleinformática do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Engenharia de Teleinformática. Área de Concentração: Sinais e Sistemas.

Orientador: Prof. Dr. Jarbas Aryel Nunes da Silveira.

FORTALEZA

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S58b Silva Filho, Jose Edilson.
Blockchain aplicado em atualização de firmwaere de nanosatélites educacionais / Jose Edilson Silva Filho. – 2023.
90 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia de Teleinformática, Fortaleza, 2023.
Orientação: Prof. Dr. Jarbas Aryel Nunes da Silveira.

1. Blockchain. 2. Firmware. 3. Nanosatélites. 4. Estações Terrestres. 5. Descentralização. I. Título.
CDD 621.38

JOSÉ EDILSON SILVA FILHO

BLOCKCHAIN APLICADO EM ATUALIZAÇÃO DE FIRMWARE DE NANOSATÉLITES EDUCACIONAIS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Engenharia de Teleinformática do Programa de Pós-Graduação em Engenharia de Teleinformática do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Engenharia de Teleinformática. Área de Concentração: Sinais e Sistemas.

Aprovada em: 13/02/2023.

BANCA EXAMINADORA

Prof. Dr. Jarbas Aryel Nunes da Silveira (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Giovanni Cordeiro Barroso
Universidade Federal do Ceará (UFC)

Prof. Dr. Rafael Vidal Aroca
Universidade Federal de São Carlos (UFSCar)

À Deus, pelo seu imenso amor e misericódia. À minha esposa e filhos pois, sem eles, não faria sentido qualquer esforço. À minha família, por sua capacidade de acreditar e investir em mim.

AGRADECIMENTOS

Agradeço em primeiro lugar à Deus por todas as graças que derramou em minha vida, em especial, o grande presente que recebi durante esta etapa de estudos que foi o nascimento do meu segundo filho. Samuel, também ofereço a você o esforço que dediquei em conciliar, estudos, vida profissional, familiar e religiosa. Espero que você e toda a nossa família usufrua os frutos da semente que plantamos estes anos. Agradeço, também, a todos os familiares e amigos que me acompanharam nesta trajetória, com abraços e beijos especiais a minha esposa, Aryana Fideles.

Agradeço ao professor Dr. Jarbas Aryel Nunes da Silveira que com atenção orientou meus trabalhos e foi além, partilhando a vida com excelentes conselhos e reflexões. Agradeço por sua confiança e por ter levado-me em projetos profissionais importantes no ramo de aplicações espaciais, meu sonho desde criança. Projetos esses que já estão próximos de sua concretização e que colocaram nossos nomes na história de nossa universidade e do nosso Estado como pessoas que "Sacudiram"[1] o espaço.

Agradeço aos colegas e alunos que participaram comigo da 1ª Olimpíada Brasileira de Satélites onde nossa equipe chamada AutoSat, a única representante a nível universitário do estado do Ceará, obteve o primeiro lugar do Nordeste na competição e cujo parte do projeto foi inspirada no meu projeto de dissertação de mestrado.

Por fim, agradeço a todos aqueles que, de alguma forma, me ajudaram na caminhada com suas palavras de motivação, orações e serviço. Que Deus me ajude sempre a seguir o tripé que tenho buscado desenvolver desde minha juventude: traballho, estudo e oração. Creio que só assim iremos mudar para melhor a história de nosso país e da humanidade.

---

[1] Alusão a um de nossos projetos de nanosatélite chamado SACODE, um acrônimo para *Sistema para Análise e Coleta de Dados Experimentais*.

"Reze como se tudo dependesse de Deus e trabalhe como se tudo dependesse de você."
(Santo Inácio de Loyola, 1491-1556)

RESUMO

O espaço tem atraído cada vez mais a atenção de governos, grandes indústrias e universidades. Uma das estratégias mais populares nos últimos anos têm sido a adoção de nanossatélites para cumprir diferentes missões, que podem funcionar isoladamente ou em constelações. As universidades se destacam entre os agentes lançadores de nanossatélites, com mais de 600 lançamentos até 2022. Diante do crescimento de entidades que controlam missões espaciais, faz-se necessário que novos métodos de comunicação entre controle e satélite sejam implementados, para gerar melhor aproveitamento das janelas de comunicação, além de mais agilidade e de segurança na transmissão de dados. Abordo neste trabalho uma arquitetura de consórcio entre estações terrestres (*ground stations* ou (GS), em inglês) para que um sistema GSaaS, ou seja, *Ground Station As a Service*, funcione com baixo custo, confiabilidade e compartilhamento de recursos. Inicialmente, simulei com o *Matlab* uma missão de nanossatélite na órbita baixa para obter os parâmetros de tempo médio de comunicação, perda de propagação e em quais ângulos a comunicação seria mais afetada por fenômenos atmosféricos. Em seguida, implementei regras de negócios para comunicação entre estação terrestre e satélite usando conceitos de contrato inteligente. Montei uma *blockchain* para fornecer a infraestrutura de descentralização e criei um *webservice* para fornecer uma API de comunicação entre o nanosatélite e a *blockchain*. Simulei o processo de atualização do *firmware* no experimento e os resultados mostram que, para um *firmware* de 301Kb, fragmentado em partes de 32 *bytes*, o nanosatélite levou 19 minutos para realizar todas as requisições dos fragmentos de *firmware* diretamente à blockchain. Isso significa que seriam necessários pelo menos 3 janelas de comunicação ativa entre a estação terrestre e o satélite para realizar toda a atividade. Quando registrei na memória do servidor os resultados das requisições à blockchain e depois enviei uma a uma ao nanosatélite o tempo diminuiu para aproximadamente 6 minutos, ou seja, teve resultado menor que a média de tempo de janela de comunicação na simulação. Outra diferença é que esta estratégia permite que as GS do consórcio se comuniquem de forma "agnóstica" com os satélites, ou seja, a privacidade do firmware será preservada pela criptografia utilizada e de propriedade das transações na blockchain. O sistema também permite que instituições que possuem estações terrestres possam compartilhar os seus recursos e serem remuneradas por isso.

Palavras-chave: *blockchain*; *firmware*; nanosatélites educacionais; estações terrenas e sistemas descentralizados

ABSTRACT

Space has increasingly attracted the attention of governments, large industries, and universities. One of the most popular strategies in recent years has been the adoption of nano-satellites to fulfill different missions, which can work alone or in constellations. Universities stand out among the agents launching nano-satellites, with more than 600 launches by 2022. Given the growth of entities that control space missions, it is necessary that new methods of communication between control and satellite be implemented to generate better use of communication windows, as well as more agility and security in data transmission. Our work proposed a consortium architecture between ground stations (GS) so that a GSaaS system, i.e. Ground Station As a Service, works with low cost, reliability and resource sharing. Initially, we simulated with matlab a nanosatellite mission in LEO orbit to obtain the parameters of average communication time, propagation loss, and at which angles the communication would be most affected by atmospheric phenomena. We then implemented business rules for communication between ground station and satellite using smart contract concepts. We set up a blockchain to provide the infrastructure for decentralization and created a webservice to provide an API for communication between the nanosatellite and the blockchain. We simulated the firmware upgrade process in our experiment. The results show that for a 301Kb firmware, fragmented into 32byte chunks, the nanosatellite took 19 minutes to perform all requests for the firmware fragments directly on the blockchain. This means that it would take at least 2 or 3 communication windows between the ground station and the satellite to perform all the activity. When we recorded in the server memory the results of the requests to the blockchain and then sent them one by one to the nanosatellite the time decreased to less than 6 minutes, less than the average communication window time in our simulation. Another difference is that we allow the consortium GSs to communicate in an "agnostic"manner with the satellites, meaning that firmware privacy will be preserved by the cryptography used and ownership of transactions in the blockchain. The system also allows institutions that own ground stations to share their resources and be remunerated for doing so.

Keywords: blockchain; firmware; educational nanosatellites; ground stations and decentralized systems

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AWS	Amazon Web Services
COTS	Commercial-Off-The-Shelf
dAPP	Decentralized Applications
GEO	Geosynchronous Earth Orbit
GS	Ground Station
GS-BC	Groud Statation Blockchain Coin
GSaaS	Ground Station As a Service
HTTP	Hypertext Transfer Protocol
I2C	Inter-Integrated Circuit
IBFT	Istambul Byzantine Fault Tolerance
INSA	Institut National des Sciences Appliquées
JSON	JavaScript Object Notation
KDE	Kernel Density Estimate
LEO	Low Earth Orbit
MEO	Medium Earth Orbit
PWM	Pulse Width Modulation
RPC	Remote Procedure Call
SoC	System on a Chip
SPI	Serial Peripheral Interface
SPIFFS	SPI Flash File System
SQL	Structured Querry Language
UML	Unified Modeling Language
WEP	Wired Equivalent Privacy
WPA	Wi-fi Protected Access

LISTA DE SÍMBOLOS

δ	Preço por minuto do uso de uma estação terrestre.
λ	Taxa de crescimento diário da *blockchain* em megabytes.
Kb	Kbytes
Mb	Megabytes

SUMÁRIO

1 INTRODUÇÃO

Explano nesta seção a motivação por deste trabalho. Apresento o escopo do trabalho e, por fim, uma visão geral sobre a estrutura do documento.

1.1 Motivação

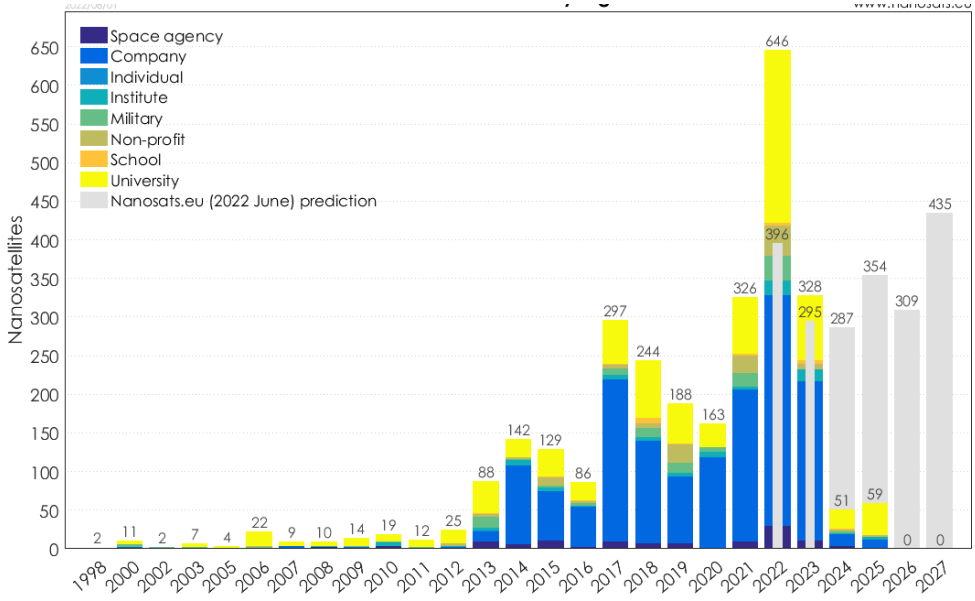
Um dos padrões de construção de satélites que se tornou popular nos últimos anos é o padrão CubeSat (NASON *et al.*, 2002) que são nanosatélites de formato cúbico e medem 10 x 10 x 10 cm. São divididos em unidades, sendo que 1 U corresponde à menor unidade do padrão. Seu formato é popular devido ao baixo custo de desenvolvimento e lançamento. Geralmente utilizam peças COTS (*Commercial-Off-The-Shelf*), que podem ser encontrados na internet. De acordo com a *nanosat database*[1], as universidades representam uma parte significativa da quota de mercado dos construtores de nanossatélites lançados em 2022. Cerca de 640 novos nanossatélites são esperados até ao final de 2022, o que representa um aumento de quase 98% face ao ano anterior. Algumas universidades não possuem estações terrestres para rastreamento, telemetria e controle de seus cubesats e precisam de parcerias públicas e privadas para alugar ou utilizar estações terrestres de outras entidades. Nesse contexto, soluções para reduzir os custos de pesquisas e missões espaciais podem aumentar a participação das universidades no ecossistema de nanossatélites.

A Figura 1 demonstra a expressiva representatividade que as universidades têm como construtoras de nanosatélites. Ainda que os números de projetos universitários tenham sido crescentes nos últimos 3 anos (2020 a 2022), os recursos financeiros e o número de estações terrestres para comunicarem-se com esses dispositivos continuam limitados e podem significar desafios sérios para o sucesso dessas missões. As estações terrestres são as antenas que enviam e recebem comando e dados de telemetria. Em geral, cada universidade ao redor do mundo que possui projetos de nanosatélites, conta com o apoio de outras instituições governamentais e/ou não governamentais para usar essas antenas. Algumas destas instituições são institutos de pesquisas e operam diversos outros satélites. A depender da frequência de operação das antenas, radio-amadores podem ajudar na missão de rastreio do cubesat, mas atualizar *firmware* ou receber alguma outra informação sensível e codificada é reservada ao controle da missão.

A Figura 2 representa o estado das missões de cubesats ao redor do mundo, ou seja,

[1] https://www.nanosats.eu

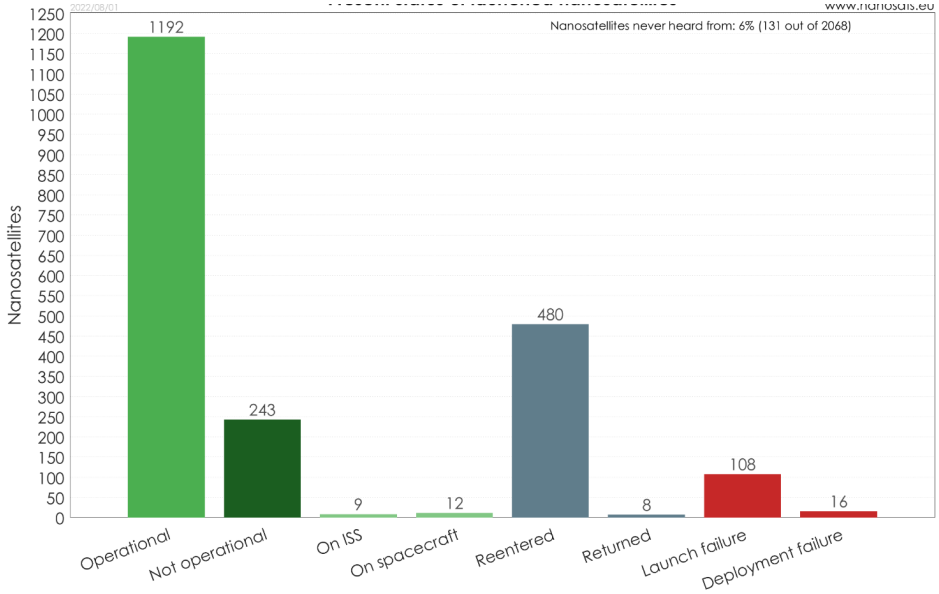
Figura 1 – Lançamento por Instituições



Fonte: https://www.nanosats.eu. Acessado em Setembro 2022 e mantido o orignal em inglês.

seus *status* de operação e exemplificam a concorrência para o uso das estações terrestres que esses nanosatélites demandam.

Figura 2 – Status das missões



Fonte: https://www.nanosats.eu. Acessado em Setembro 2022 e mantido o orignal em inglês.

Os nanosatélites seguem o mesmo padrão de funcionamento de um satélite de médio e de grande porte: trocam dados com o controle da missão, enviam e recebem dados e podem ser rastreados e terem suas órbitas alteradas. Todos esses processos dependem de antenas, operadores e tecnologias compatíveis para a realização das atividades. O processo de comunicação entre

estação terrestre e nanosatélite é bastante delicado. Primeiro, um dos bits de dado (enviado ou recebido) pode ser invertido acidentalmente durante a transmissão, por exemplo, devido a razões ambientais, como uma tempestade solar ou efeitos na atmosfera da Terra. Em segundo lugar, há outro desafio em relação à comunicação entre a estação terrestre e os nanossatélites. Cubesats operando em órbita terrestre baixa (LEO) levam entre 5 minutos a 15 minutos para esgotar a janela de comunicação com a estação terrestre (CARVALHO, 2019). Dentro de uma janela de tempo tão curta, a quantidade de dados transmitidos é mínima e a comunicação deve ser feita corretamente, posto que o nanossatélite pode levar horas para estar novamente visível para a estação na Terra. Outro ponto relevante é que os cubesats possuem um software embarcado, chamado firmware, que pode precisar ser atualizado periodicamente (GARRIDO B. GARCíA; MIGUEL, 1998) (GRECO; SNYDER, 2005), seja por causa de uma vulnerabilidade que foi descoberta e que necessite ser corrigida ou devido a novas funções que aquele nanosatélite irá realizar. Diante dos elevados números de nanosatélites em órbita e do limitado número de estações terrestres, algumas questões são relevantes para projetos espaciais, em especial, os que possuem restrições orçamentárias. Apresentamos algumas questões.

Questão 1: Como tornar mais eficiente, em questão de tempo, e seguro a comunicação entre satélite e controle da missão?

Questão 2: Como tornar acessível o serviço de estações terrestres pra que institutos de ensino, universidades e pequenas empresas do setor espacial consigam realizar as tarefas de rastreio, controle, envio e recepção de dados sem perder a segurança e confidencialidades de seus projetos?

Todos estes pontos exemplificam as complexas e desafiantes tarefas realizadas por engenheiros e pesquisadores ao construir estes pequenos satélites, além da grande tarefa de conseguir colocá-los em órbita.

1.2 Espoco do Trabalho

O objetivo deste trabalho é o desenvolvimento do conceito de descentralização do serviço de comunicação entre o controle da missão e o nanosatélite, visando aumentar a velocidade de difusão dos dados de comando e telemetria (envio dos dados, atualização de

firmware, mudança de missão ou correção de órbita). Para isso, implementei uma rede *blockchain*, proposta para ser executada de forma descentralizada nas estações terrestres, tornando a antena 'agnóstica' ao satélite, ou seja, ela se comunica com qualquer satélite que esteja autorizado na rede e faz isso usando, dentre outros, os conceitos de chaves públicas e privadas da *blockchain*. Para simplificar o desenvolvimento, escolhemos a atualização de *firmware* como parte do experimento. O protótipo desenvolvido consiste em 3 interfaces:

a) Computador simulando um servidor de uma estação terrestre.

b) Rede *blockchain* com 3 nós representando 3 estações terrestres, 1 nó RPC (*Remote Procedure Call*, do inglês) e 1 nó para o explorador de blocos (estes dois últimos conceitos são apresentados na próxima seção). A blockchain se comunica com a estação terrestre, simulada por um computador e com o nanosatélite por meio de *API* [2].

c) Nanosatélite. Um nodeMCU com ESP8266 [3] para prototipação e teste de conceito e o cubesat da PION [4], um kit educacional cujo computador de bordo possui um ESP32 e diversos sensores. [5]

1.3 Estrutura do Documento

i) Seção 2 - Fundamentos: fornece uma visão geral sobre os conceitos abordados no trabalho. Veremos um exemplo em diagrama de fluxo do processo de atualização de firmware de um cubesat em órbita, a definição e descrição do que é blockchain e seus conceitos, requisitos e exemplo de aplicações;

ii) Seção 3 - Concepção e implementação: discuto as escolhas da tecnologia de software e hardware para a implementação do protótipo;

iii) Seção 4 - Experimentos e resultados: reúne os procedimentos e dados das experiências realizadas e traz um comentário com a interpretação dos resultados obtidos;

iv) Seção 5 - Conclusão e projetos futuros: finalização do documento com um compilado de tudo o que foi tratado e o apontamento dos próximos passos.

---

[2] https://aws.amazon.com/pt/what-is/restful-api/

[3] O ESP8266 é um SoC (*System on a Chip*, em inglês) desenvolvido pela Expressif para facilitar a conectividade de placas e criação de solução dentro do ecossistema da Internet das Coisas – IoT. Possui memória *flash* de 4MB, conversor serial-USB e suporta o padrão de internet 802.11 b/g/n além dos protocolos de segurança WEP, WPA e outros. Outro detalhe interessante é que possui interfaces I2C, SPI e PWM. Possui 11 GPIO

[4] https://www.pionlabs.com.br/cubesat

[5] Uma evolução do ESP8266, focado em alta performance. Possui 4Mb de memória flash, ROM de 448 Kbytes, RAM de 520 Kbytes, *clock* máximo de 240 MHz e taxa de transferência 110 460800 bps

v) O apêndice e os anexos proporcionam conteúdo adicional e suporte para reprodução dos experimentos.

2 FUNDAMENTOS

Esta seção proporciona uma visão geral sobre os principais fundamentos teóricos deste trabalho. Discuto tópicos como nanostélites e o padrão cubesat, atualização de *firmware* em nanosatélites, tecnologia *blockchain*, com exemplos de sua aplicação em projetos aeroespaciais e outros.

2.1 Tipos de satélites

Os satélites artificiais são dispositivos desenvolvidos pelo homem que realizam suas missões e atividades no espaço. Estes equipamentos são fundamentais para observação da Terra, telecomunicações, exploração espacial e defesa.

Os satélites podem ser catalogados pelo peso. A Tabela 1 representa os sete principais agrupamentos.

Tabela 1 – Catalogação dos satélites pelo peso

Tipo	Peso
Grandes Satélites	Maior que 1000Kg
Satélites Médios	500 à 1000Kg
Minisatélites	100 à 500Kg
Microsatélites	10 à 100Kg
Nanosatélites	1 à 10Kg
Picosatélites	0,1 à 1Kg
Femtosatélites	Menor que 100g

Fonte: http://www.crn.inpe.br/conasat1/nanosatt.php

Outra característica importante são os tipos de posicionamentos que os satélites que orbitam a Terra podem estar alocados.

a) Órbita Terrestre Baixa ou LEO (*Low Earth Orbit*, em inglês): Órbita a uma altura entre 500 Km a 2.000 Km e possui uma velocidade que pode ir ale dos 25.000 Km/h. Uma característica é que a essa altura cobre uma área pequena do globo e, para aumentar a cobertura, é necessário trabalhar em conjunto com outros satélites e a esse tipo de colaboração podemos chamar de "trabalhar em constelação". Dados a altura e velocidade, em geral os satélites nessa órbita levam cerca de 90 minutos para finalizar uma volta completa ao redor da Terra. A Estação Espacial Internacional está nessa órbita.

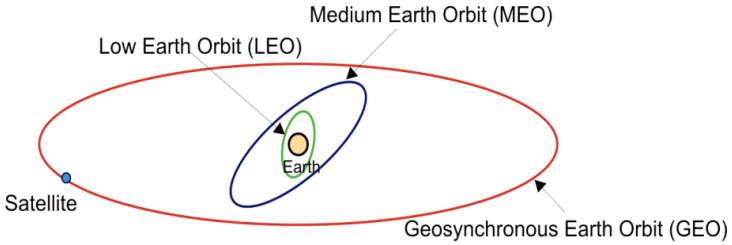
b) Órbita Terrestre Média ou MEO (*Medium Earth Orbit*): Pode também ser chamada de Órbita Circular Intermediária. Está posicionada entre 2.000 Km a 35.000 Km . As órbitas

podem variar entre circulares ou elípticas. Em geral essa órbita é usada para sistemas de comunicação e de defesa.

c) Órbita Geossíncrona ou GEO (*Geosynchronous Earth Orbit*): Satélites nessa órbita podem ser chamados de Geoestacionários. Deslocam-se com a mesma velocidade a qual a Terra rotaciona. Está aproximadamente a 35.000 Km e precisa de 24 horas para finalizar a volta completa ao redor do planeta. Com três satélites GEO, pode-se cobrir completamente as comunicações do planeta. Um exemplo de aplicação é na meteorologia.

A Figura 3 expressa os tipos de órbitas explicadas anteriormente. O foco do trabalho será na órbita LEO, visto que a maioria dos nanosatélites educacionais ou são lançados de balão meteorológicas ou estão na órbita LEO.

Figura 3 – Órbitas



Fonte: (CAKAJ *et al.*, 2011)

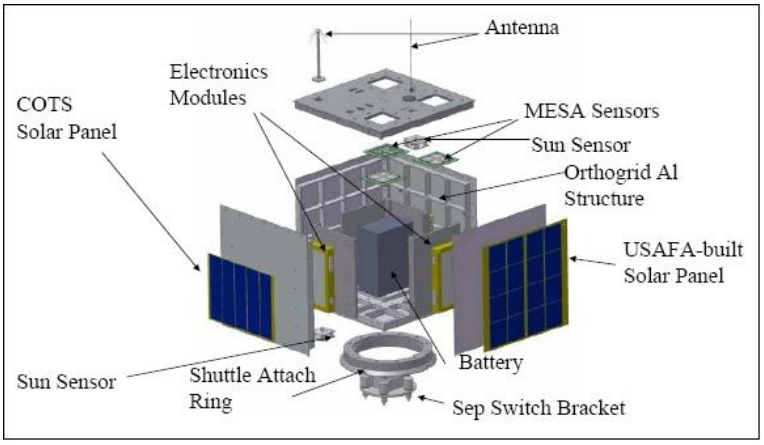
2.1.1 Padrão cubesat

Cubesat é um acrônimo das palavras em inglês *cube* e *satellite*, ou seja, cubo e satélite. Trata-se de um tipo de satélite de pequeno porte e foi proposto em 1999 pelos professores Jordi Puig-Suari e Bob Twiggs, ambos de universidades no estado da California. Eles desenvolveram um padrão de construção desses pequenos satélites para ajudar as universidades ao redor da Terra no trabalho de explorar o espaço. Com um padrão claro e objetivo, os projetos poderiam ser integrados, além de facilitar a troca de experiência e encontrar as peças no mercado.

O formato cúbico de 10 cm de aresta é chamado de "uma unidade"ou simplesmente U. Assim, um cubesat que tem apenas uma unidade é chamado de cubesat 1U. Tais satélites em miniatura podem ser colocados em órbita e lançados por foguetes ou outros veículos, como da própria Estação Espacial Internacional. Outra grande vantagem dos cubesats é que a grande maioria dos seus componentes são COTS (*Commercial off-the-shelf*, em inglês) e podem facilmente serem achados à venda na internet. A Figura 5 representa um exemplo de um cubesat

chamado Falconsat2 da Agência Espacial Europeia.

Figura 4 – Exemplo de Cubesat



Fonte: https://www.eoportal.org/satellite-missions/falconsat-2. Acessado em Setembro 2022

Em resumo, os cubesats, embora sejam pequenos satélites, sofreram positivas melhorias nos últimos anos e desenvolvem missões variadas, desde estudos cientificos até aplicações de observação da Terra, defesa e telecomunicações. Assim como todo objeto eletrônico os satélites possuem software de bordo, chamado de firmware e eventualmente podem necessitar de atualizações. Na próxima seção exploraremos esses aspecto.

Por fim, a NASA (*National Aeronautics and Space Administration*, em ingês) produziu um manterial muito interessante chamado de *CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers* [1]que mostra padrões de desenvolvimento de cubesats para quem está iniciando no ramo.

2.1.2 Atualização de firmware

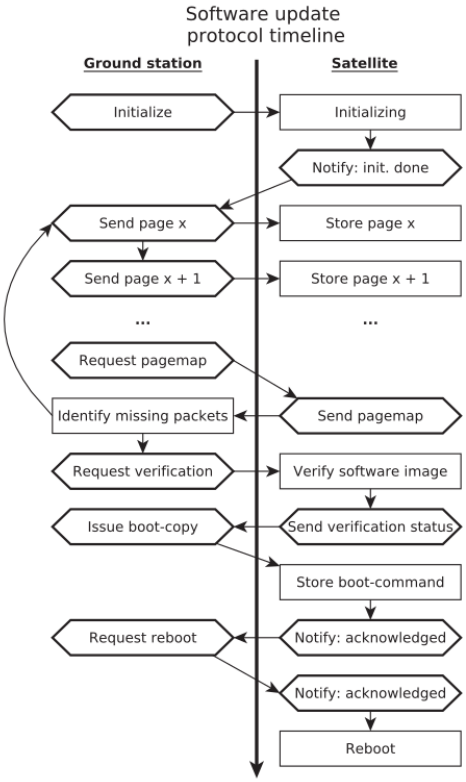
Em geral, o procedimento de atualização remota de firmware em dispositivos espaciais engloba quatro partes bem conhecidas (SUNTER *et al.*, 2016): a primeira parte é dedicada a preparar a imagem do firmware, fragmentando-o (LEINER B.; HUBER, 2007) (ROSA; RUFINO, 2010) em pequenos blocos e (BING, 2006) compactado-os para serem enviados pela estação terrestre. O segundo é o protocolo de transferência e a terceira parte está relacionada ao armazenamento na memória do nanossatélite e pós-processamento, ou seja, verificar se a imagem está completa e não corrompida, caso tenha sido alterada, uma nova requisição do pacote alterado é solicitado. Após o processo de verificação, processos internos apontam o endereço de

[1] https://www.nasa.gov/sites/default/files/atoms/files/nasa_csli_cubesat_101_508.pdf

inicialização para o endereço de memória onde o firmware recém recebido está e inicializa-se o sistema. Códigos de correção de erros (ECCs) são comumente usados na etapa de transmissão e gravação de dados de modo a garantir ao máximo o sucesso do processo (WOOD *et al.*, 2007) (SILVA *et al.*, 2020) (HAMMING, 1950).

A Figura 5 representa a linha do tempo do processo de atualização de firmware da missão do cubesat ESTCube-1, cuja missão era a observação do Sol (SUNTER *et al.*, 2016). A Tabela 2 , por sua vez, evidência alguns exemplos de missões de cubesats que tiveram a necessidade de realizar atualizações de firmware durante o seu tempo em órbita. Chamo a atenção para o AeroCube4 que precisou atualizar 150 vezes o firmware para corrigir a órbita e evitar a queda do nanosatélite prematuramente. A atualização de firmware ainda deve obedecer altos padrões de codificação e de segurança para dificultar que invasores interceptem o processo ou invadam o sistema.

Figura 5 – Fluxo de tempo para atualização de *firmware*



Fonte: (SUNTER *et al.*, 2016). Acessado em Setembro 2022

Neste projeto desenvolvi dois firmwares que realizam tarefas parecidas e simples. O primeiro firmware faz um led verde piscar e o outro um led vermelho. Me detive em focar no processo de fragmentação do firmware, registro desses fragmentos na blockchain, consulta

Tabela 2 – Exemplos de missões cubesats que atualizaram *firmware* em órbita

Nome cubesat	Tempo da missão	Nº de atualizações
AeroCube4 (GANGESTAD *et al.*, 2013) (GANGESTAD *et al.*, 2014)	18 meses	150
ESTCubete-1 (BRIDGES *et al.*, 2013)	Não informado	
STRAND-1 (BRIDGES *et al.*, 2013)	24 meses e com muitos defeitos	Não informado
UWE-3 (BUSCH *et al.*, 2013)	Não informado	3

Fonte: o autor.

na base de dados descentralizadas, solicitação dos fragmentos, remontagem da imagem e nova inicialização do sistema.

2.2 Tecnologia blockchain

Nesta seção exploraremos os principais conceitos sobre a tecnologia blockchain. Chamamos a atenção, em especial, para os conceitos de *blockchain não-permissionada, contratos inteligentes, tokens e descentralização*, pois são usados em nossos experimentos.

2.2.1 Definições principais

A Blockchain é uma lista encadeada e crescente de registros que são agrupados em blocos ligados criptograficamente um ao outro. Um bloco é criado quando os nós da rede, ou seja, os computadores ligados na rede, obtém o consenso de quem têm a autoridade de escrever um novo registro. A blockchain tem alguns componentes: dados, hash, hash anterior e metadados (carimbo de hora/data e número do bloco).

- Dados: Podem ser simples *strings*, frases, fragmentos de textos ou uma lista de transações;

- Hash: É um identificador único para o block, que funciona como uma impressão digital.

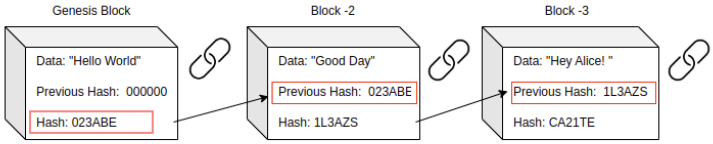
- Hash anterior: A hash do bloco anterior, se o bloco anterior não existir, o bloco atual é chamado de gênesis e a hash é uma sequência de zeros;

- Metadados: Informações sobre o número do bloco, data e hora.

Observe na Figura 6 um exemplo em alto nível de um fragmento de uma blockchain. Note que o encadeamento é fixado pela *hash* anterior que precisa ser idêntica ao campo *previous hash* no bloco atual. Uma das características mais famosas da blockchain é sua segurança. No exemplo, para que um bloco de dados já registrado seja alterado (imagine tentar alterar o

block 2), seria necessário que todos os blocos anteriores fossem alterados e validados pelos nós computacionais da rede e em uma rede com centenas ou milhares de nós (como são as blockchains públicas) o esforço computacional é tão alto que torna-se economicamente inviável.

Figura 6 – Representação em alto nível de uma *blockchain*



Fonte: Próprio autor

Existem dois tipos principais de blockchains: permissionadas e nNão-permissionadas (BRAGA, 2017). Em uma blockchain permissionada, o conjunto de registros distribuídos só podem ser acessados por algumas pessoas ou nós (computadores) que foram autorizados a fazê-lo pelo administrador, esse formato também é chamado de blockchain privada. No modelo não-permissionado, também conhecido como blockchain pública, não há restrições e um administrador não controla a participação dos outros nós. Outro conceito amplamente utilizado é o contrato inteligente ou em inglês *smart contract*. O conceito de contrato inteligente não é novo, mas a blockchain da Ethereum [2] permitiu que ele pudesse ser implementado de forma confiável. Os contratos ingelitentes são códigos executados automaticamente ao atingir marcadores específicos e são armazenados em cada nó da rede, isto é, é um armazenamento descentralizado. Usei esse padrão como base das transações que realizei. Outro conceito essencial é o de *web3* (LIU *et al.*, 2022), um nova iteração da internet baseada na tecnologia blockchain, que incorpora conceitos como a descentralização e uma economia baseada em tokens. Faz-se necessário mencionar que ESA no documento chamado *Blockchain and Earth Observation* [3] e a NASA (BEAUJARDIERE *et al.*, 2019) já manifestaram interesse em usar Blockchain.

Contratos inteligentes

O conceito de contrato inteligente é um dos fundamentos que mais atrai a indústria, academia e governos em geral. O termo *smart contract* (contrato inteligente e inglês) foi cunhada pelo cientista da computação Nick Szabo [4], na decáda de 90. Ele desejava enfatizar a possibilidade de permitir mais transparência no comércio eletrônico.

[2] https://ethereum.org/pt-br/
[3] https://eo4society.esa.int/wp-content/uploads/2019/04/Blockchain-and-Earth-Observation_White-Paper-April-2019.pdf
[4] https://firstmonday.org/ojs/index.php/fm/article/view/548

A carência de tecnologia que permitisse a evolução e propagação de sua solução tornou o termo esquecido até o surgimento da *blockchain*. Os contratos inteligentes são blocos de códigos autoexecutáveis (SUJEETHA; PREETHA, 2021) (MONTES *et al.*, 2019) que são hospedados nos nós da blockchain assim como os registros das transações. Assim, os contratos inteligentes são verdadeiras aplicações descentralizadas que processam transações e realizam escrita e consulta nos registros no banco de dados descentralizado ou blockchain. Aplicações que são executadas de forma descentralizadas na blockchain são habitualmente chamadas de dAPP (*Decentralized Applications*, do inglês).

Criptomoedas e tokens

Um dos desafios no inicio da primeira blockchain, o *Bitcoin* [5], está descrito no seu *white paper* chamado *Bitcoin: A Peer-to-Peer Eletronic Cash System* [6] que descreve qual deveria ser o incentivo para que os nós computacionais desprendessem seus recursos para resolver problemas matemáticos para autenticar transações. A ideia de criar uma moeda digital que pudesse ser trocada por algum benefício e, talvez, até por dinheiro fez sentido. Afinal, os computadores e servidores consomem energia, desgastam seus componentes e precisam de manutenção. Esse conceito evoluiu e concretizou-se no conceito de criptomoedas. Os *tokens* são uma "representação digital"de algum ativo registrado na blockchain. De certa forma, assim como o papel moeda representa uma certa quantia de uma determinada moeda, um *token* representa uma certa quantia na blockchain.

Algoritmos de consenso

O algoritmo de consenso é um mecanismo que tem o poder de dar a permissão para os nós blockchains, isto é, os computadores e servidores que executam a aplicação blockchain, trabalhem de forma organizada e segura. Ele garante que todos os nós cadastrados e validados possam concordar com uma única fonte de verdade, um exemplo é a escrita de um registro na rede, e isso deve acontecer mesmo quando algum dos nós (um ou mais) falhem. Em outras palavras, o algoritmo de consenso atribui ao sistema tolerância à falhas (PAHLAJANI *et al.*, 2019) (GAO *et al.*, 2019), por exemplo à Falha Bizantina[7].

Em uma configuração de arquitetura centralizada de sistema, uma única entidade exerce total poder sobre o sistema. Em sua maioria, neste tipo de configuração, uma única aplicação têm a capacidade de fazer qualquer alteração nos registros e isto pode representar um

[5] https://bitcoin.org/pt_BR/
[6] https://bitcoin.org/bitcoin.pdf
[7] https://wiki.hyperledger.org/display/LMDWG/Byzantine+Fault+Tolerant+Consensus

grande perigo em caso de sequestro das credenciais de *login* do sistema. Em uma configuração descentralizada, como é o caso de projetos com blockchain, é diferente. Uma das principais questões para este tipo de arquitetura é: como chegar a um acordo sobre quais nós, computadores ou entidades podem registrar informações no banco de dados e como isso será orquestrado para que não ocorra, por exemplo, de duas ou mais entidades escrevendo ao mesmo tempo no banco de dados? A solução a este desafio possibilitou a blockchain existir e mostrar-se extremamente aderente a vários ambientes e projetos.

Exitem vários tipos de algoritmos de consenso. A Tabela 3 é um resumo dos três algoritmos mais conhecidos e usados. Para simplificar usarei a sigla *A.C* para representar algorítimo de consenso.

Tabela 3 – Exemplos de Algoritmos de Consenso

Nome	Descrição
Proof of Work - PoW	-Um dos primeiros e principais *A.C* utilizados. Criado para suportar a criptomoeda *bitcoin*. Ele estabelece que, através da resolução de alguns problemas matemáticos de crescente complexidade, a entidade que primeiro resolver o problema proposto à rede, será recompensada com a capacidade de escrever o registro na Blockchain. Esse nó ou entidade ganha o nome de *nó minerador* e recebe uma recompensa por isso, no caso o *bitcoin* ou outra criptomoeda. Devido a alta complexidade, exige muito poder computacional e recurso energético.
Proof of Stake - PoS	-Proposta com uma das primeiras alternativas ao PoW, não possui o conceito de *nó minerador*. Em geral para este tipo de consenso o nó computacional da rede precisa possuir uma certa quantidade de criptomoeda da rede. Neste cenário o nó "reserva"uma certa quantidade desta criptomoeda para poder fazer o registro na Blockchain, neste momento, sua reserva não poderá ser alterada até concluir a tarefa. Funciona como uma "aposta"que cada nó faz para saber quem terá o poder de registrar dados. O nó vencedor ganha a capacidade de escrever e a recompensa de um percentual do valor da transação. Quanto mais fundos o nó guardar mais recompensa receberá.
Proof of Authority - PoA	- É um *A.C* baseado em reputação e introduz uma forma prática e eficiente para solucionar o consenso, em especial em redes permissionadas. Diferente do PoS, onde os nós colocam na aposta suas criptomoedas, no PoA eles colocam na aposta sua "reputação". A segurança da rede é reforçada pelos nós previamente validados e que arbitrariamente são sorteados para ganhar a capacidade de escrever os registros na rede. A *Mircosoft Azure* [8] tem projetos com este *A.C*. Caso se um dos nós for reconhecido como nó intruso ou invasor, ele perde a capacidade de participar da rede e é banido.

Fonte: o autor.

Outros exemplos de algoritmos de consenso são: *Proof of Burn* [9], *Proof of History*[10] da blockchain *Solana*[11] dentre outros.

[9] https://academy.binance.com/pt/articles/proof-of-burn-explained
[10] https://solana.com/news/proof-of-history
[11] https://solana.com/

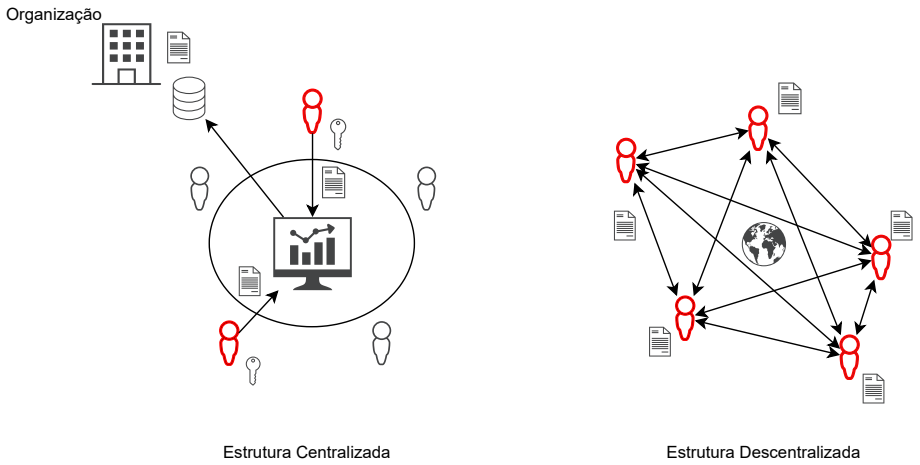
Blockchains permissionadas e não permissionadas

Antes de explorar de forma mais aprofundada as blockchains permissionadas e não permissionadas, iniciarei apresentando a diferença entre os sistemas centralizados e descentralizados.

Um sistema centralizado oferece grande poder para a entidade centralizadora escrever, atualizar, deletar e transferir registros e repostas. Para muitos projetos comerciais, especialmente que não possuem característica de aplicação crítica, esses sistemas são excelentes. São fáceis de construir, usam formatos de banco de dados que se acoplam muito bem com os outros subsistemas e podem realizar até tarefas complexas. Um exemplo é sistemas de controle de estoque, controles financeiros de empresas. Mas nem todos os projetos conseguem ser suportados por este tipo de estrutura. Neste cenário, uma das alternativas é a arquitetura descentralizada.

No cenário de descentralização, todo o sistema é espelhado em cada um dos servidores ou nós computacionais da rede. São as chamadas *blockchains* descentralizadas (PUTHAL *et al.*, 2018). A Figura 7 representa a diferença entre o conceito de uma aplicação centralizada versus uma aplicação descentralizada. Neste exemplo, os usuários em vermelho estão interagindo com o sistema. Observe que na estrutura centralizada, somente quem interage com a aplicação possui a capacidade de escrever e enviar dados. Estes dados são enviados diretamente para o banco de dados da organização. Na estrutura descentralizada, todos os participantes possuem uma cópia exata de todos os dados que foram transacionados na rede.

Figura 7 – Sistema Centralizado x Descentralizado

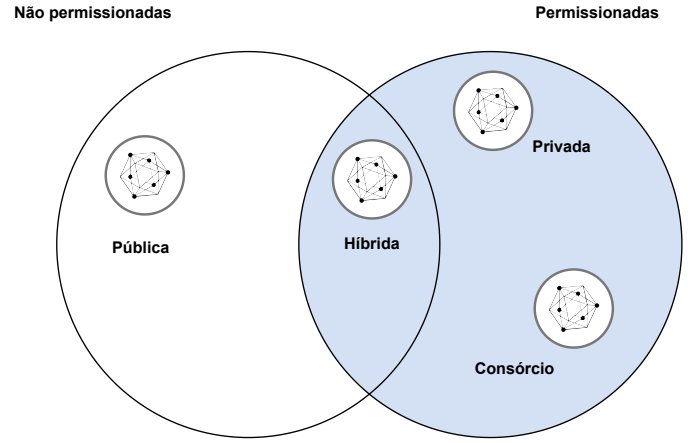


Fonte: Próprio autor

Este conceito é importante, pois o mesmo acontece com as blockchain. As blockchains podem ser públicas, híbridas ou privadas. A Figura 8 visualmente resume este conceito

que explicaremos a seguir.

Figura 8 – Tipos de Blockchain



Fonte: Próprio autor

Nas blockchains *não permissionadas ou públicas* qualquer pessoa ou entidade pode participar da rede. Neste cenário, a blockchain é completamente descentralizada e todos os nós são orquestrados por um algoritmo de consenso e obedecem todas as regras de autenticação e segurança. Todos os membros da rede têm os mesmos direitos e deveres ao acessar a blockchain. Eles são chamados de *mineradores* e recebem recompensas por participarem da rede.

Nas blockchain *permissionadas ou privadas* os membros participantes foram previamente autorizados para ingressar na rede. Não há necessidade da criação ou consumo de criptomoedas, pois não existe o conceito de *mineradores*. Assim, essas blockchains podem ser totalmente centralizadas, ou seja, a empresa ou instituição que usa a solução pode hospedar em seu servidor todos os nós da rede.

Outra possibilidade de rede permissionada é o conceito de *consórcio*. Neste formato duas ou mais instituições podem se juntar para distribuir os nós da rede em seus servidores. Logo, já não é mais uma blockchain privada centralizada, mas descentralizada. Todos os nós se comunicam entre si e são controladas por uma entidade ou grupo.

Nas blockchains *híbridas* a rede é controlada por uma única organização, mas com um nível de supervisão realizado por alguma blockchain pública, que participa realizando certas validações das transações. Um exemplo deste tipo de arquitetura é a *IBM Food Trust* [12] que atua na cadeia de suprimento de alimentos.

Criptografia na blockchain

[12] https://www.ibm.com/br-pt/products/supply-chain-intelligence-suite/food-trust

A área de conhecimento da criptografia é muito ampla. Alguns dos exemplos mais relevantes de técnicas usadas na blockchain são: criptografia simétrica, assimétrica, assinaturas digitais e funções *hash*.

A criptografia simétrica usa uma única chave para criptografar e descriptografar a informação. A criptografia assimétrica usa duas chaves diferentes, uma chave pública e uma privada. Por ter mais chaves para a assinatura, o modelo assimétrico é mais completxo e mais lento, mas atribui maior segurança.

A criptografia com *hash* é considerada uma técnica altamente segura, e aponta-se que na função *hash* não se usa chaves. O *hashing* usa um algoritmo para receber uma entrada que pode ter qualquer tamanho e produz uma saída de tamanho único, ou seja, você pode colocar uma *string* com 1 caractere ou 1000 e a saída da função terá o mesmo tamanho e quantidade de caracteres. Na blockchain do *Bitcoin* usa-se a SH-256 (desenvolvida pela Agência de Segurança Nacional dos Estados Unidos da América) e na *Ethereum* usa-se a *keccak-256*[13]. As funções *hash* possuem várias características aderentes aos projetos blockchain. Possuem baixas ocorrências de colisão (chance de duas entradas diferentes resultarem em uma mesma saída), as entradas não podem sofrer reversão (engenharia reversa para descobrir a entrada), são determinísticas (qualquer entrada 'A' sempre produzirá a mesma saída) e mesmo uma pequena alteração feita na entrada produzirá uma saída diferente, o que fortifica o conceito de incorruptibilidade dos dados armazenados. A Tabela 4, produzida pela Phemex [14], representa onde são usados as técnicas de criptografia na blockchain.

Tabela 4 – Uso de criptografias

Uso na Blockchain	Tipo de Criptografia
Transações entre os nós	Assimétrica
Mineração e verificação de transações	Hash
Imutabilidade dos registros	Hash
Proteção de dados em contratos inteligentes	Hash

Fonte: o autor.

Descentralizando a comunicação entre controle da missão e o satélite

Como completo a teoria dos fundamentos da tecnologia blockchain, o próximo passo é estudar e analisar quais poderiam ser as vantagens de utilização desta tecnologia para a descentralização da comunicação entre o controle da missão e o satélite. A Tabela 5 evidência

[13] https://keccak.team/
[14] https://phemex.com/pt/academy/what-is-blockchain-cryptography

alguns pontos levantados.

Note que a proposta do uso de *blockchain* para gerar um sistema descentralizado de controle e comunicação entre a missão de controle e o nanosatélite aumenta principalmente o tempo de disponibilidade de interação. Logo, com a criação de uma rede de estações terrestres participantes de um sistema descentralizado, os controles das missões podem comunicar-se mais rapidamente com seus satélites desfrutando da característica de comunicação*agnóstica*, isto é, uma determinada estação terrestre não se limita somente a comunicar-se com os satélites pré-determinados, mas aproveita praticamente 100% de sua disponibilidade de serviço para comunicar-se com qualquer satélite que seja participante da rede de forma automatizada.

Tabela 5 – Vantagens na descentralização na comunicação com *blockchain*

Característica	Potencial Benefício
Automação	-Automatizar, com uso de contratos inteligentes, rotinas e processos, por exemplo, calibração de sensores ou correções de órbita, podem ser executados sem intervenção humana. Pode-se, ainda, implementar no contrato inteligente a gestão automatizada das bandas de comunicação estação-satélite de forma a otimizar as janelas de comunicação.
Segurança	-A infraestrutura descentralizada com *blockchain* pode ser combinada com técnicas criptográficas modernas para preservar a privacidade de códigos, binários e transações, mantendo o código de *firmware* de cada fabricante confidencial.
Monetização	-A possibilidade de *tokenizar* (criar um ativo monetário) de rotinas e alguns processos, pode gerar uma fonte adicional de renda para as instituições que mantêm o serviço e até mesmo atrair iniciativas privadas para investir em projetos.
Disponibilidade	-Se uma das estações terrestres usadas pelo controle da missão falhar ou estiver indisponível, no sistema descentralizado o serviço continua operante equanto possuir ao menos um nó computacional ativo.
Compartilhamento de custos	-Universidades e outros projetos terão a oportunidade de compartilhar seus custos de comunicação entre estações terrestres e nanossatélites, gerando economia nos gastos e missões espaciais mais baratas.

Fonte: o autor.

A próxima seção explicará melhor como é na prática a proposta de descentralizar a comunicação entre o controle da missão e o nanosatélite propondo um consório de entidades detentoras de estações terrrestres para prestar serviço no estilo GSaaS [15], (*Ground Station As a Service*, do inglês).

Outros exemplos de uso da tecnologia blockchain

No intuito de ilustrar a aderência dessa tecnologia em diversas áreas e industrias, na Tabela 6 mostra um quadro geral com links para alguns projetos de pesquisa reunidos pelo

[15] https://www.linkedin.com/pulse/ground-station-service-gsaas-ganesh-sahai/

Blockchain Research Institute [16]. Dos mais de 100 projetos de pesquisa, reuni 5 por possuírem afinidades à indústria.

Tabela 6 – Exemplos de aplicações de *blockchain* relacionados à indústria

Tema	Link
1- Veículos Autônomos	https://www.blockchainresearchinstitute.org/project/autonomous-vehicles-and-blockchain/
2- Blockchain e Telecomunicações	https://www.blockchainresearchinstitute.org/project/blockchain-transformation-in-telecommunications
3- Conectividade Distriuída	https://www.blockchainresearchinstitute.org/project/distributed-connectivity
4- Blockchain e 5G	https://www.blockchainresearchinstitute.org/project/5g-bypass
5- Blockchain e Cidades Inteligentes: Dubai	https://www.blockchainresearchinstitute.org/project/blockchain-for-smart-city-infrastructure

Fonte: o autor.

Um outro conceito que é explorado pelo instituto como grande oportunidade para a industria é o da *trivergência* ou seja, o tripé : *blockchain*, inteligência artificial e internet das coisas [17]. As Figuras 9 e 10 representam o interesse da indústria e do mercado nesse tripé. É possível observar na Figura 9 a grande expressão que os Estados Unidos e a China possuem no setor. Isso nos chama a atenção, pois são 2 países altamente industrializados e que investem com grande impacto em tecnologia e inovação para criar produtos com forte valor agregado.

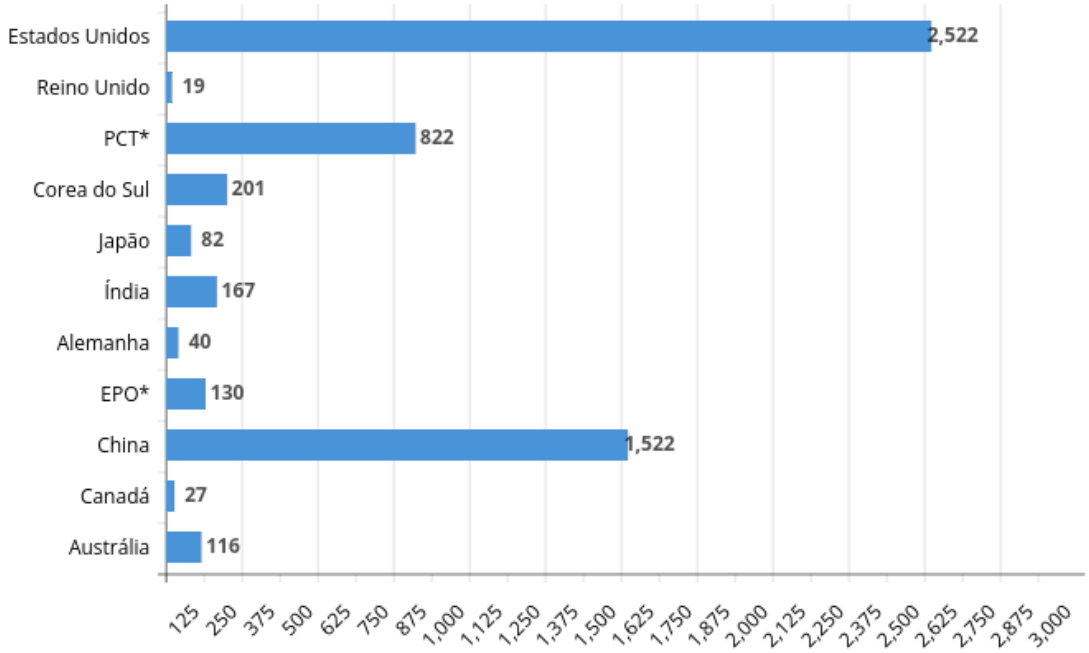
Na Figura 9 os pontos marcados com *, ou seja, PCT e EPO significam em português respectivamente Instituto Europeu de Patentes e Sistema Internacional de Patentes. Na Figura 10 citamos as empresas que mais têm investido no conceito de trivergência (seus nomes originais em inglês foram mantidos). Empresas como IBM, Intel, Americam Express e Salesforce, gigantes da tecnologia a nível mundial, são representadas no recorte desse gráfico de 2022.

Nas próximas páginas abordarei a metodologia do experimento, elencarei quais são os recursos usados para realizar os testes e analisarei os resultados de forma individualizada e geral.

[16] A base do trabalho do Blockchain Research Institute é a pesquisa. O programa de pesquisa do inclui dezenas de white papers, relatórios e estudos de caso sobre o impacto do *blockchain* na empresa, na indústria e na economia com mais de 100 projetos e centenas de pesquisadores ao redor do mundo. Sua sede é no Canadá. Site: https://www.blockchainresearchinstitute.org/

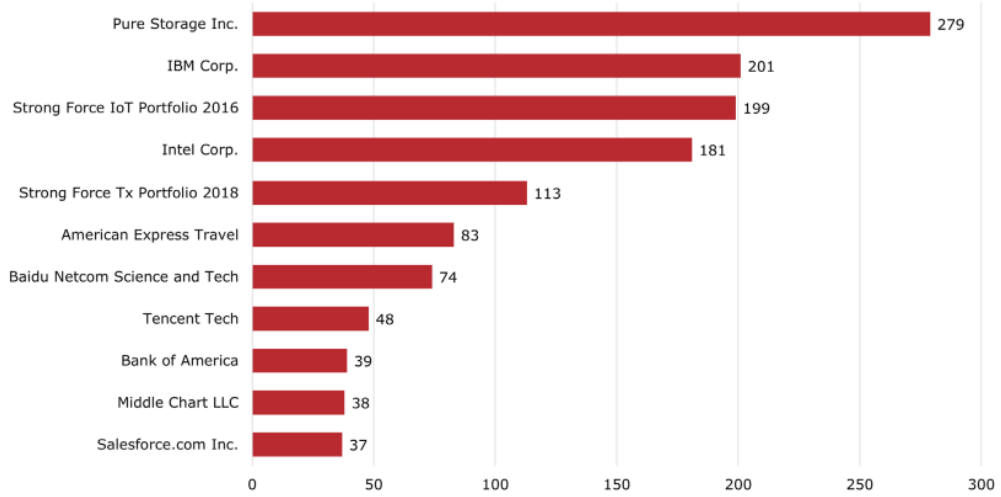
[17] https://www.blockchainresearchinstitute.org/project/patents-at-the-trivergence/

Figura 9 – Publicações de patentes com base no conceito de trivergência por país



Fonte: Adaptação do autor inspirado em: Patentscope, World Intellectual Property Organization, em 23 April 2022.

Figura 10 – Empresas que apresentam pedidos de patentes trivergentes



Fonte: Patentscope, World Intellectual Property Organization, as of 23 April 2022

3 CONCEPÇÃO E METODOLOGIA

Esta seção descreve o conceito do uso de blockchain na comunicação entre o nanosatélite e o controle da missão através de estações terrestres descentralizadas. De antemão, são apresentados os requisitos e as tecnologias que utilizei no projeto e suas funcionalidades. Em seguida,discorrerei acerca da implementação a nível de experimento do uso de blockchain em atualização de firmware de um nanosatélite.

3.1 Requisitos

Para desenvolver o conceito proposto, primeiro precisei identificar os requisitos que o projeto deve atender.

3.1.1 Requisitos Gerais

a) Implementar um sistema blockchain capaz de armazenar dados de forma descentralizada;
b) APIs no padrão REST para servirem de ponte entre o nanosatélite e o sistema blockchain;
c) Implementar banco de dados MySQL[1], linguagem baseada no SQL;
d) Implementar um protótipo em hardware para ler os dados armazenados na blockchain;
e) Fragmentar o arquivo *firmware-green.bin* e enviar para a blockchain e para o MySQL;
f) Realizar medições e obter métricas sobre o tempo para o hardware ler os dados da blockchain, tamanho de firmware, número de pacotes, tempo para executar as rotinas, tamanho de código e consumo de recursos da blockchain;
g) Implementar uma simulação em Matlab para obter dados referentes À duração de cada janela de observação.

3.1.2 Escolhendo a tecnologia

Hardware

Listamos alguns dispositivos de prototipação geral que nos permitem criar uma prova

[1] https://www.mysql.com/

de conceito aceitável antes de implementarmos o experimento em um nanosatélite. O *hardware* necessário para implementarmos a prova de conceito precisa:

a) Conexão com à internet, sem precisar acrescentar outros módulos;
b) Capacidade de processar dados;
c) Ser programável em linguagem C++ ou *Python*;
d) Facilidade em acessar memória *flash*;
e) Plataforma de desenvolvimento com *debug* e monitor.
f) Facilidade em encontrar no mercado nacional;
g) Placas de desenvolvimento que já temos no laboratório, para evitar tempo e custos adicionais na aquisição;
h) Possua um microcontrolador que já sido usado em algum de vôo (balão atmosférico ou órbital).

A Tabela 7 nos apresenta o resultado a pontuação dos itens listados. Para o sistema de pontuação o número *2* significa "Muito relevante", o número *1* para "Pouco relevante" e número *0* para "Irrelevante ou simplesmente não aplicável" para os itens elecandos.

Tabela 7 – Lista de dispositivos e pontuação

Placas	a	b	c	d	e	f	g	h	**Total**
ESP8266	2	2	2	2	2	2	2	1	**15**
BeagleBone	0	2	2	2	2	0	0	0	**8**
Arduino Leornardo	0	2	2	2	2	2	2	0	**12**
Hércules (ARM R4)	0	2	2	1	2	0	2	2	**11**
Raspberry pi 3 (ARM A9)	2	2	2	2	2	1	0	2	**13**

Fonte: o autor.

A plataforma eleita foi a ESP8266 por obter a maior pontuação.

Após a escolha do *hardware* para a prova de conceito, deveremos escolher um cubesat ou pelo menos um computador de bordo capaz de suportar nossos experimentos. Ele precisará obedecer os mesmos apontamentos (a até h) listados anteriormente. A Tabela 9 expressa a lista e os pontos obtidos. Perceba que a lista de opções é bem menor que a lista para o protótipo, um dos motivos é a capacidade de obter a placa para uso no experimento (seja por disponibilidade ou por preço).

Após a análise feita, definimos os *hardwares* que iremos usar, tanto para prova de conceito (o ESP8266) quanto para o cubesat (o PION Cubesat Educacional). Os computadores

Tabela 8 – Lista cubesats ou computadores de bordo e pontuação

Placas	A	B	C	D	E	F	G	H	**Total**
PION Cubesat Edu.	2	2	2	2	2	2	2	2	**16**
OpenOBC	0	2	2	2	2	0	1	0	**9**
OBC ISIS	0	2	2	2	2	0	0	2	**10**

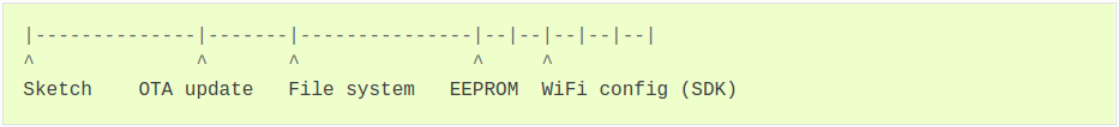
Fonte: o autor.

de bordo *OpenOBC*[2] é um computador de bordo de baixo custo (aproximadamente U$ 100.00) desenvolvido na Universidade Federal do Ceará. Já *OBC ISIS*[3] é um computador de bordo internacional amplamente usado em missões de cubesat, mas não temos em nosso laboratório e seu preço e taxa de importação não puderam ser cobertos neste trabalho.

Sistema de arquivos nos ESP8266/32

Mesmo que o sistema de arquivos esteja armazenado no mesmo *chip flash* do programa, a programação e escrita de um novo arquivo não modifica o conteúdo do sistema de arquivos. Isso permite usar essa arquitetura de arquivos para armazenar dados, arquivos de configuração e até um novo firmware. A Figura 11 representa o esquema da memória *flash* nesses dipositivos. O tamanho do sistema de arquivos depende da versão da placa. Na que temos a disposição é 4 Mb.

Figura 11 – Esquema memória *flash*



Fonte: ESP8266 CORE [4]. Acesso em Setembro de 2022

o SPIFFS é o sistema de arquivos [5] original e ideal para aplicações com restrição de espaço e RAM que utilizam muitos arquivos pequenos. No próximo capítulo iremos explicar como usamos as funções para acessar, escrever e ler a SPIFFS.

PION Cubesat Educacional e a 1ª Olímpiada Brasileira de Satélites Escolhemos colocar uma pequena subseção neste capítulo com o intuito de agradecer e, ao mesmo tempo, explicar como chegamos a obter um PION Cubesat Educacional. Durante o ano de 2021 e 2022, alguns membros do programa de pós-graduação do curso de engenharia de teleinformática da Universidade Federal do Ceará, juntamente com alguns alunos dos cursos de engenharia de

[2] https://repositorio.ufc.br/handle/riufc/24500
[3] https://www.isispace.nl/product/on-board-computer/
[5] https://arduino-esp8266.readthedocs.io/en/latest/filesystem.htmlflash-layout

telecomunicações, elétrica e mecânica concorreram na 1ª Olimpíada Brasileira de Satélites [6] e obtiveram o 1º lugar do Nordeste na competição, que previa dentre outros, o lançamento do cubesat para experimentos em balão meteorológico até a altura de 35 Km. A equipe chamada de *Autosat* recebeu um PION Cubesat Educacional para realizar seus experimentos durante a competição, após ter sido aprovada na 1ª fase. Este cubesat então ficou à disposição no laboratório do Departamento para uso em nossos experimentos.

Blockchain

Diante do objetivo proposto e dos requisitos do projeto, também realizamos um estudo para listar as *blockchains* que fossem mais aderentes ao objetivo de nosso projeto. Para isso, observamos:

a) Maturidade da *blockchain* a nível de documentação;
b) Quantidade de projetos em produção ou desenvolvimento que usam essa *blockchain*;
c) Possibilidade de criar redes permissionadas e não permissionadas;
d) Opção de escolhermos qual algorítimo de consenso desejamos usar;
e) Tamanho da comunidade de desenvolvedores;
f) Demonstre menor curva de aprendizado;
g) Capacidade de integrações com APIs;
h) Permite criar contratos inteligentes.

Tabela 9 – Lista de *blockchains* e pontuação

Blockchain	a	b	c	d	e	f	g	h	**Total**
Ethereum	2	2	2	2	2	1	1	2	**14**
Bitcoin	2	1	2	0	1	1	1	0	**8**
Solana	2	2	2	1	1	1	1	2	**12**
Hyperledger Besu	2	2	2	2	2	1	2	2	**15**

Fonte: o autor.

As duas mais bem pontuadas foram a *Ethereum* e a *Hyperledger Besu*. A *Hyperledger Besu* é uma Blockchain da família *Hyperleder* [7] apoiada, por exemplo, pela *Linux Foundation*, mas que se comporta como um *ethereum client* e acaba unindo as principais características da *Hyperledger* com a *Ethereum*. Na literatura encontramos muitos projetos usando o *Hyperledger Fabric* [8], mas nós não o colocamos em nossa lista, pois o *Hyperledger Besu* além de possuir

[6] https://www.obsat.org.br/
[7] https://www.hyperledger.org/
[8] https://www.hyperledger.org/use/fabric

semelhanças com o *Ethereum*, poderia ser mais interessante portar projetos de um para o outro, pelo fato que ambos usam a linguagem *solidity* nos contratos inteligentes, o que representa menor curva de aprendizado. Assim, escolhemos a *Hyperledger Besu* para a implementação da *blockchain*. Para aprofundar mais sobre técnicas para escolher qual *blockchain* usar recomenda-se seguir um fluxo de escolhas apartir dos requisitos do projeto, um exemplo foi apresentado no Anexo A baseado em (SONMEZ *et al.*, 2021).

3.1.3 Componentes e Modelo UML

Com o objetivo de visualizar melhor os subsistemas que se relacionam dentro do serviço da *blockchain*, apresentaremos em diagrama UML (*Unified Modeling Language*, do inglês). A UML é uma linguagem para elaboração de projetos de software. Veja no, apêndice A, o diagrama representado com as classes. Para simplificar e resumir a apresentação, a Tabela 10 nos fornece os componentes, chamados de item e uma breve descrição. Cinco classes, ou componentes, se relacionam diretamente com a classe principal e são componentes para o projeto da *blockchain*. São elas: *Tx, rlp, firmware cubesat, config e keccak*.

Tabela 10 – Componentes do diagrama UML

Item	Descrição
Tx	-Representa as transações que ocorrem na blockchain;
Rlp	-Do inglês *Recursive Length Prefix* é usada em aplicações *Ethereum*. Ele padroniza a transferência de dados entre os nós. É usado para serealizar objetos na camanda de execução do *Ethereum*;
Firmware cubesat	-É o artefato de software que é executado de forma embarcada dentro do dispositivo;
Config	-Componente que configura as *Blockchain*, por exemplo algoritimo de consenso a ser usado, quantidade de nós e etc;
Keccak	- É a criptografia usada. Para relembrar a explicação, veja o capítulo 2.

Fonte: o autor.

3.1.4 Armazenamento de dados na blockchain

No *Hyperledger Besu* um banco de dados no formato chave-valor chamado *RockDb* é usado para persistir os dados a nível de nó local. Os dados da *blockchain* incluem cabeçalhos de bloco que constituem a cadeia de blocos e isso ajuda na verificação criptográfica do estado da *blockchain*. Na maioria das *blockchains* da família da *Hyperledger* existe um conceito chamado de *World State*. Os dados armazenados na *blockchain* podem ser alterados, mas o

world state armazena qual é a versão daquele dado armazenado. A *Besu* oferece dois formatos de armazenamento [9] o *Forest of Tries* e o *Bonsai Tries*. Considerando que a *blockchain* terá um "crescimento vegetativo", isto é, novos blocos constantemente surgirão com o passar do tempo, nos detivemos em analisar a quantidade de memória exigida para sustentar o crescimento da lista de registros.

Seja λ a taxa de crescimento diária do blockchain em *megabytes*, S o tamanho de um bloco em *kilobytes*, T é o número de segundos durante a observação, no caso de 24 horas será 86.400 e t o tempo entre a geração de dois blocos em segundos. O termo t pode ser ajustado pelo desenvolvedor. Observe que λ será expresso em megabytes, logo dividimos por 1024. Da equação 3.1 Temos:

$$\lambda = \frac{S \times T}{1024 \times t} \tag{3.1}$$

Assumindo que um bloco poderá ter o tamanho de 1Kb [10] e o tempo médio entre dois blocos é de 12 segundos [11], podemos concluir que a taxa de crescimento do consumo de memória de um nó blockchain pode ser expresso pela equação 3.2.

$$\lambda = \frac{1\ kilobyte \times 60\ segudos \times 60\ minutos \times 24\ horas}{1024 \times 12 segundos} = 7,03\ Mb/dia \tag{3.2}$$

Ao analisarmos os cálculos no passo da equação 3.2, concluímos que tornar um nanosatélite um nó da rede na arquitetura tradicional *blockchain* é desafiante, pois o tamanho do recurso de memória necessária por dia para armazenar a cópia da cadeia de blocos pode ser uma barreira, visto que muitos dos cubesats possuem sérias restrições de memória. Uma outra estratégia seria tornar o nanosatélite um cliente da *blockchain* (fazendo requisições via API) e hospedar a *blockchain* nos servidores de uma estação terrestre. Seguimos este caminho em nossos experimentos.

[9] https://besu.hyperledger.org/en/stable/public-networks/concepts/data-storage-formats/?h=storage+data

[10] A documentação do *Hyperledger* nos diz que podemos escolher o tamanho do bloco, formado por n transações que desejarmos. Em nosso caso, cada transação terá como metadado um fragmento de firmware de 32bytes.

[11] O tempo médio para formação de um bloco pode ser alterado nas configurações quando se realizar o setup da Blockchain.

3.1.5 Backup e Restauração do sistema

Este é um último ponto que iremos analisar acerca da *blockchain* da *Besu*. Assim como qualquer sistema, faz-se necessário, por segurança, um sistema de *backup* e normatização para restauração de sistema.

Em um sistema *blockchain* descentralizado, os dados são replicados entre todos os nós da rede, de forma igualitária. Mas o backup da configuração e dos dados garante que o processo de uma possível restaruação do sistema seja menos traumático.

Existe suporte para *backup* e restauração de sistema[12] na *blockchain* escolhida. A documentação sugere que seja montado um volume separado para armazenar os dados. Para isso pode-se usar o comando *–data-path* para oferecer o caminho deste volume de backup no servidor usado.

Um último ponto muito útil para engenheiros e desenvolvedores de sistema é mensagens de eventos e *logs* e também esses tópicos são cobertos pela tecnologia escolhida [13].

3.2 Blockchain em atualização de firmware

Após refletirmos sobre conceitos teóricos de *blockchain*, atualização de firmware em nanosatélites e requisitos técnicos para implementar uma rede descentralizada, finalizaremos este capítulo apresentando a arquitetura planejada. Desta feita, abordaremos dois aspectos:

a) Uso de contratos inteligentes para automação de rotinas em processos de atualização de firmware e comunicação entre controle da missão e nanosatélite;
b) Como criar um ambiente descentralizado de estações terrestres usando *blockchain*

Os apontamentos desta seção nos levarão à próxima seção que implementa, através de experimentos, a prova de conceito em simulação, a implementação a nível de *hardware* e *software*, bem como a conclusão do trabalho realizado diante dos resultados obtidos.

3.2.1 Contrato Inteligente na comunicação entre nanosatélite e estação terrestre

Conforme já discorrido na seção anterior, o contrato inteligente é um algoritmo que implementa funções para a execução de determinadas tarefas. Umas das grandes diferenças entre este modelo e o tradicional é que este algoritmo é executado em uma estrutura descentralizada de

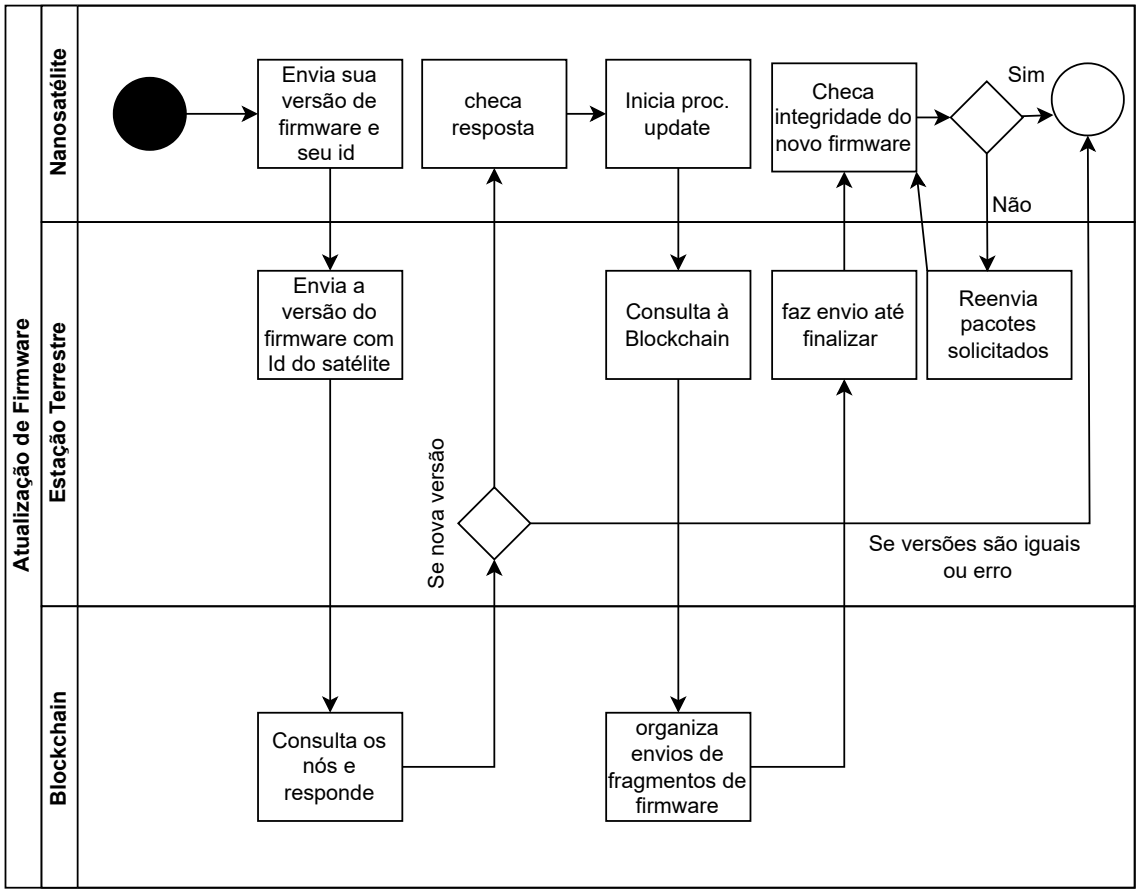
[12] https://besu.hyperledger.org/en/stable/private-networks/how-to/backup/
[13] https://besu.hyperledger.org/en/stable/public-networks/concepts/events-and-logs/

servidores e possui algumas características[14] que o torna mais aderentes em auditorias públicas e persistência do serviço, mesmo que alguns nós computacionais deixem de funcionar.

Para nosso projeto, o contrato inteligente precisará ter as funções de: registrar um fragmento de *firmware*, atribuir uma identidade ou numeração única para cada registro, possuir funções de consultas por *id* da transação, por versão ou por fabricante.

A comunicação entre o satélite e a estação terrestre é apresentada na Figura 12. Observe que separamos em 3 níveis para melhor compreensão do leitor.

Figura 12 – Fluxo de atualização de *firmware* usando *blockchain*



Fonte: Próprio autor

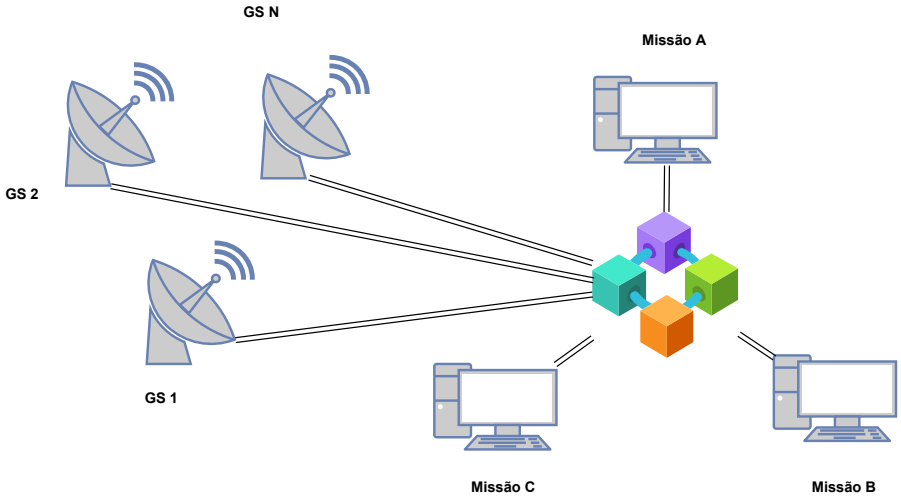
Desta feita, ainda que o nanosatélite perca a visada da estação terrestre, ele poderá recomeçar o processo de atualização de *firmware* de onde ele parou quando possuir a oportunidade de obter uma nova janela de comunicação com outra estação terrestre do consórcio, o que poderá resultar em maior eficiência e rapidez neste processo de comunicação.

[14] Uma das características é a transparência e a imutabilidade das transações.

3.2.2 Descentralização das estações terrestres

Para exemplificar resumidamente a arquitetura objetivada para a implementação da rede descentralizada de estações terrestres usaremos algumas figuras. A Figura 13 representa o consórcio de estações terrestres e o serviço realizado às missões de controle. A imagem colorida ao centro representa a *blockchain* e a Figura 14 podemos notar a comunicação de um determinado controle de missão com a *blockchain*[15].

Figura 13 – Consório de estações terrestres



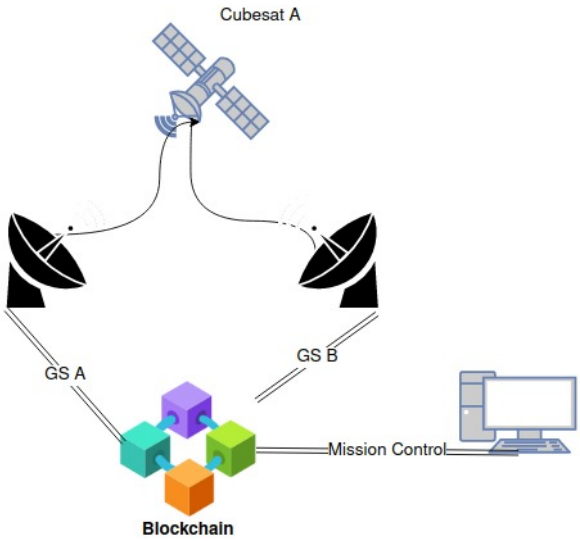
Fonte: Próprio autor

A Figura 15 expressa a *blockchain* orquestrando o serviço de comunicação com as estações terrestres GS A e GS B. Note especialmente a visão da passagem do mesmo cubesat chamado de *Cubesat A* por estações distintas enquanto continua seu processo de atualização de *firmware*.

Frise-se que o principal conceito é interpretar que a *blockchain* fornece uma camada de infraestrutura para que cada estação terrestre possua exatamente a mesma cópia de registros que as demais, mas de forma criptografada e que só é possível ser acessada pelo par controle da missão - satélite.

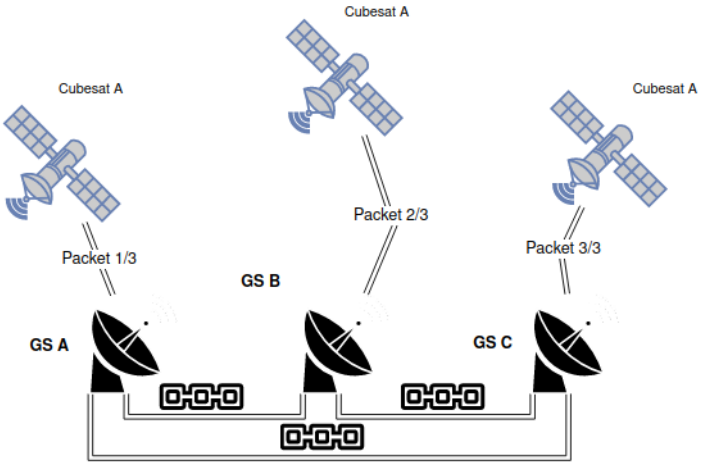
[15] Note que missões de controle de satélites diferentes acessam uma mesma rede *blockchain* para registrar firmware e requisitar informações.

Figura 14 – Missão de controle acessando o satélite *A* por meio do consórcio de estações terrestres



Fonte: Próprio autor.

Figura 15 – Descentralização das estações terrestres



Fonte: Próprio autor.

4 EXPERIMENTOS E RESULTADOS

Nesta seção apresentamos como o protótipo foi construído. O protótipo que, por vezes, também chamaremos de experimento, implementará na prática os resultados almejados e estipulados em nossa hipótese e nos dará a capacidade de confrontar os resultados teóricos, a simulação e os realizados em bancada. As subseções a seguir apresentam os passos para construir cada um dos módulos (ou subsistemas) do experimento e depois a conexão de cada, ao funcionarem conjuntamente. A Tabela 11 resume ferramentas usadas para o experimento e algumas delas serão explicadas mais a seguir.

Tabela 11 – Ferramentas usadas

Item	Descrição
Blockchain	-Hyperledger BESU 20.10.0
Linguanges usadas	-Solidity (Smart contract) and Python (API), SQL e C++
Algoritmo de Consenso	- IBFT (*Istambul Byzantine Fault Tolerance* [1])
Frameworks	-Remix-ethereum (online), Metamask (wallet), VS Code (IDE), Jupyter Notebook, Matlab e Docker Composer
Dispositivos	-PION Cubesat, ESP8266, Notebook I3 6ª geração, 12 GB RAM, 2.1Ghz (Blockchain com 3 nós e 1 rpc)

Fonte: o autor.

Para facilitar o acompanhamento desta seção teremos duas subseções, são elas: Implementações e Resultados. A subseção chamada de implementação traz a compilação do processo de desenvolvimento dos módulos e a realização dos experimentos. A seção chamada de resultados nos apresenta as consequências e as análises dos experimentos.

4.1 Implementações

A referida seção possui três frentes:

- Simulação de missão espacial com Matlab;

- Implementação em *software*: criação da blockchain e o servidor em *python* para suportar as APIs, bem como a realização do experimento e testes a nível de *software*;

- Implementação em *hardware*: Implementação dos *firmwares* no ESP8266 e no PION Cubesat.

O Apêndice E apresenta recortes de códigos usados para realizar as simulações

e experimentos, bem como as respostas em terminal dos comandos de testes. Nas próximas subseções serão feitas constantes referências a este apêndice, o que será muito útil, especialmente para àqueles que desejam reproduzir os experimentos e entender cada passo dado. Para acessar alguns dos códigos mais importantes usados neste processo acesse o *github*[2] do projeto.

4.1.1 Simulação de missão espacial com Matlab

Usando os recursos do Matlab[3] criamos uma missão espacial composta de um nanosatélite que chamamos de *SACODE BR-1* e duas estações terrestres, sendo uma chamada *UFC-PICI* com longitude e latitude do campus universitário da Universidade Federadal do Ceará no campus do PICI [4]. O objetivo foi estimar o tempo médio de comunicação entre o nanosatélite e a estação terrestre nesse ponto de localização e assim usarmos como base para os outros experimentos. Escolhemos um tempo de missão de 90 dias para colhermos uma lista de janelas de comunicação entre o nanosatélite e a estação terrestre. o Código-fonte 1 *Simulação Matlab missão do nanosatélite* (veja no Apêndice E) apresenta um exemplo de como criar uma missão espacial usando recursos do matlab. A órbita escolhida foi a órbita LEO, assim como seus parâmetros (O'KEEFE *et al.*, 2009). O código que desenvolvemos para a simulação das leituras da API e comunicação é mais complexo e extenso. Poderá ser encontrado no *github* do projeto.

O tempo de missão de 90 dias nos trouxe 396 janelas de comunicação entre satélite e estação terrestre. As informações são sintetizadas na Tabela 12. Usaremos como base para o tempo de comunicação entre a estação terrestre e o nanosatélite 657,38 segundos ou 11.35 minutos, um resultado dentro do tempo previsto (CARVALHO, 2019).

Após a implementação da estação terrestre no Brasil, acrescentamos uma segunda estação terrestre e a inserimos em uma universidade parceira de nossa instituição o INSA na cidade francesa de Rouen[5]. As Figuras 18 e 19 mostram estratégicamente um dos exemplos onde as estações terrestres localizadas nessas duas localidades deixam a menor janela de tempo de comunicação entre a missão e o nanosatélite, aproximadamente 4 minutos.

Utilizando ainda a capacidade das ferramentas do matlab, analisamos a relação da perda de propagação do sinal e o ângulo de elevação na hora da comunicação [6] e inserimos

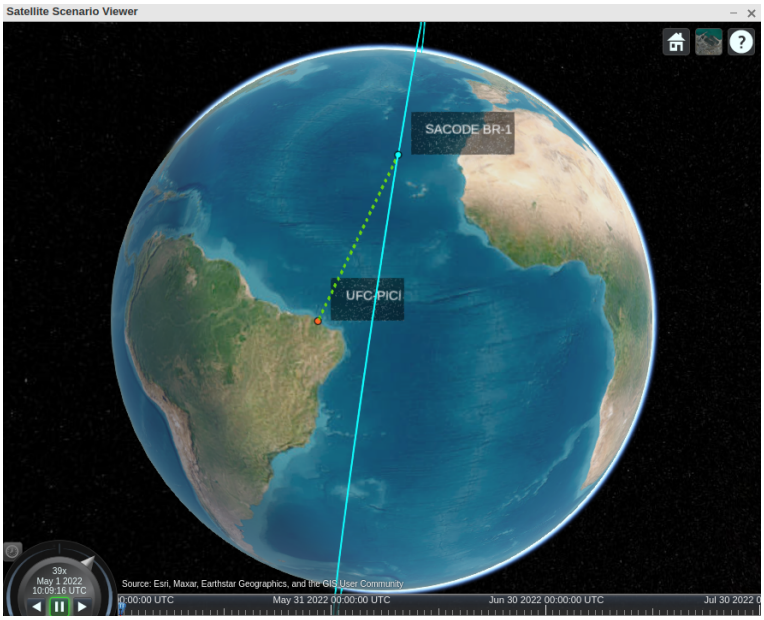
[2] https://github.com/EdilsonFilho/gs_bc_simulations.git
[3] https://matlab.mathworks.com/
[4] Av. Humberto Monte, s/n - Pici, Fortaleza - CE, 60440-593
[5] https://www.insa-rouen.fr/
[6] Usamos dados da União Internacional de Telecomunicações, Recomendação ITU-R P.618 (12/2017) para

Tabela 12 – Simulação de missão cubesat 90 dias

Item	Valores
Tempo de missão	90 dias
Início da missão	01/05/2022 às 5:36:00 UTC
Fim da missão	30/07/2022 às 05:36:00 UTC
Duração Miníma de comunicação	60 s
Duração Máxima de comunicação	840 s
Média de tempo de comunicação UFC-PICI	650,30 s
Média de tempo de comunicação INSA-Rouen	633,49 s
Nº de comunicações UFC-PICI	396 (total) e 4.4 (por dia)
Nº de comunicações INSA-Rouen	679 (total) média de 7.5 (por dia)

Fonte: o autor.

Figura 16 – Visão 1 do primeiro contato da missão



Fonte: Próprio autor

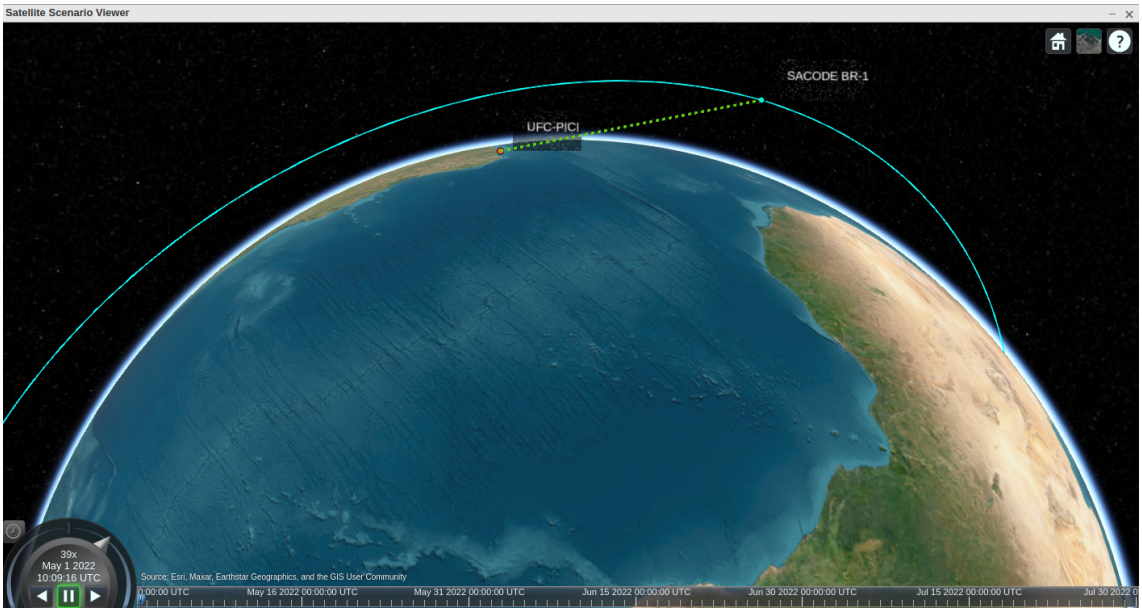
efeitos atmosféricos. Os efeitos ionosféricos nos links Terra-espaço tornam-se significativos em frequências abaixo de 1 GHz. A perda de propagação é a redução da intensidade do sinal que ocorre durante a transmissão do sinal através do ar. O ângulo de elevação é o ângulo formado entre a linha de visada do transmissor para o receptor e a horizontal. Assim, quanto maior o ângulo de elevação, menor será a distância horizontal entre o transmissor e o receptor. A Figura 20 nos mostra a diferença de perda de propagação para dois sinais enviados da Terra com 4 GHz. Uma menor perda é observada na estação do INSA.

Observe um maior efeito de cintilação [7] no gráfico que representa a estação da UFC.

criarmos o ambiente da simulação.

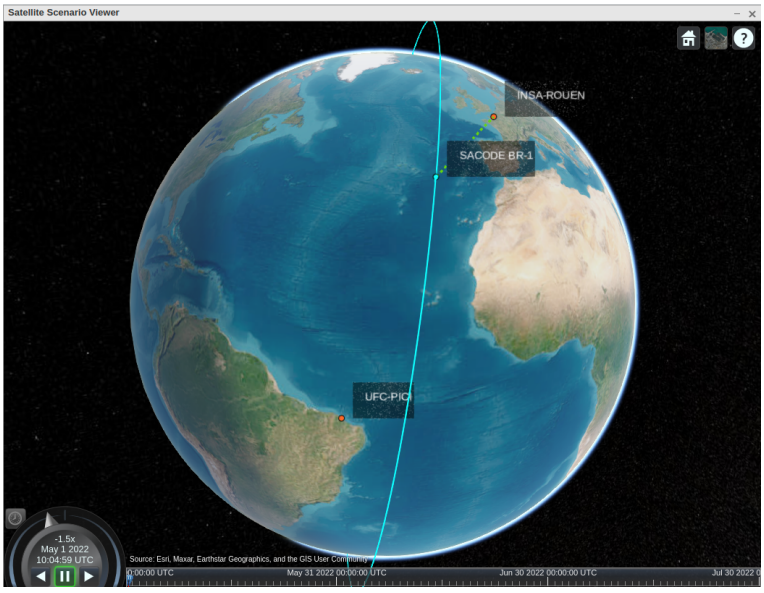
[7] A cintilação é causada por flutuações na ionosfera, uma camada da atmosfera da Terra que é ionizada pela radiação solar. Ela em geral é mais severa nas proximidades da linha do Equador, o que visivelmente foi manifesto na diferença entre os gráficos. Isso ocorre porque a ionosfera é mais densa e mais variável perto do equador, devido à maior radiação solar e à presença da anomalia equatorial. A anomalia equatorial é uma região de ionização aumentada que ocorre perto do equador magnético e é causada pela interação entre o campo

Figura 17 – Visão 2 do primeiro contato da missão



Fonte: Próprio autor

Figura 18 – Conexão entre nanosatélite e estação terrestre no INSA



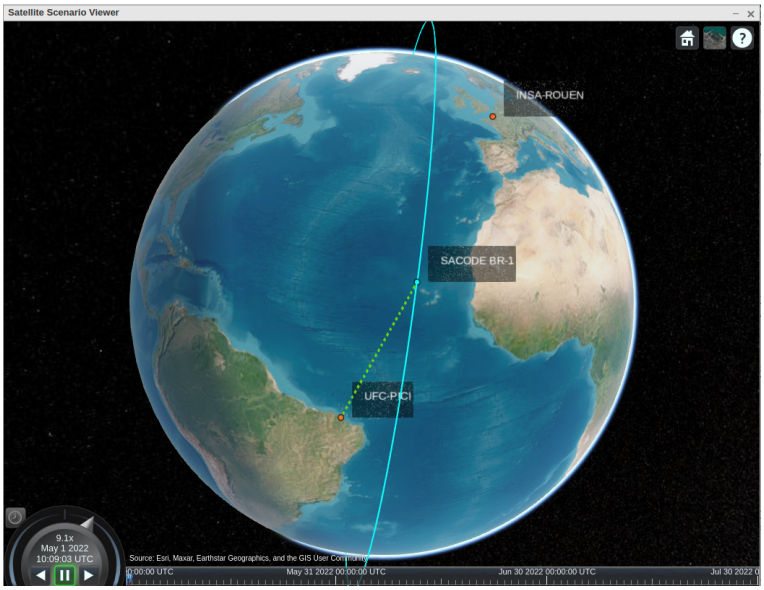
Fonte: Próprio autor

Isso era esperado pela sua proximidade com a linha do equador e uma das consequências é uma maior probabilidade de erros ou ruídos no sinal durante a comunicação nessa faixa de ângulos.

A perda de propagação é uma medida da quantidade de sinal que é perdida ao se propagar através de um meio, como o ar ou o espaço. Quanto maior a perda de propagação, menor será a intensidade do sinal recebido pelo receptor, o que pode levar a um aumento na taxa de erros de bits (BER, na sigla em inglês). Em outras palavras, a perda de propagação pode afetar negativamente a qualidade da comunicação e aumentar a BER. Por exemplo, se

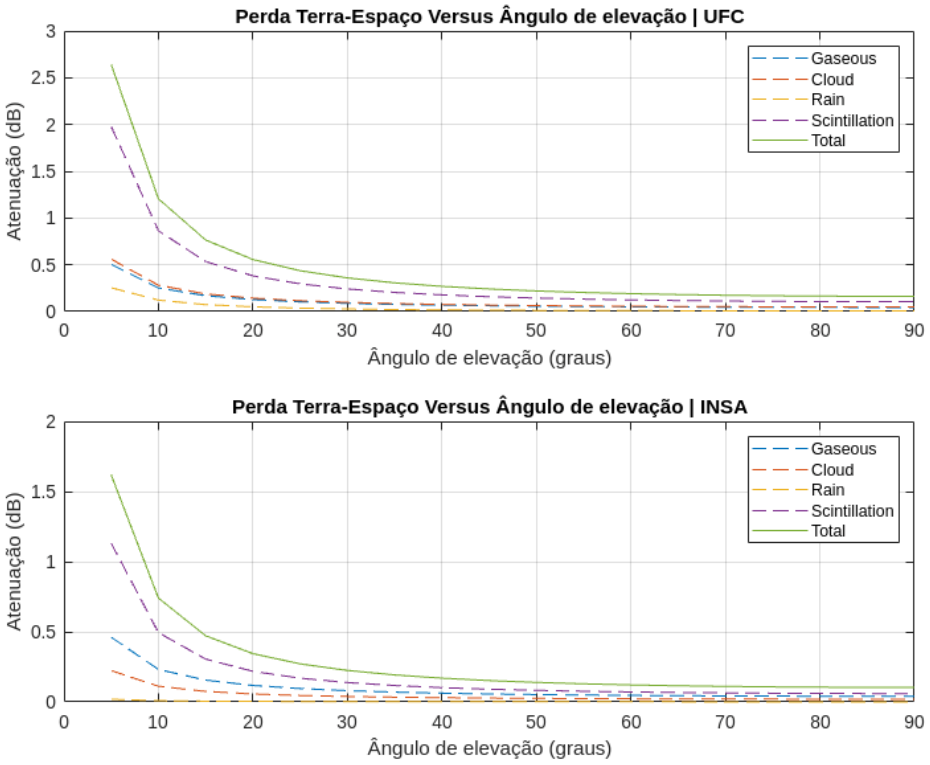
magnético da Terra e o vento solar.

Figura 19 – Conexão entre nanosatélite e estação terrestre na UFC



Fonte: Próprio autor

Figura 20 – Perda de Propagação por efeitos atmosféricos.



Fonte: Próprio autor

houver uma perda de propagação significativa devido a chuva ou cintilação, isso pode levar a um aumento na BER, pois o sinal recebido pelo receptor pode ficar muito fraco para ser decodificado corretamente. Por outro lado, se a perda de propagação for baixa, a BER pode ser baixa também, pois o sinal recebido pelo receptor será mais forte e pode ser decodificado com mais facilidade.

Essa simulação nos trouxe especialmente o número de janelas e a duração de comu-

nicação em cada janela de comunicação, ou seja, a duração do link ativo entre estação terrestres e satélite. Usaremos esses resultados mais à frente.

4.1.2 Implementação em Software

Infraestrutura blockchain

Depois de termos discutido sobre a tecnologia blockchain nas seções anteriores e de termos chegado a uma pontuação que nos permitiu decidir qual tipo de blockchain iremos adotar, passamos para a implementação em software da blockchain.

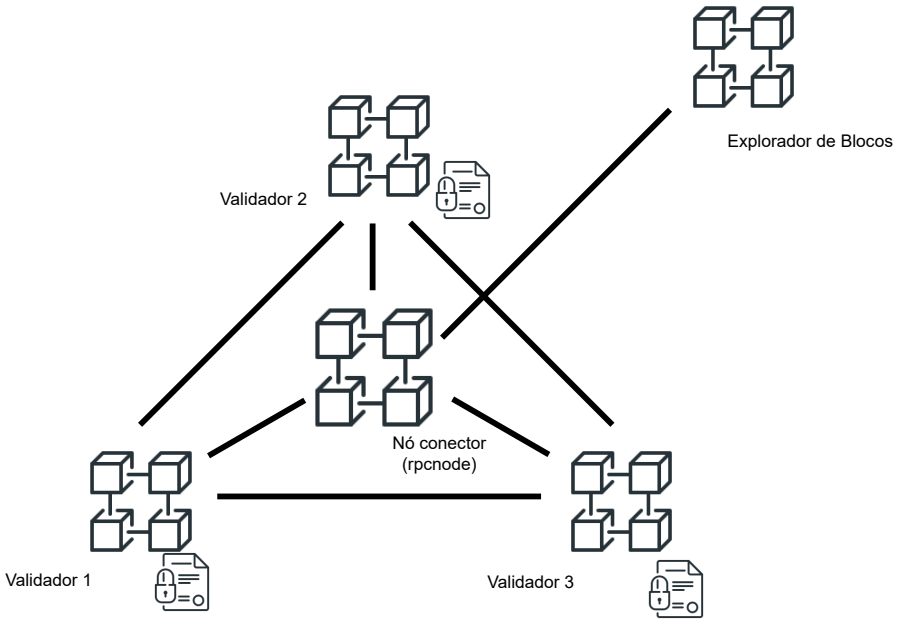
Uma vez escolhida a tecnologia *Hyperledger Besu* e seguindo sua documentação [8] o próximo passo consistiu em realizar:

a) Setup da Blockchain na máquina de desenvolvimento;
b) Criar um container *Docker*[9] para encapsular todo o software desenvolvido;
c) Instanciar as configurações básicas;
d) Criar o Contrato Inteligente ou *Smart Contract*;
e) Conectar a blockchain localmente ao framework on-line chamado *Remix-Ethereum* para compilar e subir na rede local ou *localhost* a blockchain;
f) Instanciar o *Github* para usar como repositório oficial da estrutura criada;
g) Instalar o *Flask* e criar em *Python* a infraestrutura para a construção da API.

Após executar a rede blockchain, alguns testes são necessários para verificar seu correto funcionamento. A seguir, colocaremos uma série três testes onde conferimos alguns métodos que nos garantem que a blockchain está instanciada de forma correta e em pleno funcionamento. A Figura 21 expressa a arquitetura de como os nós blockchain foram implementados. Os nós, apresentados como "validadores"fazem o papel de registrar e validar os blocos gerados. O nó central chamado de "conector"conecta todos os nós com o bloco chamado "explorador". O bloco explorador nos permite visualizar no navegador da internet, seja como *localhost* ou em produção, a criação dos blocos e também os *logs*[10]. Permite, ainda, pesquisar de forma intuitiva, como um painel de controle, as transações, os custos e outras informações. Uma vez com a arquitetura criada, nós operantes e configurações realizadas, um passo importante é testar o comportamento da rede.

[8] https://besu.hyperledger.org/en/stable/private-networks/
[9] https://www.docker.com/resources/what-container/
[10] https://besu.hyperledger.org/en/stable/public-networks/concepts/events-and-logs/

Figura 21 – Arquitetura *Blockchain* implementada



Fonte: Próprio autor

Testes na blockchain

Como a blockchain é uma infraestrutura e que está hospedada no servidor, é importante realizarmos alguns testes para obter resultados sobre o *status* da rede assim como alguns outros parâmetros. Para simplificar a explicação desse processo, todas as respostas aos testes serão chamadas de *Resposta X*, onde X é o número da resposta, as imagens e comentários mais aprofundados estão no Apêndice E. A Tabela 13 resume os testes e seus resultados esperados.

Tabela 13 – Testes na *blockchain*

Nome do teste	Resultado	Status	Resposta (ver Apêndice E)
ethChainID	Id correto	sucesso	Resposta 1
netPeerCount	3 nós	sucesso	Resposta 2
liveness	Up	sucesso	Resposta 3
readiness	Up	sucesso	Resposta 4

Fonte: o autor.

O primeiro teste que realizamos é usando o método *ethChainId*, disponivel na documentação da *Hyperledger Besu*. Esse teste é importante, pois o Id (identificador) da cadeia retornado sempre deve corresponder às informações no bloco de cabeçalho conhecido atual. O que podemos concluir resumidamente é que o teste do *ethChainId* foi um sucesso e retornou exatamente o *Id* da cadeia, o que nos traz a segurança que essa etapa está funcionando corretamente.

O segundo teste realizado foi o contador de nós da rede, o método é *netPeerCount*, e

o resultado são os números de nós. No arquivo chamado de*config.json*, na raiz de nosso projeto, nós configuramos diversas características da rede. No caso escolhemos 3 nós para a blockchain e o teste confirmou o número de nós em execução em nosso servidor. No experimento, todos esses nós estavam executando na mesma máquina por motivos práticos, mas é possível aumentar o número de nós para 5, 7, ou qualquer outro número desejado. Esses nós podem estar em outros servidores, distantes geograficamente por exemplo.

Um terceiro teste que foi realizado chama-se de *status* de atividade do servidor JSON - RPC. A Resposta 3 nos mostra o resultado desejado e obtido em todas as execuções do experimento. O JSON-RPC é um protocolo RPC em JSON e pode ser usado em vários ambientes onde transitam mensagens, por exemplo o protocolo de transferência de hipertexto o HTTP ou soquetes. Uma solicitação tem os seguintes pontos:

1- *jsonrpc* : Uma *string* especificando qual a versão do JSON-RPC está em uso. Em nosso experimento esta deverá ser sempre "2.0". Vejas nas Respostas 1 e 2.

2- *método*: Uma *string* é um nome remoto que deve ser usado para chamar a operação.

3- *parâmentros*: Se tiver, pois pode ser passado em branco, é uma estrutura JSON contendo os parâmetros para chamar a função, pode ser um objeto contendo chave valor.

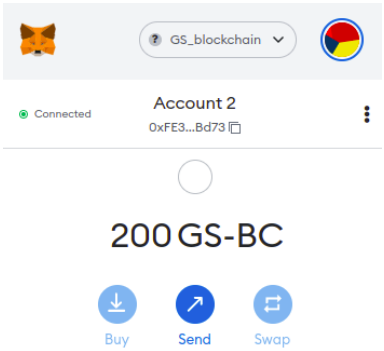
3- *resultado*: Existe em caso de chamada bem sucedida. A ausência indica a ocorrência de erro.

Por fim, o teste de *readiness* ou seja o *verificação de prontidão ou vivacidade*, por padrão, a verificação de prontidão exige que um par conectado e o nó estejam dentro de dois blocos do bloco mais conhecido. Expresso pela Resposta 4, o resultado esperado é o *status:* UP[11], o que foi verificado em todas as versões do teste.

Após todos os testes terem sido realizados com sucesso, os próximos passos para iniciar o projeto foram: configuração da rede, criar *tokens* que serão consumidos nas transações de escrita no banco de dados, no caso, na blockchain, importar esses *tokens* para uma carteira virtual. A Figura 22 apresenta o saldo de token GS-BC que serão usados nas transações de registro de fragmentos de firmware. Podemos ainda observar na mesma figura: o nome do token, o status de conexão e parte do endereço da carteira de acesso, isto é, o hexadecimal "0XFE3...Bd73", pois praticamente na blockchain o relacionamento entre os nós, os usuários e as transações são em hexadecimal.

[11] Isto é, o serviço está em operação.

Figura 22 – Tokens GS-BC



Fonte: Próprio autor

Implementando o contrato inteligente

O contrato inteligente foi desenvolvido na linguagem *solidity* [12], uma linguagem desenvolvido pela *Ethereum*. Utilizamos também a plataforma *remix-ethereum* [13], um framework para compilar o contrato inteligente, gerar o *bytecode* e o *abi* para usarmos no *webservice*. Algo muito prático para o desenvolvimento inicial de nosso experimento foi o fato de podermos conectar o *remix-ethereum* que está está hospedado na internet com a *blockchain* sendo executada em *localhost*. A orquestração que conectou estes dois ambientes foi o *metamask* [14] e o *VScode*. O código completo poderá ser acessado no repositório [15] que criamos para hospedar o código do projeto[16]. Afim de documentar o processo de implementação de um contrato inteligente e facilitar o compartilhamento de conhecimento, criamos a Tabela 14 que resume algumas seções do algoritmo do contrato inteligente que pode ser encontrado integralmente no Apêndice D. Esse conteúdo é extremamente útil para aqueles que desejam repetir o experimento e estão aprendendo a trabalhar com blockchain.

Tabela 14 – Explicação dos fragmentos de código

Item	Código-fonte (Apêndice E)
Estrutura do objeto chamado "fragment"	Código-fonte 2
Registrar um fragmento de firmwre	Código-fonte 3
Outros métodos (contar número de fragmentos e recuperar por id)	Código-fonte 4

Fonte: o autor.

[12] https://ethereum.org/pt-br/developers/docs/programming-languages/dart/getting-started-with-smart-contracts-and-solidity
[13] https://remix.ethereum.org/
[14] https://metamask.io/
[15] https://github.com/EdilsonFilho
[16] No decorrer do projeto, para maior praticidade, controle e automação, deixamos de usar o metamask e o remix e passamos a usar a biblioteca web3.py e o jupiternotebook para realizar os processo de compilação e *deploy* do contrato. Profissionalmente é o mais indicado.

o Código-fonte 2 é um fragmento de código do contrato inteligente e nos apresenta duas informações importantes: A versão da linguagem e a *struct*, como é na linguagem *C*, com os 4 elementos importantes. *version* que guarda a versão do firmware, *index* o index do fragmento armazenado, *firmware* o fragmento em si do firmware, *creator* a referência do criador do registro, neste caso seria um código identificador do controle da missão.

O Código-fonte 3 apresenta a função ou método escrito em *solidity* para armazenar o fragmento de firmware. Chamamos a atenção para a linha 5, onde fazemos uma checagem para saber se aquele index do firmware já está registrado na base de dados, no caso, se estiver, o sistema impede de duplicidade de informação, evitando consumo desnecessário de memória e ainda redundância de informação. Outro ponto que sinalizamos é o uso da *keccak256* para codificar o pacote de firmware e criar um id para aquele fragmento, isto é um ponto de reforço na segurança do item armazenado.

O Código-fonte 4 finaliza nossa apresentação dos métodos mais importantes implementados no contrato inteligente. Possui duas funções a) *getFragmentById* que retorna os elementos de *struct* e b)função *countFragments* que conta o número de fragmentos de firmware armazenados. Ele é usado no firmware do ESP8266 e no PION Cubesat para saber quantos pacotes serão enviados.

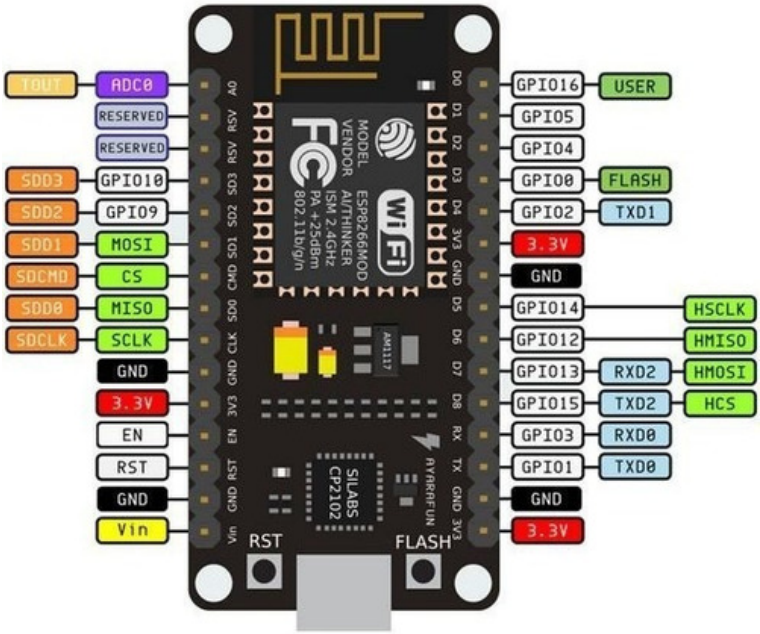
4.1.3 Implementação em Hardware

Reservamos dois momentos para a implementação em hardware. O primeiro é a utilização do ESP8266 (Figura 23) para criar um módulo rápido de prova de conceito e o segundo é a utilização do cubesat educacional da *PION*, por ser um cubesat que já realizou, tanto vôos em balão metereológio e uma versão adaptada, o *PION-BR1* foi lançado pela empresa americana *SpaceX* [17] abordo do foguete *Falcon 9* [18] para vôo subórbital [19].

A programação deste dispositivo segue o padrão de programação do firmware das placas educacionais para sistemas embarcados. Usando a interface ou programa para codificação cria-se um projeto, compila-se, e depois, faz-se o *setup* do código na máquina. O ESP8266 além de poder realizar a atualização de firmware usando a ligação serial, isto é, o cabo mini-usb, pode-se ainda usar a tecnlogia OTA (*Over-The-Air, do inglês*) ou ainda uma página *HTML* dentro do dispositivo. A linguagem de programação usada é C/C++ e uma vez o código criado, pode-se

[17] https://www.spacex.com/
[18] https://www.spacex.com/vehicles/falcon-9/
[19] https://www.tecmundo.com.br/ciencia/231954-spacex-manda-satelite-brasileiro-pion-br1-espaco.htm

Figura 23 – ESP8266 no NodeMCU v3. Lolin



Fonte: https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet. Acesso 23 de abril de 2022

faer o *download* do o arquivo.*bin* (o código binário).

Outra vantagem de usar um módulo de computador de bordo com uma comunicação *wifi* é a praticidade de focarmos nos experimentos e na codificação.

Após a prova de conceito que consistiu em, fazer requisições à blockchain usando uma API através de um sistema embarcado, o próximo passo foi a implementação no PION Cubesat. A Tabela 15 nos fornece mais informações técnicas sobre o Kit.

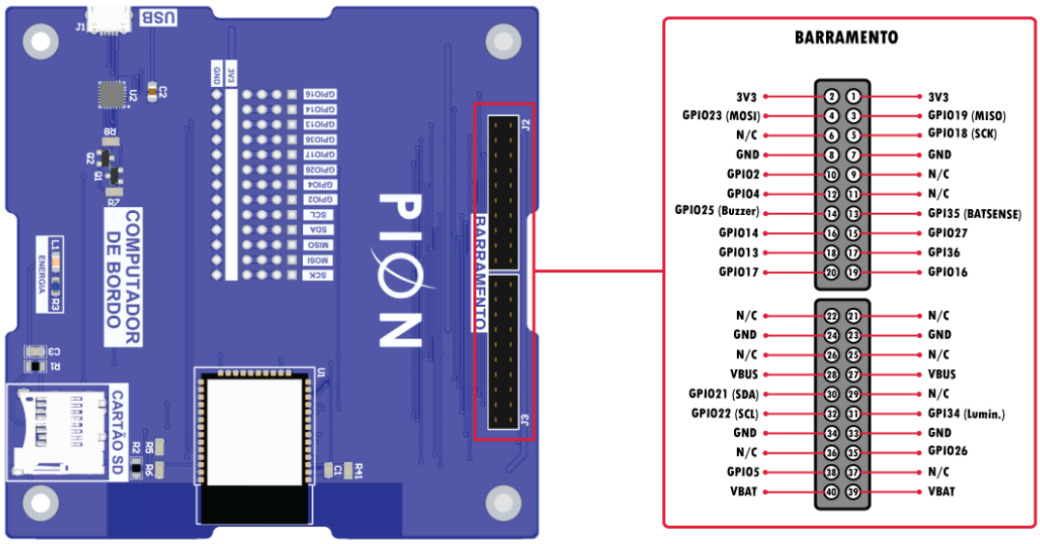
Tabela 15 – Especificações do PION Cubesat

Características e Senores	Dados e Faixa de Operação
Volume	1U - 10x10x10 cm
Massa	350g
Microcontrolador	ESP 32 (32bit /Dual Core)
Wifi	802.11 b/g/n - 2.4 2.5 GHz
Temperatura	-40 a 125ºC
Umidade	0 - 100 (em percentual)
Pressão	300 - 1100 hPa
Luminosidade	0 - 100 (em percentual)
Gás Carbônico	400 - 29206 ppm
Giroscópio	±250, ± 500, ± 1000 e ± 2000ºS
Acelerômetro	± 2g, ± 4g, ± 8g e ± 16g
Magnetômetro	± 4800µT
Nível de Bateria	0 - 100 (em percentual)

Fonte: o autor.

A Figura 24 apresenta uma visão do computador de bordo do PION Cubesat. Podemos obsevar a simplicidade do computador de bordo, que é destinado para a finalidade educacional. Ele usa um ESP32 como microcontrolador, tornando-o possível realizar vários experimentos, conectar com sensores externos e realizar tarefas mais complexas, embora não seja indicado para aplicações críticas. Um destaque é dado para a pinagem da placa.

Figura 24 – Cubesat PION



Fonte: https://www.pionlabs.com.br

A Tabela 16 apresenta alguns detalhes adicionais do computador de bordo.

Tabela 16 – Especificações Computador de Bordo

Especificações	Descrição
Microntrolador	ESP32 (32-Bit / Dual Core/ 240 MHz)
Tensão de Operação	3.3v
Baterial	Li-Po 3.7v / 400 mah
Memória Flash	4 Mb
Armazenamento Externo	Cartão microSD até 16Gb
Wifi	802.11 b/g/n 2.4 Ghz 2.5 GHz

Fonte: o autor.

4.1.4 Prototipação

Para a prova de conceito, o ESP8266 fez requisições do tipo GET direto da base de dados descentralizadas, isto é, na *blockchain* e escreveu na memória *flash*, chamada de *SPIFFS*, instanciada pela biblioteca *FS.h*[20]. Após o sucesso desta etapa, criamos uma rotina para repetir

[20] https://embarcados.com.br/spiffs-o-sistema-de-arquivos-do-esp8266-32/.

por 500 vezes o teste de requisição e coleta dados, como o tempo para fazer a requisição GET ao ler o dado da Blockchain.

Implementação da prova de conceito no ESP8266

O nosso planejamento para este experimento de implementação em hardware seguiu os passos:

a) Instanciar um *container* Docker com a *blockchain* localmente;
b) Realizar os testes 4.1.0.1;
c) Fazer o *deploy* do contrato inteligente na Blockchain, ou seja, instalar o contrato;
d) Instanciar o servidor *python* contendo o serviço de *webservice* para as APIs;
e) Fazer o *deploy* do firmware do ESP8266;
f) Realizar medições usando o Monitor Serial da IDE de desenvolvimento.

A Figura 25 representa a visão de um fragmento de firmware e traz o *index* do fragmento, o *fragment*, ou seja, o fragmento em si do código de firmware, o *version* que é a versão do firmware, o *creator* que é o código único atribuído ao controle da missão e o *package* que é a referência do pacote em relação total de pacotes da imagem do firmware. Os códigos do firmware podem ser encontrados no Apêndice E.

Figura 25 – Estrutura do fragmento de *firmware*

```
{
  "index": "1",
  "fragment": b'\x90\xe1\x00@\xa8\xde\x00@\xe8+\x00@\x00J\x00@LJ\x00@\xbf \xff?\xc9 \xff?\
      xc3 \xff?',
  "version": "0.0.1",
  "creator": "0xFE3B557E8Fb62b89F4916B721be55cEb828dBd73",
  "package": "1/9566"
}
```

Fonte: Próprio autor

Nota-se que, com base no parâmetro *package*, o firmware poderá conferir se já recebeu todos os pacotes enviados pela estação terrestre. Para compreender melhor, a Figura 12 mostra em níveis como acontece o processo de comunicação entre o nanosatélite e à blockchain, tendo como interface a estação terrestre. Assim no fluxo de acontecimentos, após toda a checagem se os pacotes chegaram com sucesso, o processo de *boot* pode ser inicializado.

Após o sucesso da requisição *GET* e do armazenamento do dado na *SPIFSS*, chegamos ao resultados expressos na Tabela 18. Alcançados os sucessos em todas as metas, passamos para implementação no PION Cubesat.

Implementação no PION Cubesat

Tabela 17 – Metas para prova de conceito

Meta	Status
Setup da Blockchain na máquina de desenvolvimento	Ok
Encapsular todo o software desenvolvido	Ok
Criar o Contrato Inteligente	Ok
Subir a Blockchain	Ok
Subir o *webservice* com API	Ok
Criar firmware do ESP8266	Ok
Ler dados da blockchain	Ok
Salvar na mémoria Flash do ESP8266	Ok
Reiniciar o sistema com novo firmware	Ok
Realizar medições	Ok

Fonte: o autor.

Uma vez provado o conceito na versão mais simples, seguimos para a implementação de *software* conectando-a com o PION Cubesat. Sinalizamos nesta etapa:

a) Mesmo que o PION Cubesat, na versão que temos em bancada, tenha realizado vôo em balão metereológico [21], o cubesat não possui antena, por isso o processo de comunicação usou *wifi*, pois iríamos realizar o teste instalando-o em uma sonda (Figura 26) como parte da competição na 1ª Olímpiada Brasileira de Satélites. Mas por questões de agenda, o lançamento em balão não coincidiu com o prazo para a realizaão de nosso experimento, desta forma o experimento foi realizado em terra;

b) Como usamos *wifi*, a arquitetura do experimento foi adaptada, veja Figura 27. Para um cenário de uso de antenas, algumas das rotinas poderão sofrer alterações, mas sem prejuízo ao objeto proposto neste trabalho;

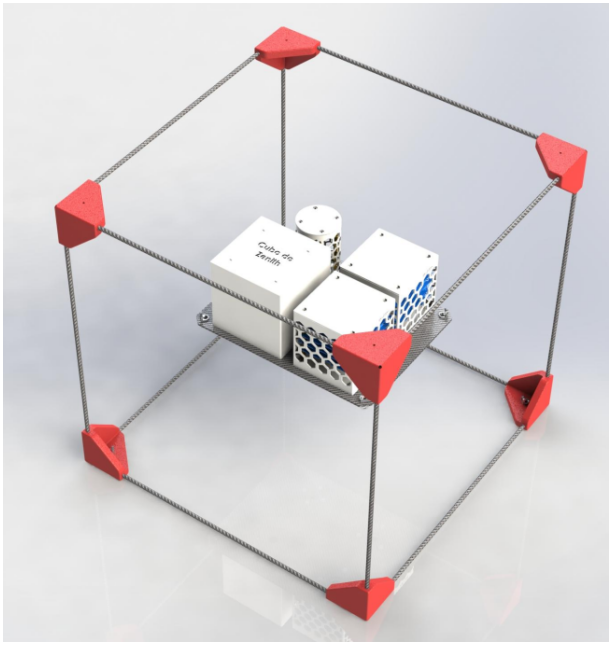
c) Por usar um ESP32 e em sua especificação sinalizar que o alcance máximo para captação de sinal *wifi* na faixa 2.4GHz é de 90m, realizamos o teste variando nossa distância até no máximo 50m do roteador.

Para as implementações no Cubesat da PION seguimos os mesmos processos da seção anterior. Mesmo que a versão do ESP8266 no nodeMCU v3 Lolin possua semelhaças com as especificações do ESP32, algumas alterações a nível de bibloteca precisaram ser realizadas. .

A Figura 26 mostra o esquemático para acoplar os cubestas planejados para a missão do balão meteorológico. A sonda também foi criada pela mesma empresa que criou o PION Cubesat. Segundo as especificações da Olimpíada Brasileira de Satélites (para relembrar o motivo de estarmos citando a olimpíada, releia o capítulo 3), cada sonda levaria 3 cubesats e 1 *camsat*

[21] https://www.gov.br/mcti/pt-br/acompanhe-o-mcti/noticias/2022/08/etapa-da-regional-da-1a-olimpiada-brasileira-de-satelites-mcti-e-realizada-em-tatui-em-sao-paulo

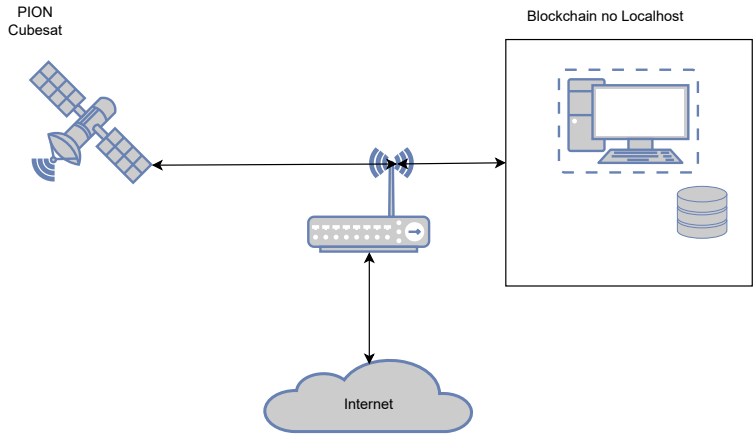
Figura 26 – Sonda para lançamento em balão



Fonte: Edital 2 da 1ª Olímpiada Brasileira de Satélites.

(formato cilíndrico na imagem). A adaptação que fizemos para realizar nosso experimento em terra pode ser encontrado na Figura 27.

Figura 27 – Arquitetura para o experimento com PION Cubesat



Fonte: Próprio autor

Em nossa adaptação, o PION cubesat usa um roteador para acessar ao *wifi* através do protocolo *http* e, assim, fazermos as requisições *GET* direto a um computador rodando o serviço do banco de dados descentralizado, a *blockchain*. Embora a figura indique o roteador ligado à internet, em nossos experimentos não usamos conexão com a internet. Usaríamos a conexão via internet se estivéssemos hospedado a blockchain em um servidor web. Outro ponto importante é que nas próximas figuras substituiremos o roteador por uma estação terrestre, pois é o que se aproximaria da realidade de forma geral.

4.2 Resultados

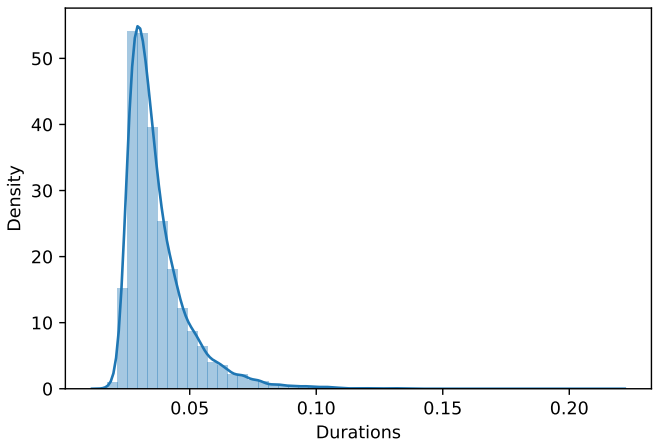
Para melhor refletir sobre os resultados, vamos apresentar em seções e subseções juntamente com seus comentários correspondentes. As seções definidas para a análise dos resultados são: nanosatélite e blockchain.

4.2.1 Nanosatélite

Medições de tempo na simulação Matlab com requisições à blockchain

Após termos criado a infraestrutura blockchain para permitir a descentralização do armazenamento e envio de dados, nosso próximo passo foi realizar alguns testes. O objetivo foi simular a nível de software a solicitação de pacotes dos fragmentos de firmware usando o *Satellite Communications Toolbox* do Matlab[22]. Reproduzimos, então, o cenário de 2 satélites requisitando simultaneamente pacotes de dados à mesma estação terrestre. Usamos como estratégia para visualizar os dados a técnica de estimativa de densidade kernel ou KDE[23]. Observe a Figura 28, o *eixo y* (ordenada) chamado de *Density* e o *eixo x* (abscissa) chamado de *Durations*. A abscissa é o compilado dos registros de duração de tempo em segundos que cada uma das requisições que o satélite simulado em Matlab fez ao servidor em *python* via API.

Figura 28 – Estimativa KDE para 1 satélite requisitando dados



Fonte: Próprio autor

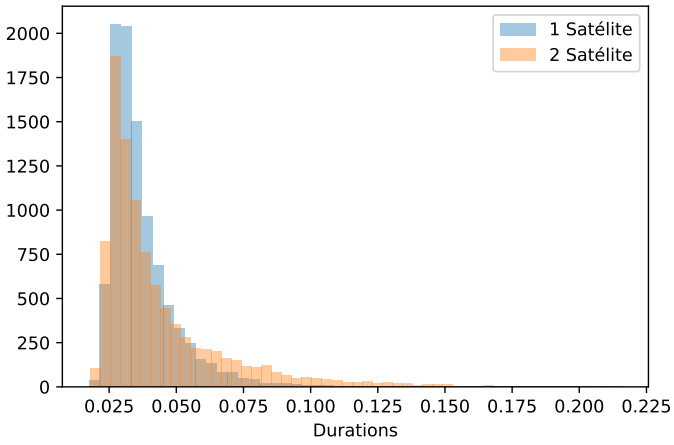
A Figura 29 é criada usando a mesma função em *python* para a figura anterior, mas desativamos o parâmetro *density*. O resultado é um gráfico onde na ordenada apresenta o número

[22] https://www.mathworks.com/products/satellite-communications.html

[23] A KDE é uma técnica para estimar a distribuição de probabilidade de um conjunto de dados baseados em uma amostra de dados.

de requisições que tiveram duração expressas na abscissa. Observe que para 2 satélites temos uma leve, mas perceptível dispersão dos registros de tempo para as requisições. Isso significa que quando mais aumentamos o número de satélites requisitando simultaneamente à mesma antena, aumentará consequentemente o tempo para que a antena responda individualmente cada satélite.

Figura 29 – Estimativa KDE para 2 satélites requisitando dados



Fonte: Próprio autor

No capítulo chamado de introdução, apresentamos 2 questões que buscamos responder ao logo de nossa jornada de experimentos. Na conclusão apresentamos uma seção chamada de *questões adicionais* com novas perguntas e suas respostas. Uma delas é exatamente como as estações terrestres identificam e respondem cada satélite. Isso faz sentido para nosso contexto que objetiva oferecer a infraestrutura para comunicação de múltiplas antenas para o mesmo satélite e o contrário, múltiplos satélites a uma mesma estação.

Medições de tempo no experimento em hardware chamando a blockchain e o banco SQL O primeiro resultado que analisamos foi o consumo de tempo para as requisições dos fragmentos de firmware feito pelo nanosatélite à Blockchain e ao banco de dados no *localhost*. Frisamos que o tamanho do firmware é de 301Kb e foi fragmentado dem 9566 fragmentos de 32Kb cada.

Após o processo de fragmentação do firmware ocorrer com sucesso, o próximo passo foi remontar toda a imagem do firmware original. Ao concluir a atividade utilizamos o algoritmo *sha1sum* para realizar o *checksum* [24] tanto do firmware original (antes da fragmentação) e o

[24] Checksum ou soma de verificação é um código usado para verificar a integridade de dados transmitidos através de um canal com ruídos ou armazenados em algum meio por algum tempo

firmware remontado. Esse experimento teve como intuito verificar se no processo de remontagem o novo firmware estava corrompido. O resultado é apresentado na Figura 30. Notemos que os dois *hashs* são idênticos, indicando que as duas imagens são iguais. O firmware original chama-se *fw-green.bin* e o remontado tem o nome de *remonte-fw.bin*.

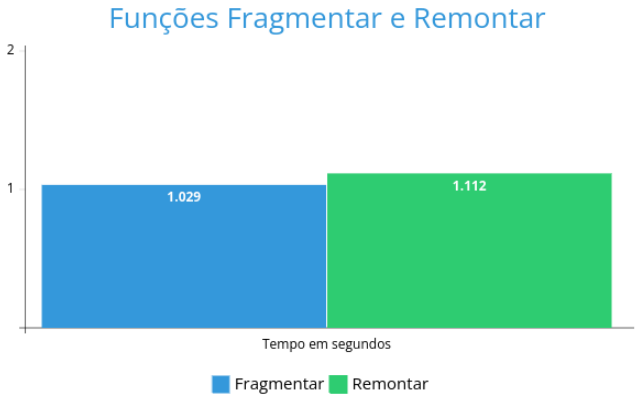
Figura 30 – Comparando o SHA1 dos dois binários

```
(web3) edilson@edilson-NH4CU03:~/Workspace/gs_blockchain$ sha1sum /home/edilson/Workspace/gs_blockchain/gs_bc/fw_green.bin
5e8992fc3ae1b6e8b118aa090c4a8488aa072c3c  /home/edilson/Workspace/gs_blockchain/gs_bc/fw_green.bin
(web3) edilson@edilson-NH4CU03:~/Workspace/gs_blockchain$ more /home/edilson/Workspace/gs_blockchain/gs_bc/remonte_fw.bin
(web3) edilson@edilson-NH4CU03:~/Workspace/gs_blockchain$ sha1sum /home/edilson/Workspace/gs_blockchain/gs_bc/remonte_fw.bin
5e8992fc3ae1b6e8b118aa090c4a8488aa072c3c  /home/edilson/Workspace/gs_blockchain/gs_bc/remonte_fw.bin
(web3) edilson@edilson-NH4CU03:~/Workspace/gs_blockchain$
```

Fonte: Próprio autor

O tempo para realizar a tarefa de fragmentar e remontar a imagem no servidor é expresso pela Figura 31. A coluna azul indica o tempo para fragmentação e a verde indica a remontagem.

Figura 31 – Tempo para executar funções no servidor da estação terrestre



Fonte: Próprio autor

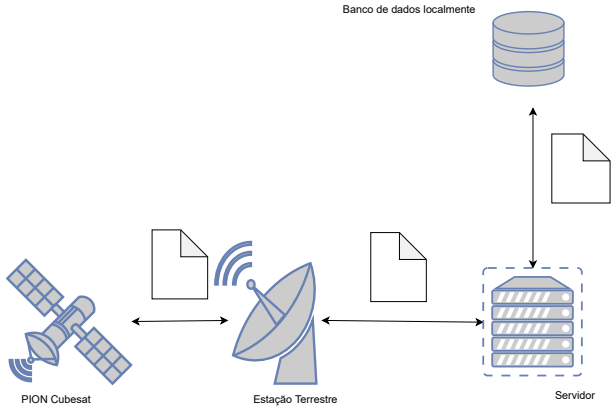
As Figuras 32 e 33 representam como foi desenvolvida a arquitetura para o experimento. A primeira representa o nanosatélite fazendo uma requisição ao servidor que, por sua vez, faz uma requisição ao banco de dados tradicional. Uma vez realizada a consulta, o banco de dados envia o fragmento solicitado ao servidor que envia para o cubesat. A segunda figura representa o mesmo modelo, mas o servidor, neste caso, se comunica com a blockchain.

A Figura 34 compila os dados de testes das 500 requisições seguidas à API pelos dispositivos. a Tabela 11 no Item Dispositivo mostra as configurações do servidor.

Tal surpreendeu, pois esperávamos que o tempo para ler o arquivo na *blockchain* fosse muito maior do que o de ler sem fazer solicitações à *blockchain*. Para entender mais como a blockchain da *Hyperledger Besu* guarda os dados, podemos encontrar em sua documentação[25].

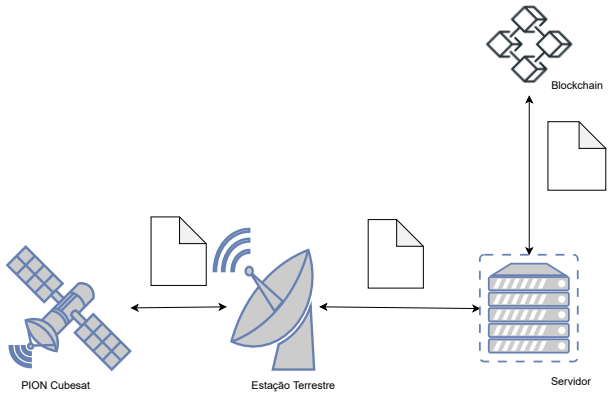
[25] https://besu.hyperledger.org/en/stable/public-networks/concepts/data-storage-formats/?h=storage+databonsai-

Figura 32 – Arquitetura de consulta e envio pelo banco de dados



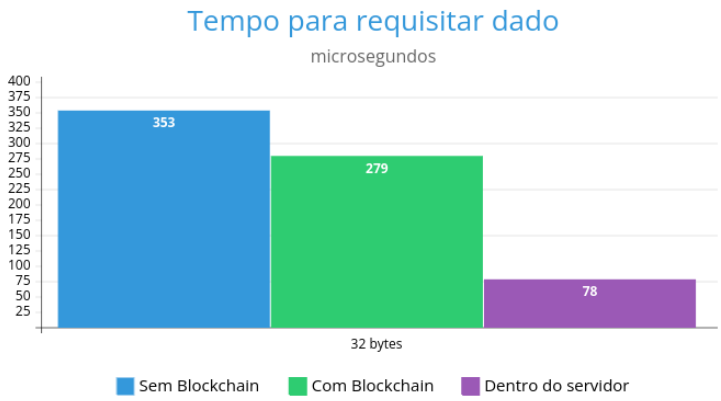
Fonte: Próprio autor

Figura 33 – Arquitetura de consulta e envio pela *blockchain*



Fonte: Próprio autor

Figura 34 – Comparação de tempos de solicitação.



Fonte: Próprio autor

O formato de armazenamento chave-valor que a blockchain usa permite que algumas consultas sejam realizadas de forma mais rápida do que em um banco de dados relacional, à exemplo do MySql. O experimento acima tinha como controle realizar a mesma chamada 500 vezes, mas era o mesmo fragmento de dado chamado repetidamente. Faltávamos chamar seguidamente todos os fragmentos e esse foi nosso próximo passo no experimento.

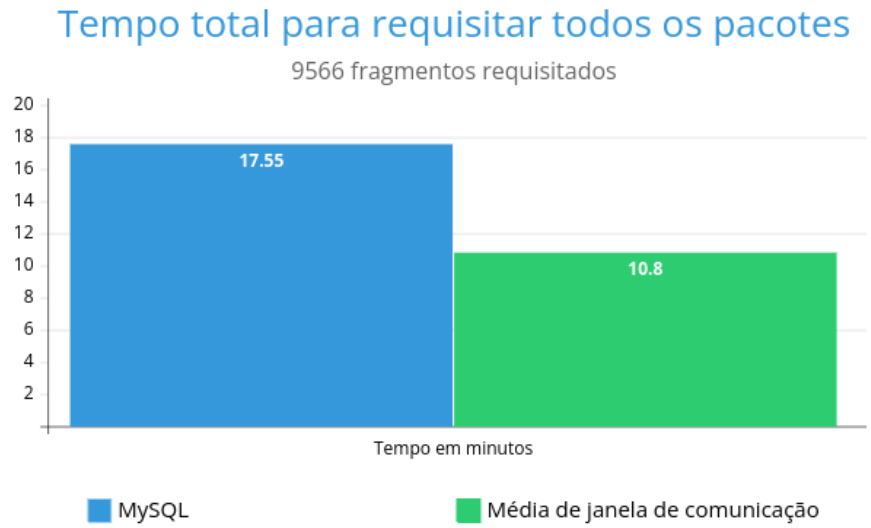
tries

Neste novo teste, usamos o mesmo firmware funcional com 301Kb [26] que foi fragmentado em 9566 partes de 32 bytes e instanciado em um servidor com um banco de dados MySQL (Figura 32). O nanosatélite comunicou-se com o roteador que emula uma estação terrestre e esta, por sua vez, comunicou-se com o banco de dados em MySQL.

Uma observação é que nos testes optamos por não gravar o dado na flash, posto que devido o volume de requisições, poderíamos danificar o dispositivo. O último passo foi compilar o tempo total decorrido para realizar a totalidade das requisições dos fragmentos de firmware.

Levando em conta o resultado de média de tempo do link ativo entre estação terrestre e nanosatélite que obtivemos na simulação em matlab, criamos um gráfico que confronta os intervalos de tempo de fazer a requisição e a média de tempo janela de cominicação ativa. O gráfico da Figura 40 traz na cor azul o tempo para completar as 9566 requisições do cubesat direto do bando de dados MySql.

Figura 35 – Tempo total para concluir requisições



Fonte: Próprio autor

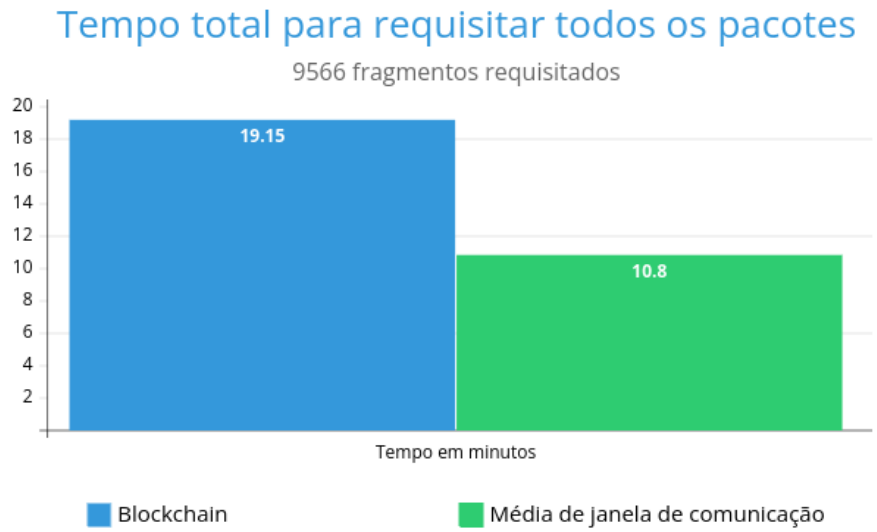
O resultado mostra que requisitar cada fragmento no modelo nanosatélite -> estação terrestre -> banco de dados levou muito mais tempo que a janela de comunicação apresentada em verde. É verdade que, em um ambiente real, o sistema de comunicação seria através de antenas e este resultados poderá certamente ser maior, quando introduzimos, por exemplo, as atenuações e perdas de propagação do sinal.

Feito o procedimento de requisitar os fragmentos de firmware armazenados em um banco de dados MySQL, realizamos o experimento novamente mas, desta vez, realizando as

[26] Este firmware possui as funções e bibliotecas para o computador de bordo conectar-se à internet, acender leds, realizar contagem de tempo e alguns laços para marcação de rotinas internas

chamadas direto à *blockchain*. Figura 36.

Figura 36 – Tempo total para concluir requisições



Fonte: Próprio autor

Foram realizadas o mesmo número de requisições ao mesmo servidor anterior e já no primeiro resultado notamos que, embora a diferença de tempo entre o tempo total para chamar o MySQL e a blockchain sejam pequenos, isso representa muito mais do que a média de tempo de 1 janela de comunicação, ou seja, seriam necessárias mais passagens pela estação terrestre até completar a atividade. Se a missão de controle deste cubesat tiver à disposição somente 1 estação terrestre, a depender do momento que o comando de telemetria fosse acionado, poderiam durar horas até o próximo link ativo fosse estabelecido.

Observamos que, em ambos os casos, serão necessários mais um de uma janela de comunicação para enviar a totalidade dos pacotes de dados. Com base na simulação realizada usando o *Satellite Communication Toolbox* do Matlab, durante um recorte de 90 dias de missão, foram realizadas 396 comunicações entre o satélite e a estação terrestre. Considerando que seriam necessários cerca de 20 minutos para completar a totalidade dos envios de pacotes (levando em conta retransmissão de pacote e uma margem de identificação), investigamos quantas janelas de comunicações seguidas seriam necessárias para completa a atividade. Assim usando a coluna *duration* que contabiliza em segundos o tempo de link ativo entre estação terrestre e satélite, chegamos ao resultado que no mínimo 2 e no máximo três janelas de comunicação seriam necessárias. Levando em conta o tempo total da missão (dentro do recorte de 90 dias) chegamos o resultado:

O resultado é aceitável e condizente com as missões atuais. Tomando ainda os dados

Tabela 18 – Disponibilidade de janelas de comunicação

Nº de janelas necessárias	Total disponível na missão
2	145 grupos de janelas seguidas
3	35 grupos de janelas seguidas

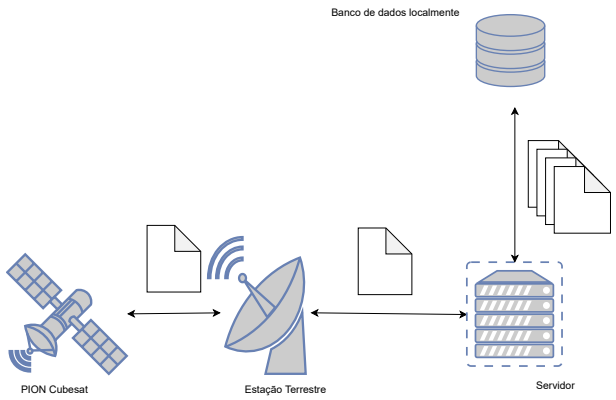
Fonte: o autor.

da simulação do Matlab, realizamos os cálculos necessários e descobrimos que o menor tempo entre duas janelas de comunicações foi de 1 hora e 26 minutos e o maior tempo foi te 10 horas e 43 minutos. É fácil perceber que quando se aumenta o número de estações terrestres, aumenta-se o tempo de comunicação entre missão de controle e satélite.

> No experimento realizado e descrito acima, os dados foram lidos em tempo real do servidor via API e enviados para o satélite. Como seriam os resultados se os dados fossem lidos do banco de dados e/ou da *blockchain* e salvos em tempo de execução e só depois enviados para o satélite?

Isso então nos levou a mudar a forma de como os dados são consultados na Blockchain e na banco de dados MySQL. Arquitetamos um novo experimento onde, uma vez iniciada a solicitação de dados, todos os dados são requisitados e ficam prontos no servidor para serem enviados. Depois são enviados 1 a 1 até concluir a tarefa. Veja Figura 37.

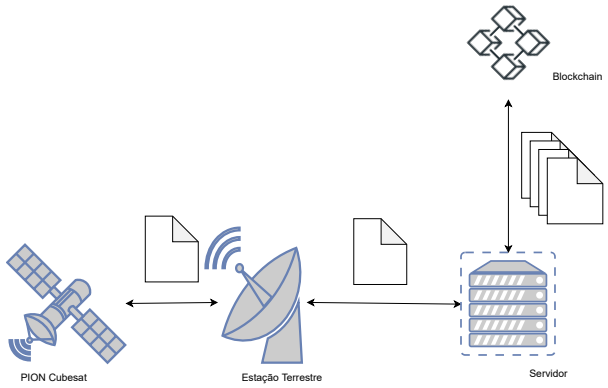
Figura 37 – Obter completamente o *firmware* no banco de dados MySQL



Fonte: Próprio autor

O mesmo processo também se aplica na arquitetura expresso pela Figura 38. Neste caso, a diferença que podemos notar é que a consulta não é feita em um banco de dados tradicional, mas direto na blockchain, isto é, solicita-se todos os fragmentos de código que estão armazenados na *blockchain*, organiza-os e remonta-se a imagem do binário na memória do servidor em tempo de execução.

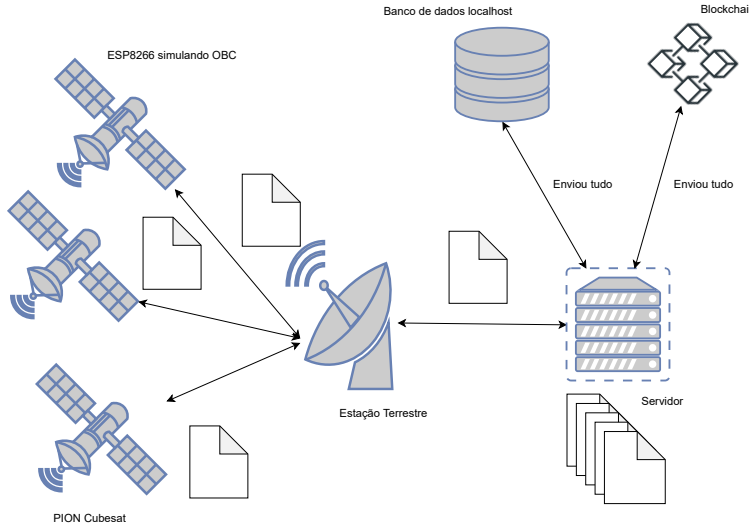
Figura 38 – Obter completamente o *firmware* da *blockchain*



Fonte: Próprio autor

Neste etapa do experimento fizemos o primeiro teste apenas com o cubesat fazendo requisições ao servidor, na segunda acrescentamos um concorrente, isto é, o ESP8266 emulando um computador de bordo. Já na terceira parte, colocamos um notebook também realizando chamadas. O objetivo foi medir a interferência de sistemas concorrentes durante as requisições. A Figura 39 apresenta a arquitetura de múltiplos dispositivos fazendo requisições.

Figura 39 – Arquitetura de consulta por três dispositivos ao banco de dados

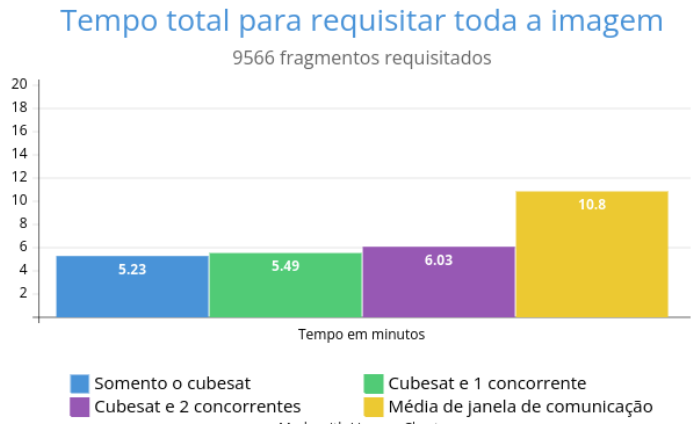


Fonte: Próprio autor

Como ambos os processos, MySQL e blockchain, virtualizaram na memória do servidor em tempo de execução a resposta de todas as consultas, no fim tivemos, para o envio de dados do servidor (estação terrestre) para o nanosatélite intervalo de tempo satisfatório, concluindo a totalidade de envio de dados dentro da janela de comunicação, veja Figura 40 o resultado dos 3 experimentos. Em azul o compilado do primeiro teste com apenas o cubesat realizando chamadas. O tempo alcançado está bem abaixo da janela de comunicação, o que daria

margem para reenvio de pacotes de dados corrompidos durante a transmissão, caso estivessémos usando sistema de rádio. Note que nas colunas verde e roxa podemos notar a interferência de sistemas concorrentes, mas ambos ainda dentro da janela de comunicação.

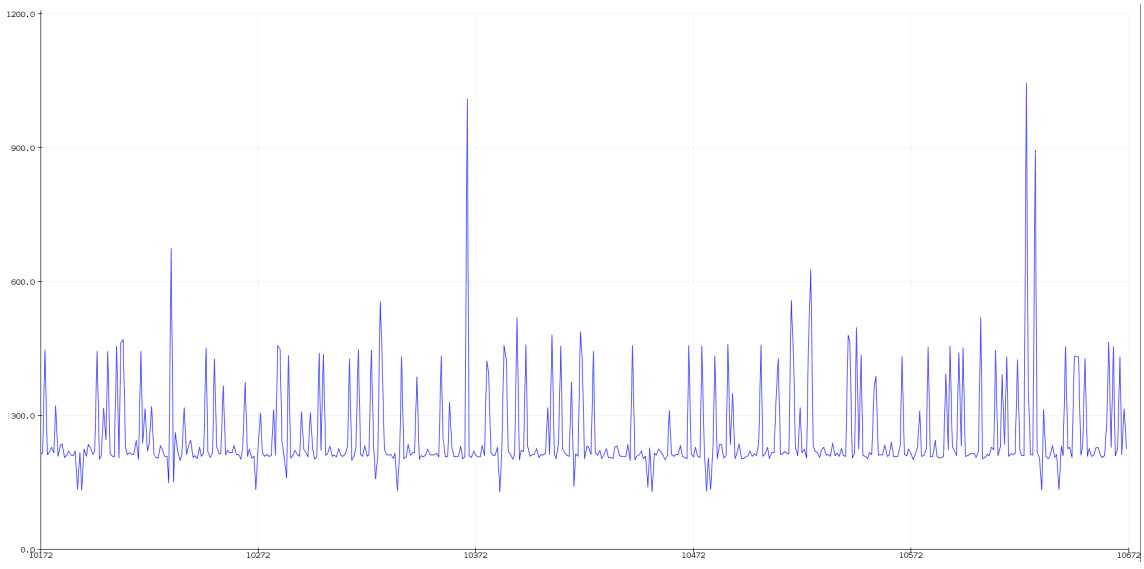
Figura 40 – Tempo de consulta por três dispositivos ao Banco de dados



Fonte: Próprio autor

A série apresentada na Figura 41 possui como eixo x (horizontal) o número de requisições e o eixo y (vertical) o tempo em microsegundos.

Figura 41 – Série temporal com dois dispositivos fazendo requisições



Fonte: Próprio autor usando dados da IDE Arduino

Veja que foram realizadas mais de 10.000 requisições e nesse recorte temporal destacamos a existência de 5 picos acima dos 600 microsegundos. Tais picos ocorreram quando o servidor precisou atender as múltiplas requisições. Usamos no dispositivo concorrente ao Cubesat um período de chamada à API a cada 0,2 segundos. Por fim realizamos mais testes de requisições à API desta vez com 3 dispositivos, onde além do cubesat e do ESP8266, usamos

uma aplicação chamada *Insomnia* [27] rodando dentro do próprio servidor para fazer requisições. Para este teste rodamos uma bateria de 32.000 requisições. Vejas as imagens e comentários no Apêndice B.

Finalizamos esta seção com uma questão interessante. *Se em ambos os cenários, usando MySQL e usando blockchain, as consultas foram todas realizadas e os resultados virtualizados em memória do servidor, qual o motivo de usar blockchain?*.

A resposta está na capacidade que a *blockchain* tem de criar a infraestrutura de descentralização da aplicação e a opção de criação de uma camada de negócios automatizada, que permite que os detentores de estações terrestres gerem receita ao compartilhar suas antenas e estações. No paradigma de banco de dados centralizados, MySQL ou outro no estilo, teríamos que encontrar artifícios para mitigar os desafios de oferecer alta disponibilidade de serviço, especialmente nos momentos de registro de informações no banco (evitar duplicidade de dados e etc). Outro ponto é a necessidade da implementação de camadas extras de tolerância à falhas e segurança, algo já nativo em aplicações blockchain.

Recurso energético e potência

O PION Cubesat é composto por diversos sensores e componentes que consomem recurso energético. Segundo o fabricante do microcontrolador, a corrente é de 30mA quando está totalmente ligado, mas pode ter um pico de 350mA durante, por exemplo, a incialização do sistema ou no processo de busca e conexão ao *wifi*. A potência de saída é de 22.5bBm (250mW) para o WiFi e de +6dBm para o *Bluetooth*, em nossos experimentos não fizemos uso da função *Bluetooth*. No cenário de uso constante do dipositivo, uma bateria de 2500mAh duraria no máximo 35 horas, ou seja, menos que dois dias. Conforme visto na seção de introdução, um nanosatélite em órbita LEO a cada 90 minutos completa um ciclo ao redor da Terra, sendo assim, ele terá constantemente acesso à luz do Sol para recarregar as baterias.

Taxa de erro na implementação em hardware

Em nosso experimento, utilizamos uma arquitetura que conectou via *wifi* o PION cubesat ao *localhost*, isto é, a emulação de nossa estação terrestre hospedada em nossa máquina e servidor. Certamente a implementação de comunicação via antenas nos trará resultados mais próximos ao ambiente real. Em nosso trabalho durante as 500 vezes que realizamos os testes de requisições à Blockchain via API, não identificamos inversões de bits ou diferença entre o dado armazenado na blockchain e o dado armazenado na flash do nanosatélite ou do ESP8266.

[27] https://insomnia.rest/

O roteador de internet usado no experimento é um Tp-Link Acher 20 com velocidade de *wifi* na faixa 2.4GHz de 300 Mbps (802.11n). Este cenário então nos concede resultados de experiência em bancada que divergirão do cenário de uso, por exemplo, de antenas. Por isso iremos retornar a esta reflexão no capítulo final e referenciar como trabalho futuro.

4.2.2 Blockchain

Custo das transações Como explanado anteriormente, a blockchain é um conjunto de nós, ou seja, computadores, que realizam diversas tarefas, entre elas escrever e ler dados. As transações de leitura inicialmente não possuem custos mas as de escrita, sim, pois exigem mais da rede.

Logo, um dos resultados que buscamos em nosso experimento foi o custo (em tokens) para armazenar os fragmentos de firmware na rede. Um primeiro resultado encontrado é o custo em dólares, veja Tabela 19. O *gás* é o valor cobrado por cada transação no Ethereum. Os preços do gás são indicados em *gwei*, uma denominação proprietária de ETH em que cada *gwei* é igual a 0,00000001 ETH. Esse valor pode variar de acordo com diversos fatores, como a demanda da rede e o tipo de operação. Considerando que cada fragmento tem 32 bytes de tamanho criamos a Tabela 19 que lista o custo em *gás* e em dólares para registrar pedaços de firmware. Observe a correspondência entre o tamanho do firmware e o custo da transação em dólares americanos. Embora haja variação no preço de uma criptomoeda como a ETH, as instituições podem prever em seus projetos quanto gastarão em processos de atualização de firmware. O preço do ETH em relação ao dólar foi de setembro de 2022.

Tabela 19 – Custos para armazenar fragmentos de *firmware* na *blockchain*

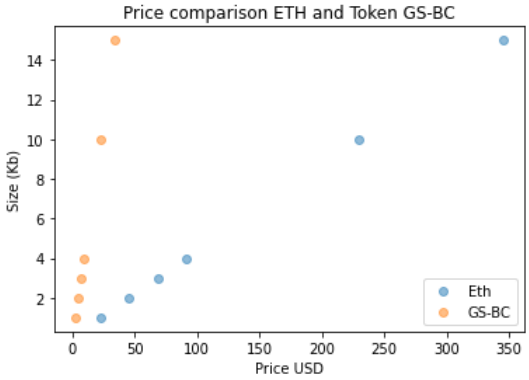
Tamanho do *firmware* (Kb)	Nº Fragmentos	Gás	ETH	USD
1	32	9017687.5	0.017206	22.99
2	63	18035375.0	0.034412	45.98
3	94	27053062.5	0.051618	68.97
4	125	36070750.0	0.068823	91.96

Fonte: o autor.

Token GS-BC Algo que nossa arquitetura permite é a criação de tokens, ou seja, ativos digitais, que podem substituir o ETH (*Ethereum)*. Isso é interessante posto que os preços das transações não ficariam vinculados às flutuações do ETH. Ao mesmo tempo, permite uma nova forma de rentabilizar o projeto e também criar negócios em cima da solução.

A Figura 42 mostra a perspectiva de utilização do Token GS-BC para a rede privada do consórcio. Ademais, sugere que é economicamente mais atraente para projetos privados como é nosso alvo, por exemplo, universidades e pequenas e médias empresas privadas. Um resultado adicional pode ser encontrado na Tabela 20.

Figura 42 – Comparação entre ETH e o Token GS-BC



Fonte: Próprio autor

Viabilidade de Descentralização de estações terrestres

Tomando por base uma missão tradicional, por exemplo, uma missão que usa antena UHF/VHF para a comunicação entre o controle da missão e o nanosatélite, podemos projetar o custo para a aquisição desse equipamento e comparar com as atuais soluções de mercado, bem como, com nossa solução proposta. Para efeitos de cálculo, não consideramos operadores técnicos, custo de energia, etc. A alternativa comercial para locação de estações terrenas escolhemos o serviço da AWS (*Amazon Web Services*, do inglês), o Amazon Ground Station. [28].

Para realizarmos o cálculo do custo por minuto estação terrestre, comprada, depreciarmos o valor total da estação pelo tempo de missão, onde δ representa o valor em dólares USD por minuto. P é o preço de aquisição da antena e Tm é o tempo de missão em minutos. Assim, vêm a equação 4.1.

$$\delta = \frac{P}{Tm} \tag{4.1}$$

Para uma missão de 18 meses diante da compra de uma estação terrestre no valor de

[28] https://aws.amazon.com/pt/ground-station/

USD 69,750.00 [29] teremos:

$$\delta = \frac{69,750.00}{18x30x24x60} = \frac{0.0896USD}{min} \tag{4.2}$$

Usando a mesma equação, podemos calcular o δ para AWS e o P para a proposta de descentralização de antenas. Segue preço AWS e GS-BC respectivamente:

$$\delta = \frac{29,700.00}{18x30x165} = \frac{0.33USD}{min} \tag{4.3}$$

$$\delta xTm = P = 1.00x1430x18 = 25,740.00USD \tag{4.4}$$

No cálculo da equação 4.4 podemos notar que USD 1.00 é o custo por minutos (no sistema de descentralização de estações terrestres) e 1430 é a média (em minutos) do link ativo da missão cubesat simulada em 90 dias.

A Tabela 20 compila os resultados dos cálculos (4.2, 4.3 e 4.4). É interessante notar que mesmo a *AWS Ground Station* tendo um valor final com preço intermediário, por mês ela oferece quase 10 vezes menos tempo de antena do que a alternativa que usa GS-BC.

Tabela 20 – Custo para usar (comprar ou alugar) estação terrestre VHF/UHF

Opção	Missão	Custo por minuto (USD)	Minutos por mês	Preço Final	Preço/mês (USD)
Tradicional	18 meses	0.08	43.800	69,750.00	3,875.00
AWS GS	18 meses	0.33	165	29,700.00	1,650.00
GS-BC	18 meses	1.00	1,460	25,740.00	1,430.00

Fonte: o autor.

A AWS tem um padrão de qualidade alto e embora a princípio seu custo final em um pacote de 12 meses seja de USD 19.800,00 para uma missão de 18 meses, foi estimado em USD 29.700,00 com apenas 165 minutos por mês. Em nosso estudo, além de propor o uso descentralizado das estações terrestres, criamos um token GS-BC e adicionalmente propusemos a oferta do serviço de uso do GS a USD 1.00 por minuto. Nossos cálculos indicam que, em uma missão de 18 meses, o custo do serviço compartilhado pode representar uma economia de

[29] https://www.cubesatshop.com/product/full-ground-station-kit-vhfuhf/

63.1% do preço de compra de uma estação UHF/VHF. Torna-se uma opção, pois permite um melhor controle na gestão do recurso de aluguel de estações terrenas, é mais barato que outras alternativas e ainda pode se tornar uma fonte de receita para instituições que possuem estações terrenas para aluguel no consórcio

Gerando receita com a estação terrestres

Temos comentado que a descentralização das estações terrestres, no modelo apresentado, pode oferecer uma fonte de receita para a instituição detentora da antena. Isto se dá pelo fato do sistema descentralizado poder ser regido por contratos inteligentes específicos e do uso dos *tokens* GS-BC. Desta feita, instituições que não possuem antenas, mas que desejam usar o serviço poderão adquirir estes *tokens*, ou criptoativos, para pagar as transações de registro na rede. Por meio dos contratos inteligentes é automatizado este processo de prestação de serviço (alocação da antena) e consumo (por parte da missão de controle), no formato de GSaaS (*Ground Station As a Service*, do inglês) [30].

[30] https://www.linkedin.com/pulse/ground-station-service-gsaas-ganesh-sahai/

5 CONCLUSÕES E TRABALHOS FUTUROS

Iniciamos este trabalho apresentando dois questionamentos que engenheiros e desenvolvedores de missões cubesat trazem em seus projetos, são elas:

Questão 1: Como tornar mais eficiente, em questão de tempo, e seguro a comunicação entre satélite e controle da missão?

Questão 2: Como tornar acessível o serviço de estações terrestres pra que institutos de ensino, universidades e pequenas empresas do setor espacial consigam realizar as tarefas de rastreio, controle, envio e recepção de dados sem perder a segurança e confidencialidades de seus projetos?

Nosso trabalhou refletiu sobre alternativas diante das soluções existentes para os questionamentos ora mencionados.

5.1 Questão 1

Na seção 3, concluímos que, para alguns projetos, pode ser inviável tornar um nanosatélite um nó *blockchain*, especialmente pela questão de armazenamento de dados, isto é, cada nanosatélite possuir uma cópia de cada registro feito na rede. Do lado oposto, tornar as estações terrestres participantes de um consórcio de nós blockchain mostrou-se interessante e com baixo custo de implementação. As simulações no matlab e os experimentos de bancada usando o nanosatélite da PION nos mostraram a possibilidade do uso do blockchain para processos de comunicação, um exemplo implementado foi a atualização de firmware. A nível de consumo de tempo, estimou-se em simulação e em bancada, a necessidade de 2 a 3 janelas de comunicação para o processo de envio de todos os fragmentos para compor a nova imagem de firmware, nesse ponto nada diferente do padrão. Quando aumentamos o número de estações terrestres, o tempo para a realização do processo de atualização de firmware diminui, o que torna a ideia de consórcio descentralizado de estações terrestres muito interessante e com forte apelo ao mercado.

Os resultados que encontramos confirmam a eficiência e a aderência da tecnologia blockchain na tarefa de atualização de firmware, ao criar uma infraestrutura descentralizada de estações terrestres, pois oferece ao satélite mais oportunidades de se comunicar com o controle

da missão através de uma rede de antenas que possuem uma cópia exata dos mesmos dados e, ao mesmo tempo, um protocolo seguro de comunicação entre as partes.

5.2 Questão 2

As atualizações de firmware são um componente essencial de missões espaciais bem-sucedidas. Considerando as restrições orçamentárias que diversos projetos universitários sofrem na instabilidade do mercado mundial, soluções que permitam o compartilhamento de custos de missões espaciais são sempre bem-vindas. Nosso projeto não oferece apenas uma solução para o problema de missões espaciais universitárias mais baratas mas, também, para iniciativas industriais e comerciais. Oferecemos uma solução robusta e segura em uma infraestrutura confiável para *upload* e atualização de firmware para dispositivos espaciais, no caso estudado, para cubecats. Implementamos um token chamado GS-BC com um valor fixo inicial de 10% de ETH ou outro valor indexado em uma moeda como o dólar. Concluímos também que os custos de registro do firmware na rede blockchain são aceitáveis diante das facilidades e segurança do sistema.

5.3 Questões adicionais

Como nossa proposta objetiva fornecer o serviço no formato GSaaS, podemos trazer algumas questões adicionais.

Questão 3: Como lidar com o acesso simultâneo de satélites de missões diferentes à mesma estação terrestre?

Essa é uma ótima questão, pois quando mais de um satélite acessa a mesma estação terrestre, ou seja, ambos compartilham um determinado espaço de tempo sendo visíveis pela mesma estação terrestre, isso pode causar interferência entre os dois sinais de comunicação. Isso pode afetar a qualidade do sinal recebido ou enviado e pode causar erros na decodificação dos dados.

Há várias maneiras de lidar com a interferência de múltiplos sinais em uma estação terrestre. Uma forma é usar técnicas de multiplexação de frequência, onde a cada satélite é atribuída uma frequência diferente para se comunicar com a estação terrestre. Deste modo, os sinais de cada satélite podem ser separados e decodificados independentemente. Outra forma é usar técnicas de multiplexação por divisão de tempo, onde os sinais de cada satélite

são transmitidos em momentos diferentes, permitindo que a estação terrestre os receba e os decodifique sequencialmente.

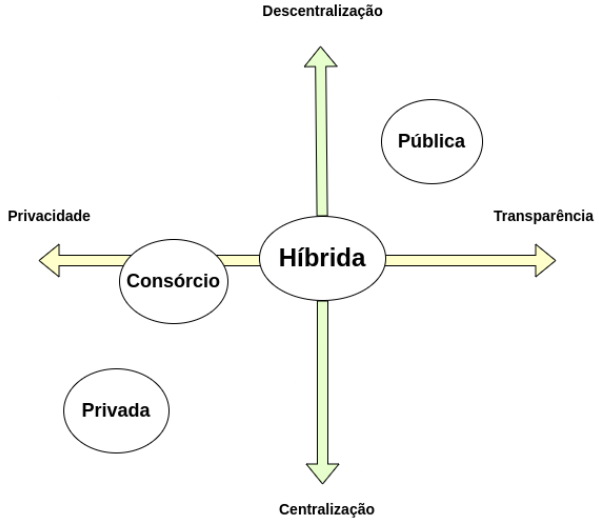
É importante frisar que essas técnicas de multiplexação requerem um planejamento cuidadoso da banda de frequência e do tempo de transmissão, para evitar que os sinais sofram interferência. Além disso, é importante garantir que as estações terrestres e os satélites estejam sincronizados corretamente.

Além disso, existe também a possibilidade de utilizar técnicas de codificação e modulação para diferenciar os sinais entre os satélites, tornando mais fácil para a estação terrestre diferenciar e decodificar os sinais.

Questão 4: No modelo proposto como se relacionam privacidade e o modelo de consórcio de estações terrestres?

A questão 4 surgiu durante nossos estudos e implementações dos experimentos. Foi natural nos perguntarmos como se relacionam os paradigmas de blockchain e os conceitos de privacidade e transparência. Algumas revisões de literatura sobre o tema têm trazido luzes sobre esse "equilíbrio difícil", mas podemos sintetizar com base em estudos recentes (SONMEZ *et al.*, 2021) em um quadro geral. A Figura 43 nos apresenta um compilado prático.

Figura 43 – Quadrante de paradigmas *blockchain* e relação com privacidade x transparência



Fonte: Adaptado pelo autor, inspirado em (SONMEZ *et al.*, 2021)

Nossa proposta pela escolha do modelo de consórcio traz três características, cada uma com pesos diferentes. A *privacidade*, escolhemos por que é um fator relevante e uma preocupação real das missões espacias. A forte presença do fator *centralizado*, pois corrobora com o quesito de segurança, ou seja, fortes características de uma entidade reguladora para orquestrar a

entrada dos membros (instituições educacionais, empresas e provedores de serviço - estações terrestres) e os nós computacionais. Mesmo com a forte característica de centralização, o modelo de consórcio permite que os nós estejam presentes em diferentes locais, dentre os membros autorizados, o que fornece então, mesmo que leve, mas a característica de *descentralização*.

5.4 Projeto Futuro

Nosso experimento utilizou uma arquitetura muito parecida com as usadas em internet das coisas, assim os conceitos do nosso projeto também podem ser aplicados em dispositivos que se enquadram em Internet das coisas, como também para a internet das coisas em movimento. Um exemplo é a atualização de firmware de veículos autônomos, como carros, ônibus, drones etc.

Como projeto futuro, queremos ampliar o estudo e implementação de tecnologia para esses outros processos, como aquisição de dados e calibração de sensores, utilizando infraestrutura centralizada de estações terrestres e blockchain.

Almejamos ainda:

a) Implementar comunicação via rádio, utilizando antenas tanto no nanosatélite como na estação terrestre;
b) Implementar nós blockchains em outros dispositivos como *raspberry pi Model 4* e criar um *cluster* com pelo menós 3 nós ligados em redes diferentes;
c) Implementar o serviço de cliente blockchain, que fizemos no PION Cubesat educacional, em outros Computadores de bordos, especialmente com outros processadores. Um exemplo é usar a placa de desenvolvimento chamada *Hércules* [1] da Texas Instruments ou no *ISIs* [2].

[1] https://software-dl.ti.com/hercules/hercules$_d$*ocs*/*latest*/*hercules*/*index.html*
[2] https://www.cubesatshop.com/product/isis-on-board-computer/

REFERÊNCIAS

BEAUJARDIERE, J. de L.; MITAL, R.; MITAL, R. Blockchain application within a multi-sensor satellite architecture. In: **IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium**. [*S. l.*: *s. n.*], 2019. p. 5293–5296.

BING, B. A fast and secure framework for over-the-air wireless software download using reconfigurable mobile devices. **IEEE Communications Magazine**, v. 44, n. 6, p. 58–63, 2006.

BRAGA, A. M. Tecnologia blockchain: fundamentos, tecnologias de segurança e desenvolvimento de software. **Campinas: CPQD**, 2017.

BRIDGES, C.; YEOMANS, B.; IACOPINO, C.; FRAME, T.; SCHOFIELD, A.; KENYON, S.; SWEETING, M. Smartphone qualification linux-based tools for cubesat computing payloads. In: **2013 IEEE Aerospace Conference**. [*S. l.*: *s. n.*], 2013. p. 1–10.

BUSCH, S.; SCHILLING, K.; BANGERT, P.; REICHEL, F. Robust satellite engineering in educational cubesat missions at the example of the uwe-3 project. **IFAC Proceedings Volumes**, Elsevier, v. 46, n. 19, p. 236–241, 2013.

CAKAJ, S.; KAMO, B.; KOLIÇI, V.; SHURDI, O. The range and horizon plane simulation for ground stations of low earth orbiting (leo) satellites. **Int. J. Commun. Netw. Syst. Sci.**, v. 4, n. 9, p. 585–589, 2011.

CARVALHO, R. A. Optimizing the communication capacity of a ground station network. **Journal Aerospace Technology and Management**, SciELO Brasil, v. 11, 2019.

GANGESTAD, J.; HARDY, B.; HINKLEY, D. Operations, orbit determination, and formation control of the aerocube-4 cubesats. 2013.

GANGESTAD, J. W.; ROWEN, D. W.; HARDY, B. S. Forest fires, sunglint, and a solar eclipse: Responsive remote sensing with aerocube-4. In: IEEE. **2014 IEEE Geoscience and Remote Sensing Symposium**. [*S. l.*], 2014. p. 3622–3625.

GAO, S.; YU, T.; ZHU, J.; CAI, W. T-pbft: An eigentrust-based practical byzantine fault tolerance consensus algorithm. **China Communications**, v. 16, n. 12, p. 111–123, 2019.

GARRIDO B. GARCíA, A. A. N.; MIGUEL, J. Minisat01 on-board software maintenance. In: **DASIA 98-Data Systems in Aerospace**. [*S. l.*: *s. n.*], 1998. v. 422, p. 65.

GRECO, M. E.; SNYDER, J. F. Operational modification of the mars explorations rover's flight software. **IEEE Systems, Man and Cybernetics, International Conference**, 2005.

HAMMING, R. Error detecting and error correcting codes. **The Bell System Technical Journal**, v. 29, p. 147–160, 1950.

LEINER B., S. M. O. R.; HUBER, B. A comparison of partitioning operating systems for integrated systems. **Proceedings of 26th International Conference on Computer Safety, Reliability, and Security (SAFECOMP)**, p. 342–355, 2007.

LIU, Z.; XIANG, Y.; SHI, J.; GAO, P.; WANG, H.; XIAO, X.; WEN, B.; LI, Q.; HU, Y.-C. Make web3.0 connected. **IEEE Transactions on Dependable and Secure Computing**, v. 19, n. 5, p. 2965–2981, 2022.

MONTES, J. M.; RAMIREZ, C. E.; GUTIERREZ, M. C.; LARIOS, V. M. Smart contracts for supply chain applicable to smart cities daily operations. In: **2019 IEEE International Smart Cities Conference (ISC2)**. [*S. l.*: *s. n.*], 2019. p. 565–570.

NASON, I.; PUIG-SUARI, J.; TWIGGS, R. Development of a family of picosatellite deployers based on the cubesat standard. In: IEEE. **Proceedings, IEEE Aerospace Conference**. [*S. l.*], 2002. v. 1, p. 1–1.

O'KEEFE, K.; PETOVELLO, M.; CAO, W.; LACHAPELLE, G.; GUYADER, E. Comparing multicarrier ambiguity resolution methods for geometry-based gps and galileo relative positioning and their application to low earth orbiting satellite attitude determination. Hindawi Publishing Corporation, 2009.

PAHLAJANI, S.; KSHIRSAGAR, A.; PACHGHARE, V. Survey on private blockchain consensus algorithms. In: **2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)**. [*S. l.*: *s. n.*], 2019. p. 1–6.

PUTHAL, D.; MALIK, N.; MOHANTY, S.; KOUGIANOS, E.; YANG, C. The blockchain as a decentralized security framework [future directions]. **IEEE Consumer Electronics Magazine**, v. 7, p. 18–21, 03 2018.

ROSA, J. J.; RUFINO, J. Exploiting air composability towards spacecraft onboard software update. **Actas do INForum-Simpósio de Informática**, 2010.

SILVA, F.; MUNIZ, A.; SILVEIRA, J.; MARCON, C. Clc-a: An adaptive implementation of the column line code (clc) ecc. In: **2020 33rd Symposium on Integrated Circuits and Systems Design (SBCCI)**. [*S. l.*: *s. n.*], 2020. p. 1–6.

SONMEZ, R.; SÖNMEZ, F. Ö.; AHMADISHEYKHSARMAST, S. Blockchain in project management: a systematic review of use cases and a design decision framework. **Journal of Ambient Intelligence and Humanized Computing**, Springer, p. 1–15, 2021.

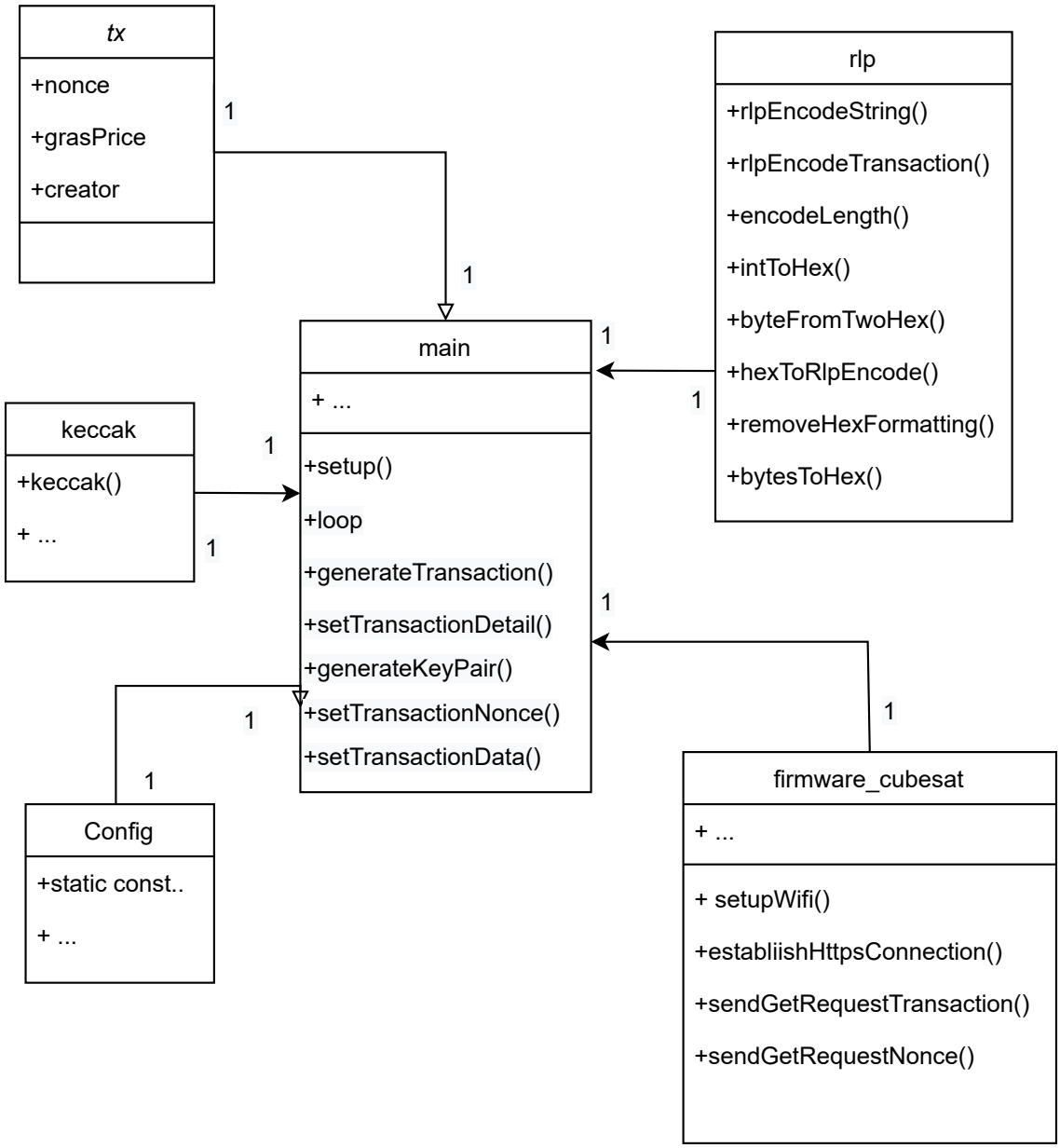
SUJEETHA, R.; PREETHA, C. A. S. D. A literature survey on smart contract testing and analysis for smart contract based blockchain application development. In: **2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)**. [*S. l.*: *s. n.*], 2021. p. 378–385.

SUNTER, I.; SLAVINSKIS, A.; KVELL, U.; VAHTER, A.; KUUSTE, H.; NOORMA, M.; KUTT, J.; VENDT, R.; TARBE, K.; PAJUSALU, M.; VESKE, M.; ILVES, T. Firmware updating systems for nanosatellites. **IEEE Aerospace and Electronic Systems Magazine**, v. 31, n. 5, p. 36–44, 2016.

WOOD, L.; EDDY, W.; IVANCIC, W.; MCKIM, J.; JACKSON, C. Saratoga: a delay-tolerant networking convergence layer with efficient link utilization. In: . [*S. l.*: *s. n.*], 2007. p. 168 – 172. ISBN 978-1-4244-0938-9.

APÊNDICE A – MODELOS UML

Figura 44 – Diagrama UML Modelo

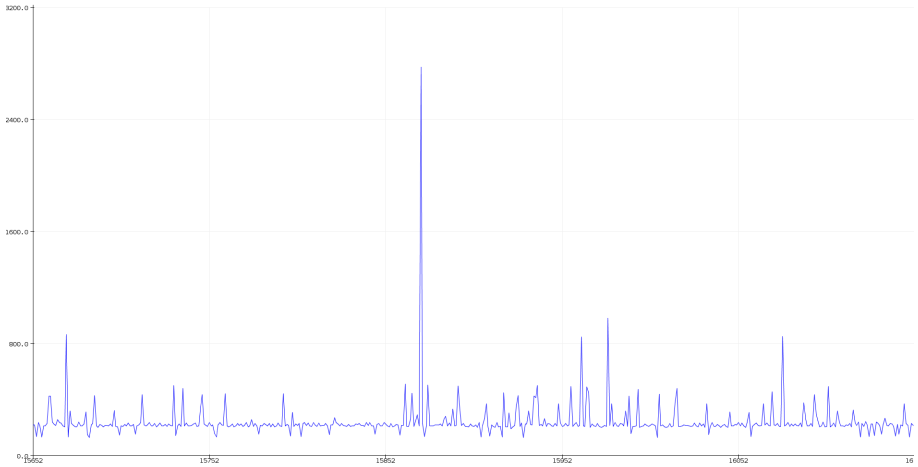


Fonte: Próprio autor

APÊNDICE B – SÉRIES TEMPORAIS DOS EXPERIMENTOS DE REQUISIÇÕES

Nas figuras 45 e 46 o eixo Y é o tempo em milesegundos e o eixo X o número da requisição. Observe na Figura 45 o elevado pico na série correspondendo a um elevado tempo de resposta. No ato do experimento 3 dispositivos faziam requisições simultaneamente ao servidor. O pico não se repete ao longo da série, provavelmente por limitações do computador usado nos experimentos.

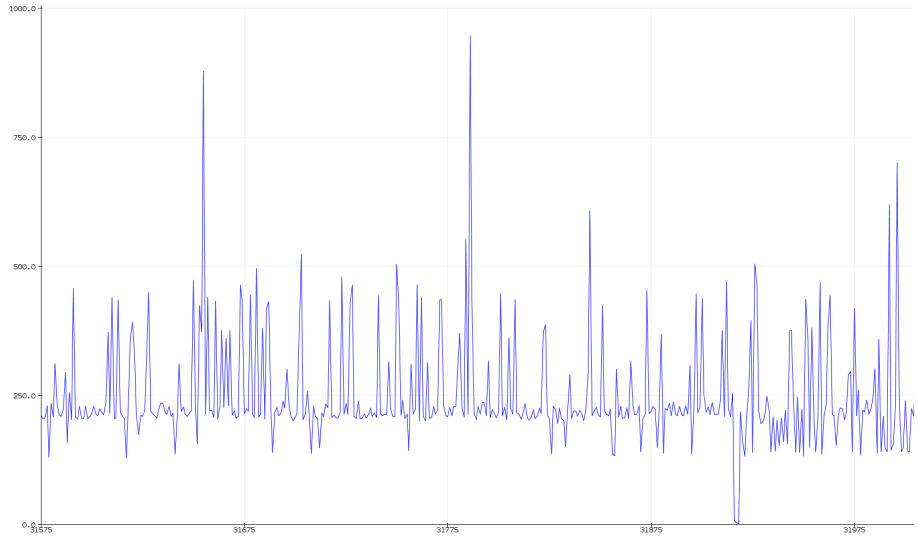
Figura 45 – Série temporal com múltiplos dispositivos



Fonte: Próprio autor usando dados da IDE Arduino

Note que o gráfico vai a zero milesegundos quando o servidor foi desligado.

Figura 46 – Registro do desligamento e religação do servidor



Fonte: Próprio autor usando dados da IDE Arduino

APÊNDICE C – JUPITERNOTEBOOK DE REGISTRO DE DADO NA BLOCKCHAIN

Figura 47 – Fragmento de código para registro de fragmento na *blockchain*

```
In [43]: register_firmware = contract_instance.functions.createFragmentFirmware(fragment, str(transaction_index), '0.1.1').build_transaction(
             {
                 "gasPrice": w3.eth.gas_price,
                 "gas": est_gas * 2,
                 "chainId": chain_id,
                 "from": my_wallet,
                 "nonce": nonce + transaction_index,

             }
         )

         # transaction_index +=1

         print(register_firmware)

{'value': 0, 'gasPrice': 100000000000, 'gas': 296310, 'chainId': 1337, 'from': '0xFE3B557E8Fb62b89F4916B721be55cEb828dBd73', 'nonce': 3,
'to': '0x9a3DBCa554e9f6b9257aAa24010DA8377C57c17e', 'data': '0xb8af9b4490e10040a8de0040e82b0040004a00404c4a0040bf20ff3fc920ff3fc320ff3f00
000000000000000000000000000000000000000000000000000000000000600000000000000000000000000000000000000000000000000000000000000000a000000000000
0000000000000000000000000000000000000000000000000001320000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000005302e312e31000000000000000000000000000000000000000000000000000000'}

In [44]: signed_transaction = w3.eth.account.sign_transaction(
                 register_firmware, private_key = private_key
             )
         signed_transaction

Out[44]: SignedTransaction(rawTransaction=HexBytes('0xf9014c0385174876e80083048576949a3dbca554e9f6b9257aaa24010da8377c57c17e80b8e4b8af9b4490e10040
a8de0040e82b0040004a00404c4a0040bf20ff3fc920ff3fc320ff3f00000000000000000000000000000000000000000000000000000000000006000000000000000000
000000000000000000000000000000000000000000000a0000000000000000000000000000000000000000000000000000000000000001320000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000005302e312e310000000000000000000000000
00000000000000000000000000000820a95a0a6239824867c57e6b9d2e1c81e5208798d0f1067fd5cede05b7655d3e5867ff8a062a3bf5c12e7aed3d7fe8863e9796a3664
88a1a39c05c980d0cca20ef1853d21'), hash=HexBytes('0x29262f331c12062d673fbe04e007e207d472bb4a2990082679d4c982a5947c39'), r=7514682256225525
5081389204050823676885125191843849760848683821369789354770424, s=446159759485772431570537666363415513592183336883044977163400659769636246
80737, v=2709)

In [45]: signed_transaction.rawTransaction

Out[45]: HexBytes('0xf9014c0385174876e80083048576949a3dbca554e9f6b9257aaa24010da8377c57c17e80b8e4b8af9b4490e10040a8de0040e82b0040004a00404c4a0040b
f20ff3fc920ff3fc320ff3f00000000000000000000000000000000000000000000000000000000000000600000000000000000000000000000000000000000000000000000
000000000000a00000000000000000000000000000000000000000000000000000000000000001320000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000005302e312e31000000000000000000000000000000000000000000000000000000820a
95a0a6239824867c57e6b9d2e1c81e5208798d0f1067fd5cede05b7655d3e5867ff8a062a3bf5c12e7aed3d7fe8863e9796a366488a1a39c05c980d0cca20ef1853d21')
```

Fonte: Próprio autor.

Figura 48 – Fragmento de código com informações da transação registrada na *blockchain*

```
In [49]: tx_receipt

Out[49]: AttributeDict({'blockHash': HexBytes('0x652c6af5031f0c0af1c8e367014c2818026ef5161a5c99afadfe4bf6071b3b25'),
 'blockNumber': 156,
 'contractAddress': None,
 'cumulativeGasUsed': 165755,
 'from': '0xFE3B557E8Fb62b89F4916B721be55cEb828dBd73',
 'gasUsed': 165755,
 'effectiveGasPrice': 100000000000,
 'logs': [AttributeDict({'address': '0x9a3DBCa554e9f6b9257aAa24010DA8377C57c17e',
   'topics': [HexBytes('0x4b50d218cc5b8036db20cffe18dc64086af8e5b0ac421905f78e111a387342a3')],
   'data': '0x000000000000000000000000fe3b557e8fb62b89f4916b721be55ceb828dbd73ad7c5bef027816a800da1736444fb58a807ef4c9603b7848673f7e3a68e
b14a590e10040a8de0040e82b0040004a00404c4a0040bf20ff3fc920ff3fc320ff3f00000000000000000000000000000000000000000000000000000000000000a00000
0000000000000000000000000000000000000000000000000000000e000000000000000000000000000000000000000000000000000000000000000013200000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000005302e312e31000000000000
000000000000000000000000000000000000000000',
   'blockNumber': 156,
   'transactionHash': HexBytes('0x29262f331c12062d673fbe04e007e207d472bb4a2990082679d4c982a5947c39'),
   'transactionIndex': 0,
   'blockHash': HexBytes('0x652c6af5031f0c0af1c8e367014c2818026ef5161a5c99afadfe4bf6071b3b25'),
   'logIndex': 0,
   'removed': False})],
 'logsBloom': HexBytes('0x000000000000000000000000000000000000000000000000000000000000000000200000000000000000004000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000800000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000080000000000000000000000000000000001000000000000000000000000000000000000000000000'),
 'status': 1,
 'to': '0x9a3DBCa554e9f6b9257aAa24010DA8377C57c17e',
 'transactionHash': HexBytes('0x29262f331c12062d673fbe04e007e207d472bb4a2990082679d4c982a5947c39'),
 'transactionIndex': 0})

In [48]: ##checando  quantos registros tem na Blockchain. Como é recém criada, esperamos 0 como resposta
         number = contract_instance.functions.countFragments().call() ##contando fragmentos registrados.
         print(number)

1

In [ ]: #w3.eth.get_block('latest')
        #para saber mais
        #https://web3py.readthedocs.io/en/stable/web3.eth.account.html#sign-a-transaction
```

Fonte: Próprio autor.

APÊNDICE D – CÓDIGOS DO CONTRATO INTELIGENTE

Nessão seção apresentamos o contrato inteligente e suas principais funções.

Contrato Inteligente parte 1

```
pragma solidity >=0.4.0 <0.9.0;

struct Fragment {
        string version;
        string index;
        bytes32 firmware;
        address creator;
}

contract StorageFirmware {
    (Alguns imports...)
    address private owner;
    // logs
    event LogCreateFragment(address creator, bytes32 key, bytes32 firmware,  string index,
         string version);
    event LogFragment(bytes32 key, bytes32 firmware, address creator,  string index,
        string version);
    // util
    mapping(bytes32 => Fragment) private fragment_db;
    constructor() {
        //usar para controle de acesso na plataforma
        owner = msg.sender;
    }
    function createFragmentFirmware(bytes32  _firmware, string memory _index, string memory
         _version)
    public
    returns(bytes32)
    {
        bytes32 id = keccak256(abi.encodePacked(_index));
        fragment_set.insert(id);

        Fragment storage p = fragment_db[id];
        p.firmware = _firmware;
        p.index = _index;
        p.version = _version;
        p.creator = msg.sender;
        emit LogCreateFragment(msg.sender, id , _firmware , _index,  _version);

        return id;
    }
    (...continua)
```

Contrato Inteligente parte 2

```
    (...continuando)
    function getFragmentById(bytes32 id)
    public view
    returns(string memory _index, bytes32 firmware, string memory version, address creator)
         {
        require(fragment_set.exists(id), "Can't retrive a firmware that doesn't exist.");

        Fragment storage p = fragment_db[id];

        return(p.index, p.firmware, p.version, p.creator);
    }

    function countFragments() public view returns(uint count) {
        return fragment_set.count();
    }


    function getFragmentsAtIndex(uint index) public view returns(bytes32 key) {
        return fragment_set.keyAtIndex(index);
    }


    function getFragmentAddressByFirmware(bytes32 _firmware)
    public view
    returns(bytes32 id) {
        bytes32 id = keccak256(abi.encodePacked(_firmware));
        require(fragment_set.exists(id), "Can't retrive a firmware that doesn't exist.");

        return(id);
    }
}
```

APÊNDICE E – RECORTE DE CÓDIGOS E RESPOSTAS DO TERMINAL

E.0.1 Simulação Matlab de missão espacial

Código-fonte 1 – Simulação Matlab missão do nanosatélite

```
startTime = datetime(2020,5,1,11,36,0);
stopTime = startTime + days(90);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
lat = -3.7484767;
lon = -38.5788343;
gs = groundStation(sc,lat,lon,"Name","UFC-PICI");
semiMajorAxis = 7155000;
eccentricity = 0.0001030;
inclination = 98.5;
rightAscensionOfAscendingNode = 161.5;
argumentOfPeriapsis = 0;
trueAnomaly = 263.19;
sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
        rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly,"Name","SACODE BR-1")
            ;
ac = access(sat,gs);
intvls = accessIntervals(ac);
intvls;
play(sc)
```

E.0.2 Respostas aos teste da infraestrutura blockchain

Veja a Resposta 1 ao teste *ethChainId*. O método *eth-chainId JSON-RPC* no Ethereum retorna o ID da cadeia atual, que é usado para identificar a rede Ethereum específica à qual o cliente está conectado. O *blockchain* ID é usado para evitar ataques de repetição, que ocorrem quando uma transação que foi assinada em uma rede é transmitida para outra rede e processada lá.

Resposta 1 - Teste chainId

```
// comando: curl -X POST --data '{"jsonrpc":"2.0","method":"eth_chainId","params":[],"id
    ":1}' localhost:8545
{
  (...)
  "result" : "0x539"
}
```

O valor 0x539 (1337 em decimal) é o ID da cadeia para o testnet Rinkeby, que é um *testnet* Ethereum público projetado para teste e experimento. Ele usa um mecanismo de consenso *PoA* em que as transações são confirmadas por um grupo de nós confiáveis chamados "autores", em vez de mineradores. Rinkeby foi um dos primeiros *testnet* Ethereum e é amplamente utilizado.

É mister mencionar que o chainId pode ser diferente a depender da rede à qual você está se conectando. Para ter o acesso a outros exemplos de redes, podemos acessar a própria documentação da ferramenta[1].

Resposta 2 - Teste peerCount

```
// comando: curl -X POST --data '{"jsonrpc":"2.0","method":"net_peerCount","params":[],"id
    ":1}' localhost:8545
// resposta
{
  "jsonrpc" : "2.0",
  "id" : 1,
  "result" : "0x3"
}
```

Resposta 3 - Teste Liveness

```
// url no navedador: http://localhost:8545/liveness
// resposta
{
  "status": "UP"
}
```

No contexto do Hyperledger Besu, "vivacidade"ou *readiness* refere-se à capacidade da rede de continuar processando transações e obtendo consenso. Um "teste de vivacidade"é um teste usado para determinar se uma rede é capaz de continuar processando transações e obtendo consenso. Há várias maneiras de realizar um teste de vivacidade em uma rede Hyperledger Besu, algumas delas são:

a) Enviando um grande número de transações para a rede e medindo quanto tempo leva para que elas sejam processadas e incluídas em um bloco;

b) Monitorando os blocos e transações da rede para garantir que eles sejam adicionados a uma taxa constante;

[1] https://besu.hyperledger.org/en/stable/public-networks/concepts/network-and-chain-id/

c) Medir o tempo que um nó leva para sincronizar com o restante da rede após ser reiniciado.

Em nosso contexto trabalhamos em verificar o *status* da rede durante o processo de registro de informação. A *Resposta 4* apresenta o resultado esperado e que foi encontrado durante todo o experimento.

Resposta 4 - Teste Readiness

```
// url no navedador: http://localhost:8545/readiness
// resposta
{
  "status": "UP"
}
```

É importante executar esses testes para garantir que a rede esteja funcionando sem problemas e seja capaz de lidar com diferentes tipos de falhas e casos extremos.

E.0.3 Fragmentos código fonte do contrato inteligente

Código-fonte 2 – Struct do Objeto fragmento

```
pragma solidity >=0.4.0 <0.9.0;
struct Fragment {
        string version;
        string index;
        bytes32 firmware;
        address creator;
}
```

Código-fonte 3 – Função para registrar fragmento de firmware

```
function createFragmentFirmware(bytes32 _firmware, string memory _index, string memory
    _version)
    public
    returns(bytes32)
    {
        bytes32 id = keccak256(abi.encodePacked(_index));
        fragment_set.insert(id); // Note that this will fail automatically if the key
            already exists.
        Fragment storage p = fragment_db[id];
        p.firmware = _firmware;
        p.index = _index;
        p.version = _version;
        p.creator = msg.sender;
        emit LogCreateFragment(msg.sender, id , _firmware , _index,  _version);
        return id;
    }
```

Código-fonte 4 – Outros métodos do contrato inteligente

```

function getFragmentById(bytes32 id)
    public view
    returns(string memory _index, bytes32 firmware, string memory version, address creator)
         {
        require(fragment_set.exists(id), "Can't retrive a firmware that doesn't exist.");

        Fragment storage p = fragment_db[id];

        return(p.index, p.firmware, p.version, p.creator);
    }

    function countFragments() public view returns(uint count) {
        return fragment_set.count();
    }
```

E.0.4 Códigos e respostas referentes ao firmware

o Códifo-fonte 5 traz fragmentos do firmware implementado para escrever o dado lido pela função GET na flash do ESP8266. Suplantamos apresentar o teste de conexão com o WIFI, por ser algo bastante conhecido da comunidade. O termo *payload* representa a carga de dado que chegou da requisição GET e o termo *path* é o caminho do diretório dentro da flash onde a informação está armazenada.

O Código-fonte 6 expressa o momento que é instânciado objetos do tipo *http wifiCliente* e é possível ver ainda a url usada para a realiação da requisição GET. A url é o endereço público do *localhost* que utilizamos bem como a porta 5000 usada pelo *flask*.

Código-fonte 5 – Fragmento de firmware para escrecer na memória SPIFFS

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
(...)
#include <FS.h> // Biblioteca que nos permite acessar a flash

void writeFile(String palyload, String path) {
  //Abre o arquivo para escrita ("a") para sobreescrever seria "r".
  //Sobreescreve o conte do do arquivo
  File rFile = SPIFFS.open(path,"a");
  if(!rFile){
    Serial.println("Erro ao abrir arquivo!");
  } else {

    Serial.print("gravou estado: ");
    Serial.println(payload);
    Serial.print("saindo da funcao de escrever.....");

  }
  rFile.close();
}
```

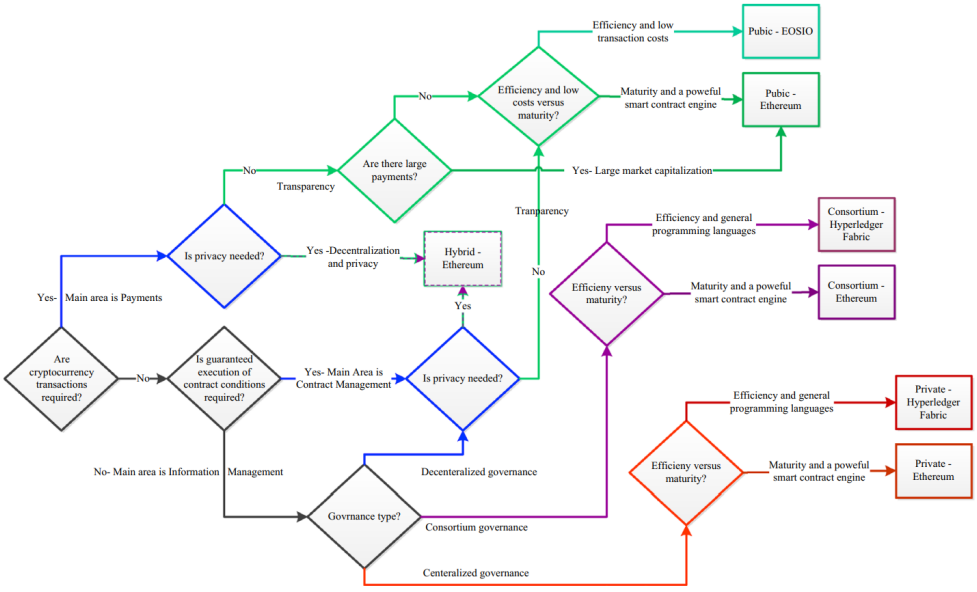
Código-fonte 6 – Fragmento de código do firmware fazendo requisições GET ao servidor

```
1 if (WiFi.status() == WL_CONNECTED) {
2     //cria a requisi  o http passando o URL da api node
3     HTTPClient http;
4     WiFiClient wifiClient;
5     http.begin(wifiClient, "http://192.168.0.105:5000/");
6     int httpCode = http.GET();
7     if (httpCode > 0) {
8         payload = http.getString();// gives us the message
9         (...)
```

ANEXO A – FLUXO DE ESCOLHA EM PROJETOS BLOCKCHAIN

A Figura 49 nos dá uma opção interessante para escolher alguns *frameworks* para desenvolver projetos em *blockchain*.

Figura 49 – Fluxo para escolha de framework de *blockchain* a ser usada em projetos



Fonte: (SONMEZ *et al.*, 2021)