Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Infraestrutura de Observabilidade para a Rede Bitcoin: Quantificando a Latência e o Overhead

David Herbert de Souza Brito

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. José Edil Guimarães Medeiros

Brasília
2025

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Infraestrutura de Observabilidade para a Rede Bitcoin: Quantificando a Latência e o Overhead

David Herbert de Souza Brito

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. José Edil Guimarães Medeiros (Orientador)
ENE/UnB

Prof. Eduardo Adilio Pelinson Alchieri
CIC/UnB

Sr. Bruno Ely Reis Garcia
Vinte Um

do Curso de Engenharia da Computação

Brasília, 08 de dezembro de 2025

Dedicatória

De início, dedico este trabalho a Deus, nosso criador, pois sem Ele nada disso seria possível. Nas maiores dificuldades, senti que Ele ouvia e atendia às minhas preces; e, nos momentos de alegria, senti Sua presença celebrando comigo.

À minha família, meu alicerce, que me deu apoio incondicional durante todos esses anos, pois uma jornada sozinho e longe de casa não é nada fácil, ainda mais quando não os temos por perto. Tudo isso foi, é e será por vocês. **Ao meu pai**, que sempre me aconselhou, ensinou-me a olhar para frente e a pensar no futuro para tomar decisões, "puxando a minha orelha" quando precisava. Sem ele, eu não teria chegado até aqui, pois, quando eu queria desistir, ele me fez enxergar o motivo de continuar (ainda bem). Obrigado, pai, por nunca ter me deixado faltar nada e por sempre acreditar que eu era capaz. Espero, um dia, retribuir tudo. **À minha mãe**, por sempre me ligar e se preocupar se eu estou comendo, dormindo ou estudando direito; por sempre me ajudar antes que eu "derrubasse a cozinha" ou tomasse algum remédio errado. Obrigado por todo o carinho e cuidado, mãe. A senhora deixou toda essa jornada mais fácil; sem você, não me sustentaria aqui. Espero, um dia, retribuir tudo. **Ao meu irmão**, que, apesar das "brigas" — que sempre existem entre irmãos —, me ajudou muito quando morou comigo, fazendo-me companhia. É muito bom saber que há alguém que se inspira em você e o admira; é por você também que continuo a melhorar. Espero, um dia, retribuir tudo.

Aos meus amigos: vocês não sabem o quanto me ajudaram. Mesmo sem precisarem dizer nada, estar com vocês era o que me fazia esquecer os problemas e descontrair. Sem as brincadeiras, as noites e as resenhas, acho que já teria enlouquecido. Além de todas as noites mal dormidas, as quais usamos para estudar; sem a ajuda de vocês, eu não teria chegado até o fim. Sou imensamente grato.

UnB, que, como dizemos, "molda caráter". Os desafios de cada semestre nos forjaram como profissionais e, graças a esta instituição, meu amadurecimento foi exponencial. É uma honra ter feito parte e me formar em uma universidade de tamanho prestígio.

Esta foi uma jornada que, por vezes, pareceu eterna, mas que agora percebo ter passado rápido, deixando muitas saudades. Foi uma fase crucial em minha vida, na qual caí, levantei-me, aprendi e cresci, tornando-me mais forte para alcançar os sonhos que almejo.

Olho para trás com um misto de emoções e a certeza de que faria tudo de novo. Obrigado!

Agradecimentos

Inicialmente, gostaria de expressar minha gratidão ao professor Edil, que expandiu minha visão em relação ao Bitcoin. Sem a sua orientação e excelente trabalho, minha entrada no universo técnico desta criptomoeda não teria sido possível. Graças ao seu incentivo, tenho o desejo de me aprofundar cada vez mais nesta área tão vasta. Obrigado, professor, pelos ensinamentos e por ter sido um dos melhores mestres que já tive.

Estendo meus agradecimentos a Satoshi Nakamoto — independentemente de sua real identidade — pela criação de um sistema de tamanha magnitude e importância para a humanidade. Acredito fielmente que este é um dos passos essenciais para atingirmos a liberdade e a estabilidade financeira mundial.

Agradeço também ao 0xB10C, desenvolvedor do *Peer Observer*. O seu monitoramento da rede Bitcoin não apenas nos inspirou a realizar esta pesquisa, como também foi fundamental, somado aos seus feedbacks e contribuições diretas.

Por fim, sou grato aos meus amigos Maycon e Lucas, que colaboraram com o projeto do *Peer Observer* conteinerizado. O auxílio de vocês facilitou imensamente a configuração do ambiente de desenvolvimento. Sem esse suporte técnico e parceria, a conclusão deste trabalho não seria possível.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Este trabalho propõe o desenvolvimento de uma infraestrutura de observabilidade baseada em microsserviços para a rede Bitcoin, desenhada especificamente para isolar e comparar as latências de comunicação nas camadas de rede e de aplicação. O mérito da proposta reside na solução do problema da opacidade na composição do atraso de propagação, oferecendo uma instrumentação granular que permite distinguir, em tempo real, gargalos de infraestrutura física de ineficiências de processamento do *software*. A validação da arquitetura foi realizada em um ambiente determinístico controlado (*regtest*) orquestrado via Docker, onde a correlação de métricas coletadas por sondas ativas P2P e ICMP comprovou a capacidade da ferramenta de quantificar o *Protocol Overhead* e fornecer diagnósticos visuais precisos sobre a saúde do nó. Como desdobramentos futuros, vislumbra-se a aplicação do sistema na *Mainnet* para análise de latência em escala global, a integração de métricas de *hardware* para inferência de causalidade e a implementação de mecanismos de alerta baseados em detecção de anomalias estatísticas.

Palavras-chave: Bitcoin, Observabilidade, Latência, Monitoramento P2P

Abstract

This work proposes the development of a microservices-based observability infrastructure for the Bitcoin network, specifically designed to isolate and compare communication latencies across network and application layers. The proposal's merit lies in resolving the opacity regarding the composition of propagation delays, offering granular instrumentation that enables real-time distinction between physical infrastructure bottlenecks and software processing inefficiencies. The architecture was validated in a controlled deterministic environment (*regtest*) orchestrated via Docker, where the correlation of metrics collected by active P2P and ICMP probes demonstrated the tool's capability to quantify *Protocol Overhead* and provide precise visual diagnostics regarding node health. Future work envisions the system's deployment on *Mainnet* for global-scale latency analysis, the integration of *hardware* metrics for causality inference, and the implementation of alerting mechanisms based on statistical anomaly detection.

Keywords: Bitcoin, Observability, Latency, P2P Monitoring.

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Contextualização

O conceito de redes descentralizadas remonta às décadas de 1960 e 1970, com o desenvolvimento da ARPANET, uma rede projetada para ser resiliente a falhas, distribuindo o controle entre múltiplos nós independentes [1]. Avanços subsequentes, como o *Hashcash* proposto por Adam Back em 1997 para *proof-of-work* [2], e ideias de moedas digitais por Wei Dai e Nick Szabo em 1998 [3, 4], pavimentaram o caminho para sistemas financeiros distribuídos. Essas inovações culminaram no Bitcoin, que integrou esses elementos em uma solução prática para transações eletrônicas sem autoridade central.

O protocolo Bitcoin foi introduzido em 2008 por Satoshi Nakamoto no *whitepaper* intitulado *Bitcoin: A Peer-to-Peer Electronic Cash System*. Sua proposta central é possibilitar que participantes desconhecidos coordenem um registro público, imutável e auditável de transações financeiras sem depender de autoridades centrais, resolvendo o problema do gasto duplo exclusivamente com mecanismos criptográficos e computacionais [5]. Para isso, o sistema combina regras de consenso distribuído, prova de trabalho (*Proof-of-Work*) e uma estrutura de dados encadeada que registra permanentemente todas as transações já realizadas — a *Blockchain*.

Mais do que um ativo digital, o Bitcoin opera essencialmente como um protocolo de mensageria ponto a ponto, no qual a informação que circula entre os nós consiste majoritariamente em transações e blocos [5]. As mensagens fundamentais deste sistema são as transações, estruturas de dados que consistem em campos como versão, entradas (*inputs*), saídas (*outputs*) e *lock time*. As entradas referenciam saídas não gastas de transações anteriores (UTXOs), representando os fundos que estão sendo gastos, enquanto as saídas especificam os valores a serem transferidos e as condições para seu gasto futuro. Essas condições são codificadas em um pequeno programa na linguagem *Script*, que define regras de validação baseadas em criptografia assimétrica. Essa semântica assemelha-se a

uma entrada contábil, onde um valor é debitado de uma ou mais fontes e creditado a novos destinatários sem intermediários. Para garantir a autenticidade, a transação é assinada pelo detentor legítimo da chave privada correspondente. Ao ser criada, a transação é transmitida em *broadcast* pela rede e, ao atingir cada nó, passa por uma validação inicial: assinaturas, formato, ausência de gasto duplo local e verificação de que suas entradas são válidas. Uma vez aprovada, é inserida no *mempool* — o conjunto de transações pendentes — onde permanece até ser incluída em um bloco minerado. A figura 1.1 demonstra o que é uma UTXO e como ela é gasta em uma transação.

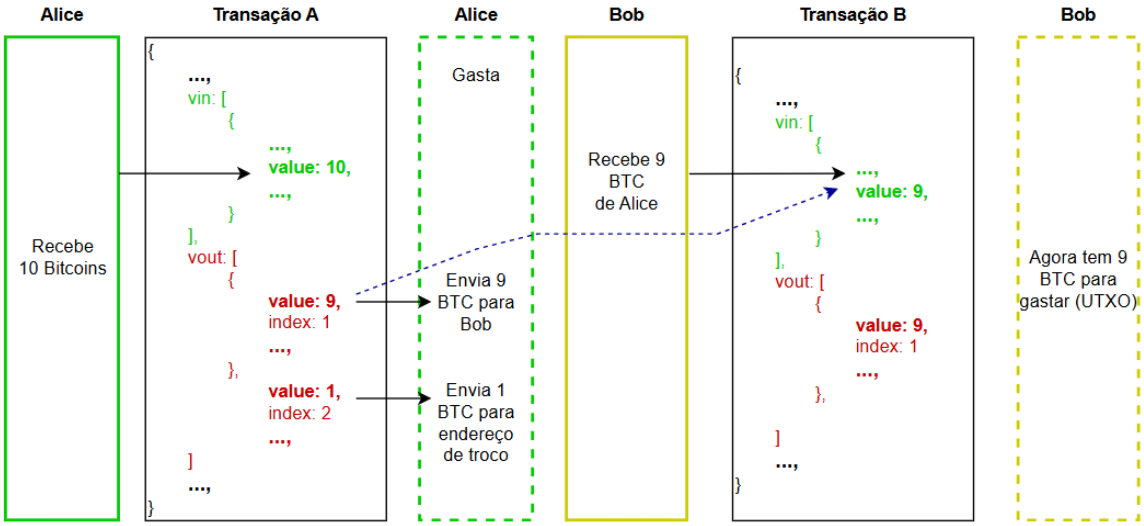

Figura 1.1: Dinâmica de entradas e saídas nas transações.

As regras de consenso do protocolo, que determinam como os participantes convergem para um único histórico global, incluem: (i) o algoritmo de prova de trabalho baseado em SHA-256, que exige esforço computacional mensurável para propor novos blocos; (ii) a emissão monetária limitada a 21 milhões de bitcoins, com *halving* da recompensa a cada 210.000 blocos; (iii) o encadeamento criptográfico dos blocos, no qual cada cabeçalho inclui o *hash* do bloco anterior, garantindo resistência a alterações retroativas; e (iv) a validação em dois estágios das transações — primeiro de forma individual por cada nó (*mempool*), depois de forma global quando mineradores selecionam e ordenam transações em blocos, resolvendo descentralizadamente o problema do gasto duplo por meio da ordenação canônica que emerge da prova de trabalho [5].

A rede Bitcoin é composta principalmente por nós completos (*full nodes*) e por *mineradores*, ambos desempenhando papéis distintos e complementares. Os *full nodes*, além de executar todas as funções ditas em parágrafos anteriores, são dispositivos que mantêm

uma cópia integral da *blockchain* [5]. Esses nós estabelecem conexões P2P e propagam informações usando mecanismos de *gossip flooding*: transações, blocos, inventários, `pings` e `pongs` são retransmitidos para que, eventualmente, todos os participantes recebam os dados relevantes [6]. Mineradores, por sua vez, são nós especializados que empregam grande capacidade computacional para resolver o *puzzle* de trabalho (*proof-of-work*). O minerador que encontra a solução primeiro propaga seu bloco para toda a rede; os nós o verificam e, se válido, o adicionam à sua cópia local da *blockchain*. Como recompensa, o minerador recebe o subsídio do bloco e taxas das transações incluídas nele.

A utilização de uma topologia P2P sem qualquer ponto central de controle confere ao Bitcoin notável resiliência e disponibilidade desde seu lançamento. Blocos são produzidos, em média, a cada dez minutos, graças ao ajuste dinâmico da dificuldade a cada 2016 blocos, o que mantém o ritmo de emissão mesmo diante de variações no poder computacional total da rede. A combinação entre prova de trabalho, regras de consenso e disseminação P2P transforma a *blockchain* em um livro-razão distribuído, transparente, resistente à adulteração e acessível a qualquer participante, sem que nenhuma entidade única possua controle sobre o sistema.

No entanto, essa arquitetura implica que a segurança e a eficiência do sistema dependem diretamente da propagação oportuna e confiável de mensagens entre os nós da rede. Atrasos na disseminação de blocos e transações impactam a competição entre mineradores, afetam o tempo de confirmação percebido pelos usuários e podem abrir espaço para ataques baseados em latência e estratégias de isolamento de nós. Assim, monitorar continuamente o comportamento da rede torna-se fundamental para entender o estado de saúde da rede, detectar anomalias e avaliar seu desempenho em um ambiente global descentralizado [6].

1.2 Importância do monitoramento da rede Bitcoin

A natureza descentralizada e aberta da rede Bitcoin impõe um desafio inerente à sua auditabilidade, uma vez que inexiste uma entidade central responsável por sua supervisão. Essa característica expõe simultaneamente o sistema a vetores de ataque específicos de redes P2P. Nesse contexto, o monitoramento independente torna-se um mecanismo de defesa crucial, permitindo a avaliação contínua da saúde do sistema e a detecção de anomalias operacionais, como atrasos na propagação de blocos ou congestionamentos que afetam a confirmação de transações.

A observabilidade da rede é a principal linha de defesa contra ameaças à segurança e à confiança dos participantes. O monitoramento contínuo viabiliza a identificação de comportamentos maliciosos, como padrões de *flooding* que visam esgotar recursos com-

putacionais. Entre os riscos mitigáveis, destacam-se o ataque *Eclipse*, que isola um nó controlando suas conexões para manipular informações [7]; o ataque *Sybil*, no qual identidades falsas são criadas para subverter a reputação ou o funcionamento da rede [8]; e ataques de Negação de Serviço (*DDoS*) direcionados a mineradores para reduzir sua competitividade [9, 6].

Além da segurança, o desempenho da rede é vital para a estabilidade do consenso. Latências elevadas na propagação de blocos aumentam drasticamente a taxa de blocos órfãos, resultando em desperdício de energia e *hash power* para mineradores honestos [10]. A relevância crítica desse monitoramento foi evidenciada historicamente no *fork* de 2013, causado por incompatibilidades entre versões do Bitcoin Core [11]. Naquela ocasião, a ausência de métricas distribuídas de rede em tempo real retardou o diagnóstico do comportamento divergente, sublinhando que atrasos amplificados podem comprometer momentaneamente a consistência da *blockchain*.

Entretanto, a relevância do monitoramento estende-se aos diversos participantes do ecossistema, e não apenas aos mineradores. Para eles, a propagação rápida de transações é um imperativo econômico para otimizar a seleção de taxas (*fees*) e manter a competitividade. Simultaneamente, operadores de nós completos dependem da observabilidade para garantir a integridade da validação e a disponibilidade do serviço [12], enquanto pesquisadores necessitam de dados granulares para compreender a topologia dinâmica da rede e propor melhorias de escalabilidade no protocolo [13].

Apesar da existência de exploradores de blocos como *Mempool.space* e ferramentas analíticas como *Chainalysis*, estas soluções tendem a oferecer visões macroscópicas e agregadas. Para diagnósticos de infraestrutura e pesquisa de resiliência, são necessárias ferramentas *open-source* de monitoramento direto e granular, como o *peer-observer*, capazes de coletar dados brutos de conexão e mensagens P2P [14].

Particularmente no contexto de desempenho, artigos como os de Neudecker et al. demonstram que a métrica de latência de rede, quando analisada isoladamente, é insuficiente para a caracterização completa do sistema. A comparação entre `pings` do protocolo Bitcoin e `pings` ICMP oferece uma visão aprofundada, permitindo a segregação entre os atrasos introduzidos pela infraestrutura física e aqueles gerados pelo processamento interno dos nós [6]. Essa distinção é fundamental para a compreensão dos gargalos operacionais e para a avaliação da real resiliência da rede frente ao estresse computacional.

1.3 Comparação entre Pings Bitcoin e ICMP

Uma abordagem comum para avaliar a conectividade e latência em redes distribuídas baseia-se fundamentalmente no envio de mensagens de sondagem (`pings`). No contexto

do Bitcoin, distinguem-se dois mecanismos de aferição que operam em camadas diferentes do modelo OSI: o `ping` via protocolo ICMP (Camada de Rede) e o `ping` via protocolo Bitcoin (Camada de Aplicação).

O ICMP (*Internet Control Message Protocol*) é um protocolo utilizado para diagnóstico e controle de erros na infraestrutura da Internet. Ferramentas de rede utilizam mensagens do tipo *Echo Request* e aguardam o retorno de um *Echo Reply* para calcular o tempo de ida e volta (*Round-Trip Time* - RTT) [15]. Por ser processado diretamente pelo *kernel* do sistema operacional, o RTT medido via ICMP reflete essencialmente o atraso de propagação, roteamento e congestionamento dos enlaces físicos, com influência mínima do *software* que está sendo executado no hospedeiro.

Em contraste, o protocolo Bitcoin implementa seu próprio mecanismo de *keep-alive* na camada de aplicação. A partir da versão 0.13 do Bitcoin Core, cada mensagem `ping` encapsula um *nonce* (identificador único aleatório de 64 bits), obrigando o nó receptor a processar o pacote e devolver uma mensagem `pong` contendo o mesmo identificador [16]. A recepção do `pong` confirma a disponibilidade do par e permite o cálculo do RTT da aplicação. Diferentemente do ICMP, essa resposta é gerada pelo *daemon* do Bitcoin em espaço de usuário (*user space*). Consequentemente, o tempo medido incorpora não apenas o trânsito na rede, mas também o atraso de processamento interno do nó — incluindo a desserialização da mensagem, verificação de filas e agendamento de CPU [17].

Pesquisas anteriores, como os trabalhos de Neudecker et al. (2016), demonstra que a comparação entre essas duas métricas é uma ferramenta analítica poderosa. Enquanto o `ping` ICMP estabelece uma linha de base da qualidade da infraestrutura, o `ping` do protocolo Bitcoin oferece uma visão da latência efetiva percebida pela aplicação [6]. A diferença aritmética entre esses valores permite isolar o "atraso do cliente", uma métrica que quantifica o *overhead* computacional introduzido pelo *software* do nó, independentemente das condições da rede subjacente.

1.4 Definição do Problema

Embora os impactos econômicos e de segurança causados pela latência na rede Bitcoin já estejam bem estabelecidos na literatura, conforme discutido na seção anterior, persiste um desafio técnico fundamental: a opacidade na composição desse atraso. O tempo total de propagação de uma mensagem em um sistema distribuído não é uma métrica monolítica; ele é a soma vetorial do atraso de transmissão da rede física e do tempo de processamento da aplicação (*software* Bitcoin Core).

O problema central abordado neste trabalho reside na incapacidade das ferramentas atuais em distinguir essas duas componentes. Estudos como os de Neudecker et al.

(2016) indicam que variações na implementação do cliente, concorrência de processos ou sobrecarga de *hardware* podem introduzir atrasos de processamento significativos, que, na ausência de instrumentação adequada, são frequentemente e erroneamente diagnosticados como problemas de rede [6].

Identifica-se, portanto, uma lacuna crítica de observabilidade. As soluções de monitoramento existentes concentram-se predominantemente em métricas macroscópicas (como taxa de *hash* global e tamanho da *blockchain*) ou em métricas de rede puras, ignorando o comportamento interno do nó. Falta aos operadores e pesquisadores uma ferramenta acessível que permita segregar, em tempo real e de forma comparativa, a latência imposta pela infraestrutura de transporte (ICMP) daquela imposta pelo protocolo da aplicação (P2P). Sem essa distinção, torna-se inviável diagnosticar com precisão a causa raiz de gargalos de desempenho, comprometendo a eficiência e a manutenção da infraestrutura da rede [17, 6].

1.5 Estratégia e Proposta

O objetivo primordial deste trabalho é construir uma infraestrutura de observabilidade capaz de isolar, quantificar e comparar o custo computacional (*overhead*) intrínseco ao protocolo Bitcoin em relação à latência física da rede, sendo o *overhead* calculado usando a latência de ambos. A concepção desta abordagem inspira-se na "monitoring wishlist" de 0xB10C (item *ping-pong*), que reconhece a relevância persistente do fenômeno e propõe a sua contabilização sistemática como uma métrica valiosa para a saúde da rede [18]. Para materializar essa proposta e garantir que os resultados sejam tecnicamente aferíveis, a estratégia de desenvolvimento foi estruturada em quatro pilares fundamentais, implementados através de uma arquitetura de microsserviços orquestrada via *Docker*:

1. **Estabelecimento de um Ambiente Determinístico:** Para eliminar ruídos externos e garantir a reprodutibilidade dos testes, isolou-se o objeto de estudo — o *software* Bitcoin Core — em um contêiner Docker operando na rede *regtest*. Esta escolha assegura que qualquer latência medida seja proveniente do sistema local e não de flutuações da internet pública.

2. **Extração Ativa da Latência de Aplicação:** Para mensurar com precisão o tempo de processamento do protocolo, utilizou-se o *P2P Extractor* do *Peer Observer*. Este componente opera injetando periodicamente mensagens `ping` e registrando o tempo de resposta (`pong`). A arquitetura emprega serialização eficiente via *Protobuf* para publicar os eventos no sistema de mensageria NATS, permitindo que o componente `metrics` os consuma assincronamente e os exponha no formato padro-

nizado do Prometheus. Essa estratégia de desacoplamento é crucial para garantir que a instrumentação ocorra em alta *performance*, minimizando o impacto da coleta sobre o desempenho do nó monitorado.

3. **Definição da Linha de Base de Infraestrutura:** Para estabelecer um referencial comparativo, integrou-se o *Blackbox Exporter* ao ecossistema de monitoramento. Esta ferramenta executa sondagens ICMP (`pings` de rede) simultâneas contra o mesmo contêiner do nó Bitcoin. Ao capturar a resposta diretamente do *kernel* do sistema operacional, o exportador fornece a métrica de latência de transporte pura, isolada do processamento da aplicação, expondo-a no formato padronizado para coleta pelo Prometheus.

4. **Consolidação e Análise Comparativa:** A transformação de eventos discretos em inteligência visual ocorre através da centralização dos dados no Prometheus, que extrai métricas tanto do componente `metrics` quanto do *Blackbox Exporter*. O processo culmina na construção de *dashboards* no Grafana. Esta etapa não se limita a plotar as séries temporais de latência ($RTT_{Bitcoin}$ e RTT_{ICMP}) no mesmo eixo; ela também processa e exibe indicadores de média de latência para cada camada. Estes valores médios consolidados servem como base de cálculo para as métricas derivadas de eficiência — especificamente o *Protocol Overhead* (Sobrecarga do Protocolo) médio, definido como a diferença aritmética entre as médias de latência da aplicação e da rede — e para a distribuição percentual de latência (*Breakdown*), que oferece uma visão holística do custo computacional do protocolo.

Em termos de implementação, esta proposta estende o conceito original do *Peer Observer* ao introduzir o monitoramento comparativo de ICMP. A orquestração destes componentes cria um ecossistema onde a latência de aplicação (Bitcoin) e a latência de rede (ICMP) estão disponíveis simultaneamente, prontas para a análise comparativa que fundamenta este trabalho.

1.6 Escopo

Convém delimitar claramente o escopo deste trabalho. O foco reside primordialmente na engenharia da infraestrutura de observabilidade e na validação da metodologia de comparação entre as latências de aplicação e de rede. Não serão realizadas análises estatísticas inferenciais complexas sobre o comportamento global da rede Bitcoin, nem modelagens dos efeitos econômicos destes atrasos sobre as receitas de mineradores ou a segurança do consenso. Em particular, não se busca definir qual seria o atraso "normal" da rede, mas sim fornecer o instrumental técnico para que tal atraso possa ser mensurado e decomposto.

Excluem-se também do escopo otimizações de *performance* do *software*, simulações em larga escala ou integrações com ferramentas proprietárias como *Chainalysis*.

Em vez disso, o trabalho concentra-se na entrega de uma plataforma funcional capaz de processar os dados coletados e gerar métricas derivadas essenciais, especificamente o *Protocol Overhead* e o perfil de distribuição de latência. Assim, a contribuição principal é a definição e implementação de um ambiente de monitoramento que não apenas intercepta as latências de `pings` do protocolo Bitcoin e ICMP, mas as expõe de forma síncrona e comparável, estabelecendo a base empírica para que operadores e pesquisadores possam realizar diagnósticos precisos sobre a eficiência de nós na rede.

Capítulo 2

Arquitetura de Monitoramento

Este capítulo fundamenta a arquitetura de monitoramento proposta, descrevendo os componentes tecnológicos essenciais para a sua implementação. A estrutura do texto reflete o ciclo de vida da informação no sistema, iniciando-se pela caracterização do nó Bitcoin e da ferramenta de instrumentação *Peer Observer*.

Na sequência, abordam-se os mecanismos de transporte e serialização (NATS e *Protobuf*) que garantem o desacoplamento e a eficiência da comunicação. Por fim, detalha-se a pilha de observabilidade — composta pelo módulo `metrics`, Prometheus e Grafana — responsável por transformar os eventos brutos capturados em métricas estruturadas e visualizações analíticas.

2.1 Nó do Bitcoin

Um nó Bitcoin é uma instância de *software* que participa da rede ponto a ponto (P2P), executando as regras definidas pelo protocolo. Seu papel central é validar transações e blocos, manter uma cópia atualizada da *blockchain* e retransmitir mensagens para seus pares, contribuindo para o consenso distribuído sem depender de qualquer autoridade central [5]. Cada nó opera de forma autônoma, verificando assinaturas digitais, assegurando a inexistência de gasto duplo e garantindo que todas as transações e blocos recebidos obedeçam integralmente às regras do protocolo.

Do ponto de vista operacional, um nó mantém três componentes principais: o banco de dados da *blockchain*, o *mempool* (que armazena transações ainda não mineradas) e a camada de comunicação P2P. A rede utiliza conexões TCP entre pares e um conjunto padronizado de mensagens, como `version`/`verack` para *handshake*, `inv` e `getdata` para anúncio e solicitação de dados, e `block` e `tx` para propagação de blocos e transações. Essa estrutura de comunicação garante que cada participante contribua simultaneamente para a disseminação das informações e para a verificação coletiva do estado da rede.

2.2 Peer Observer

O *Peer Observer* é uma ferramenta de monitoramento e observabilidade *open-source* desenvolvida em Rust, projetada para auditar eventos internos e de rede do Bitcoin Core, a implementação de referência do protocolo Bitcoin. Sua arquitetura adota um modelo modular e desacoplado de produtores e consumidores, interconectados por um barramento de mensagens.

O funcionamento do sistema baseia-se na captura de eventos em diferentes camadas da aplicação — desde chamadas de sistema de baixo nível até interações de rede de alto nível. Esses eventos são processados e normalizados para um formato comum antes de serem distribuídos para análise. A Figura 2.1 ilustra o fluxo de dados da arquitetura, que se inicia na extração junto ao nó e culmina na visualização e análise pelas ferramentas consumidoras. Os tópicos subsequentes abordarão detalhadamente cada passo.

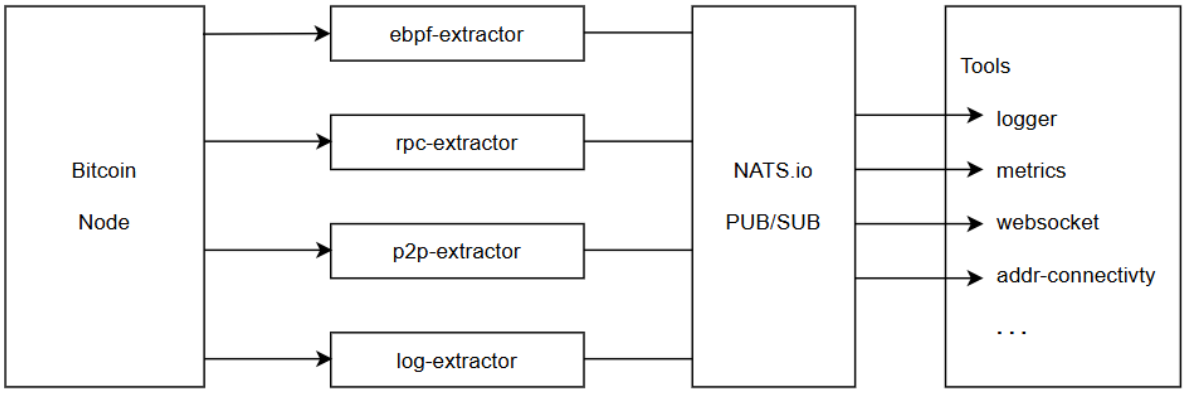

Figura 2.1: Diagrama da arquitetura do Peer Observer (Fonte: [14]).

2.2.1 Extatores

A camada de coleta é composta por módulos denominados **extratores**, que operam como agentes produtores de dados. Cada extrator especializa-se em uma interface ou mecanismo de introspecção específico do Bitcoin Core, serializando os dados capturados no formato binário estruturado *Protocol Buffers* (Protobuf). Posteriormente, essas mensagens são publicadas em tópicos específicos em um servidor NATS, um *middleware* de mensageria leve e de alto desempenho.

Os principais extratores implementados e suas metodologias de captura são:

- **eBPF-extractor**: Este módulo utiliza a tecnologia eBPF (*Extended Berkeley Packet Filter*) para instrumentação de baixo nível e alta performance. Ele se acopla aos

tracepoints USDT (*User Statically-Defined Tracing*) inseridos no código do Bitcoin Core, utilizando a biblioteca `libbpf-rs`. Isso permite a interceptação de eventos críticos — como o recebimento e envio de mensagens P2P, abertura e fechamento de conexões e alterações no *mempool* — diretamente no espaço de usuário, com sobrecarga mínima para o sistema hospedeiro. Devido à sua natureza, requer privilégios elevados no sistema e suporte a partir da versão 29.0 do Bitcoin Core.

- **RPC-extractor**: Complementando os dados em tempo real, este extrator foca em dados de estado (*stateful data*). Utilizando a biblioteca `corepc`, ele realiza consultas periódicas (polling) à interface RPC (*Remote Procedure Call*) do nó. Sua função principal é obter snapshots do estado interno que não são facilmente capturados por eventos discretos, como a lista completa de pares conectados (`getpeerinfo`) e estatísticas consolidadas do *mempool*.

- **P2P-extractor**: Diferentemente dos anteriores que observam o nó passivamente, este extrator atua como um participante ativo da rede (um *honeynode* ou par simulado). Ele estabelece conexões via protocolo Bitcoin P2P na interface local (*localhost*) para minimizar a latência de rede externa e realiza o *handshake* completo. Sua função primária é a metrologia de rede, enviando mensagens `ping` periódicas e mensurando o tempo de resposta (`pong`) para detecção de anomalias de latência e disponibilidade.

- **Log-extractor**: Atua na camada de persistência de texto, monitorando e analisando o arquivo *debug.log* do Bitcoin Core em tempo real. Ele extrai informações estruturadas de eventos que podem não estar cobertos pelos *tracepoints* ou pela API RPC, servindo como uma fonte redundante ou complementar de auditoria.

2.2.2 Protobuf

Protocol Buffers é um mecanismo de serialização de dados estruturados desenvolvido pelo Google. Serve para codificar informações de forma compacta e eficiente, independentemente de plataforma e linguagem. Em vez de utilizar formatos textuais mais verbosos, como JSON ou XML, o *Protobuf* emprega uma representação binária otimizada. Segundo a documentação oficial, o *Protobuf* proporciona mensagens significativamente menores e com maior velocidade de processamento, além de oferecer geração automática de código para diversas linguagens de programação [19].

Na prática, isso significa que mensagens *Protobuf* ocupam menos espaço na rede e exigem menor tempo para serialização e desserialização em comparação com formatos textuais tradicionais. Por exemplo, ao serializar um campo numérico ou textual, o *Pro-*

tobuf omite nomes de campos e aplica técnicas como codificação *varint* para inteiros, resultando em *payloads* compactos e mais eficientes [19].

Do ponto de vista técnico, o *Protobuf* exige a definição prévia de esquemas em arquivos `.proto`, nos quais cada mensagem e seus respectivos campos tipados (inteiros, *strings*, mensagens aninhadas, etc.) são declarados. Um compilador específico gera código-fonte em diferentes linguagens de programação a partir desses esquemas. Durante a execução, os dados estruturados são serializados para *bytes* conforme o esquema definido. Esse processo garante interoperabilidade entre módulos, como extratores e ferramentas, mesmo quando implementados em linguagens distintas. Além disso, o *Protobuf* preserva compatibilidade entre versões: novos campos podem ser adicionados sem comprometer consumidores mais antigos, já que a identificação dos dados é feita por números de campo, e não por nomes textuais.

2.2.3 NATS

O NATS é um sistema de mensageria *open-source* projetado para comunicação distribuída de alta *performance*, oferecendo baixa latência, simplicidade operacional e forte resiliência [20]. Atuando como um *middleware* orientado a mensagens, permite que componentes de sistemas distribuídos troquem dados de forma assíncrona e desacoplada. O sistema baseia-se na troca de mensagens discretas entre aplicações cliente e servidores. Os servidores podem operar isoladamente ou formar *clusters* e *superclusters* geograficamente distribuídos, garantindo coerência global e escalabilidade. As conexões são estabelecidas via URL, suportando protocolos como TCP, TLS e *WebSocket*, com diversos mecanismos de autenticação.

O paradigma central de comunicação é o modelo *publish-subscribe*. Nele, *publishers* enviam mensagens para canais lógicos denominados *subjects* — identificadores textuais hierárquicos compostos por *tokens* separados por pontos (ex: `peer.latency.ping`). Os *subscribers* registram interesse nesses padrões e recebem as mensagens correspondentes. Diferentemente de outros sistemas, os *subjects* não dependem da topologia da rede nem exigem configuração prévia de filas ou *brokers*, tornando o roteamento dinâmico e de baixa complexidade.

A flexibilidade do endereçamento é ampliada pelo suporte a curingas (*wildcards*): o caractere `*` corresponde a um único nível hierárquico, enquanto o `>` corresponde a um ou mais níveis subsequentes (ex: `sensor.>` abrange `sensor.temp` e `sensor.pressure.room1`). Essa expressividade permite assinaturas granulares ou categóricas conforme a necessidade da aplicação.

Em seu modo *core*, o NATS opera com semântica *at-most-once*, adotando o modelo *fire-and-forget*, no qual mensagens são entregues apenas aos *subscribers* ativos no mo-

mento, sem retenção, priorizando a velocidade. Para cenários que exigem persistência e garantia de entrega, o sistema oferece o *JetStream*, uma extensão que suporta garantias *at-least-once* ou *exactly-once*, confirmação de mensagens, *replay*, *streams* e *key-value stores*. Entre suas vantagens, destacam-se o binário compacto (um servidor pode ter menos de 20 MB), eficiência de recursos e facilidade de escalabilidade horizontal.

2.2.4 Consumidores

Uma vez que os eventos são publicados no NATS, as aplicações independentes, classificadas como **ferramentas** (*tools*), atuam como consumidores de dados. Essas ferramentas se inscrevem em tópicos específicos do NATS, desserializam as mensagens e executam processamentos personalizados. As principais ferramentas incluem:

- **Ferramenta `metrics`**: É o componente central para observabilidade quantitativa. Ela consome o fluxo de eventos serializados, agrega os dados em memória e os expõe como métricas de séries temporais compatíveis com o padrão Prometheus. A análise detalhada da implementação deste componente será aprofundada em uma seção subsequente.
- **Ferramenta `logger`**: Funciona como um depurador do fluxo de dados, imprimindo-as de forma legível no `stdout`. Ela permite a aplicação de filtros por tipo de mensagem, sendo essencial para verificação de integridade do *pipeline* e diagnóstico rápido.
- **Ferramenta `websocket`**: Atua como uma ponte para interfaces web, convertendo os eventos binários do NATS para o formato JSON e publicando-os via protocolo *WebSocket*. Isso facilita a integração com *dashboards* de visualização em tempo real que não suportam nativamente o protocolo NATS ou Protobuf.
- **Ferramenta `connectivity-check`**: Focada na análise de topologia, esta ferramenta extrai os endereços IP contidos nas mensagens de anúncio de pares (como mensagens `addr` ou `addrv2`) capturadas pelos extratores e executa testes ativos de alcançabilidade. Isso permite verificar se os nós anunciados na rede P2P são válidos e acessíveis publicamente.

2.2.5 Ferramenta Metrics do Peer Observer

A ferramenta `metrics` do *Peer Observer* é responsável por coletar, processar e expor métricas da rede Bitcoin P2P para sistemas de monitoramento como o Prometheus [14]. Sua função principal consiste em consumir eventos exibidos pelo NATS — tais como mensagens de `ping`, `pong`, conexões, anúncios de transações ou blocos — e transformá-los em métricas estruturadas e quantificáveis.

O funcionamento da `metrics` baseia-se no modelo de mensageria assíncrona: os eventos capturados pelos *extractors* (extratores) nos *nodes* são serializados e publicados em um sistema de mensagens, como o NATS. A ferramenta `metrics` atua como assinante (*subscriber*) desses tópicos, escutando continuamente os eventos pertinentes, correlacionando dados como *timestamp* de envio e recebimento para calcular latências (por exemplo, RTT de `pings`) e agregando essas informações segundo um modelo dimensional de métricas.

Uma vez processados, os valores são organizados em tipos de métricas compatíveis com o Prometheus: histogramas para latência, contadores para número de eventos de conexão ou desconexão, *gauges* para estado atual de *peers*, entre outros. Em seguida, a ferramenta expõe essas métricas por meio de um *endpoint* HTTP no formato de exposição padrão do Prometheus (`/metrics`), de modo que o servidor Prometheus pode coletá-las periodicamente para ingestão em sua base de séries temporais.

Além disso, a `metrics` suporta a definição de *labels* (rótulos) que permitem diferenciar métricas segundo variáveis relevantes, como a identidade do *node*, o identificador do *peer* ou o tipo de evento. Isso possibilita consultas refinadas no Prometheus, como o acompanhamento de latência por *peer* específico ou a comparação de comportamento entre diferentes *nodes*.

A arquitetura da `metrics` é projetada para ser leve e resiliente, evitando bloqueios no *pipeline* de coleta: mesmo que a publicação de eventos seja intensa, o uso de NATS e *Protobuf* garante que o processamento de métricas permaneça eficiente. Também é possível configurar a ferramenta para lidar com falhas, por exemplo, monitorando o número de eventos perdidos ou acumulando métricas internamente até a reconexão com o barramento de mensageria.

Em síntese, a ferramenta `metrics` traduz dados brutos de atividades P2P da rede Bitcoin em métricas observáveis e estruturadas, habilitando monitoração contínua e análise histórica por meio de sistemas como o Prometheus. Esse componente é essencial para permitir visibilidade sobre latências, padrões de propagação e comportamento de nós, contribuindo tanto para a operação quanto para a pesquisa de segurança na rede Bitcoin.

2.3 Prometheus

O Prometheus é um sistema *open-source* projetado para fornecer funcionalidades de monitoramento e armazenamento de métricas como dados de séries temporais, registrando informações com um *timestamp*. Em vez de receber dados passivamente, o Prometheus coleta (*scraping*) métricas ativamente de *endpoints* configurados via HTTP. Ou seja, cada exportador (como o `metrics` do *Peer Observer*) expõe um *endpoint* que o servidor Prometheus consulta periodicamente [21]. Essa arquitetura *pull* significa que é o Prometheus

quem solicita os dados em intervalos regulares, simplificando o modelo de descoberta de alvos e evitando a dependência de conexões de entrada nos nós monitorados [21].

Conforme as especificações oficiais, o Prometheus armazena dados como séries temporais multidimensionais. Cada métrica coletada contém um nome e pares chave-valor chamados *labels* (dimensões) [21]. Por exemplo, a métrica de latência de `ping` pode ter *labels* indicando o `peer_id` ou `instance`, permitindo consultas sofisticadas (como filtrar latência por nó, por dia, etc.). O modelo de dados flexível e a linguagem de consulta PromQL tornam o Prometheus poderoso para análise histórica e alertas. Além disso, o Prometheus não depende de armazenamento distribuído; cada servidor é autônomo, o que reforça a confiabilidade na coleta contínua mesmo que outros componentes falhem [21].

2.4 Grafana

O Grafana é uma plataforma multiprotocolo de código aberto (*open-source*) especializada em observabilidade, análise e visualização de dados métricos. Diferentemente de sistemas de armazenamento, o Grafana adota uma arquitetura agnóstica em relação à persistência dos dados: ele atua como uma camada de apresentação unificada que se conecta a diversas fontes de dados externas, denominadas *datasources*, sem exigir a migração ou duplicação das informações [22].

Sua estrutura interna divide-se em um *backend* (escrito em Go), responsável pelo gerenciamento de conexões, autenticação e roteamento de consultas (*proxying*), e um *frontend* interativo, que renderiza as visualizações no navegador do usuário. Esta arquitetura permite que o Grafana execute consultas complexas diretamente nas bases de dados originais — como o Prometheus — e transforme os resultados brutos em *insights* visuais através de um motor de processamento de dados em tempo real [22].

Em contextos de monitoramento de infraestrutura distribuída, a função do Grafana transcende a simples plotagem de gráficos. Ele permite a composição de painéis (*dashboards*) dinâmicos que correlacionam métricas de diferentes origens no mesmo eixo temporal. Além disso, a ferramenta suporta a aplicação de expressões matemáticas sobre os vetores de dados retornados pelas consultas. Isso possibilita, por exemplo, calcular métricas derivadas — como a diferença aritmética entre a latência de rede e a latência de aplicação — no momento da visualização, sem a necessidade de armazenar esses valores intermediários no banco de dados.

Capítulo 3

Implementação da Infraestrutura de Observabilidade

Este capítulo detalha a implementação prática da infraestrutura de observabilidade proposta, descrevendo o processo de engenharia realizado para isolar e quantificar as latências de rede e de aplicação. A metodologia de desenvolvimento fundamenta-se na orquestração de microsserviços via *Docker*, uma abordagem escolhida para garantir o isolamento de recursos e a reprodutibilidade do experimento em um ambiente controlado.

A exposição estrutura-se seguindo o ciclo de vida do dado na arquitetura: inicia-se pela definição da topologia de rede e configuração do nó Bitcoin (objeto de estudo), avança para os mecanismos de extração e transporte de eventos (sondas P2P e ICMP), e encerra-se com as etapas de agregação, persistência e visualização das métricas.

3.1 Arquitetura dos Fluxos de Monitoramento

Para concretizar a estratégia definida na Seção 1.5, a solução foi estruturada como um conjunto de microsserviços orquestrados, onde cada componente opera como uma entidade autônoma dentro de uma rede virtual privada. A containerização via Docker, além de facilitar a implantação, desempenha um papel crítico no design experimental ao garantir o isolamento de recursos através de *Linux Namespaces*. Esse mecanismo assegura que a execução das ferramentas de monitoramento (como o *Blackbox Exporter*) não concorra deslealmente por CPU e memória com o nó monitorado, minimizando o "efeito do observador"e preservando a integridade das medições.

A comunicação entre os serviços é viabilizada por uma topologia de rede *bridge* interna. Diferentemente da rede *host*, esta configuração isola o tráfego de monitoramento da interface pública da máquina hospedeira e provê um serviço de descoberta automática através do servidor DNS embarcado do Docker. Tal funcionalidade permite que os componentes

localizem uns aos outros por seus nomes de serviço (ex: `bitcoin-node`, `metrics`), abstraindo a complexidade do endereçamento IP dinâmico, o que é crucial para a estabilidade dos alvos de coleta do Prometheus.

3.1.1 Pipeline de Monitoramento de Aplicação (P2P)

O monitoramento da latência intrínseca ao protocolo Bitcoin ocorre através de um fluxo de eventos orientado a mensagens, desenhado para capturar o tempo exato de processamento das mensagens de controle da rede. Este pipeline é composto por três estágios principais:

1. **Interceptação e Extração:** O contêiner denominado `bitcoin-node` executa a implementação de referência *Bitcoin Core* (`bitcoind`), configurada para operar na rede de testes local (*regtest*). Coabitando este ambiente, o processo p2p-extractor [14] estabelece uma conexão TCP persistente com a interface local do nó. Diferente de um *sniffer* passivo (programa que escuta o tráfego de dados em uma rede sem interferir nele, simplesmente capturando pacotes de dados que passam por ele), este extrator participa ativamente do protocolo: ele injeta mensagens `ping` contendo um *nonce* (identificador único) e aguarda a resposta `pong` correspondente. O cálculo da latência é realizado pela diferença $t_{pong} - t_{ping}$, capturando todo o ciclo de *handshake*, serialização e desserialização da mensagem pelo nó.

2. **Transporte Assíncrono (NATS):** Para evitar que operações de I/O (entrada e saída) bloqueantes no extrator afetem a precisão das medições de tempo, utiliza-se o NATS como sistema de mensageria. O extrator publica os eventos de latência no tópico apropriado imediatamente após o cálculo. O NATS atua como um *buffer* de alto desempenho, desacoplando a produção do dado de seu consumo e armazenamento.

3. **Normalização de Métricas:** O serviço `metrics` consome do NATS e transforma os eventos brutos recebidos em séries temporais compatíveis com o padrão OpenMetrics. Especificamente, ele expõe a métrica `peerobserver_p2pextractor_ping_duration_nanoseconds`

   mantendo o valor em nanossegundos para preservar a precisão original da captura antes da ingestão pelo banco de dados.

A Figura 3.1 apresenta o diagrama de blocos da solução, detalhando o pipeline de dados que compõem o essa parte do sistema.

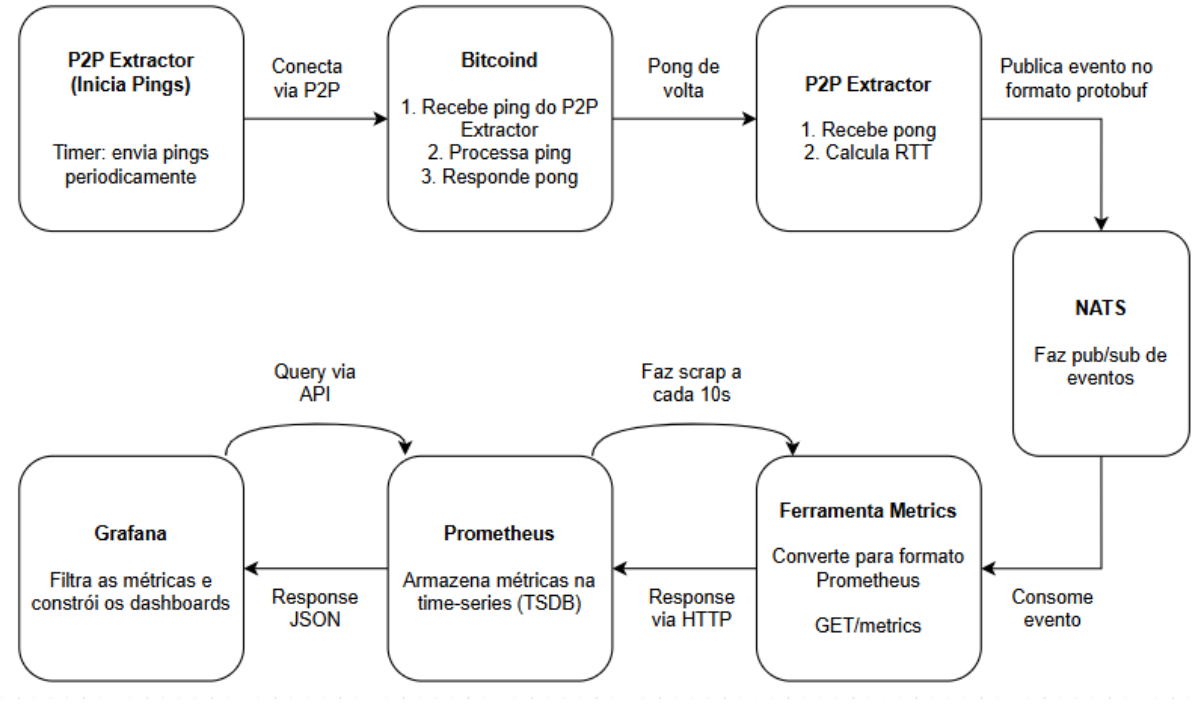

Figura 3.1: Diagrama da arquitetura do Pipeline da aplicação (Fonte: [23]).

3.1.2 Pipeline de Monitoramento de Rede (ICMP)

Paralelamente ao fluxo de aplicação, o sistema executa uma sondagem ativa de rede para estabelecer uma linha de base (*baseline*) de latência de infraestrutura. Este processo é gerenciado pelo **Blackbox Exporter** [24], um componente que permite sondar terminais através de protocolos como HTTP, HTTPS, DNS, TCP e, crucialmente para este trabalho, ICMP.

O fluxo de execução deste pipeline difere do anterior por ser iniciado externamente (modelo *pull*):

- O Prometheus é configurado com um *job* específico (`ping_peers_ICMP`) que instrui o Blackbox Exporter a sondar o alvo `bitcoin-node`.
- Ao receber a solicitação, o Blackbox resolve o nome de domínio `bitcoin-node` para o IP interno do contêiner (ex: `172.18.0.5`) utilizando o DNS do Docker.
- Um pacote *ICMP Echo Request* é enviado, e o exportador mede as fases distintas da operação: `setup`, `resolve` (tempo de DNS) e `RTT` (*Round-Trip Time*).
- O resultado é retornado ao Prometheus na métrica `probe_icmp_duration_seconds`, isolando o componente `RTT`.

Esta separação metodológica é vital: enquanto o P2P Extractor mede a "rede + aplicação", o Blackbox Exporter mede estritamente a "rede". A diferença aritmética entre os dois valores revela o custo computacional do protocolo Bitcoin.

A Figura 3.2 apresenta o diagrama de blocos da solução, detalhando o pipeline de dados que compõem o essa parte do sistema.

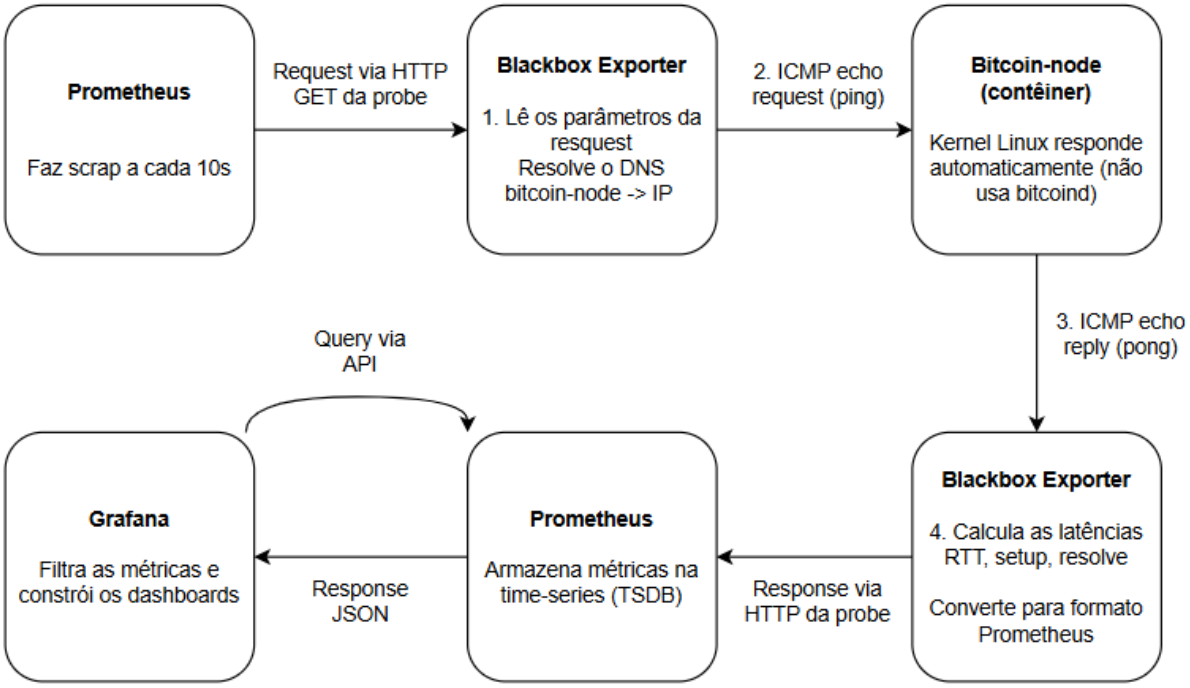

Figura 3.2: Diagrama da arquitetura do Pipeline da rede ICMP (Fonte: [23]).

3.1.3 Agregação e Camada de Visualização

A convergência dos dados ocorre no Prometheus, configurado como um banco de dados de séries temporais (*Time Series Database* - TSDB). O servidor realiza a coleta (*scraping*) dos dados de ambos os pipelines em intervalos sincronizados de 10 segundos, como pode ser observado nas Figuras 3.1 e 3.2. Esta frequência foi definida para oferecer granularidade suficiente para a detecção de picos de latência, sem sobrecarregar o armazenamento ou a CPU dos contêineres.

A camada final de apresentação é provida pelo Grafana, que consulta o Prometheus para gerar visualizações em tempo real. No painel de controle desenvolvido, consultas em *PromQL* (Linguagem de Consulta do Prometheus) são aplicadas para normalizar as unidades de medida — convertendo os nanossegundos do extrator P2P e os segundos do ICMP para uma unidade comum (milissegundos). O resultado consiste na plotagem de duas linhas sobrepostas em um mesmo plano cartesiano, o que permite uma análise

visual imediata tanto da correlação quanto das discrepâncias entre a latência de rede e a latência da aplicação Bitcoin. Além disso, foram produzidas plotagens adicionais a partir do refinamento dos dados brutos obtidos, as quais estão descritas no item "*Consolidação e Análise Comparativa*" da Seção 1.5.

3.2 Configuração e Operação do Nó Bitcoin

O nó Bitcoin constitui o objeto central de estudo deste trabalho, atuando como o elemento passivo sobre o qual as métricas de latência são aferidas. Nesta arquitetura, o nó não desempenha o papel de agente monitorador, mas sim de alvo: ele recebe as sondas de verificação (mensagens do protocolo), processa-as internamente e devolve as respostas, permitindo a mensuração externa do tempo de resposta da aplicação.

As especificações de construção e inicialização do contêiner Docker que encapsula o nó serão detalhadas a seguir.

3.2.1 Especificação da Imagem e Ambiente de Execução

Diferentemente de implementações que utilizam imagens genéricas, este projeto emprega uma imagem Docker construída especificamente para fins de observabilidade avançada. A base do sistema é o *Ubuntu*, sobre a qual foi compilado manualmente o `Bitcoin Core v29.0`.

O processo de compilação do binário `bitcoind` incluiu a habilitação do suporte a *User Statically-Defined Tracing* (USDT). Essa configuração é fundamental para conferir versatilidade ao ambiente: embora este trabalho priorize a extração via protocolo P2P, a presença dos rastreadores USDT e das ferramentas de análise de *kernel* (como `bcc-tools` e `bpftrace`) assegura que a imagem suporte nativamente o `ebpf-exporter` para auditorias de baixo nível, assim como os outros extractors que não possuem exigência de kernel.

Além disso, a imagem adota uma estratégia de empacotamento monolítico para a suíte de monitoramento, integrando todos os binários do ecossistema *peer-observer*. Isso inclui não apenas o `p2p-extractor` e o `ebpf-exporter`, mas também o `rpc-exporter` (para coleta de métricas via interface de comando) e o `log-extractor` (para análise de eventos de registro). Essa abordagem garante que o ambiente de execução seja agnóstico quanto ao método de coleta, permitindo a alternância entre diferentes estratégias de monitoramento sem a necessidade de reconstrução da infraestrutura.

3.2.2 Definição da Rede: Regression Test Mode (Regtest)

O ambiente de rede selecionado para os experimentos foi o Regtest (*Regression Test Mode*). Esta escolha é definida através de variáveis de ambiente passadas ao contêiner durante a orquestração e processadas pelo *script* de inicialização, que será detalhado adiante.

A utilização da *Regtest* justifica-se pela necessidade de um ambiente estritamente determinístico. Ao criar uma *blockchain* local e privada, isolada da internet pública, elimina-se a variabilidade estocástica introduzida pela propagação de blocos e transações de terceiros na *Mainnet* ou *Testnet*. Isso permite que a latência medida reflita exclusivamente o desempenho do nó e da infraestrutura containerizada, sem ruídos externos.

3.2.3 Orquestração e Inicialização do Serviço

A lógica de operação do nó não depende de um arquivo de configuração estático (`bitcoin.conf`). Em vez disso, o comportamento do serviço é governado por um *script* de ponto de entrada (*entrypoint*) que orquestra a inicialização dos processos com base em argumentos de linha de comando. O código integral do script pode ser verificado na referência [23].

No cenário de monitoramento via P2P, o fluxo de inicialização segue uma ordem de dependência estrita para garantir o estabelecimento correto da topologia de monitoramento:

1. **Inicialização do Extrator:** O processo `p2p-extractor` é iniciado primeiramente em modo de escuta, aguardando conexões em uma interface de rede local. Abaixo, apresenta-se o pseudocódigo simplificado do comando para que se execute o extrator [23]:

```
/usr/local/bin/p2p-extractor \
--nats-address HOST:PORTA_NATS \
--p2p-network regtest
--p2p-address HOST:PORTA_P2P_EXTRACTOR
```

2. **Execução do Nó:** O `bitcoind` é iniciado subsequentemente, configurado através do argumento `-addnode` para estabelecer uma conexão persistente com o extrator local. O pseudocódigo a seguir demonstra os argumentos essenciais utilizados para instruir o nó [23].

```
/usr/sbin/runuser -u bitcoin -- bitcoind \
-regtest \
```

```
-addnode HOST:PORTA_EXTRATOR
```

A orquestração precisa destes comandos é determinante para a integridade da arquitetura. O `p2p-extractor` é configurado através de três parâmetros essenciais: `-nats-address`, que define o destino de publicação dos eventos no barramento de mensagens; `-p2p-network`, que assegura a conformidade com o protocolo da rede *regtest*; e `-p2p-address`, que efetivamente abre o *socket* para aceitar conexões de entrada.

Em contrapartida, a aplicação da flag `-addnode` no comando do nó inverte a lógica tradicional de monitoramento passivo: ela instrui o nó Bitcoin a iniciar ativamente a conexão em direção ao extrator, estabelecendo-o como um par (*peer*) prioritário e persistente. Uma vez consolidado este túnel TCP, o extrator inicia a injeção periódica de mensagens `ping`. O nó Bitcoin, operando de forma reativa, processa estas solicitações e devolve os respectivos `pongs`, ciclo que permite o cálculo de alta precisão do RTT (*Round-Trip Time*) na camada de aplicação.

3.2.4 Persistência e Verificação de Saúde

A integridade dos dados do experimento é mantida através do mapeamento de volumes do Docker para o diretório de dados do Bitcoin (`/home/bitcoin/.bitcoin`). Este volume persiste a *blockchain* (pastas `blocks` e `chainstate`), os arquivos de *log* (`debug.log`) e as credenciais de autenticação RPC (arquivos *cookie*) entre reinicializações do contêiner.

Para garantir que o sistema de monitoramento apenas colete dados válidos, foi implementado um mecanismo de verificação de saúde (*healthcheck*) [23]. Este procedimento executa periodicamente consultas à interface RPC do nó (comando `getblockchaininfo`) para confirmar a disponibilidade do serviço antes de marcá-lo como saudável para o orquestrador Docker.

3.3 Coleta de Eventos pelo Peer Observer

Uma vez estabelecida a conexão TCP entre o nó Bitcoin e o extrator (conforme descrito na Seção 3.2), inicia-se a fase de operação do protocolo. O componente responsável por esta etapa é o `p2p-extractor` [14], um módulo desenvolvido em Rust que não atua apenas como um observador passivo, mas simula o comportamento de um *Full Network Peer*.

A função primordial deste componente nesta arquitetura é isolar a latência da camada de aplicação. Para isso, ele implementa uma máquina de estados que gerencia o ciclo de vida da conexão e executa um algoritmo de sondagem ativo, cujas etapas de negociação e medição são detalhadas a seguir.

3.3.1 Handshake e Estabelecimento de Sessão

A comunicação via protocolo Bitcoin exige uma etapa de autenticação mútua conhecida como *handshake*. Imediatamente após o acoplamento do soquete TCP, o extrator e o nó trocam mensagens de controle para validar a compatibilidade das versões do software.

O processo inicia-se com o envio da mensagem `version`, na qual o extrator declara sua versão de protocolo e capacidades. Embora o extrator não ofereça serviços de armazenamento de blocos (`ServiceFlags::NONE`), ele é reconhecido pelo nó monitorado como um par válido. A sessão é considerada ativa somente após a troca recíproca das mensagens `verack` (*Version Acknowledge*), momento em que o canal é liberado para o tráfego de mensagens de comando.

3.3.2 Mecanismo de Medição de Latência (Ping/Pong)

A metodologia central para a captura da latência de aplicação reside na manipulação dos campos da mensagem de `ping` do protocolo Bitcoin. Por definição, uma mensagem `ping` carrega um campo de dados de 64 bits denominado *nonce*, cujo propósito original é identificar a conexão e detectar auto-conexões. O protocolo estipula que, ao receber um `ping`, o nó deve responder obrigatoriamente com uma mensagem `pong` contendo exatamente o mesmo *nonce* recebido [16].

O extrator explora essa característica determinística para realizar a medição de tempo. Em vez de utilizar um número aleatório, o extrator injeta no campo *nonce* o *timestamp* exato do momento de envio (t_{envio}), com precisão de nanossegundos.

O ciclo de medição, executado periodicamente (por padrão a cada 10 segundos), obedece ao seguinte fluxo lógico:

1. **Envio:** O extrator captura o tempo atual do sistema (T_0), serializa-o como um inteiro de 64 bits não sinalizado (`u64`) e o encapsula no *payload* da mensagem `ping`, despachando-a para o nó.
2. **Processamento Remoto:** O nó Bitcoin recebe a mensagem, realiza a desserialização, valida o *checksum*, identifica o comando e prepara a resposta. Este intervalo compreende o atraso de processamento da aplicação que se deseja mensurar.
3. **Retorno:** O nó envia a mensagem `pong` contendo o mesmo valor T_0 no seu *payload*.
4. **Cálculo:** Ao receber o `pong`, o extrator captura o novo tempo atual (T_1) e realiza a diferença aritmética:

$$Latencia = T_1 - T_0 \tag{3.1}$$

O valor resultante engloba o tempo de serialização, o trânsito na interface de rede local e, crucialmente, o tempo de CPU gasto pelo *Bitcoin Core* para processar a mensagem na camada de aplicação.

3.3.3 Serialização e Exportação de Eventos

Para garantir que o processo de extração não introduza gargalos de performance, os dados brutos da medição não são processados localmente. Imediatamente após o cálculo, o valor da latência é estruturado em um objeto de transferência de dados definido via *Protocol Buffers* (Protobuf), resultando em *payloads* compactos (aproximadamente 6 bytes para um evento de `ping`) e tipagem forte, o que facilita a evolução do esquema de dados sem quebrar a compatibilidade com os consumidores.

O evento serializado é então publicado de forma assíncrona no barramento de mensagens NATS, sob o tópico `p2p_extractor`. Esta abordagem desacopla a lógica de medição (que exige alta precisão temporal) da lógica de armazenamento e indexação, delegando a persistência e a formatação final das métricas para o componente subsequente da arquitetura.

3.4 Processamento das Métricas pelo Metrics

Enquanto o extrator atua na fronteira da rede capturando eventos efêmeros, o componente `metrics` desempenha o papel de agregador de estado e tradutor de protocolos. Sua função arquitetural é consumir o fluxo contínuo de eventos binários provenientes do barramento de mensagens e transformá-los em um modelo de dados dimensional e persistente em memória, acessível via HTTP.

Esta seção detalha o ciclo de vida da informação dentro deste componente, desde a ingestão assíncrona até a exposição síncrona para sistemas de monitoramento.

3.4.1 Ingestão e Deserialização de Eventos

O componente atua como um assinante (*subscriber*) no ecossistema NATS. Ao iniciar, o serviço estabelece uma conexão persistente com o servidor de mensageria e registra uma assinatura curinga (*wildcard*) em todos os tópicos relevantes [23]. Esta abordagem de desacoplamento temporal permite que a coleta de dados (no extrator) e o processamento analítico (no `metrics`) ocorram em ritmos distintos, prevenindo que picos de tráfego na rede P2P saturem a capacidade de processamento da métrica.

O fluxo de ingestão inicia-se quando um *payload* binário chega através do *socket* do NATS. O componente utiliza as definições de esquema do *Protocol Buffers* para deco-

dificar a sequência de bytes, reconstruindo a estrutura de dados original (o evento) em memória. Este processo de deserialização converte a representação de transporte eficiente em estruturas de dados nativas da linguagem (Rust structs), que contêm não apenas os valores medidos (como a latência), mas também metadados cruciais como o *timestamp* de origem e identificadores do emissor.

3.4.2 Roteamento e Atualização de Estado em Memória

Uma vez instanciado o objeto do evento, o sistema executa uma lógica de roteamento baseada no tipo do evento. O `metrics` mantém em memória um registro centralizado (*Registry*), uma estrutura de dados que armazena o estado atual de todos os indicadores monitorados. Diferentemente de um banco de dados tradicional, este registro reside na memória volátil e utiliza tipos atômicos (*AtomicU64*, *AtomicI64*) para garantir a integridade dos dados em um ambiente concorrente sem a necessidade de bloqueios (*locks*) pesados [25].

Para o caso específico da latência , descrita na seção anterior, o manipulador (*handler*) roteia o evento do tipo `PingDuration`. O valor da duração, recebido em nanossegundos, é utilizado para atualizar uma métrica do tipo *Gauge* denominada `p2pextractor_ping_duration_nanoseconds`.

- **Métricas de Estado (Gauge):** Para valores que podem oscilar (como latência ou número de conexões), o manipulador substitui o valor anterior pelo atual.
- **Métricas Incrementais (Counter):** Para eventos de contagem (como número de mensagens enviadas), o manipulador incrementa atomicamente o valor existente.
- **Métricas de Distribuição (Histogram):** Para análises estatísticas, os valores são observados e distribuídos em "recipientes"(*buckets*) pré-definidos para cálculo de percentis.

Este modelo de atualização em memória assegura que o componente sempre possua uma *snapshot* atualizado do comportamento do nó Bitcoin, pronta para ser consultada.

3.4.3 Exposição via Servidor HTTP

A etapa final do processamento é a exposição dos dados. O componente `metrics` implementa um servidor HTTP *multithreaded* que escuta na porta configurada (padronizada como 8001 na arquitetura do projeto) [14]. Este servidor não realiza processamento ativo contínuo; ele reage sob demanda (modelo *Pull*) às requisições de coleta do Prometheus.

Quando uma requisição `GET /metrics` é recebida, o servidor dispara a rotina de coleta (*gather*) no registro de memória. Este procedimento percorre todas as famílias de métricas armazenadas, coleta seus valores atômicos atuais e seus respectivos rótulos (*labels*).

Em seguida, um codificador de texto (*TextEncoder*) serializa esses dados estruturados para o formato de exposição padrão do Prometheus. O resultado é um fluxo de texto contendo linhas de ajuda (`# HELP`), tipo (`# TYPE`) e as séries temporais com seus valores atuais, conforme exemplificado abaixo:

```
# HELP peerobserver_p2pextractor_ping_duration_nanoseconds ...
# TYPE peerobserver_p2pextractor_ping_duration_nanoseconds gauge
peerobserver_p2pextractor_ping_duration_nanoseconds 713000
```

Esta resposta é enviada via HTTP para o coletor, completando o ciclo de transformação do dado: de um evento binário assíncrono na rede interna para uma métrica textual síncrona e padronizada para observabilidade.

3.5 Coleta de Métricas pelo Prometheus

O Prometheus atua como o componente centralizador da arquitetura de observabilidade, operando sob um modelo de coleta ativa (*Pull Model*). Diferentemente de sistemas baseados em envio (*Push*), onde os agentes remetem dados para o servidor, o Prometheus assume a responsabilidade de conectar-se periodicamente aos alvos configurados para realizar a extração (*scraping*) dos dados. Esta escolha arquitetural simplifica o controle de fluxo e a descoberta de serviços, permitindo que o servidor central dite o ritmo da ingestão e evite sobrecarga.

Independentemente da origem dos dados — seja da aplicação ou da sondagem de rede — ambos os *endpoints* respondem utilizando o padronizado *Prometheus Exposition Format* [21]. Isso permite que o *parser* do Prometheus processe dados heterogêneos de forma unificada, validando tipos e extraindo rótulos automaticamente.

3.5.1 Configuração dos Jobs de Coleta

A operação do sistema é governada pelo arquivo de configuração `prometheus.yml`, que define os trabalhos (*jobs*) de coleta. Para atender aos objetivos deste trabalho — a comparação entre latência de aplicação e latência de rede —, foram configurados dois fluxos de coleta distintos que operam em paralelo, ambos com um intervalo de varredura sincronizado em 10 segundos [23].

O primeiro *job*, denominado `metrics`, conecta-se diretamente ao servidor HTTP do componente descrito na seção anterior (Seção 3.4). Sua função é ingerir as métricas de

latência da aplicação Bitcoin ($RTT_{Bitcoin}$), já processadas e expostas no formato de texto padrão do Prometheus.

O segundo trabalho, denominado `ping_peers_ICMP`, é responsável pela aferição da latência de rede pura (RTT_{ICMP}). Diferentemente da coleta passiva de métricas expostas, este *job* instrui o componente *Blackbox Exporter* [24] a realizar uma sondagem ativa contra o nó Bitcoin.

3.5.2 Sondagem Ativa com Blackbox Exporter

A implementação deste mecanismo exige uma configuração avançada de reescrita de rótulos (*relabeling*) no Prometheus. O objetivo é transformar o alvo lógico (o contêiner `bitcoin-node`) em um parâmetro de consulta para a ferramenta de sondagem. O fluxo técnico detalhado desta operação ocorre nas seguintes etapas:

1. **Definição do Alvo:** O Prometheus identifica o alvo `bitcoin-node` na sua lista de configuração.
2. **Transformação de Relabeling:** Antes de iniciar a conexão, regras de reescrita de rótulos modificam o endereço de destino. O endereço original do alvo é movido para um parâmetro de URL (`?target=bitcoin-node`), e o endereço de conexão é substituído pelo endereço do serviço do exportador (ex: `blackbox-exporter:9115`).
3. **Execução da Sonda:** O Prometheus envia uma requisição HTTP GET para o Blackbox Exporter. Este, por sua vez, lê o módulo configurado (`icmp`) e dispara um pacote *ICMP Echo Request* (`ping`) contra o IP resolvido do contêiner do Bitcoin.
4. **Resposta em Nível de Kernel:** O pacote ICMP chega à interface de rede do contêiner do nó. Crucialmente, este pacote é processado e respondido diretamente pela pilha de rede do *kernel* Linux, que envia automaticamente um *ICMP Echo Reply* (`pong`). Este processo ocorre inteiramente em espaço de núcleo (*kernel space*), sem qualquer interação com o processo do *daemon* `bitcoind`. Esta distinção garante que a métrica coletada represente estritamente a latência da infraestrutura de rede, isolada do tempo de processamento da aplicação.
5. **Cálculo e Exposição:** O Blackbox recebe o pacote de resposta e calcula o tempo total de ida e volta (*Round-Trip Time*). O resultado é exposto ao Prometheus sob o *namespace* `probe_*`, especificamente na métrica `probe_icmp_duration_seconds`. Esta métrica utiliza rótulos de fase (`phase`) para segmentar o tempo em etapas distintas de operação: `resolve` (resolução de DNS), `setup` (preparação do socket) e `rtt` (tempo de ida e volta).

O resultado das séries temporais pode ser observado abaixo:

```
# HELP probe_icmp_duration_seconds Duration of ICMP request by phase
# TYPE probe_icmp_duration_seconds gauge
probe_icmp_duration_seconds{phase="resolve"} 0.000823
probe_icmp_duration_seconds{phase="setup"} 0.000078
probe_icmp_duration_seconds{phase="rtt"} 0.000156
```

3.5.3 Persistência em Séries Temporais

Uma vez coletados, os dados de ambos os *jobs* são normalizados e persistidos no banco de dados de séries temporais (TSDB) interno do Prometheus. O TSDB utiliza uma estrutura de blocos (*chunks*) imutáveis.

Cada amostra armazenada é indexada unicamente por seu conjunto de rótulos. O resultado final deste processo é a existência, no mesmo banco de dados, de duas séries temporais comparáveis: `peerobserver_p2pextractor_...` (Aplicação) e `probe_icmp_duration_...` (Rede). A consolidação destas fontes viabiliza as consultas vetoriais que serão aplicadas na etapa de visualização.

3.6 Visualização no Grafana

A etapa final da arquitetura de monitoramento é a transformação dos dados de séries temporais em informação visual interpretável. O componente responsável por esta função é o Grafana, que se conecta ao Prometheus para extrair, processar e renderizar as métricas coletadas.

A escolha desta ferramenta fundamenta-se em dois pilares técnicos essenciais para o projeto: a capacidade de provisionamento automatizado, que elimina configurações manuais frágeis, e o suporte nativo à linguagem PromQL, permitindo a execução de operações vetoriais matemáticas diretamente sobre os dados durante a consulta, sem a necessidade de pré-processamento no banco de dados.

3.6.1 Provisionamento e Conexão com a Fonte de Dados

Para garantir a reprodutibilidade e a integridade do ambiente de visualização, adotou-se uma estratégia de "Infraestrutura como Código"(*Infrastructure as Code*). Ao invés de configurar a conexão via interface gráfica, o Grafana é inicializado através de um sistema de provisionamento que lê arquivos de configuração YAML pré-definidos [23].

Durante o ciclo de *startup* do contêiner, o serviço carrega automaticamente a definição da fonte de dados (*Datasource*). O Prometheus é registrado como a fonte padrão, configurado para acesso via *proxy* interno do Grafana. Simultaneamente, a definição estrutural do *dashboard* — um objeto JSON contendo a especificação de todos os painéis, variáveis e consultas — é importada para o banco de dados interno da aplicação. Esta abordagem assegura que o ambiente de análise esteja operacional e padronizado imediatamente após a orquestração dos contêineres.

3.6.2 Construção do Painel Comparativo

O núcleo da visualização é a comparação direta e síncrona entre as latências de aplicação e de rede. Para materializar esta análise, foi desenvolvido um painel de séries temporais (*Time Series*) que plota múltiplas consultas no mesmo eixo cartesiano temporal.

Como as fontes de dados geram valores em grandezas distintas — o P2P Extractor reporta em nanossegundos e o exportador ICMP em segundos —, o motor de consulta do Grafana aplica funções de normalização em tempo de leitura para converter ambas as métricas para uma unidade comum (milissegundos):

- **Série $RTT_{Bitcoin}$:** Consulta a métrica de aplicação, aplicando a divisão por 10^6 para conversão de nanossegundos para milissegundos.
- **Série RTT_{ICMP}:** Consulta a métrica de rede do *Blackbox*, aplicando a multiplicação por 10^3 para conversão de segundos para milissegundos.

A sobreposição visual destas duas linhas permite a inspeção imediata da correlação entre picos de rede e picos de processamento do nó.

3.6.3 Métricas Derivadas e Análise de Overhead

Além da plotagem dos dados brutos, a capacidade analítica do sistema é expandida através de uma camada de processamento de métricas derivadas. Diferentemente das séries temporais que exibem valores instantâneos, estas métricas fundamentam-se na agregação temporal dos dados para fornecer indicadores de tendência e eficiência.

O processo inicia-se pelo cálculo da **Média de Latências** para cada fluxo. Utilizando funções de agregação vetorial (`avg_over_time`) sobre a janela de observação selecionada, o sistema suaviza os ruídos momentâneos da rede e estabelece valores de referência consolidados para o período ($\overline{RTT}_{Bitcoin}$ e $\overline{RTT}_{ICMP}$).

Estas médias constituem a base matemática para o cálculo da métrica mais crítica deste estudo: o *Protocol Overhead* (Sobrecarga do Protocolo). Este indicador isola o

tempo estritamente gasto pelo *software* Bitcoin para processar a mensagem, sendo obtido através da diferença aritmética entre as médias consolidadas, conforme a expressão:

$$\Delta_{Overhead} = \overline{RTT}_{Bitcoin} - \overline{RTT}_{ICMP} \tag{3.2}$$

A partir desta derivação, gera-se o gráfico de distribuição (*Latency Breakdown*). Este componente visual utiliza os valores médios calculados para segmentar a latência total em proporções percentuais, oferecendo uma visão holística que contrasta o custo da infraestrutura física com o custo computacional imposto pelo protocolo P2P.

3.7 Considerações Finais

A implementação descrita ao longo deste capítulo consolidou uma *pipeline* completa de observabilidade para a rede Bitcoin, desenhada especificamente para isolar e quantificar os componentes de latência em sistemas distribuídos. A arquitetura final, fundamentada na orquestração de microsserviços via Docker, integrou com sucesso componentes heterogêneos — desde a execução do nó Bitcoin e extratores de protocolo até sistemas de agregação e visualização de séries temporais [23].

O ambiente desenvolvido atende rigorosamente aos requisitos de isolamento e reprodutibilidade definidos na proposta. A estratégia de conteinerização, aliada ao provisionamento automatizado do Grafana e à configuração declarativa do Prometheus, assegura que o experimento possa ser instanciado de forma determinística, eliminando a fragilidade inerente a configurações manuais. Adicionalmente, a utilização do NATS como *middleware* de mensageria provou-se essencial para garantir o desacoplamento entre a coleta de eventos em alta frequência e o processamento analítico, conferindo resiliência e estabilidade ao fluxo de dados [14].

Sob a perspectiva funcional, a solução demonstrou capacidade para capturar, de forma síncrona e contínua, as duas dimensões críticas de desempenho:

- A latência da aplicação ($RTT_{Bitcoin}$), obtida através da interação ativa com o protocolo P2P.
- A latência da infraestrutura (RTT_{ICMP}), aferida via sondagem *Blackbox*.

Com a infraestrutura plenamente operacional e validada pelos mecanismos de verificação de saúde (*healthchecks*), o sistema encontra-se apto a gerar a massa de dados necessária para a análise comparativa. As métricas coletadas e processadas por esta arquitetura constituem a base empírica que será explorada e discutida no capítulo subsequente.

Capítulo 4

Resultados

O capítulo anterior detalhou a engenharia da infraestrutura de monitoramento. Este capítulo dedica-se à validação empírica desta arquitetura, demonstrando sua eficácia operacional e analítica dentro do ambiente controlado da rede *Regtest*.

O objetivo desta exposição é comprovar que a solução desenvolvida transcende a especificação teórica, entregando resultados tangíveis em duas dimensões críticas para o monitoramento de sistemas distribuídos. Primeiramente, avalia-se a robustez do *pipeline* de coleta, assegurando que os eventos de latência sejam capturados, transportados e persistidos de forma estável e contínua. Em segundo lugar, apresenta-se a materialização visual da discrepância entre a latência de infraestrutura (RTT_{ICMP}) e a latência de aplicação ($RTT_{Bitcoin}$), evidenciando quantitativamente o custo de processamento do protocolo que motivou este estudo.

4.1 Validação Operacional do Ambiente

Antes de proceder à análise quantitativa das latências, é imperativo validar a integridade funcional da infraestrutura implementada. A confiabilidade dos dados apresentados neste trabalho depende diretamente da estabilidade das conexões internas entre os microsserviços e da correta execução dos protocolos de rede.

Esta seção apresenta as evidências operacionais de que o ambiente orquestrado via Docker comportou-se conforme o planejado, garantindo que o *pipeline* de dados (explícitos na seção 3.1) — desde a injeção de mensagens P2P até a sondagem ICMP — estivesse desobstruído e livre de artefatos de concorrência.

4.1.1 Verificação do Fluxo de Mensagens P2P

A premissa fundamental para a medição da latência de aplicação é o estabelecimento e a manutenção de uma sessão TCP entre o extrator e o nó Bitcoin. Conforme definido no capítulo 3, a topologia adota uma conexão reversa onde o nó é instruído a se conectar ao extrator.

A Figura 4.1 demonstra o log de operação inicial do componente `p2p-extractor`. O registro evidencia a sequência de inicialização bem-sucedida na porta 8555, destacando primeiramente a confirmação da conexão com o barramento de mensagens NATS na porta 4222. Isso assegura que o extrator esteja operacional e apto a transmitir os dados de telemetria antes de interagir com a rede P2P.

```
Starting p2p-extractor setup...
Starting p2p-extractor on 0.0.0.0:8555...
2025-11-25T15:02:36.647Z INFO  [p2p_extractor] Using network magic for: regtest
2025-11-25T15:02:36.647Z INFO  [p2p_extractor] Ping measurements enabled: true
2025-11-25T15:02:36.647Z INFO  [p2p_extractor] Ping measurements interval: 10s
2025-11-25T15:02:36.647Z DEBUG [p2p_extractor] Connecting to NATS server at nats://nats:4222..
2025-11-25T15:02:36.657Z INFO  [p2p_extractor] Connected to NATS server at nats://nats:4222
2025-11-25T15:02:36.657Z DEBUG [p2p_extractor] Starting TCP listener on 0.0.0.0:8555..
2025-11-25T15:02:36.657Z INFO  [p2p_extractor] P2P-extractor listening on 0.0.0.0:8555
Launching Bitcoin node in regtest mode (connecting to p2p-extractor)...
```

Figura 4.1: Log do P2P-extractor evidenciando a sua inicialização e conexão com o NATS (Fonte: [23]).

Na sequência do processo, observa-se a aceitação da conexão de entrada proveniente do contêiner `bitcoin-node` na porta 38040, conforme ilustrado na Figura 4.2. A estabilidade desta conexão valida a eficácia do parâmetro `-addnode` utilizado na orquestração e confirma que a rede interna do Docker está permitindo o tráfego na porta configurada, estabelecendo o canal necessário para a injeção de mensagens de ping.

```
2025-11-25T15:02:39.092Z INFO  [p2p_extractor] accepted a new connection from: 127.0.0.1:38024
2025-11-25T15:02:39Z init message: Done loading
2025-11-25T15:02:39Z 0 addresses found from DNS seeds
2025-11-25T15:02:39Z dnsseed thread exit
2025-11-25T15:02:39.141Z DEBUG [127.0.0.1:38024] mismatch in the P2P message magic: this could mean the node is on a different network or it's attempting a
P2Pv2 connection..
2025-11-25T15:02:39.141Z WARN  [127.0.0.1:38024] error decoding message: Network magic mismatch: got=6526d534, expected=fabfb5da
2025-11-25T15:02:39.141Z INFO  [p2p_extractor] closing connection: '127.0.0.1:38024'
2025-11-25T15:02:39.592Z INFO  [p2p_extractor] accepted a new connection from: 127.0.0.1:38040
2025-11-25T15:02:39Z New manual v1 peer connected: version: 70001, blocks=0, peer=1
Bitcoin node connected to p2p-extractor successfully
2025-11-25T15:02:49.597Z DEBUG [127.0.0.1:38040] processing the ping message took: 788714ns
2025-11-25T15:02:59.596Z DEBUG [127.0.0.1:38040] processing the ping message took: 778321ns
```

Figura 4.2: Log do P2P-extractor evidenciando a aceitação da conexão iniciada pelo nó Bitcoin (Fonte: [23]).

Uma vez estabelecido o canal, a validação do fluxo de dados ocorre no componente consumidor. A Figura 4.3 apresenta os registros do serviço `metrics`, confirmando a recepção contínua de eventos do tipo `PingDuration`. A presença destes logs comprova que o ciclo completo do protocolo — envio de `ping`, processamento pelo nó, retorno de `pong` e publicação no NATS — está ocorrendo sem interrupções ou perdas de pacotes que poderiam comprometer a amostragem.

```
# HELP peerobserver_p2pextractor_ping_duration_nanoseconds The time it takes for a connected Bitcoin node to respo
nd to a ping with a pong in nanoseconds.
# TYPE peerobserver_p2pextractor_ping_duration_nanoseconds gauge
peerobserver_p2pextractor_ping_duration_nanoseconds 1442748
```

Figura 4.3: Trecho de log do serviço Metrics confirmando a ingestão e processamento dos eventos de latência gerados (Fonte: [23]).

4.1.2 Consistência da Coleta ICMP

Paralelamente ao fluxo de aplicação, a validação do monitoramento de infraestrutura focou na capacidade do *Blackbox Exporter* em resolver e alcançar o contêiner alvo dentro da rede virtual. A integridade deste teste confirma que o isolamento proporcionado pelos *Linux Namespaces* não impediu a visibilidade de rede necessária para o diagnóstico.

A Figura 4.4 exibe o resultado de uma sondagem (*probe*) bem-sucedida. O registro explicita que o exportador foi capaz de resolver o nome de domínio `bitcoin-node` para seu endereço IP interno e receber a resposta do *Echo Reply*. O código de sucesso (`probe_sucess 1`) e a duração registrada validam que a latência de base está sendo coletada sem interferência de bloqueios de *firewall* ou falhas de roteamento interno do Docker.

```
# HELP probe_icmp_duration_seconds Duration of icmp request by phase
# TYPE probe_icmp_duration_seconds gauge
probe_icmp_duration_seconds{phase="resolve"} 0.00099273
probe_icmp_duration_seconds{phase="rtt"} 0.000217986
probe_icmp_duration_seconds{phase="setup"} 7.4265e-05
# HELP probe_success Displays whether or not the probe was a success
# TYPE probe_success gauge
probe_success 1
```

Figura 4.4: Log de verificação do Blackbox Exporter demonstrando o sucesso da sonda ICMP contra o contêiner do nó (Fonte: [23]).

4.2 Exposição e Visualização das Métricas

A validação operacional descrita na seção anterior garantiu a integridade do fluxo de dados. No entanto, o objetivo central deste trabalho não se limita à captura de eventos, mas reside na capacidade de expô-los de forma comparativa e inteligível – como mencionado na Seção 1.5. Esta seção apresenta a materialização final da arquitetura proposta: o *dashboard* de monitoramento, alimentado por uma coleta de dados realizada ininterruptamente ao longo de um período de 12 horas.

Através da normalização temporal e da aplicação de funções de agregação vetorial, o sistema converteu séries temporais heterogêneas em visualizações que evidenciam o comportamento do protocolo Bitcoin em relação à infraestrutura de rede subjacente, alinhando-se à metodologia de análise temporal explorada por Neudecker et al. [6].

4.2.1 Monitoramento Comparativo em Tempo Real

A visualização primária, responsável por concretizar a proposta de comparação direta entre as camadas de aplicação e de rede, é apresentada na Figura 4.5. Este gráfico de séries temporais plota, no mesmo eixo cartesiano, a evolução do $RTT_{Bitcoin}$ (linha superior, em azul) e do RTT_{ICMP} (linha inferior, em laranja) durante a janela de observação de 12 horas.

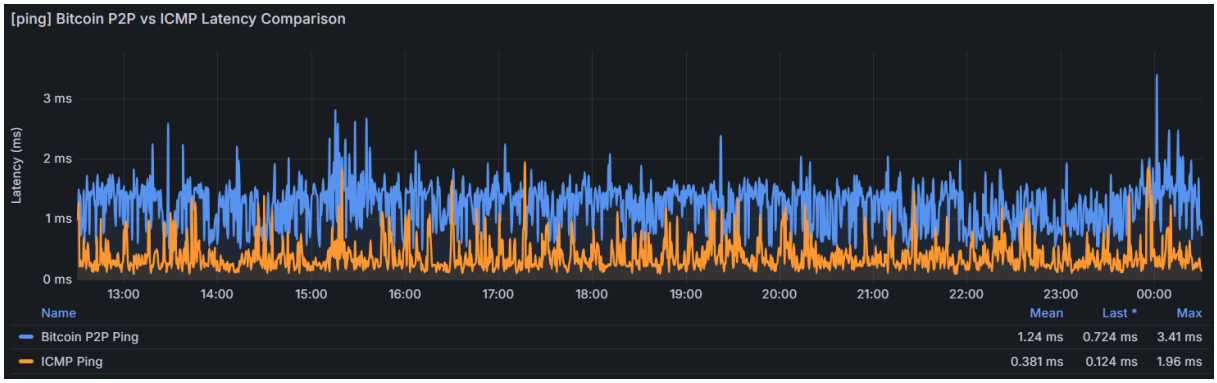

Figura 4.5: Evolução temporal comparativa entre latência de aplicação (Bitcoin) e latência de rede (ICMP) em uma janela de 12 horas (Fonte: [23]).

A renderização deste painel comprova a eficácia do motor de consulta PromQL em normalizar grandezas de ordens diferentes — nanossegundos para o protocolo P2P e segundos para o ICMP — para uma unidade comum de milissegundos. O comportamento visual observado revela linhas que têm uma forte correlação temporal: flutuações na infraestrutura de rede (linha laranja) refletem-se imediatamente na latência da aplicação (linha azul). Contudo, a persistência de um distanciamento vertical entre as curvas confirma

visualmente a existência de um custo de processamento fixo. Este fenômeno corrobora as observações de Neudecker et al. [6], que identifica esse diferencial como o tempo de processamento interno do nó (d_{proc}), validando a hipótese de que a latência percebida pelo par ($Peer$) é consistentemente superior à latência de transporte.

4.2.2 Isolamento do Overhead de Processamento

Para quantificar a eficiência do nó para além da análise visual das tendências, o sistema consolida as métricas médias do período. A Figura 4.6 apresenta os indicadores de média absoluta para a aplicação e para a rede. Estes valores, calculados sobre a totalidade das amostras das 12 horas, servem como base fundamental para os cálculos de eficiência.

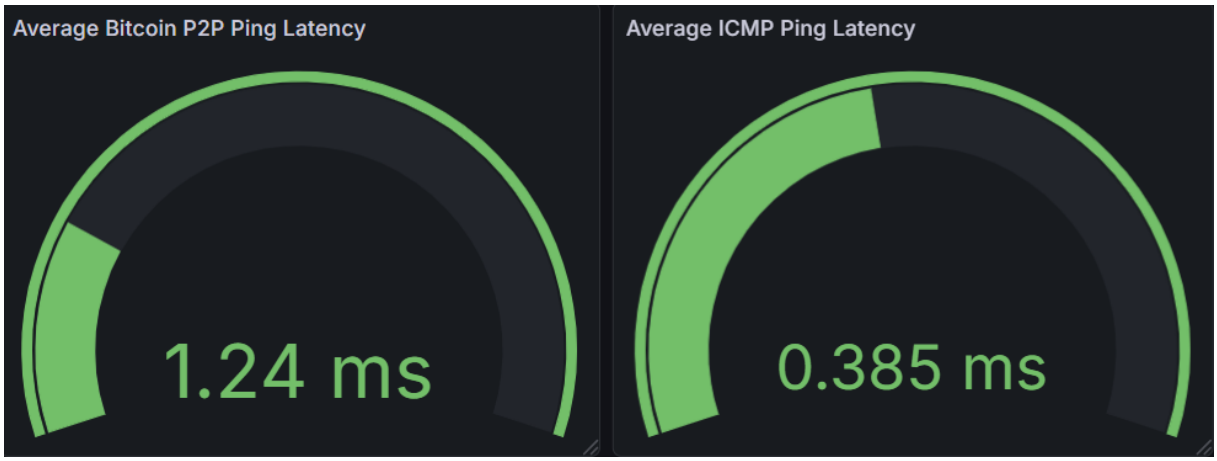

Figura 4.6: Indicadores de latência média consolidada para o protocolo Bitcoin e infraestrutura ICMP (Fonte: [23]).

A partir destas médias consolidadas, o sistema processa uma nova informação derivada: o *Protocol Overhead*. Esta métrica é calculada pela diferença aritmética entre a média de latência do Bitcoin e a média de latência do ICMP ($\overline{RTT}_{Bitcoin} - \overline{RTT}_{ICMP}$), sendo apresentada nos painéis de instrumentação da Figura 4.7.

O painel de medição (*Gauge*), à esquerda, exibe o valor absoluto do atraso introduzido pelo *software*, registrando uma média de aproximadamente $0,853$ ms. Complementarmente, o gráfico de distribuição (*Pie Chart*), à direita, ilustra a proporção percentual entre o tempo gasto na rede física e o tempo consumido pelo processamento do protocolo.

Esta visualização oferece uma resposta imediata sobre a eficiência operacional do nó, métrica que inexiste em ferramentas de monitoramento padrão. A decomposição apresentada permite distinguir se um eventual gargalo é proveniente de congestionamento de rede (aumentando a fatia do ICMP) ou de estresse computacional no processamento de mensagens (aumentando a fatia do *Overhead*).

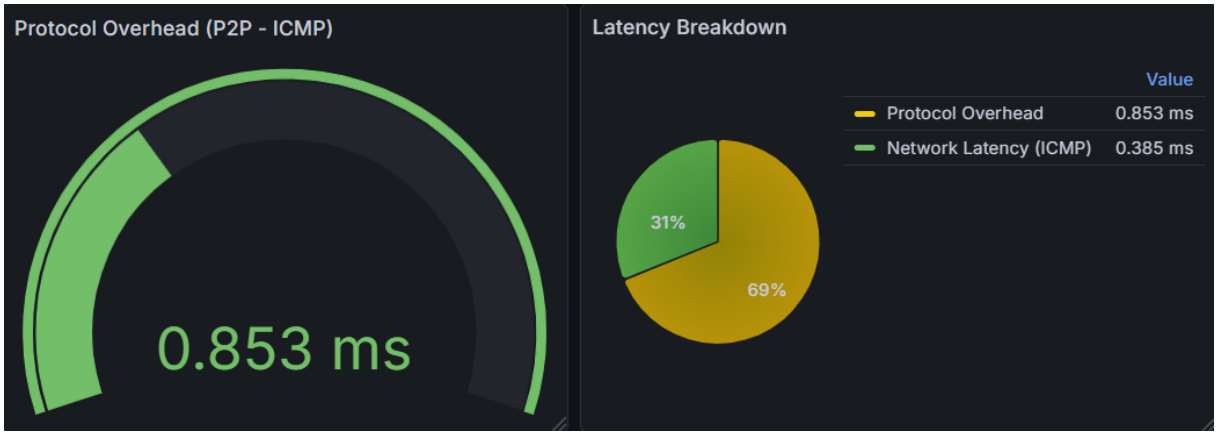

Figura 4.7: Quantificação da eficiência do nó: Indicador de Overhead e Distribuição de Latência derivados das médias do período (Fonte: [23]).

4.3 Diagnóstico e Interpretabilidade dos Dados

A mera exposição de métricas em séries temporais, embora necessária, é insuficiente para a operação eficiente de nós em produção. A utilidade prática da ferramenta desenvolvida reside na sua capacidade de transformar dados brutos em diagnósticos imediatos. A arquitetura de monitoramento proposta preenche a lacuna de observabilidade ao fornecer "assinaturas visuais"claras, permitindo que operadores realizem a triagem de anomalias distinguindo, com precisão visual, falhas de infraestrutura de gargalos de aplicação.

4.3.1 Identificação de Padrões de Latência

A interpretação da Figura 4.5 (Séries Temporais) baseia-se no comportamento relativo entre as curvas de latência de aplicação ($RTT_{Bitcoin}$) e de rede (RTT_{ICMP}). A ferramenta foi projetada para que a correlação — ou a falta dela — entre estas séries indique a causa raiz de uma degradação de desempenho:

- **Assinatura de Rede (Correlação):** Quando ambas as linhas sofrem elevação simultânea mantendo a correlação visual, o diagnóstico indica um problema exógeno ao nó. Se a latência de rede sobe e a latência da aplicação acompanha esse movimento na mesma proporção, a degradação é atribuída à infraestrutura de transporte (congestionamento de rota, latência de *link*). Neste cenário, a ferramenta isenta o *software* Bitcoin Core de responsabilidade, evitando esforços desnecessários de depuração interna.
- **Assinatura de Aplicação (Divergência):** Quando as linhas divergem — especificamente, quando a linha azul ($RTT_{Bitcoin}$) apresenta picos ou elevação gradual

enquanto a linha laranja (RTT_{ICMP}) permanece estável e baixa — o problema é isolado no *software*. Visualmente, isso se traduz em um aumento da área entre as curvas, indicando que o nó está consumindo tempo excessivo em operações de *user space* (como validação de blocos pesados ou *garbage collection*), independentemente da qualidade da rede.

Esta segregação visual permite uma triagem instantânea, substituindo a complexidade da correlação manual de *logs* de texto e carimbos de tempo por uma análise gráfica intuitiva.

4.3.2 Interpretação da Eficiência e Saúde do Sistema

Para além da análise de tendências, a ferramenta oferece mecanismos para a avaliação instantânea da saúde do sistema através dos Painéis da Figura 4.7. A interpretação destes dados é facilitada pela aplicação de Limiares Semânticos (*Thresholds*) configurados no Grafana, que atuam como um "semáforo"para a operação do nó.

O indicador de *Protocol Overhead* (Figura 4.7) traduz valores numéricos em estados de urgência:

- **Estado Verde (Nominal):** Indica que o atraso de processamento é marginal, sinalizando que o sistema opera com folga de recursos.
- **Estado Vermelho (Crítico):** Indica que o tempo de processamento ultrapassou limites aceitáveis de segurança (ex: $>$ 50ms em ambiente local), sugerindo travamento de processos (*deadlocks*) ou exaustão severa de recursos computacionais.

Complementarmente, o Gráfico de Distribuição (*Latency Breakdown* - Figura 4.7) orienta decisões estratégicas de investimento em infraestrutura. Uma predominância da fatia de "Rede"sugere a necessidade de melhoria na conectividade (banda larga ou localização geográfica), enquanto uma predominância da fatia de "Overhead"indica que a otimização deve ocorrer no *hardware* do servidor (CPU/Memória).

Em suma, a ferramenta desenvolvida entrega *insights* acionáveis, permitindo que a gestão da infraestrutura Bitcoin migre de uma abordagem reativa, baseada em falhas, para uma abordagem proativa, baseada em evidências quantitativas de eficiência.

Capítulo 5

Conclusão

A trajetória percorrida neste trabalho partiu da identificação de uma lacuna crítica de observabilidade na rede Bitcoin e culminou na materialização de uma ferramenta capaz de preenchê-la. O Capítulo 1 estabeleceu a premissa central da pesquisa: a latência na rede Bitcoin operava sob uma opacidade sistêmica, onde a ausência de instrumentação adequada tornava impossível distinguir entre atrasos impostos pela infraestrutura de rede e ineficiências intrínsecas ao *software* do nó. Os Capítulos 2 e 3 fundamentaram e detalharam a engenharia da solução, enquanto o Capítulo 4 demonstrou a validação operacional e a capacidade analítica da arquitetura desenvolvida.

A solução proposta rompeu essa opacidade ao orquestrar uma infraestrutura de monitoramento baseada em microsserviços, cumprindo integralmente a estratégia definida na Seção 1.5. O **estabelecimento de um ambiente determinístico** foi plenamente atingido através da containerização do nó Bitcoin e sua configuração na rede *regtest*, conforme detalhado na Seção 3.2. Essa abordagem isolou com sucesso o experimento de flutuações externas da internet, permitindo que as variações de latência observadas fossem atribuídas exclusivamente ao comportamento do *software* e da pilha de rede local.

No que tange à **extração ativa da latência de aplicação**, o componente *P2P Extractor* demonstrou eficácia na manutenção de conexões persistentes e na injeção periódica de mensagens `ping`, validando o fluxo de captura descrito nas Seções 3.3 e 3.4. A utilização do NATS e *Protobuf* garantiu o desacoplamento necessário, evitando que a instrumentação introduzisse gargalos no nó monitorado. Simultaneamente, a **definição da linha de base de infraestrutura** foi assegurada pela integração do *Blackbox Exporter* (Seção 3.5.2), que forneceu métricas de ICMP estáveis e livres do processamento da aplicação, estabelecendo o "zero de referência"fundamental para o estudo.

Por fim, a **consolidação e análise comparativa** materializou-se na construção dos *dashboards* no Grafana (Seção 3.6). A centralização dos dados no Prometheus (Seção 3.5) permitiu não apenas a plotagem síncrona das séries temporais, mas também o cálculo em

tempo real do *Protocol Overhead*. Como demonstrado nos resultados (Capítulo 4), essa visualização transformou dados brutos em informação diagnóstica, evidenciando de forma clara a discrepância temporal entre a camada de rede e a camada de aplicação.

O objetivo principal do trabalho foi, portanto, plenamente atingido. A formulação da métrica de *Protocol Overhead* constitui a prova definitiva de que a ferramenta é capaz de isolar e quantificar o custo computacional do protocolo Bitcoin. Mais do que a simples coleta de dados, a solução converteu métricas técnicas em inteligência operacional: a capacidade de distinguir visualmente a latência de rede da latência de aplicação fornece aos operadores *insights* de diagnóstico imediatos, permitindo a detecção de anomalias e gargalos de desempenho que passariam despercebidos em abordagens de monitoramento tradicionais.

5.1 Limitações

Embora a arquitetura proposta tenha cumprido o objetivo de isolar e quantificar o *overhead* do protocolo, a validade dos resultados deve ser interpretada à luz das restrições inerentes ao design experimental. A primeira e mais significativa limitação refere-se ao ambiente de rede utilizado. A validação foi realizada exclusivamente em modo *regtest* (Seção 3.2.2), uma rede local determinística. Se, por um lado, essa escolha foi metodologicamente vital para eliminar ruídos estocásticos e estabilizar o cálculo do custo computacional, por outro, ela impede a observação de fenômenos característicos da *Mainnet*, como a latência geográfica, a propagação por múltiplos saltos e a variação imprevisível de largura de banda. Portanto, os valores absolutos de latência de rede (RTT_{ICMP}) aqui apresentados não refletem a realidade de um nó operando na internet pública, servindo apenas como linha de base laboratorial.

Uma segunda restrição reside na escalabilidade da coleta P2P. O protótipo do *P2P Extractor* foi configurado para manter uma relação de monitoramento um-para-um (1:1) com o nó alvo. Em um cenário de produção real, um nó Bitcoin mantém conexões simultâneas com dezenas de pares. A ferramenta, em sua versão atual, não implementa a concorrência necessária para gerenciar múltiplas conexões de saída e entrada simultaneamente, o que limita sua aplicação para a análise de topologia de rede ou para a inferência de latência média de vizinhança (neighbourhood latency).

Por fim, há uma limitação operacional relacionada à complexidade de orquestração. A decisão arquitetural de utilizar microsserviços desacoplados (NATS, Prometheus, Grafana, Extratores) aumentou significativamente a carga cognitiva e operacional de implantação em comparação a ferramentas de monitoramento monolíticas. A necessidade de gerenciar múltiplos contêineres, volumes de persistência e redes virtuais cria uma barreira de en-

trada técnica, exigindo que o operador possua conhecimentos prévios em orquestração de contêineres para replicar o ambiente de observabilidade proposto.

5.2 Trabalhos Futuros

As fronteiras delimitadas para este estudo abrem caminhos claros para a evolução da ferramenta e da metodologia de análise. O passo subsequente mais lógico consiste na **Validação na Mainnet**. A implantação da arquitetura em um nó público permitiria confrontar o comportamento determinístico observado em laboratório com a dinâmica estocástica da rede real. Para que essa análise seja estatisticamente relevante, propõe-se a execução de uma **Coleta de Dados Longitudinal**, operando o sistema ininterruptamente por um período de 6 a 12 meses. Isso possibilitaria a criação de uma base histórica robusta para classificar os níveis de *overhead* em diferentes condições de mercado e congestionamento da rede.

Para elevar a precisão do diagnóstico de causalidade, sugere-se a **Correlação com Métricas de Hardware**. A integração de ferramentas como o *cAdvisor* ao *dashboard* permitiria sobrepor dados de consumo de CPU, memória e I/O aos gráficos de latência. Dessa forma, seria possível provar empiricamente se uma elevação no *Protocol Overhead* é consequência direta de exaustão de recursos (ex: uso de CPU em 100%), eliminando ambiguidades na triagem de problemas.

No âmbito da operação autônoma, a ferramenta deve evoluir para incluir **Alertas Inteligentes**. A configuração do *Alertmanager* do Prometheus permitiria notificar operadores não apenas sobre falhas binárias (como a queda de um par, dessincronização do nó ou indisponibilidade do NATS), mas também sobre anomalias estatísticas, disparando avisos quando o *overhead* desviar significativamente do padrão (desvio padrão) estabelecido pela linha de base.

A interface de visualização também comporta evoluções através de **Painéis Avançados**. A implementação de mapas de calor (*Heatmaps*) facilitaria a identificação de padrões de latência ao longo do dia, enquanto o uso de alertas visuais dinâmicos (mudança de cores ou intermitência) aumentaria a perceptibilidade de estados críticos.

Para aumentar a abrangência e precisão do monitoramento, é necessária a implementação da **Medição de Latência para Outros Tipos de Mensagens**. O atual mecanismo baseia-se estritamente em mensagens de controle (`ping`), que possuem processamento leve. A expansão da coleta para incluir mensagens de dados, como `inv`, `tx` e `block`, permitiria avaliar o comportamento do nó diante de cargas de trabalho mais onerosas. Isso possibilitaria correlacionar o aumento da latência com o tempo gasto nas

rotinas de desserialização e validação de blocos e transações, oferecendo uma visão mais completa do impacto do protocolo sobre o desempenho da rede.

Por fim, para garantir a confiabilidade do *software* em ambientes de produção, é imprescindível a implementação de uma suíte de **Testes Automatizados**. Isso deve abranger testes de unidade para os *scripts* de orquestração, validação da integridade dos JSONs do Grafana e, crucialmente, testes de integração que verifiquem o fluxo completo de dados entre o `bitcoind` e o serviço de métricas, assegurando que atualizações futuras não comprometam a observabilidade.

Referências

[1] Leiner, Barry M, Vinton G Cerf, David D Clark, Robert E Kahn, Leonard Kleinrock, Daniel C Lynch, Jon Postel, Larry G Roberts e Stephen Wolff: *A brief history of the internet*. ACM SIGCOMM Computer Communication Review, 39(5):22–31, 2009. 1

[2] Back, Adam: *Hashcash - a denial of service counter-measure*, 2002. `http://www.hashcash.org/papers/hashcash.pdf`, Original proposal in 1997, technical report in 2002. 1

[3] Dai, Wei: *b-money*, 1998. `http://www.weidai.com/bmoney.txt`. 1

[4] Szabo, Nick: *Bit gold*, 1998. `http://unenumerated.blogspot.com/2005/12/bit-gold.html`, Conceptualized in 1998, blog post from 2005. 1

[5] Nakamoto, Satoshi: *Bitcoin: A peer-to-peer electronic cash system*, 2008. `https://bitcoin.org/bitcoin.pdf`, Whitepaper. 1, 2, 3, 9

[6] Neudecker, Till, Philipp Andelfinger e Hannes Hartenstein: *Timing analysis for inferring the topology of the bitcoin p2p network*, 2016. `https://www.dsn.kastel.kit.edu/publications/files/323/bitcoin_timing_analysis_dsn.pdf`. 3, 4, 5, 6, 34, 35

[7] Heilman, Ethan, Alison Kendler, Aviv Zohar e Sharon Goldberg: *Eclipse attacks on bitcoin's peer-to-peer network*. Em *USENIX Security*, 2015. `https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-heilman.pdf`. 4

[8] Douceur, John: *The sybil attack*. Em *IPTPS*, 2002. `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/IPTPS2002.pdf`. 4

[9] Eyal, Ittay e Emin Gün Sirer: *Majority is not enough: Bitcoin mining is vulnerable*. Em *International Conference on Financial Cryptography and Data Security*, páginas 436–454. Springer, 2014. 4

[10] Decker, Christian e Roger Wattenhofer: *Information propagation in the bitcoin network*. Em *IEEE P2P 2013 Proceedings*, páginas 1–10, 2013. `http://www.gsd.inesc-id.pt/~ler/docencia/rcs1314/papers/P2P2013_041.pdf`. 4

[11] Narayanan, Arvind: *Analyzing the 2013 bitcoin fork: centralized decision-making saved the day*, 2015. `https://blog.citp.princeton.edu/2015/07/28/analyzing-the-2013-bitcoin-fork-centralized-decision-making-saved-the-day/`. 4

[12] Longchamp, Yves, Saurabh Deshpande e Ujjwal Mehra: *Classification and importance of nodes in a blockchain network*, 2020. https://aminagroup.com/research/classification-and-importance-of-nodes-in-a-blockchain-network/. 4

[13] Essaid, Meryam, SeJin Park e Hongtaek Ju: *Visualising bitcoin's dynamic p2p network topoogy and performance*. Em *Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019. https://ieeexplore.ieee.org/document/8751305. 4

[14] 0xB10C: *peer-observer: Tool to monitor p2p anomalies using bitcoin core honeynodes*, 2025. https://github.com/0xB10C/peer-observer. 4, 10, 13, 17, 22, 25, 30

[15] Tanenbaum, Andrew S. e David J. Wetherall: *Computer Networks*. Pearson, 5th edição, 2011. 5

[16] Bitcoin Core Developers: *Bitcoin developer documentation: P2p networking*, 2023. https://developer.bitcoin.org/reference/p2p_networking.html. 5, 23

[17] Neudecker, Till: *Analysis and characterization of the bitcoin network*, 2015. https://www.dsn.kastel.kit.edu/bitcoin/publications/bitcoin_network_characterization.pdf. 5, 6

[18] 0xB10C: *monitoring wishlist – issue #8*. GitHub. https://github.com/0xB10C/project-ideas/issues/8, acesso em 2025-11-20. 6

[19] Google Developers: *Protocol buffers documentation: Overview*. Protobuf Documentation, 2008. https://protobuf.dev/overview/. 11, 12

[20] Collison, Derek: *Nats: Connective technology for adaptive edge & distributed systems*. NATS Documentation, 2010. https://docs.nats.io/nats-concepts/what-is-nats. 12

[21] Prometheus developers: *Prometheus — overview*, 2025. https://prometheus.io/docs/introduction/overview/. 14, 15, 26

[22] Grafana Labs: *Grafana cloud documentation*, 2025. https://grafana.com/docs/grafana-cloud/. 15

[23] ClubeBitcoinUnB: *peer-observer-docker: Docker setup for peer-observer*, 2025. https://github.com/ClubeBitcoinUnB/peer-observer-docker. 18, 19, 21, 22, 24, 26, 28, 30, 32, 33, 34, 35, 36

[24] Prometheus Project: *Blackbox prober exporter*, 2025. https://github.com/prometheus/blackbox_exporter. 18, 27

[25] Rust Project Developers: *std::sync::atomic — atomic types*, 2025. https://doc.rust-lang.org/std/sync/atomic/index.html. 25