DISSERTAÇÃO DE MESTRADO PROFISSIONAL

# *Byzantine Fault Tolerant to Health* (BFT-H) - Proposta de um algoritmo de consenso direcionado às especificidades de sistemas de gestão de saúde baseados em *Blockchain*


**FABRICIO RODRIGUES FREIRE**

**Orientador Prof. Dr. William Ferreira Giozza**

**Coorientador Prof. Dr. Carlo Kleber Silva Rodrigues**



Programa de Pós-Graduação Profissional em Engenharia Elétrica

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

# *Byzantine Fault Tolerant to Health (BFT-H)* - Proposta de um algoritmo de consenso direcionado às especificidades de sistemas de gestão de saúde baseados em *Blockchain*

***Byzantine Fault Tolerant to Health (BFT-H) - Proposal for a consensus algorithm with focus on systems specificities Blockchain-based Healthcare Management***

**Fabricio Rodrigues Freire**

**Orientador: Prof. Dr. William Ferreira Giozza, ENE/UnB**

**Coorientador: Prof. Dr. Carlo Kleber Silva Rodrigues, CMCC/UFABC**

PUBLICAÇÃO: PPEE.MP.076

BRASÍLIA-DF

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO PROFISSIONAL

# *Byzantine Fault Tolerant to Health* (BFT-H) - Proposta de um algoritmo de consenso direcionado às especificidades de sistemas de gestão de saúde baseados em *Blockchain*

**FABRICIO RODRIGUES FREIRE**

**Orientador Prof. Dr. William Ferreira Giozza**

**Coorientador Prof. Dr. Carlo Kleber Silva Rodrigues**

*Dissertação de Mestrado Profissional submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica*

Banca Examinadora

Prof. Dr. William Ferreira Giozza, ENE-FT/UnB _____
*Presidente*

Prof. Dr. Robson de Oliveira Albuquerque, ENE-FT/UnB _____
*Examinador Interno*

Prof. Dr. André Castelo Branco Soares, Universidade Federal do Piauí (UFPI) _____
*Examinador externo*

Prof. Dr. Rafael Rabelo Nunes, ENE-FT/UnB _____
*Suplente*

**FICHA CATALOGRÁFICA**

FREIRE, FABRICIO RODRIGUES
*Byzantine Fault Tolerant to Health* (BFT-H) - Proposta de um algoritmo de consenso direcionado às especificidades de sistemas de gestão de saúde baseados em *Blockchain* [Distrito Federal] 2024.
xvi, 163 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2024).
Dissertação de Mestrado Profissional - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

| | |
|---|---|
| 1. *Blockchain* | 2. Saúde |
| 3. Algoritmo de Consenso | 4. Segurança |
| I. ENE/FT/UnB | II. Título (série) |

**REFERÊNCIA BIBLIOGRÁFICA**

FREIRE, F.R. (2024). *Byzantine Fault Tolerant to Health (BFT-H) - Proposta de um algoritmo de consenso direcionado às especificidades de sistemas de gestão de saúde baseados em Blockchain* . Dissertação de Mestrado Profissional, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 163 p.



FABRICIO RODRIGUES FREIRE
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

"Existe no mundo um único caminho por onde só tu podes passar. Onde leva? Não perguntes, segue-o!"

Friedrich Nietzsche

## AGRADECIMENTOS

A Deus, por nunca desemparar um filho rebelde e sempre apresentar respostas para situações aparentemente sem solução. Muito obrigado por me guiar nos momentos difíceis sem jamais soltar a minha mão.

Aos meus irmãos, Elenildo e Iuri, que sempre me apoiaram em todas as etapas da minha vida. Vocês são exemplos de retidão e conduta moral para mim, sem os quais eu nunca teria chegado até aqui. O amor que sinto por vocês é incondicional.

À minha irmanzinha Fernanda (*in memorian*), por sempre ter orgulho do irmão mais velho, mesmo que injustificado. O mundo ficou mais triste sem o seu sorriso e senso de humor peculiar. Nunca me esquecerei de você.

À minha mãe Helena (*in memorian*), por ter me dado a educação que não pode receber, amado, orientado, sofrido a cada vez que eu partia e festejado a cada chegada. Sua força foi uma inspiração para mim e com você entendi a importância do trabalho e de fazer sempre por merecer. Te amo demais.

Aos amigos Gueison e Vovó Monqunha, pelo carinho e orações constantes, me mostrando que não é preciso falar que gostam de você para que esse afeto seja sentido, e mesmo de longe torcem pela minha felicidade e vitórias.

À minha sogra Solange, meus cunhados Diego e Danillo, e "cunhada"Alana, por compartilhar refeições deliciosas e pelas conversas descontraídas, proporcionando momentos que me fizeram esquecer dos problemas, tornando a vida mais leve e agradável.

Ao amigo Nataniel Simich, pelo apoio diário, conselhos, e por fazer oportunidades se tornarem realidade, me incentivando a iniciar o mestrado e permitindo o ingresso na docência. Obrigado por sua amizade.

Aos meus orientadores, Professor Dr. Willian Ferreira Giozza e Professor Dr. Carlo Kleber Silva Rodrigues. Sinto uma profunda admiração pelos senhores, que me deram broncas e fizeram cobranças, mas sobretudo acreditaram em mim e me trouxeram a tranquilidade necessária para chegar até aqui. Obrigado mestres.

À amiga Suenia, que se tornou uma irmã neste curso. Obrigado pela presença e apoio em todos os momentos, ficando feliz com as conquistas e ajudando a carregar o peso nos desafios do PPEE. Sua garra e perseverança são motivadoras.

Ao amigo Alexandre Mundim, meu respeito e admiração, não só pela competência indiscutível, mas pelo autruísmo e dedicação ao outro, capaz de largar os curtos momentos de lazer e descanso com a família no fim de semana, para apoiar um companheiro em dificuldade. Obrigado por sua amizade.

Aos professores Georges, Demétrio e Rafael Rabelo, mestres com os quais tive a honra de conviver durante as disciplinas. A competência dos senhores vai muito além da área técnica e se faz presente na motivação com que ensinam, na atenção aos que estão em dificuldades e na bondade com que tratam seus alunos. Os senhores são uma inspiração para mim.

À amiga, conterrânea e ex-chefe Dayse Albuquerque, a minha admiração e respeito por tratar de ques-

tões sensíveis com tanta classe e não deixar que as adversidades a façam perder a sensibilidade ao olhar para o outro e por tratar tudo com competência, força e uma elegância quase sobrenatural. Obrigado pelo apoio e orientações.

E por fim, meu agradecimento à mulher da minha vida, minha doce Bárbara, pela compreensão ao ter momentos de lazer suprimidos, por me apoiar em cada loucura, torcer pelo meu sucesso, sofrer junto nas madrugadas em que eu estudava e cuidar de mim quando eu não conseguia fazer isso. Não só esse curso, mas a vida não seria possível sem você ao meu lado. Te amo mais que tudo.

## RESUMO

A gestão de dados em sistemas de gestão de saúde distribuídos baseados na tecnologia Blockchain apresenta uma complexidade significativa, necessitando da adoção de soluções que garantam a segurança, escalabilidade e consistência das informações. Este trabalho propõe o algoritmo BFT-H (*Byzantine Fault Tolerance to Health*), um algoritmo adaptativo de consenso bizantino concebido especificamente para sistemas de gestão de saúde distribuídos privados. A principal inovação do algoritmo proposto reside na utilização do mecanismo ADAN (*Adaptive Decentralized Asynchronous Node-slicing*) para reduzir a complexidade da comunicação entre os nós da rede de $O(n^2)$ para $O(n)$ por meio do fatiamento dinâmico da rede. O BFT-H também inclui um sistema de reputação que avalia o comportamento histórico e atual dos nós, viabilizando a formação adaptativa de grupos para o processamento paralelo das transações. A validação experimental foi realizada com o simulador NS3 em redes compostas por até 5.000 nós, utilizando métricas como o tempo de consenso, a latência e a vazão (*throughput*). Obteve-se por exemplo uma redução de 33% no tempo necessário para atingir o consenso com o BFT-H quando comparado ao algoritmo PBFT e 80% em relação ao algoritmo Raft. A latência com o BFT-H por exemplo apresentou diminuições de 27,5% e 63,8%, respectivamente, mesmo sob condições caracterizadas por alta latência. Por outro lado, a vazão com o BFT-H, em redes extensas, superou o PBFT em 31,4% e mais que dobrou quando comparado ao Raft, demonstrando conformidade com os requisitos de tolerância a falhas bizantinas. Os resultados obtidos reforçam a viabilidade do BFT-H como uma solução eficaz para o consenso distribuído em sistemas de gestão da saúde em larga escala, equilibrando eficiência operacional e segurança.

**Palavras-chave: *Blockchain*, Consenso Bizantino, Sistemas de Saúde Distribuídos, Fatiamento de Rede, Sistema de Reputação**

## ABSTRACT

*Data management in distributed healthcare management systems based on Blockchain technology presents significant complexity, requiring the adoption of solutions that ensure the security, scalability, and consistency of information. This work proposes the BFT-H (Byzantine Fault Tolerance to Health) algorithm, an adaptive Byzantine consensus algorithm designed specifically for private distributed healthcare management systems. The main innovation of the proposed algorithm lies in the use of the ADAN (Adaptive Decentralized Asynchronous Node-slicing) mechanism to reduce the communication complexity between network nodes from $O(n^2)$ to $O(n)$ through dynamic network sharding. The BFT-H protocol also includes a reputation system that evaluates the historical and current behavior of nodes, enabling the adaptive formation of groups for parallel transaction processing. The experimental validation was carried out using the NS3 simulator in networks of up to 5.000 nodes, using metrics such as consensus time, latency, and throughput. The results showed, for example, a 33% reduction in the time required to reach consensus with BFT-H compared to the PBFT algorithm, and an 80% reduction compared to the Raft protocol. Latency with BFT-H also presented decreases of 27.5% and 63.8%, respectively, even under conditions characterized by high latency. On the other hand, the throughput with BFT-H in extensive networks surpassed PBFT by 31.4% and more than doubled compared to Raft, while maintaining the guarantees associated with Byzantine security. The obtained results reinforce the viability of BFT-H as an effective solution for distributed consensus in large-scale healthcare management systems, balancing operational efficiency and security.*



***Keywords: Blockchain, Byzantine Consensus, Distributed Healthcare Systems, Network Slicing, Reputation System.***

# SUMÁRIO

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE QUADROS

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ADAN | Adaptive Decentralized Asynchronous Node-slicing |
| BFT | Byzantine Fault Tolerance |
| BFT-H | Byzantine Fault Tolerance to Health |
| CFT | Crash Fault Tolerance |
| DDoS | Distributed Denial of Service |
| DLT | Distributed Ledger Technology |
| EHR | Electronic Health Records |
| GDPR | General Data Protection Regulation |
| HIPAA | Health Insurance Portability and Accountability Act |
| HSHB | Hybrid Secure Health Blockchain IoMT - Internet of Medical Things |
| IoT | Internet of Things |
| LGPD | Lei Geral de Proteção de Dados |
| MSP | Membership Service Provider |
| NS3 | Network Simulator 3 |
| P2P | Peer-to-peer |
| PBFT | Practical Byzantine Fault Tolerance |
| PME | Prontuário Médico Eletrônico |
| PoS | Proof of Stake |
| PoW | Proof of Work |
| RTT | Round Trip Time |
| SUS | Sistema Único de Saúde |
| TPS | Transactions Per Second |
| UBS | Unidade Básica de Saúde |
| UPA | Unidade de Pronto Atendimento |
| VRF | Verifiable Random Function |

# 1 INTRODUÇÃO

## 1.1 CONTEXTUALIZAÇÃO DO TEMA

A gestão de dados em sistemas de gestão de saúde distribuídos enfrenta desafios crescentes relacionados à escalabilidade, segurança e consistência das informações. Por exemplo, o Sistema Único de Saúde (SUS), com sua extensa rede composta por mais de 5.500 hospitais, 40.000 Unidades Básicas de Saúde (UBS), 500 Unidades de Pronto Atendimento (UPA), 27 secretarias estaduais e 5.570 secretarias municipais de saúde, processa diariamente milhões de registros médicos e realiza aproximadamente 12 milhões de internações por ano (1). Esta infraestrutura heterogênea reflete a complexidade do sistema, onde unidades com diferentes capacidades tecnológicas precisam interoperar. Vão de pequenas UBS em áreas remotas até hospitais terciários em grandes centros urbanos.

## 1.2 PROBLEMA

A gestão de dados entre unidades de saúde enfrenta um desafio fundamental: como garantir que informações críticas dos pacientes estejam disponíveis e atualizadas em tempo real, independentemente do local de atendimento. Esta questão é particularmente relevante em situações de emergência, quando o acesso imediato ao histórico médico pode ser decisivo para o tratamento. No contexto do Sistema Único de Saúde (2), em que diferentes estabelecimentos precisam compartilhar informações constantemente, torna-se crucial desenvolver uma solução que funcione de maneira eficiente mesmo com infraestruturas tecnológicas distintas. A tecnologia *Blockchain* surge como uma alternativa para este cenário, principalmente por sua capacidade de manter registros seguros e inalteráveis de forma descentralizada. No entanto, sua aplicação em sistemas de gestão de saúde de grande escala encontra um obstáculo significativo: as limitações dos algoritmos de consenso existentes. Os algoritmos atualmente disponíveis não conseguem atender adequadamente às necessidades destes sistemas. O algoritmo PBFT (*Pratical Byzantine Fault Tolerant*) (3), por exemplo, torna-se inviável em redes maiores devido ao crescimento quadrático da complexidade de suas comunicações, enquanto o algoritmo *Raft* (4), embora de implementação mais simples, acaba criando gargalos ao concentrar todo o processamento em um único ponto.

## 1.3 OBJETIVOS

O objetivo central deste trabalho é propor um algoritmo de consenso bizantino que integre reputação e fatiamento adaptativo, sem sacrificar as garantias de segurança, a fim de aprimorar o desempenho dos sistemas de gestão de saúde baseados em *Blockchain*. Os objetivos específicos incluem o desenvolvimento de um mecanismo dinâmico para fatiamento da rede que mantenha as propriedades necessárias para tolerância a falhas bizantinas. Também está prevista a implementação de um sistema reputacional

que otimize tanto a seleção quanto a organização dos nós participantes, além do estabelecimento de um processo consensual em duas fases, capaz de preservar as garantias da segurança com menor sobrecarga comunicacional. Adicionalmente, busca-se validar experimentalmente o algoritmo em situações que simulem características típicas dos sistemas distribuídos na área de gestão de saúde, avaliando sua eficácia em condições operacionais realistas.

## 1.4 JUSTIFICATIVA

O desenvolvimento de um novo algoritmo de consenso bizantino específico para sistemas de gestão de saúde justifica-se pela crescente necessidade de compartilhamento seguro e eficiente de dados médicos entre diferentes instituições. A fragmentação atual dos registros de saúde, onde cada unidade mantém suas próprias bases de dados, dificulta a continuidade do cuidado e pode resultar em decisões clínicas baseadas em informações incompletas ou desatualizadas.

O BFT-H (*Byzantine Fault Tolerance to Health*) foi concebido para endereçar estas questões, apresentando uma arquitetura que se adapta às diferentes realidades do sistema de saúde brasileiro. Seu mecanismo ADAN (*Adaptive Decentralized Asynchronous Node-slicing*) reduz significativamente o overhead de comunicação entre os nós através do fatiamento dinâmico da rede, permitindo que mesmo unidades com recursos computacionais limitados possam participar efetivamente do sistema (5).

Além disso, a implementação de um sistema de reputação possibilita identificar e priorizar nós mais confiáveis, aspecto crucial em um ambiente onde a disponibilidade e integridade dos dados são essenciais. Esta característica é particularmente relevante durante emergências médicas ou surtos epidêmicos, quando o sistema precisa manter sua eficiência mesmo sob carga elevada.

A redução da complexidade de comunicação de $O(n^2)$ para $O(n)$ propiciada pelo BFT-H representa uma contribuição prática para viabilizar a implementação de sistemas *Blockchain* em larga escala no setor de gestão de saúde. Esta melhoria possibilita a criação de uma rede verdadeiramente integrada de prontuários eletrônicos, beneficiando diretamente a qualidade do atendimento ao paciente.

## 1.5 PUBLICAÇÕES RESULTANTES DESTA PESQUISA

Durante o desenvolvimento deste trabalho, foram publicados dois artigos científicos que apresentam resultados parciais da pesquisa:

- FREIRE, Fabricio R.; GIOZZA, William F.; RODRIGUES, Carlo K. da Silva. *Proposta de um Algoritmo de Consenso para Plataformas Blockchain em Sistemas de Gestão de Saúde Privados*. RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação, n. E57, p. 103-116, abr. 2023.
- FREIRE, Fabricio R.; GIOZZA, William F.; RODRIGUES, Carlo K. da Silva. *Towards Consensus Algorithm for Healthcare Management Systems in Blockchains*. Current Trends in Computer Sciences & Applications, v. 2, n. 4, p. 228-234, 2023.

## 1.6 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação está organizada em cinco capítulos, conforme descrito a seguir O Capítulo 1 apresentando esta introdução. O Capítulo 2 apresenta uma fundamentação teórica sobre *Blockchain*, algoritmos consensuais e suas aplicações nos sistemas de gestão de saúde. O Capítulo 3 detalha a arquitetura assim como os componentes do algoritmo proposto BFT-H. No Capítulo 4 são descritos os experimentos de avaliação de desempenho realizados seguidos por uma análise dos resultados obtidos. Por fim, no Capítulo 5 são apresentadas as conclusões deste trabalho juntamente com sugestões para futuros trabalhos de pesquisa nesta área.

# 2 REVISÃO DA LITERATURA

Neste capítulo são apresentados os principais aspectos que caracterizam a tecnologia *Blockchain*, contemplando seus conceitos fundamentais, os algoritmos de consenso e suas classificações, com ênfase em algoritmos baseados em votação como o PBFT (*Pratical Byzantine Fault Tolerant*)(3) e o *Raft* (4). A aplicação da tecnologia *Blockchain* no contexto de sistemas de gestão de saúde é aqui discutida, abordando as diferentes arquiteturas para registros eletrônicos e as soluções em países de alto índice populacional, sob requisitos de desempenho como escalabilidade, privacidade e segurança.

Este capítulo tem por objetivo contextualizar a proposta do algoritmo BFT-H (*Byzantine Fault Tolerant to Health*), justificando sua relevância no cenário atual da digitalização dos procedimentos de saúde pela aderência aos objetivos da Estratégia Global sobre Saúde Digital 2020-2025 da Organização Mundial da Saúde que enfatiza a importância de desenvolver soluções tecnológicas que aprimorem os sistemas de gestão de saúde de forma global (6).

A revisão da literatura neste capítulo fornece a base para entender os desafios atuais nos algoritmos de consenso da tecnologia *Blockchain* em sistemas de gestão de saúde. Inclui a definição de critérios para avaliar o desempenho, como tempo para consenso, latência e vazão (*throughput*). Os artigos estudados foram obtidos no período de 02 de setembro a 1º de novembro de 2024, acessando a consulta de periódicos disponível no portal da CAPES. Estes elementos teóricos fundamentam a proposta do algoritmo BFT-H, descrita no Capítulo 3 e a metodologia experimental empregada na avaliação de desempenho do BFT-H, utilizando o simulador NS3, apresentada no Capítulo 4.

## 2.1 TECNOLOGIA *BLOCKCHAIN*

### 2.1.1 Conceitos Fundamentais

A tecnologia *Blockchain*, introduzida em 2008 com a proposta do *Bitcoin* (7), representa uma inovação disruptiva na gestão de dados distribuídos. Fundamentalmente, uma *Blockchain* constitui uma estrutura de dados distribuída e imutável (Figura 2.1), que emprega técnicas criptográficas avançadas para criar uma lista encadeada de blocos de transações, formando um livro-razão (*ledger*) descentralizado e resistente a adulterações (8).

A tecnologia *Blockchain* representa uma ruptura na forma como gerenciamos informações digitais. Funcionando como um livro-razão distribuído, esta tecnologia permite que todas as transações sejam registradas de forma cronológica, transparente e imutável, sem a necessidade de uma autoridade central. Cada participante da rede mantém uma cópia completa deste registro, criando um sistema naturalmente resistente a falhas e adulterações. Esta característica descentralizada, combinada com a organização sequencial dos dados, estabelece um novo paradigma para a confiabilidade e segurança das transações digitais (9).

As transações são agrupadas em blocos, unidades maiores de armazenamento. Sua estrutura comumente contém um cabeçalho com metadados, como *timestamp*, versão do algoritmo, *hash* do bloco anterior e a raiz da árvore de Merkle das transações, além do corpo delas. Cada bloco está criptograficamente ligado

ao anterior via função *hash*, formando a "cadeia de blocos"(10). A Figura 2.1 exemplifica a estrutura de uma rede *Blockchain*.

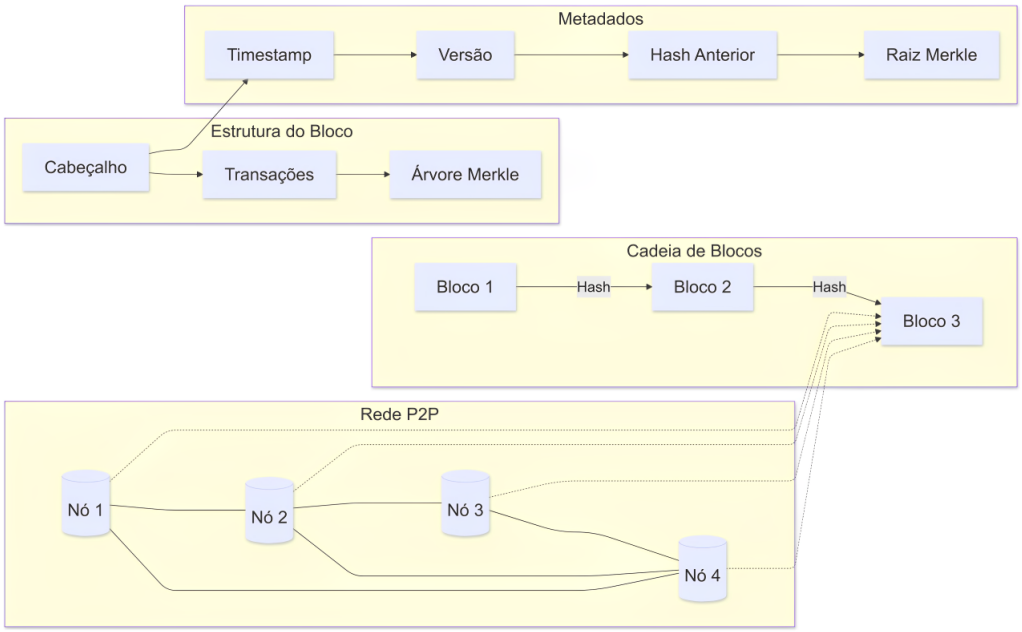


*Figura 2.1: Estrutura de uma rede Blockchain*

A infraestrutura da *Blockchain* é sustentada por uma rede *peer-to-peer* (P2P) que permite a interação direta entre os participantes e elimina a necessidade de intermediários. Nesse arranjo, cada integrante possui uma cópia atualizada de todos os registros, possibilitando que as transações sejam validadas coletivamente pela própria rede. Essa descentralização distribui não só a responsabilidade entre os envolvidos, mas também fortalece a segurança do sistema como um todo. O encadeamento criptográfico das informações em blocos consecutivos resulta em um registro praticamente imutável. Cada novo bloco está matematicamente interligado ao anterior, formando uma cadeia na qual qualquer tentativa de modificação seria rapidamente identificada pelos demais participantes da rede (11).

A transparência é obtida por meio do registro das prescrições de maneira distribuída e acessível a todos os participantes autorizados da rede, o que possibilita um acompanhamento seguro das transações independentemente de uma entidade central. No que se refere à rastreabilidade, o sistema documenta a origem e o destino das prescrições e dos medicamentos de forma imutável, permitindo a auditoria dos dados médicos com o consentimento do paciente (12).

No que diz respeito à permissão para uso, as redes de *Blockchain* podem ser divididas nas categorias permissionadas e não permissionadas. A primeira exige uma autorização prévia para a participação, proporcionando um controle de acesso mais rigoroso, além de geralmente oferecerem melhor eficiência e desempenho em termos de rapidez nas transações e consumo energético. A outra permite a participação de qualquer usuário, o que favorece uma maior descentralização, embora isso possa comprometer a eficiência e a velocidade das transações (13).

### 2.1.2 Algoritmos de Consenso

#### 2.1.2.1 Visão Geral

Os algoritmos de consenso são componentes fundamentais da tecnologia *Blockchain*, desempenhando um papel crucial na manutenção da consistência e na segurança da rede distribuída (14). Esses algoritmos permitem que os nós participantes da rede cheguem a um acordo sobre o estado atual do sistema, mesmo na presença de falhas ou nós maliciosos (15).

Os principais objetivos dos algoritmos de consenso incluem garantir a consistência dos dados, prevenir ataques maliciosos e manter a integridade da rede *Blockchain* (16). Existem diversos algoritmos de consenso, cada um com suas próprias características, vantagens e desvantagens (17).

Os algoritmos de consenso em *Blockchain* podem ser classificados em duas categorias principais: baseados em prova e baseados em votação. Na primeira categoria, encontram-se o *Proof of Work* (PoW)(18) e o *Proof of Stake* (PoS)(19). O PoW, utilizado pelo *Bitcoin*, exige que os participantes resolvam problemas computacionais complexos para validar transações, demandando significativo poder computacional e energia. Já o PoS seleciona validadores com base na quantidade de criptomoedas que possuem como garantia (*stake*), oferecendo maior eficiência energética. O *Ethereum* (20) é um bom exemplo disso. Na segunda categoria, os algoritmos baseados em votação como o *Practical Byzantine Fault Tolerance* (PBFT) e o *Raft* utilizam um sistema de votos entre os participantes para alcançar consenso, sendo mais adequados para redes permissionadas onde os participantes são conhecidos. Enquanto os algoritmos baseados em prova são mais escaláveis e adequados para redes públicas, os baseados em votação oferecem maior eficiência em velocidade de transação, mas com limitações de escalabilidade (21, 22, 23, 24).

A escolha do algoritmo de consenso adequado depende das necessidades específicas da aplicação, considerando fatores como o tamanho da rede, requisitos de desempenho, nível de descentralização desejado e tolerância a falhas (15, 17). Em sistemas de gestão de saúde baseados em *Blockchain*, por exemplo, fatores como privacidade e velocidade de transação são particularmente relevantes na seleção do algoritmo de consenso (16).

#### 2.1.2.2 Classificação

A classificação dos mecanismos de consenso em sistemas *Blockchain* envolve múltiplos critérios técnicos que refletem seu funcionamento e características operacionais. (25) propõem uma taxonomia hierárquica com cinco níveis: Falha, Decisão, Ordem, Tipo e Sistema, permitindo uma categorização mais granular e técnica. O nível de falha, fundamental para a robustez do sistema, distingue entre algoritmos com *Crash Fault Tolerance* (CFT) e *Byzantine Fault Tolerance* (BFT). Os algoritmos CFT são projetados para lidar com falhas simples onde os nós param de funcionar completamente (falha por parada), enquanto os algoritmos BFT podem tolerar comportamentos arbitrários e potencialmente maliciosos dos nós, incluindo a transmissão deliberada de informações incorretas ou contraditórias. Nível de Decisão os classifica quanto ao grau de centralização, variando de centralizados a descentralizados. Tipo identifica mecanismos específicos como *Proof of Work* (PoW), *Proof of Stake* (PoS) e *Practical Byzantine Fault Tolerance* (PBFT), enquanto o nível de Sistema se refere às implementações reais desses algoritmos em diferentes cadeias *Blockchains*. Além desta taxonomia, outros critérios técnicos são empregados na classificação dos algo-

ritmos de consenso. A finalidade distingue-os entre probabilísticos, como PoW, que oferecem garantia estatística de consenso, e determinísticos, como PBFT, que fornecem garantia imediata. O permissionamento diferencia algoritmos que permitem participação irrestrita daqueles que restringem a participação a nós pré-aprovados, afetando diretamente a arquitetura e o desempenho dos sistemas *Blockchain*, como discutido por (26). A escalabilidade é outro critério importante, que classifica os algoritmos com base em sua capacidade de escalar vertical ou horizontalmente. (14) abordam este aspecto ao propor um algoritmo baseado em *Double-DAG* (23) para melhorar a escalabilidade em ambientes *Industrial Internet of Things* (IIoT) heterogêneos, onde está enquadrada também sistemas para gestão da saúde. A eficiência energética tornou-se um critério de classificação importante, distinguindo algoritmos de alto consumo, como PoW, de alternativas mais eficientes, como PoS e PBFT. O mecanismo de seleção de líder é outro fator de classificação, diferenciando algoritmos que selecionam líderes para propor blocos, como PoW e PoS, daqueles que operam sem líderes explícitos, como PBFT. A latência de confirmação e o throughput categorizam algoritmos baseado em desempenho temporal. Eles realizam medições para confirmar transações irreversíveis e o número daquelas processadas por tempo, respectivamente. A resistência a ataques é um critério de classificação que avalia a capacidade dos algoritmos de resistir a diferentes tipos de ameaças, como ataques de 51% ou ataques *Sybil* (27). (28) abordam este aspecto ao otimizar o PBFT para cenários de transações de propriedade intelectual, demonstrando como as características de segurança podem ser adaptadas para aplicações específicas. Esta otimização resultou em uma redução de 30% no tempo de consenso e um aumento de 25% no *throughput* em comparação com o PBFT padrão, mantendo o nível de resistência a ataques bizantinos.

### 2.1.3 Algoritmos de Consenso baseados em votação

Algoritmos de votação em *Blockchain* fundamentam-se em um processo eleitoral entre os nós participantes para alcançar consenso, no qual os nós votam para eleger temporariamente um líder encarregado da proposição de novos blocos e da coordenação do processo de validação. Essa abordagem contrasta com métodos baseados em prova que empregam competição direta entre líderes através de recursos computacionais (como no *Proof of Work*) ou econômicos (como no *Proof of Stake*) para determinar quem possui o direito de propor o próximo bloco. (29) ressaltam que algoritmos de consenso baseados em votação são concebidos para ambientes com nós conhecidos, proporcionando tolerância a falhas bizantinas. A implementação desses algoritmos busca equilibrar segurança, eficiência e descentralização por meio da escolha de nós confiáveis para a validação de transações e criação de blocos. As características principais incluem menor consumo energético, maior velocidade nas transações e adaptabilidade a redes com participantes identificáveis.

#### 2.1.3.1 *Practical Byzantine Fault Tolerance (PBFT)*

O algoritmo *Practical Byzantine Fault Tolerance* (PBFT), originalmente proposto por (3), representa um marco significativo na evolução dos algoritmos de consenso distribuído. O PBFT opera em um modelo de sistema assíncrono e é capaz de tolerar até f nós bizantinos em uma rede de $3f + 1$ nós, garantindo

segurança e vivacidade sob condições de comportamento bizantino.

O algoritmo PBFT é estruturado em três fases principais conforme ilustrado na Figura 2.1: Pré-Preparo (*PRE-PREPARE*), Preparo (*PREPARE*) e Confirmação (*COMMIT*). Em *PRE-PREPARE*, o nó primário (Líder), selecionado mediante um mecanismo de rotação baseado no número da visão atual, envia uma mensagem contendo o número de sequência atribuído à requisição do cliente. As fases subsequentes de *PREPARE* e *COMMIT* envolvem a troca de mensagens entre os nós (Réplicas) para estabelecer consenso sobre a ordem das operações (30).

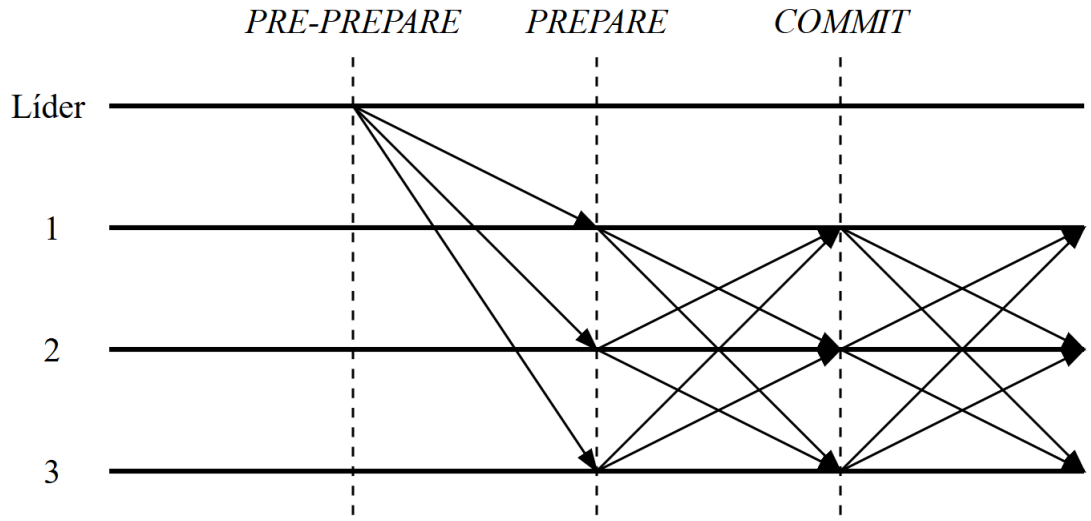


*Figura 2.2: Fluxo de mensagens do PBFT*

A complexidade de comunicação do PBFT é $O(n^2)$, onde $n$ é o número de nós na rede. Esta característica pode resultar em sobrecarga significativa em sistemas de larga escala, limitando sua aplicabilidade em ambientes com alto número de participantes (31). Para mitigar esta limitação, diversas otimizações têm sido propostas, incluindo abordagens de hierarquização de nós, onde estes são organizados em grupos menores com líderes que intermediam a comunicação entre grupos, e mecanismos de seleção de nós baseados em reputação.

(32) propõem uma variante do PBFT adaptada para ambientes de *edge computing*, denominada R-PBFT (*Reputation-based PBFT*). Esta abordagem introduz um mecanismo de reputação dinâmica para a seleção do nó primário e otimiza o protocolo de consistência, reduzindo as fases de consenso do algoritmo. A implementação do R-PBFT demonstrou uma redução de 12,2% no tempo de treinamento em comparação com o PBFT tradicional em um cenário de aprendizado federado descentralizado.

(33) apresentam o ST-PBFT (*Shard Transaction PBFT*), uma otimização do PBFT para cenários de transações de propriedade intelectual. O ST-PBFT implementa um mecanismo de fragmentação baseado em tipos de transação e introduz um grupo de coordenação de consenso para gerenciar o processo de validação entre os fragmentos. Esta abordagem demonstrou uma redução significativa na complexidade de comunicação e um aumento na eficiência do consenso.

(34) propõem uma versão leve do PBFT para aplicações de saúde baseadas em *Blockchain*. Esta implementação incorpora um modelo de confiança Eigen (35) para selecionar nós confiáveis para participar no consenso e utiliza uma Função Aleatória Verificável (VRF) para a seleção do nó primário. Os resultados

experimentais indicaram uma melhoria na eficiência do consenso e na tolerância a falhas em comparação com o PBFT tradicional.
Apesar das otimizações, propostas nos trabalhos discorridos acima, o PBFT e suas variantes ainda enfrentam desafios significativos em termos de escalabilidade e eficiência em redes de larga escala. A busca por soluções que equilibrem segurança, desempenho e descentralização continua sendo um campo ativo de pesquisa na área de sistemas distribuídos e *Blockchain*.

#### 2.1.3.2 *Raft*

O algoritmo *Raft*, proposto por (36), é um algoritmo de consenso projetado para ser mais compreensível que o algoritmo Paxos (37), mantendo as garantias de segurança e desempenho. Seu funcionamento baseia-se em três estados principais para os nós conforme ilustrado na Figura 2.3: Seguidor (*Follower*), Candidato (*Candidate*) e Líder (*Leader*). O mecanismo de eleição de líderes no *Raft* opera por meio de um sistema de temporização aleatória, conhecido como election timeout. Cada seguidor (*Follower*) é dotado de um temporizador que se inicia com um valor aleatório, geralmente variando entre 150 e 300 milissegundos. Caso um seguidor não receba qualquer sinal de vida (*heartbeat*) do líder vigente dentro desse intervalo, ele interpreta que o líder pode ter falhado ou se encontra inacessível. Nessa circunstância, o seguidor altera seu estado para candidato (*Candidate*) e dá início a uma nova eleição. A adoção de temporizações aleatórias é crucial para prevenir que múltiplos seguidores lancem eleições ao mesmo tempo, o que poderia resultar em uma divisão dos votos (*split vote*), impedindo assim que qualquer candidato alcance a maioria necessária para ascender à liderança. O algoritmo utiliza o conceito de "termo maior"como contador monotônico de períodos de liderança: quando um nó detecta um termo maior que o seu, ele automaticamente retorna ao estado de seguidor e atualiza seu termo, buscando estabelecer a unicidade da liderança através de mecanismos de consenso e prevenindo inconsistências após partições de rede. O Candidato vencedor que obtiver a maioria dos votos se torna o novo Líder (4).

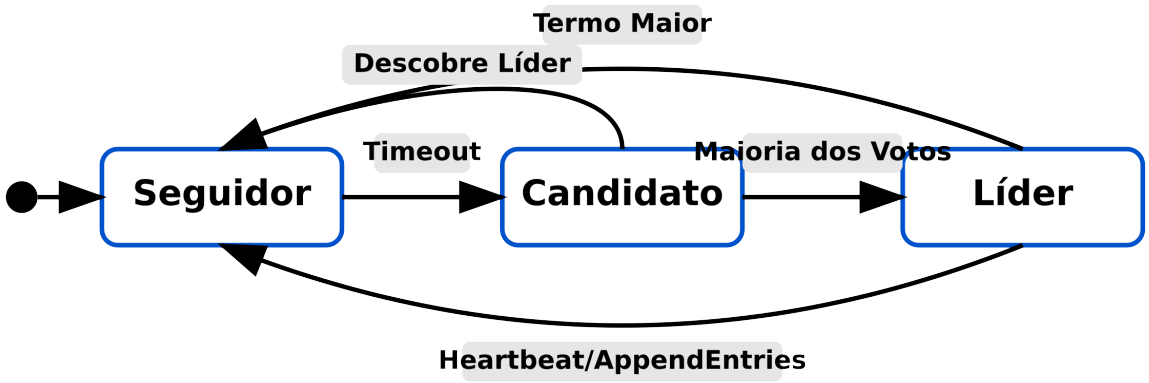


*Figura 2.3: Fluxo de mensagens do Raft*

O algoritmo *Raft* desagrega o problema do consenso em três subproblemas distintos: a replicação do histórico (*log*), a eleição de um líder e a segurança. No que tange à replicação do *log*, o líder recebe comandos dos clientes, os incorpora ao seu *log* e realiza a replicação para os seguidores. A manutenção da consistência é assegurada por meio de índices e termos: os índices são números sequenciais (iniciando em

1) que designam a posição de cada entrada no *log*, enquanto os termos representam contadores monotônicos que assinalam diferentes períodos de liderança. Um exemplo é quando um evento de um líder falhar e um novo for eleito, o termo é incrementado de 1 para 2. Essa interseção entre índice e termo possibilita ao líder identificar com precisão inconsistências - por exemplo, se um seguidor possui em seu índice 5 uma entrada correspondente ao termo 1, enquanto o líder apresenta neste mesmo índice uma entrada do termo 2, isso sinaliza uma divergência que requer correção (38). O algoritmo preserva a integridade do *log* através de verificações de consistência durante os processos de replicação e na eleição de novos líderes.

A segurança no *Raft* é assegurada por restrições nos processos de eleição e Confirmação (*Commit*). Um candidato só pode ser eleito se seu *log* contiver todos os registros confirmados ("*committed*") em termos anteriores. Além disso, o Líder só pode confirmar (*commit*) registros que foram criados durante seu mandato atual (seu próprio termo) - por exemplo, se um líder com termo 3 encontra entradas não confirmadas dos termos 1 e 2, ele deve primeiro replicar essas entradas para a maioria dos seguidores antes de poder confirmá-las. Esta restrição previne que diferentes líderes tomem decisões conflitantes sobre entradas de *log* de termos passados (39). Essas regras garantem que, uma vez confirmado, um registro permanecerá no *log* mesmo com mudanças de liderança.

O *Raft* introduz o conceito de consenso conjunto ("*joint consensus*") como um mecanismo para gerenciar alterações na composição do grupo de nós participantes do *cluster* de forma segura. Uma situação em que isso ocorre é quando se faz necessário adicionar ou remover servidores do grupo de consenso. Neste processo, o sistema transita por um estado intermediário onde as antigas e novas configurações coexistem, garantindo que não haja inconsistências durante a transição (4). Isso permite que o *cluster* seja reconfigurado de forma dinâmica sem comprometer a segurança ou a disponibilidade do sistema.

Uma característica importante do *Raft* é sua capacidade de lidar com falhas de rede e partições. O algoritmo mantém a consistência forte mesmo em cenários de partição de rede, garantindo que apenas um líder possa confirmar novos registros. Quando a partição é resolvida, os nós da partição minoritária seguem o líder da partição majoritária, mantendo a consistência global (38).

O desempenho do *Raft* é comparável ao de outros algoritmos de consenso, como o Paxos, mas com a vantagem de ser mais fácil de entender e implementar. Em termos de latência, o *Raft* requer apenas um *round trip* entre o líder e a maioria dos seguidores para fazer a confirmação de um registro, o que é eficiente para muitos casos de uso (39). No entanto, o *Raft* pode ter desafios de escalabilidade em *clusters* muito grandes devido à necessidade de comunicação com a maioria dos nós para cada operação de confirmação.

### 2.1.4 Principais ameaças em sistemas distribuídos baseados na tecnologia *Blockchain*

As ameaças à segurança em sistemas *Blockchain* podem ser classificadas conforme a camada em que atuam: camada de rede, camada de consenso e camada de aplicação, conforme evidenciado por (40). Na camada de rede, os principais ataques incluem *man-in-the-middle*, *Distributed Denial of Service* (DDoS), ataques de repetição e particionamento de rede. Ataques DDoS são particularmente críticos quando o tempo de resposta do consumidor de recursos excede a velocidade de envio das requisições do atacante, comprometendo assim a disponibilidade do sistema (41). Na camada de consenso, destacam-se três tipos principais de ataques: o *double spending*, em que um atacante tenta reutilizar a mesma unidade de valor em múltiplas transações; os ataques de 51%, que ocorrem quando um grupo controla mais da metade do poder computacional ou dos votos da rede, permitindo reverter ou bloquear transações; e os ataques de

*timestamp*, nos quais invasores manipulam a ordem temporal das transações para obter benefícios ilícitos (42). Na camada de aplicação, as vulnerabilidades mais comuns concentram-se em *smart contracts* e no acesso não autorizado. Ataques dessa natureza podem ser representados como $S(C) \rightarrow S'(C)$, onde $S(C)$ representa o estado do contrato e $S'(C)$ o estado atualizado após a exploração de fragilidades como overflow de inteiros e reentrada (43). Outra característica importante trazida por essas explorações é a sua natureza evolutiva. As ameaças, também conhecidas como vetores ou superfície de ataque, têm caráter dinâmico e exigem atualizações regulares nas defesas. Tal adaptabilidade se aplica especialmente aos contratos inteligentes que precisam evoluir à medida que surgem novas vulnerabilidades. A combinação destes vetores de ataque remete ao conceito de ataque em profundidade apresentado por (40). A seguir são explicados os conceitos de defesa em profundidade, estratégia que possibilita a implementação de múltiplos níveis de proteção para mitigar riscos em sistemas *Blockchain*. Em seguida, retoma-se o tema das ameaças, abordando suas características marcantes e as formas de mitigação.

## 2.2 *BLOCKCHAIN* APLICADA AO CONTEXTO DA SAÚDE

### 2.2.1 Arquiteturas *Blockchain* para Registros Eletrônicos de Saúde

O *Healthchain*, proposto por (44), implementa criptografia homomórfica e controle de acesso baseado em atributos para o compartilhamento seguro de dados de saúde em larga escala. Seu mecanismo de consenso distribuído e esquema de assinatura em anel para autenticação permitem aos pacientes manterem o controle sobre suas informações médicas, facilitando simultaneamente o acesso autorizado por profissionais de saúde.

(45) desenvolveram o *SS-Health*, integrando *Blockchain* e computação de borda. Este sistema utiliza contratos inteligentes para governar o compartilhamento de dados e um mecanismo de consenso personalizado otimizado para redes de saúde. O *SS-Health* viabiliza o monitoramento remoto de pacientes e a correlação de eventos médicos críticos entre instituições, aprimorando a colaboração e resposta a emergências.

No âmbito da Internet das Coisas Médicas (IoMT), o *BAKMP-IoMT*, proposto por (46), emprega criptografia de curva elíptica e funções *hash* para estabelecer canais de comunicação seguros entre dispositivos médicos implantados, servidores pessoais e em nuvem. Seu mecanismo de prova de trabalho leve para consenso na *Blockchain* assegura a integridade dos dados e resistência a ataques como *man-in-the-middle* e *replay*.

(47) apresentaram o *BCHealth*, uma arquitetura que utiliza duas cadeias separadas para políticas de acesso e transferências de dados. O sistema implementa um esquema de criptografia baseado em atributos para proteção de privacidade e uma estratégia de clustering para lidar com escalabilidade e latência.

A arquitetura descentralizada de (48), baseada no *Hyperledger Fabric* (49), conecta três organizações hospitalares por um único canal. O sistema implementa *Membership Service Provider (MSP)* para controle de acesso, múltiplos contratos inteligentes para diferentes *stakeholders*, certificados X.509 para autenticação e armazenamento de dados em *CouchDB* e no *Blockchain*. Essa arquitetura aborda questões de interoperabilidade, permitindo o compartilhamento seguro de dados entre hospitais (48).

### 2.2.2 Soluções baseadas em *Blockchain* em países de alto índice populacional

Em países com alta densidade populacional, soluções *Blockchain* têm sido propostas para otimizar a gestão de registros eletrônicos de saúde (EHR). (50) desenvolveram um sistema baseado no *Hyperledger Fabric* para o contexto indiano, implementando uma rede *Blockchain* privada com múltiplos hospitais como nós. O sistema utiliza contratos inteligentes para gerenciar permissões e acesso aos dados, armazenamento *off-chain* para imagens médicas volumosas e o mecanismo de consenso *Practical Byzantine Fault Tolerance (PBFT)*. Experimentos demonstraram eficiência no processamento de transações, com tempos de resposta na ordem de milissegundos para a maioria das operações.

Na Malásia, (51) analisaram a implementação de *EHRs* baseados em *Blockchain*, examinando iniciativas governamentais como o *National Digital Health Blueprint* e o *National Health Stack*. Os autores propõem um modelo conceitual para integração de *Blockchain* no sistema de saúde malaio, visando melhorar a troca de informações de saúde e o controle do paciente sobre seus dados médicos. Entretanto, identificaram barreiras à adoção, incluindo resistência dos profissionais de saúde, necessidade de reformas regulatórias e desafios técnicos de integração com sistemas legados.

(52) realiza uma comparação entre duas plataformas destinadas ao gerenciamento de prontuários médicos eletrônicos (PMEs) no âmbito do Sistema Único de Saúde (SUS), ambas fundamentadas na tecnologia *Blockchain*. A principal distinção está no algoritmo de consenso empregado, pois uma das plataformas faz uso do algoritmo de consenso por votação (PBFT), enquanto a outra opta pelo consenso com um algoritmo baseado em provas (PoW). O estudo analisa os critérios de eficiência, disponibilidade, integridade e confidencialidade, chegando à conclusão de que a plataforma que utiliza o algoritmo de votação é mais eficiente e menos resiliente, enquanto a plataforma que utiliza o PoW oferece maior disponibilidade. O trabalho enfatiza a utilização da tecnologia *Blockchain* como um meio para aprimorar a segurança e a disponibilidade dos PMEs dentro do SUS, levando em consideração as implicações práticas da Lei Geral de Proteção de Dados (LGPD)(53).

### 2.2.3 Consenso e Escalabilidade

(29) propõem um método de compartilhamento seguro de dados de saúde baseado em *Blockchain* híbrida (HSHB). Este método divide as transações de compartilhamento em cadeia privada e cadeia de aliança, de acordo com diferentes entidades de compartilhamento. O estudo apresenta uma política de controle de acesso a dados de saúde (HDAC) com base na identidade e no propósito dele.

Os autores implementam um índice baseado em árvore B+ para consultas eficientes em dados históricos de saúde. A árvore B+ é uma estrutura de dados hierárquica que otimiza o acesso através de um arranjo balanceado de chaves. Seus registros são armazenados nas folhas e conectados por ponteiros. Este índice possibilita buscas e consultas de intervalo com complexidade logarítmica. O sistema também implementa reencriptação *proxy* para viabilizar o compartilhamento seguro de dados entre instituições. Esta técnica permite a transformação de dados cifrados entre diferentes chaves públicas sem exposição do conteúdo em texto claro (54). Assim, é possível reencriptar dados sem comprometer as chaves criptográficas originais.

Os contratos inteligentes são usados para processar dados sensíveis antes de serem compartilhados com terceiros. O estudo também propõe um mecanismo de verificação de integridade de dados que permite aos usuários verificarem se os dados foram adulterados ou descartados pelo servidor em nuvem.

Os resultados experimentais mostram que o esquema DLT (*Distributed Ledger Technology*) com *Blockchain* híbrida (HSHB) tem vantagens significativas em termos de sobrecarga computacional, *throughput* e latência quando comparado com sistemas DLT baseados em *Ethereum* (55). Por exemplo, o HSHB alcançou um *throughput* de cerca de 100 transações por segundo (TPS) com 100 nós, enquanto o *Ethereum* ficou abaixo de 20 TPS nas mesmas condições.

### 2.2.4 Privacidade e Segurança

(56) propõem uma *Blockchain* de provisionamento de soberania de dados para serviços de saúde remotos. O sistema utiliza uma abordagem de *Blockchain* híbrida, combinando cadeias públicas e privadas. A cadeia pública armazena informações não sensíveis sobre médicos e hospitais, enquanto a cadeia privada contém dados sensíveis dos pacientes, diagnósticos e histórico de prescrições.

Foram implementados diversos mecanismos de segurança, como uma política de pseudônimos para proteger a identidade dos pacientes, a delegação controlada de permissões, que permite aos médicos acessarem os dados de pacientes mediante liberação. Também é estabelecida mecanismos de criptografia para proteção das informações constantes nos prontuários e assinaturas digitais para garantir a autenticidade e integridade das informações.

O sistema proposto utiliza contratos inteligentes para automatizar o processamento de dados sensíveis antes do compartilhamento. Os autores argumentam que sua abordagem fornece um equilíbrio entre transparência e privacidade, permitindo o compartilhamento seguro de dados de saúde entre diferentes instituições.

Apesar do potencial transformador da *Blockchain* na área da saúde, existem desafios significativos e limitações que precisam ser superados para uma implementação bem-sucedida. Estes desafios abrangem aspectos técnicos, operacionais e regulatórios.

Um dos principais desafios técnicos é a escalabilidade. (57) destacam que, à medida que o volume de transações aumenta, a performance da rede *Blockchain* pode ser afetada. Em seus experimentos com *Hyperledger Fabric*, os autores observaram que o aumento no número de nós e transações pode levar a um aumento na latência e uma diminuição no *throughput*. Isso sugere que a escalabilidade continua sendo um obstáculo significativo para a adoção em larga escala de soluções *Blockchain* em saúde.

A interoperabilidade entre diferentes sistemas *Blockchain* e sistemas legados de saúde é outro desafio crucial. (58) aponta que a falta de padrões unificados para implementação de *Blockchain* em saúde pode levar à criação de silos de informação, dificultando a troca eficiente de dados entre diferentes instituições e sistemas. Isso pode limitar o potencial da *Blockchain* para melhorar a colaboração e a continuidade do cuidado em saúde.

A segurança e privacidade dos dados, embora sejam pontos fortes da *Blockchain*, também apresentam desafios únicos no contexto da saúde. (40) abordam a complexidade de garantir a segurança em ambientes de Internet das Coisas Médicas (IoMT), onde a integração de dispositivos IoT introduz novos vetores de ataque que precisam ser cuidadosamente mitigados. A necessidade de equilibrar a transparência inerente da *Blockchain* com os requisitos de privacidade dos dados de saúde é um desafio contínuo.

O consumo de energia é outra preocupação significativa, especialmente para implementações de *Blockchain* que utilizam mecanismos de consenso intensivos em computação, como o Proof of Work. (58) menciona que isso pode entrar em conflito com os objetivos de sustentabilidade de muitas organizações de saúde.

Além disso, há questões regulatórias e de conformidade que precisam ser abordadas. As leis de proteção de dados e privacidade, como o GDPR (*General Data Protection Regulation*) (59) na Europa e a HIPAA (*Health Insurance Portability and Accountability Act of 1996*) (60) nos Estados Unidos, impõem requisitos estritos sobre como os dados de saúde podem ser armazenados e compartilhados. Adaptar os sistemas *Blockchain* para cumprir essas regulamentações, mantendo suas características essenciais, é um desafio complexo (58).

## 2.3 SÍNTESE E IMPLICAÇÕES

A revisão da literatura evidencia desafios significativos no uso de algoritmos de consenso bizantino em sistemas de saúde distribuídos. O PBFT, embora ofereça garantias robustas de segurança, apresenta complexidade de comunicação $O(n^2)$ que limita sua escalabilidade. O *Raft*, apesar de sua complexidade linear $O(n)$, sofre com gargalos devido ao processamento sequencial e centralizado no líder. Estas limitações são particularmente críticas no contexto do SUS, que necessita coordenar milhares de unidades de saúde com diferentes capacidades computacionais.

As soluções existentes para registros eletrônicos de saúde, como o *Healthchain* e o *SS-Health*, enfatizam a importância da privacidade e do controle de acesso, mas não endereçam adequadamente o desafio da escalabilidade do consenso. Implementações em países com alta densidade populacional, como demonstrado por (50) na Índia, evidenciam que o PBFT tradicional, mesmo em redes privadas, apresenta limitações quando o número de nós aumenta significativamente.

O BFT-H propõe-se a preencher esta lacuna através de três inovações principais. Primeiro, introduz o mecanismo ADAN (*Adaptive Decentralized Asynchronous Node-slicing*) que mantém a complexidade $O(n)$ do *Raft*, mas elimina o gargalo da liderança centralizada através do fatiamento dinâmico da rede. Segundo, implementa um sistema de reputação que otimiza a formação dos grupos de consenso, característica ausente tanto no PBFT quanto no *Raft*. Terceiro, estabelece um processo de consenso em duas fases que preserva as garantias bizantinas do PBFT com menor overhead de comunicação.

Em relação aos requisitos específicos de sistemas de saúde, o BFT-H alinha-se às necessidades identificadas por (29, 56) para compartilhamento seguro de dados médicos. Seu mecanismo de fatiamento adaptativo permite que unidades com diferentes capacidades computacionais participem efetivamente da rede, enquanto o sistema de reputação fornece uma camada adicional de segurança para identificação de comportamentos maliciosos.

A proposta se diferencia das otimizações existentes do PBFT, como o R-PBFT e ST-PBFT, por oferecer uma solução integrada que aborda simultaneamente escalabilidade, segurança e adaptabilidade às características heterogêneas do sistema de saúde. Em vez de focar apenas na redução do número de fases ou na fragmentação estática, o BFT-H implementa um mecanismo dinâmico que se ajusta às condições da rede e ao comportamento dos participantes.

Embora não seja o foco deste trabalho, esta abordagem mostra-se particularmente relevante para o cenário brasileiro, onde o SUS necessita integrar mais de 45.000 unidades de saúde com diferentes níveis de infraestrutura tecnológica. O BFT-H propõe endereçar este desafio mantendo as garantias de segurança necessárias para dados sensíveis de saúde, enquanto oferece a escalabilidade e adaptabilidade requeridas

para operação em larga escala.

# 3 ALGORITMO PROPOSTO - BFT-H

Este capítulo apresenta a proposta de um algoritmo de consenso bizantino adaptativo, o BFT-H (*Byzantine Fault Tolerance to Health*) desenvolvido especificamente para sistemas distribuídos de gestão de saúde baseados em *Blockchain*.

## 3.1 CARACTERÍSTICAS GERAIS DO BFT-H

### 3.1.1 Objetivos do BFT-H

O algoritmo BFT-H foi desenvolvido para contornar as limitações dos algoritmos bizantinos convencionais em redes de grande escala, com especial ênfase em sistemas de gestão de saúde distribuídos. O BFT-H foi especificado para atender aos seguintes objetivos (Figura 3.1):

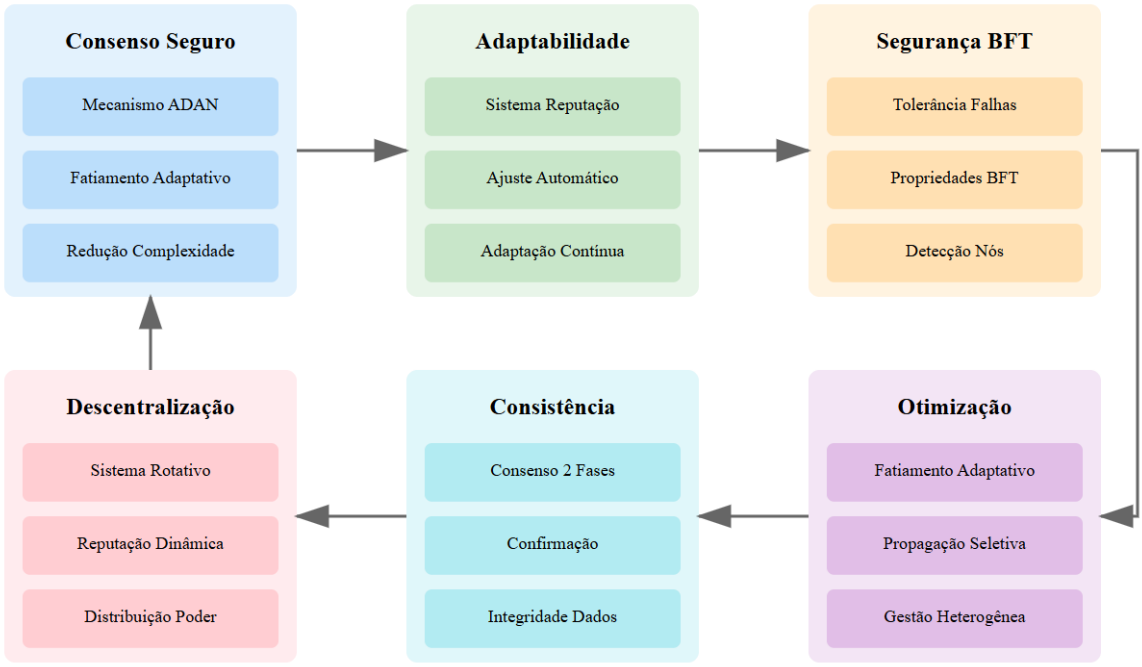


*Figura 3.1: Objetivos do BFT-H*

a) Consenso Tolerante a Falhas: propor mecanismos para um consenso tolerante a falhas bizantinas e eficaz em estruturas amplas, utilizando o mecanismo ADAN (*Adaptive Decentralized Asynchronous Node-slicing*) que aplica o fatiamento adaptativo dos nós disponíveis. Essa estratégia diminui a complexidade da comunicação quadrática típica dos mecanismos de consenso BFT para um nível mais manejável, permitindo que o algoritmo mantenha seu desempenho mesmo à medida que a rede se expande.

b) Adaptabilidade: garantir uma adaptabilidade dinâmica às transformações na rede. Os sistemas de

saúde distribuídos são frequentemente caracterizados por variações nas cargas e mudanças frequentes nas condições dos nós participantes. O BFT-H atende a essa demanda através de seu sistema de reputação dinâmica e mecanismos automáticos de ajuste, possibilitando que o algoritmo se adeque continuamente às alterações nas condições da rede sem sacrificar sua eficiência ou segurança.

c) Segurança BFT: buscar atender aos requisitos de tolerância a falhas bizantinas. Apesar das inovações introduzidas em termos de escalabilidade e adaptabilidade, o BFT-H mantém as propriedades de segurança associadas aos algoritmos BFT tradicionais. O BFT-H é capaz de tolerar até

$$f = (n - 1) \tag{3.1}$$

nós bizantinos, assegurando a precisão do consenso mesmo diante da presença de nós maliciosos ou falhos.

d) Otimização: otimizar o uso dos recursos computacionais e da rede. Através do fatiamento adaptativo e da propagação seletiva das mensagens, o algoritmo reduz a sobrecarga tanto na comunicação quanto no processamento em cada nó participante. Tal otimização é especialmente significativa em ambientes heterogêneos onde os nós podem apresentar capacidades computacionais e de comunicação distintas.

e) Consistência: assegurar a consistência e a finalidade das operações dentro de um contexto distribuído. O algoritmo implementa um mecanismo de consenso em duas fases que garante que todas as operações sejam executadas consistentemente e que seus resultados sejam definitivos assim que confirmados pela rede. Essa garantia é crucial para sistemas de saúde, nos quais a integridade e a consistência dos dados são essenciais.

f) Descentralização: promover uma descentralização efetiva por meio de um sistema rotativo de liderança fundamentado em reputação. Este mecanismo evita a concentração do poder decisório entre um grupo fixo de nós, ao mesmo tempo em que preserva a eficiência do processo consensual. A seleção dos líderes baseia-se em critérios objetivos relacionados ao desempenho e à confiabilidade dos participantes.

### 3.1.2 Premissas do BFT-H

O BFT-H opera sob um conjunto de premissas que definem as condições necessárias para sua operação em sistemas distribuídos de gestão de saúde:

i) Modelo de Comunicação - modelo de sistema distribuído assíncrono com $n$ nós participantes, onde cada nó possui um par único de chaves criptográficas para autenticação e assinatura digital. Esta escolha endereça os desafios de latência variável e a conectividade intermitente em redes de saúde distribuídas. Conectividade - cada nó deve manter conexões com um subconjunto significativo da rede, garantindo que mesmo após o fatiamento, exista conectividade suficiente para a propagação de mensagens. As conexões entre nós devem suportar comunicação bidirecional, permitindo tanto o envio quanto o recebimento de mensagens. O protocolo de comunicação não exige sincronização global de relógios, mas assume que os nós possuem relógios locais com deriva limitada.

ii) Requisitos Criptográficos - cada nó deve implementar mecanismos para proteção de suas chaves criptográficas e ser capaz de verificar as assinaturas digitais de outros nós. O sistema de reputação demanda que os nós mantenham registros locais das interações e comportamentos observados, permitindo o cálculo e a atualização das pontuações de reputação. A integridade das mensagens deve ser garantida por mecanismos criptográficos.

iii) Adaptabilidade - para o funcionamento do mecanismo de fatiamento ADAN - cada nó deve ser capaz de manter e atualizar sua pontuação de reputação localmente, calcular e validar o fatiamento da rede de forma independente, propagar informações de fatiamento para outros nós e participar do processo de consenso quando selecionado.

iv) Recursos - os nós devem atender aos requisitos mínimos de recursos computacionais e capacidades de armazenamento, necessários para processar e verificar assinaturas digitais, manter histórico local de operações e estados, executar cálculos de fatiamento e reputação e participar efetivamente do processo de consenso.

v) Disponibilidade - os nós devem manter um nível mínimo de tempo online para se envolverem efetivamente no consenso. A falha em manter essa disponibilidade afeta diretamente a reputação do nó e sua participação nas fatias ativas.

vi) Recuperação de falhas - o sistema assume a existência de mecanismos para recuperação de falhas, permitindo que nós que experimentem desconexões temporárias possam se recuperar e sincronizar seu estado com a rede. A recuperação deve ser realizada sem comprometer a segurança ou a consistência do sistema.

A Tabela 3.1 resume as premissas e os requisitos operacionais fundamentais para o funcionamento do BFT-H, associando-os diretamente com os problemas enfrentados em sistemas de gestão de saúde baseados em Blockchain.

*Tabela 3.1: Requisitos do BFT-H e Problemas típicos em Sistemas de Saúde*

| Categoria | Requisito | Problema |
|---|---|---|
| Modelo de Sistema | Rede Assíncrona: Tolerância a atrasos, $$n > 3f + 1 \quad (3.2)$$ nós, $$f = \frac{n-1}{3} \quad (3.3)$$ nós bizantinos | Latência variável entre unidades de saúde geograficamente distribuídas e problemas de conectividade em áreas remotas |
| Conectividade | Comunicação Resiliente:<br>Canais ponto a ponto<br>Tempo mínimo online<br>Recuperação pós-falhas | Continuidade do serviço durante emergências médicas e necessidade de acesso ininterrupto a dados críticos de pacientes |
| Segurança | Autenticação e Integridade: Par de chaves por nó, Assinaturas digitais e Validação de mensagens | Necessidade de garantir privacidade dos dados dos pacientes (LGPD/ HIPAA/ GDPR) e prevenir alterações não autorizadas em registros médicos |
| Adaptabilidade | Adaptabilidade: Fatiamento dinâmico, Sistema de reputação e Propagação eficiente | Variação na demanda de serviços de saúde (picos de emergência, surtos epidêmicos) e necessidade de escala dinâmica |
| Recursos | Capacidade Computacional: Processamento criptográfico, Armazenamento local e Gestão de estado | Heterogeneidade de infraestrutura entre diferentes instituições de saúde (hospitais grandes versus unidades de saúde básica) |
| Disponibilidade | Manutenção de fatias ativas com: $$\|S\| \geq \left\lfloor \frac{\|N\|}{2} \right\rfloor + 1 \quad (3.4)$$ nós e monitoramento contínuo do tempo de atividade através do sistema de reputação | Diversidade de infraestruturas no SUS, que inclui desde pequenas UBS em áreas remotas até hospitais terciários em grandes centros, exigindo que o sistema se mantenha operacional mesmo com variações na disponibilidade das unidades |
| Recuperação de Falhas | Monitoramento contínuo das fatias com detecção de mudanças e preservação de operações em andamento durante reorganizações | Alta demanda por disponibilidade contínua em sistemas de saúde, onde falhas podem impactar diretamente o atendimento ao paciente, especialmente em emergências ou durante encaminhamentos entre unidades |

## 3.2 ARQUITETURA DO BFT-H

A arquitetura do BFT-H foi projetada para atender aos requisitos específicos de sistemas de saúde baseados em *Blockchain*, oferecendo uma estrutura modular e adaptativa que permite o consenso eficiente em redes de larga escala. A arquitetura do BFT-H integra quatro componentes principais, ADAN, Sistema de Reputação Dinâmica, Eleição de Líder e Fases de Consenso que trabalham em conjunto para fornecer as garantias necessárias de segurança, escalabilidade e desempenho.

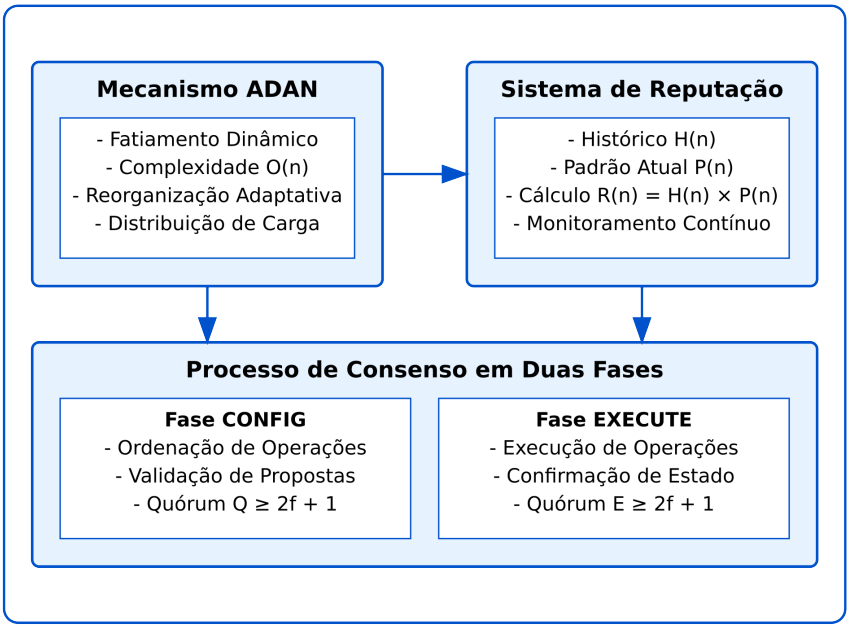


*Figura 3.2: Arquitetura do BFT-H*

No início desta arquitetura está o mecanismo ADAN, responsável pelo fatiamento adaptativo da rede que trabalha em conjunto com um Sistema de Reputação para otimizar a participação dos nós. Estes componentes são complementados por um mecanismo de Eleição de Líder baseado em reputação e realizada de forma assíncrona, e por um mecanismo de Consenso em duas fases (*CONFIG* e *EXECUTE*) que garantem a ordenação e a execução consistente das operações na rede.

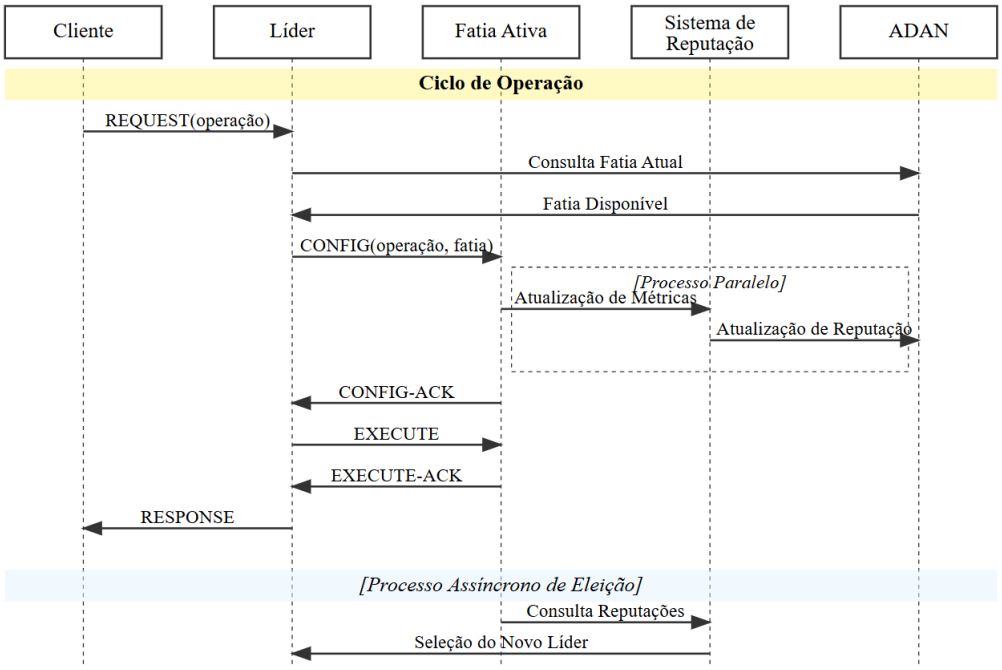


*Figura 3.3: Fluxo de comunicação do BFT-H*

O diagrama mostrado na Figura 3.3 representa o fluxo completo de um processo de consenso no algoritmo BFT-H, incluindo o processo assíncrono de escolha do líder. O ciclo se inicia quando um Cliente envia uma requisição (*REQUEST*) ao Líder atual.

Antes de dar andamento à operação, o Líder consulta o mecanismo ADAN para identificar a fatia ativa de nós disponíveis. Com a fatia configurada, o Líder inicia a fase *CONFIG*, enviando a proposta da operação para a Fatia Ativa. Ao mesmo tempo, o Sistema de Reputação atualiza suas métricas com base no comportamento dos nós e comunica ao ADAN sobre possíveis reconfigurações. A Fatia Ativa responde com *CONFIG-ACK*, permitindo que o Líder prossiga para a fase *EXECUTE*. Após receber o *EXECUTE-ACK* da Fatia Ativa, o Líder encaminha a resposta final ao Cliente (*RESPONSE*).

De maneira assíncrona e após concluir o consenso, ocorre a eleição do novo líder, onde se consulta o Sistema de Reputação para escolher, entre os nós com maior pontuação de reputação, aquele que assumirá a liderança no próximo ciclo de consenso.

Cada um destes componentes arquiteturais, suas interações e o fluxo de dados entre eles, são detalhados a seguir, destacando como a arquitetura do BFT-H endereça os desafios específicos encontrados em sistemas de gestão de saúde distribuídos. A integração harmoniosa destes elementos procura equilibrar as necessidades conflitantes de segurança, eficiência e escalabilidade.

### 3.2.1 *Adaptive Decentralized Asynchronous Node-slicing* (ADAN)

A principal inovação do BFT-H reside no uso do mecanismo denominado ADAN (*Adaptive Decentralized Asynchronous Node-slicing*), que, apesar de manter uma complexidade assintótica linear $O(n)$, reduz consideravelmente o número total de mensagens trocadas ao operar com aproximadamente a metade dos nós ($\frac{n}{2}$) em cada ciclo de consenso.
Essa redução, embora não modifique a classe de complexidade do algoritmo BFT-H, produz um efeito prático significativo quando comparado com algoritmos de consenso tradicionais. Por exemplo, em uma rede composta por 1000 nós, o PBFT tradicional demanda cerca de 1.000.000 mensagens em cada rodada de consenso $\mathcal{O}(n^2)$, enquanto o BFT-H necessita apenas de aproximadamente 500 mensagens por rodada ($\frac{n}{2}$) conforme mostrado na Tabela 3.2. Tal otimização reveste-se de especial importância nos sistemas de gestão de saúde distribuídos, tendo em vista que a eficiência da comunicação é fundamental para o desempenho do sistema.

| **Algoritmo** | **Complexidade Teórica** | **Mensagens para n=1000** |
|---|---|---|
| PBFT | $O(n^2)$ | $\sim$1.000.000 |
| *Raft* | $O(n)$ | $\sim$1.000 |
| BFT-H | $O(n)$ | $\sim$500 |

*Tabela 3.2: Complexidade e eficiência da comunicação do BFT-H*

O processo completo do sistema de reputação e fatiamento da rede, executado pelo ADAN consiste das seguintes etapas, conforme mostrado pelo diagrama contido na Figura 3.4:

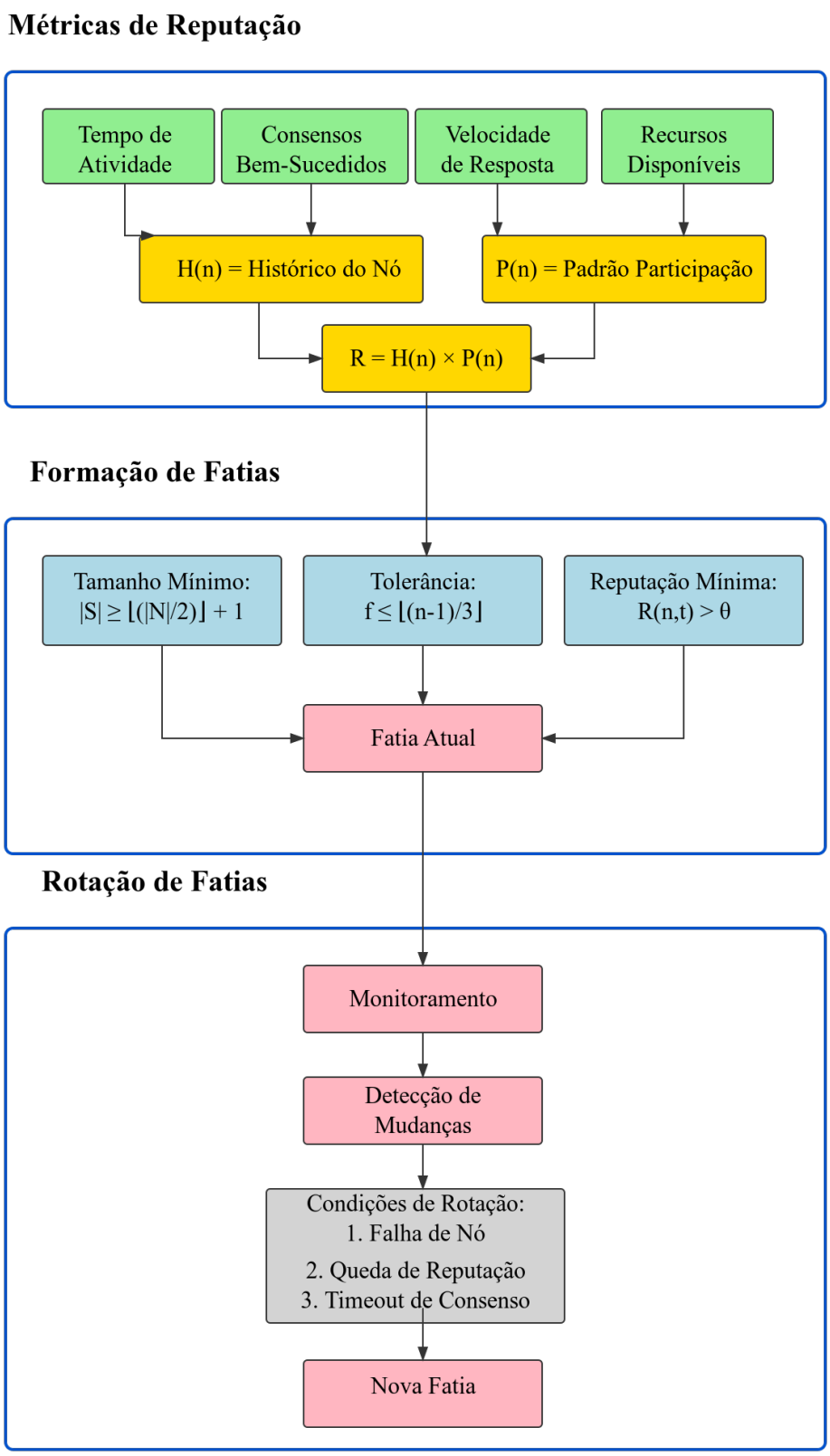


*Figura 3.4: Processo de fatiamento da rede pelo ADAN*

a) Coleta de Métricas de Reputação – o processo inicia-se com a coleta contínua de quatro métricas fundamentais: o tempo de atividade do nó, a quantidade de consensos bem-sucedidos, a velocidade

média de resposta e os recursos computacionais disponíveis.

b) Cálculo de Reputação - as métricas de reputação alimentam o processo de cálculo da reputação do nó que considera duas dimensões principais: seu histórico acumulado $H(n)$ e seu padrão atual de participação $P(n)$. O histórico $H(n)$ é construído a partir das quatro métricas coletadas, enquanto o padrão de participação $P(n)$ reflete seu comportamento mais recente. A reputação final $R$ é obtida pela multiplicação destes dois fatores, conforme será visto em detalhes na Subseção 3.2.2.

c) Formação de Fatias - a reputação calculada influencia diretamente a formação das fatias no sistema, que deve atender a três critérios fundamentais:

i) cada segmento deve manter um tamanho mínimo de

$$|S| \geq \left\lfloor \frac{|N|}{2} \right\rfloor + 1 \text{ nós.} \tag{3.5}$$

ii) o sistema deve tolerar até f falhas bizantinas, em que

$$f \leq \left\lfloor \frac{n-1}{3} \right\rfloor \tag{3.6}$$

iii) apenas nós cujas reputações $R(n,t)$ superem um limiar $\theta$ podem participar das fatias.

d) Rotação de Fatias - Por fim, o diagrama (Figura 3.4) mostra o processo contínuo de rotação das fatias. O segmento atual é continuamente monitorado para detecção de alterações relevantes. Quando ocorrem eventos como falha de nó, queda acentuada de reputação ou timeout de consenso, uma nova segmentação é formada. Durante esta transição, o sistema preserva todas as operações em andamento, garantindo a continuidade do serviço. Este processo forma um ciclo contínuo, no qual a nova fatia passa a ser monitorada, mantendo o sistema adaptativo. O mecanismo monitora métricas relacionadas à formação e eficiência das fatias, como tamanho atual, distribuição de reputação dentro das fatias, efetividade da comunicação intra-fatia e a necessidade de reorganização.

O processo de adaptação ADAN tem as dimensões e ações caracterizadas na Tabela 3.3.

*Tabela 3.3: Dimensões do mecanismo de adaptação*

| Dimensão | Ações |
|---|---|
| Adaptação Temporal | Frequência de recálculo<br>Janelas de observação<br>Intervalos de reorganização<br>Ajuste do tamanho das fatias |
| Adaptação de Fatias | Redistribuição de nós<br>Atualização de limiares de reputação<br>Critérios de formação de fatias |
| Adaptação Estrutural | Políticas de sobreposição<br>Regras de transição |

*Quadro 3.1: Mecanismo de Adaptação do ADAN*

**Entrada:** Estado das fatias $E$, Métricas $M$, Configurações de fatiamento $C$

**Saída:** Nova configuração de fatias $C'$

1. Para cada período de avaliação $t$:
    2. $M \leftarrow$ *ColetarMétricasDasFatias()*
    3. Se *NecessitaReorganização*($M$):
        4. $\Delta C \leftarrow$ *CalcularNovoFatiamento*($M, E$)
        5. $C' \leftarrow$ *AplicarNovoFatiamento*($C, \Delta C$)
        6. Se *ValidarFatiamento*(C'):
            7. *PrepararNovasFatias*(C')
        8. Senão:
            9. *ManterFatiamentoAtual*()
    10. *AtualizarEstadoDasFatias*(E)

O algoritmo de adaptação do ADAN é apresentado no Quadro 3.1.

### 3.2.2 Sistema de Reputação

O sistema de reputação utiliza a fórmula $R(n) = H(n) \times P(n)$, onde $R(n)$ é a reputação do nó $n$, $H(n)$ é o histórico acumulado do nó, considerando seu comportamento ao longo do tempo e $P(n)$ é o padrão atual de participação do nó na rede. O histórico $H(n)$ é calculado considerando o tempo de atividade do nó, a quantidade de consensos bem-sucedidos, a velocidade média de resposta.
Para isso, utiliza-se a fórmula:

$$H(n) = \sum_i w_i x_i(n) \tag{3.7}$$

onde $i \in 1, 2, 3, 4$. Cada componente $x_i(n)$ representa uma métrica normalizada:

$$x_1(n) = \frac{At(n)}{At_{max}} \tag{3.8}$$

para o tempo de atividade,

$$x_2(n) = \frac{Cs(n)}{Cs_{total}} \tag{3.9}$$

para os consensos bem-sucedidos,

$$x_3(n) = \frac{Vr_{min}}{Vr(n)} \tag{3.10}$$

para a velocidade de resposta e

$$x_4(n) = \frac{Rd(n)}{Rd_{max}} \tag{3.11}$$

para os recursos disponíveis. Os pesos $w_i$ definem a relevância de cada métrica, respeitando as condições

$$\sum_i w_i = 1e0 \leq w_i \leq 1 \tag{3.12}$$

com $H(n)$ variando entre [0,1]. Essa abordagem assegura que todas as métricas sejam normalizadas antes da aplicação dos pesos, possibilitando uma avaliação justa do histórico do nó. O diagrama constante na Figura 3.5 representa a máquina de estados que governa o Sistema de Reputação do BFT-H, cujos estados são:

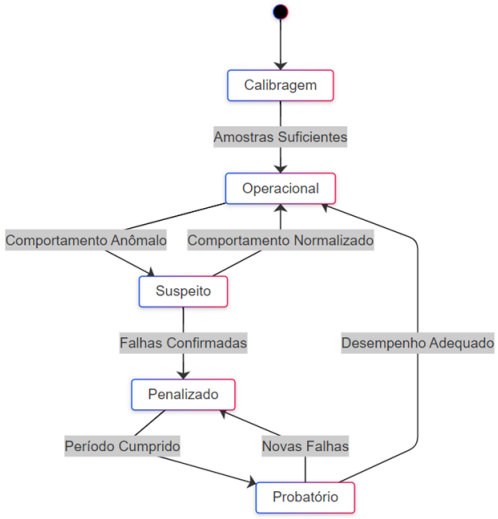


*Figura 3.5: Diagrama de estados do sistema de reputação*

- Calibragem - quando um nó ingressa na rede, ele começa no estado de Calibragem, onde o sistema coleta métricas iniciais e acumula um número mínimo de interações $k$ para estabelecer seu perfil comportamental base, onde $k$ é um parâmetro configurável que deve ser dimensionado de acordo com as características da rede e requisitos de segurança do sistema. Este período inicial é necessário para estabelecer um perfil comportamental básico do nó.

- Operacional - após coletar amostras suficientes, o nó passa ao estado Operacional, onde sua reputação $R(n)$ mantém-se acima de um limiar estabelecido e sua participação ocorre normalmente na rede. Durante sua operação regular, caso o nó apresente comportamentos fora do padrão esperado, ele transita para o estado Suspeito.

- Suspeito - no estado Suspeito, dois caminhos são possíveis: se o nó normaliza seu comportamento, retorna ao estado Operacional; caso as falhas se confirmem, avança para o estado Penalizado.

- Penalizado - a partir do estado Penalizado, após cumprir um período determinado, o nó passa para um estado Probatório, onde tem a chance de demonstrar melhoria em seu comportamento.

- Probatório - durante o período Probatório, se o nó mantiver um desempenho adequado, retorna ao estado Operacional. Contudo, caso apresente novas falhas durante este período, retorna ao estado Penalizado.

Este ciclo de estados possibilita que o sistema mantenha uma supervisão meticulosa sobre o comportamento dos nós ao mesmo tempo que oferece oportunidades de recuperação para nós que apresentaram falhas temporárias.
O sistema implementa verificações de segurança em sua estrutura, garantindo que o tamanho das fatias e o número de votos válidos atendam aos requisitos mínimos para a tolerância bizantina. Este processo é codificado por meio de estruturas de verificação que validam tanto as fatias quanto as rodadas de consenso, conforme especificado no Quadro 3.2.
O mecanismo de garantias de segurança tem como objetivo primordial validar e confirmar o consenso em um conjunto de nós (ou participantes) dentro de um sistema distribuído. Inicialmente, verifica-se se uma determinada "fatia"do sistema possui um número suficiente de nós ativos para operar com segurança, calculando a quantidade máxima tolerável de falhas. Em seguida, realiza-se a análise do consenso entre os participantes por meio da contagem dos votos válidos, assegurando que as decisões sejam tomadas apenas quando houver uma concordância adequada entre os membros.
É importante ressaltar que este sistema de reputação funciona como um mecanismo auxiliar que busca identificar padrões de comportamento dos nós, baseando-se em métricas observáveis de desempenho. Embora possa contribuir para a detecção de comportamentos potencialmente problemáticos, as pontuações de reputação devem ser interpretadas como indicadores e não como garantias absolutas de confiabilidade, uma vez que comportamentos históricos positivos não asseguram necessariamente condutas futuras; variações nas métricas podem resultar de fatores diversos, não apenas de ações maliciosas; e o próprio mecanismo de cálculo de reputação pode estar sujeito a manipulações.

*Quadro 3.2: Garantias de Segurança*

**Garantias de Segurança**

1. Para cada rodada de consenso $r$:
    2. $V \leftarrow$ *ObterVotosAtivos()*
    3. $F \leftarrow$ *CalcularFalhasMaximas*$(V)$
    4. Se *NecessitaValidacao*(V, F):
        5. $Q \leftarrow$ *CalcularQuorumNecessario*$(F)$
        6. $VV \leftarrow$ *ContarVotosValidos*$(r)$
        7. Se $VV \geq (3 * F + 1)$:
            8. *PrepararNovaRodada*(r)
            9. *AtualizarConsenso*(Q)
        10. Senão:
            11. *ManterRodadaAtual*()
    12. *AtualizarEstadoVotos*(V)

### 3.2.3 Consenso em duas fases

O consenso no BFT-H ocorre em duas fases distintas: fase de Configuração (*CONFIG*) e fase de Execução (*EXECUTE*). O algoritmo define estruturas específicas para as mensagens trocadas em cada fase, controlando aspectos como número da visão atual, sequência de operações e *hash* das transações, garantindo assim tanto a ordem total quanto a consistência do estado entre os nós participantes.
A Figura 3.6 representa a sequência de mensagens trocadas entre Cliente, Líder, Fatia Ativa e Réplicas durante o processo de consenso no algoritmo BFT-H:

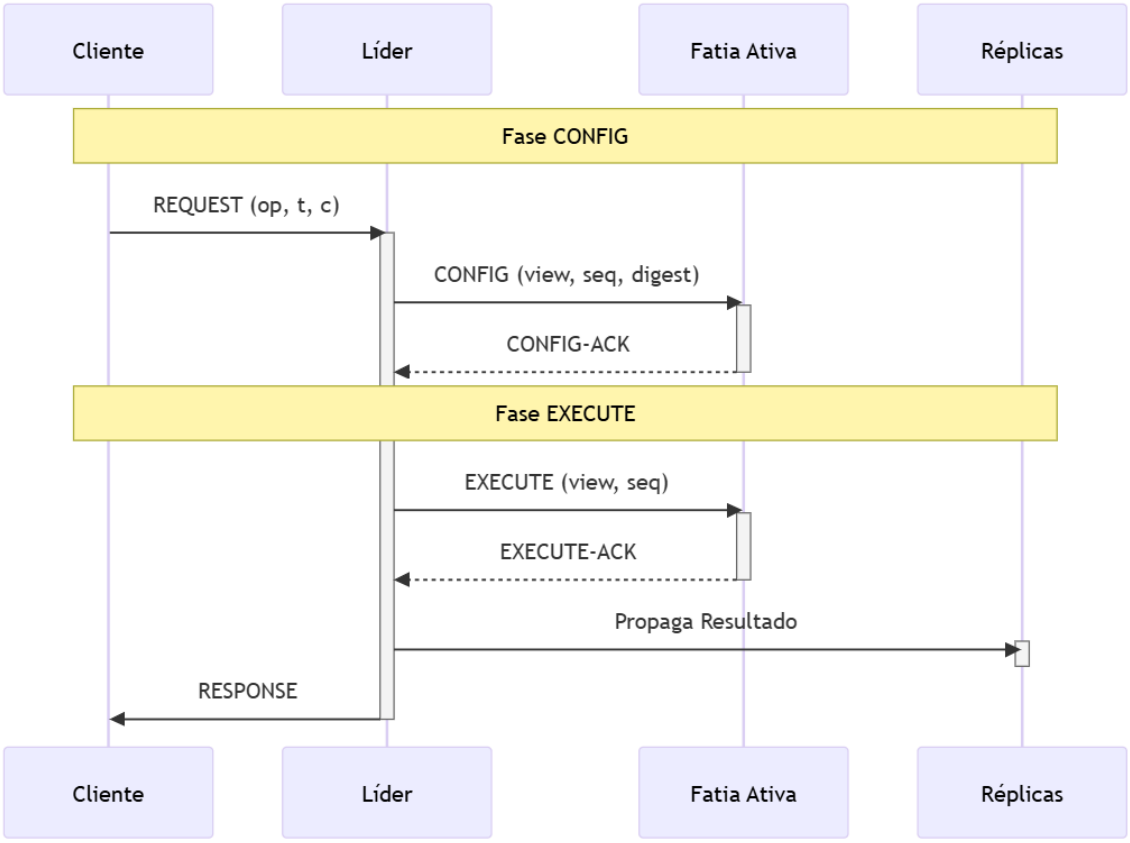


*Figura 3.6: Diagrama de sequência para obtenção do consenso no BFT-H*

I. Fase de Configuração (*CONFIG*):

a. O processo de consenso inicia quando um Cliente envia uma requisição *REQUEST* ao nó Líder, contendo a operação desejada ($op$), o *timestamp* ($t$) e o identificador do cliente ($c$).

b. Ao receber a requisição, o Líder inicia a primeira fase do consenso enviando uma mensagem *CONFIG* para todos os nós da Fatia Ativa. A mensagem *CONFIG* contém a visão atual (*view*), o número de sequência da operação (*seq*) e o *hash* da operação (*digest*). Os nós da Fatia Ativa, ao receberem a mensagem *CONFIG*, validam a proposta e, se considerada válida, respondem ao Líder com uma notificação *CONFIG-ACK*. O Líder aguarda a recepção de um quórum mínimo de validações $Q$, onde

$$Q \geq 2f + 1 \tag{3.13}$$

sendo $f$ = número máximo de falhas bizantinas toleradas.

II. Fase de Execução (*EXECUTE*):

a. Após receber as confirmações necessárias, o Líder inicia a segunda fase de consenso, enviando uma mensagem *EXECUTE* para a Fatia Ativa. Os nós da Fatia Ativa realizam a operação localmente e respondem com mensagens *EXECUTE-ACK*. O algoritmo requer um quórum mínimo de execuções confirmadas $E$, onde

$$E \geq 2f + 1 \tag{3.14}$$

sendo

$$f = \left\lfloor \frac{n-1}{3} \right\rfloor \tag{3.15}$$

e $n$ = número total de nós. Esse quórum de mensagens *EXECUTE-ACK* garante que a operação foi executada corretamente por um número suficiente de nós honestos para manter a consistência do sistema.

b. Por fim, o Líder propaga o resultado para todas as Réplicas do sistema, garantindo que mesmo os nós fora da Fatia Ativa mantenham seus estados atualizados, e envia a Resposta final ao Cliente que originou a requisição. As transições entre as fases do consenso são controladas por uma máquina de estados finita que garante as propriedades de acordo, validade, terminação e ordem total.

A integração entre o sistema de reputação e o processo de consenso no BFT-H ocorre como ilustrado na Figura 3.7:

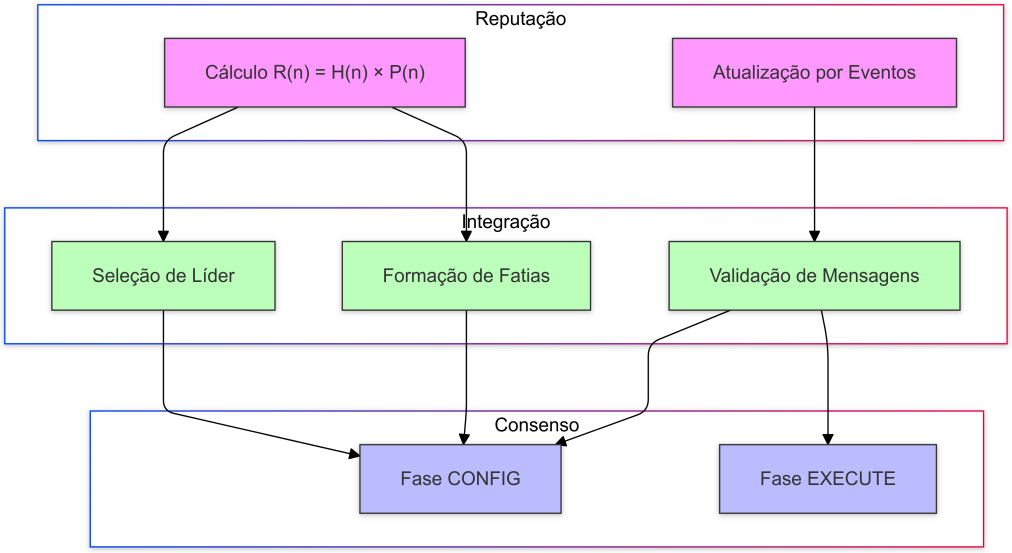


*Figura 3.7: Integração entre reputação e consenso*

a) Reputação - sistema de Reputação (Figura 3.7) é representado por dois componentes principais: o cálculo, que utiliza a fórmula

$$R(n) = H(n) \times P(n) \tag{3.16}$$

e o mecanismo de atualização baseado em eventos do algoritmo.

b) Integração - A reputação influencia o consenso por intermédio de três pontos de integração fundamentais:

- Primeiro, na seleção do líder, quando nós com maior reputação têm prioridade para coordenar o consenso.
- Segundo, na formação das fatias, quando apenas nós com reputação acima do limiar estabelecido, podem participar.
- Terceiro, na validação de mensagens, quando a reputação dos nós é considerada durante o processo de verificação.

c) Consenso - Estes pontos de integração afetam diretamente as duas fases do consenso: *CONFIG e EXECUTE*. A fase *CONFIG* é influenciada por todos os pontos de integração, pois depende da liderança escolhida, da composição da fatia e da validação das mensagens. Já a fase *EXECUTE* é afetada principalmente pela validação das mensagens, que garante a execução correta das operações.

O processo de consenso utiliza uma máquina de estados finitos para controlar as transições entre fases. O sistema inicia no estado Inicial e, ao receber uma requisição *REQUEST*, move para a fase *CONFIG*, onde o líder distribui a mensagem *CONFIG* e aguarda $2f + 1$ *CONFIG-ACKs*. Com as confirmações necessárias, avança para a fase *EXECUTE*, onde distribui a mensagem *EXECUTE* e aguarda $2f + 1$ *EXECUTE-ACKs* antes de finalizar.

Em caso de falhas ou timeouts durante qualquer uma das fases (*CONFIG ou EXECUTE*), o sistema entra em estado de TrocaVisão, onde um novo Líder é eleito preservando as operações em andamento. Após a

eleição, retorna à fase *CONFIG* com a nova visão. A integração entre o sistema de reputação e o processo de consenso é feita através de um mecanismo de seleção de Líder que prioriza nós com maior reputação. Esta arquitetura permite que o BFT-H mantenha as garantias de segurança bizantinas enquanto otimiza o desempenho por meio do fatiamento adaptativo da rede. O sistema de reputação, integrado ao mecanismo ADAN e ao processo de consenso em duas fases, cria um algoritmo robusto e eficiente, especialmente adequado para os requisitos de sistemas de gestão de saúde distribuídos.
A Tabela 3.4 resume os tipos de mensagens trocadas durante o processo de consenso do BFT-H.

*Tabela 3.4: Tipos de mensagens trocadas durante o consenso*

| **Tipo** | **Origem** | **Destino** | **Conteúdo** | **Propósito** |
|---|---|---|---|---|
| *REQUEST* | Cliente | Líder | *op, t, e* | Solicita execução de operação |
| *CONFIG* | Líder | Fatia | *view, seq, digest* | Propõe ordenação da operação |
| *CONFIG-ACK* | Fatia | Líder | *view, seq, digest, i* | Confirma aceitação do *CONFIG* |
| *EXECUTE* | Líder | Fatia | *view, seq* | Autoriza execução da operação |
| *EXECUTE-ACK* | Fatia | Líder | *view, seq, i* | Confirma execução da operação |
| *VIEW-CHANGE* | Fatia | Todos | *view+1, P, i* | Inicia troca de visão |
| *NEW-VIEW* | Novo Líder | Fatia | *view+1, P* | Estabelece nova visão |

**Onde:**

*op* - operação solicitada

*t* - *timestamp*

*c* - identificador do cliente

*view* - número da visão atual

*seq* - número de sequência

*digest* - *hash* da operação

*i* - identificador do nó

*P* - conjunto de operações pendentes

## 3.3 ANÁLISE E GARANTIAS DO BFT-H

O BFT-H foi projetado para atender às propriedades fundamentais de sistemas distribuídos bizantinos estabelecidas em algoritmos tolerantes à falha bizantina, enquanto implementa mecanismos adicionais de segurança sobre os quais discorremos ao longo deste capítulo.

A propriedade de Acordo estabelece que dois nós honestos não podem confirmar diferentes valores para a mesma posição na sequência de operações. Esta garantia é mantida pelo requisito de quórum duplo: $2f+1$ confirmações tanto na fase *CONFIG* quanto na fase *EXECUTE*, assegurando sobreposição de pelo menos $f + 1$ nós honestos entre quaisquer dois quóruns.

A Validade garante que o algoritmo continua fazendo progresso desde que exista pelo menos uma fatia ativa com $2f + 1$ nós honestos. A validação ocorre em múltiplos níveis: primeiro na verificação de assinaturas pelo líder, depois na validação pela fatia ativa. O sistema de reputação visa fortalecer esta propriedade através da priorização de nós com histórico consistente nas fatias ativas.

A Ordem Total assegura que todos os nós honestos processam as operações na mesma sequência. O algoritmo implementa esta garantia através da numeração única atribuída na fase *CONFIG* e da propagação ordenada após confirmação. Durante mudanças de fatia, o novo grupo herda o último estado confirmado, garantindo que todas as réplicas, mesmo aquelas fora da fatia ativa, mantenham uma visão consistente do sistema.

O sistema de reputação atua como mecanismo adicional de segurança ao influenciar três aspectos críticos do algoritmo. O primeiro refere-se à seleção do líder, priorizando nós com maior pontuação para coordenar o consenso. O segundo estabelece um limiar mínimo $\theta$ para participação nas fatias ativas. O terceiro permite atuação preventiva ao limitar a participação de nós que apresentem comportamentos suspeitos antes mesmo de sua identificação definitiva como maliciosos.

Os experimentos indicam que as propriedades fundamentais do algoritmo podem ser preservadas com o fatiamento dinâmico, desde que três condições sejam observadas. Primeiro, cada fatia deve preservar a condição $n \geq 3f + 1$, mantendo a proporção necessária entre nós honestos e bizantinos. Segundo, os quóruns devem manter $2f + 1$ confirmações, garantindo a interseção de nós honestos. Terceiro, o sistema de reputação deve identificar corretamente comportamentos bizantinos, possibilitando a identificação e potencial isolamento de nós que apresentem comportamentos divergentes dos padrões estabelecidos.

## 3.4 APLICABILIDADE EM SISTEMAS DE SAÚDE

Esta seção analisa a aplicabilidade do BFT-H em sistemas de gestão de saúde distribuídos em larga escala. Por exemplo, a rede SUS (1) que processa diariamente milhões de registros médicos e realiza aproximadamente 12 milhões de internações por ano, demanda um sistema distribuído robusto e confiável para integrar os níveis de atenção à saúde primária, secundária e terciária.

A heterogeneidade da rede SUS, que engloba desde Unidades Básicas de Saúde (UBS) até hospitais terciários de alta complexidade, encontra suporte no mecanismo ADAN do BFT-H. A formação adaptativa de fatias permite que estabelecimentos com diferentes capacidades computacionais e de infraestrutura participem efetivamente do sistema. Por exemplo, uma UBS com recursos computacionais limitados pode

participar da rede mantendo seus registros consistentes com um hospital regional que possui infraestrutura mais robusta.

O sistema calcula a reputação de cada unidade de saúde utilizando a fórmula de reputação definida na Seção 3.2.2, que considera tanto o histórico acumulado quanto o padrão atual de participação de cada unidade na rede. Por exemplo, um hospital regional que mantém consistentemente seus registros atualizados e responde rapidamente a requisições de prontuários terá um $H(n)$ elevado.

A reputação leva em conta a participação da unidade em consensos, como na validação de registros médicos. O tempo de disponibilidade na rede é crucial para acessar informações em emergências. Além disso, a velocidade de resposta é importante em encaminhamentos entre unidades e os recursos computacionais disponibilizados também são considerados.

Em uma rede do SUS, onde existem desde UBSs até hospitais de alta complexidade, o sistema mantém a consistência dos dados mesmo quando algumas unidades estão temporariamente indisponíveis ou sobrecarregadas. Por exemplo, durante um surto epidêmico, quando uma UBS pode experimentar sobrecarga, o sistema continua operando por meio de outras unidades com reputação adequada na mesma região.

A consistência garantida pelo algoritmo seria importante para funcionamento do sistema de referência e contrarreferência do SUS. Quando um paciente é encaminhado de uma UBS para um hospital especializado, o BFT-H assegura que todas as informações do prontuário estejam disponíveis e atualizadas, mesmo em cenários onde parte da rede experimenta instabilidades. A Figura 3.8 exemplifica o funcionamento do BFT-H aplicado ao processamento de registros médicos dentro do unidades de saúde.

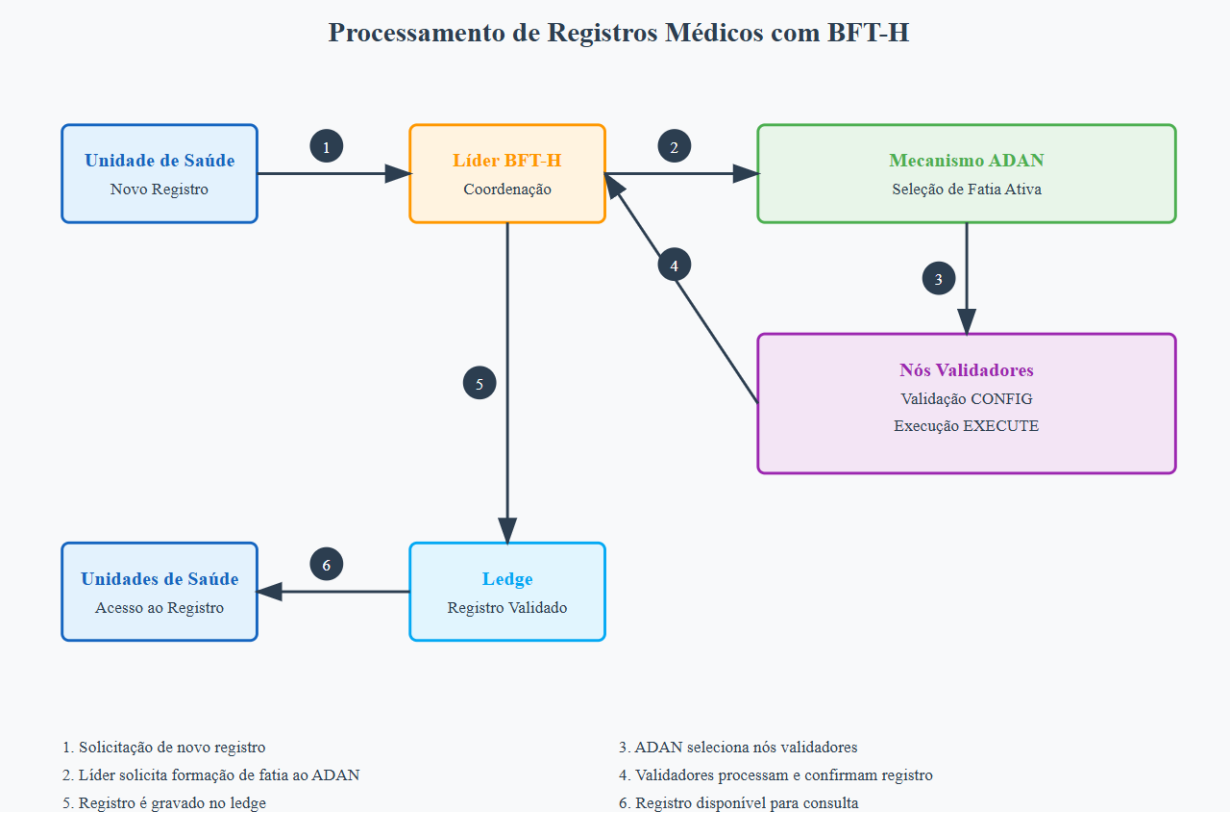


*Figura 3.8: Processamento de registros médicos*

# 4 AVALIAÇÃO DE DESEMPENHO DO BFT-H

A avaliação de desempenho do BFT-H foi realizada por meio de experimentos utilizando a técnica de simulação e uma comparação com resultados da literatura com outros algoritmos similares.

## 4.1 CONFIGURAÇÃO GERAL DOS EXPERIMENTOS

### 4.1.1 Ferramenta de Simulação

A avaliação experimental do BFT-H utilizou o simulador de eventos discretos *Network Simulator 3* (NS3) (61), capaz de modelar com precisão as complexidades de redes distribuídas em larga escala. O NS3 suporta modelos de mobilidade e protocolos de roteamento em redes ad hoc, permitindo, por meio de sua arquitetura extensível, a implementação de módulos específicos para algoritmos *Blockchain*. O uso do NS3, caracterizado por uma arquitetura multicamada, viabiliza uma modelagem granular das interações entre o algoritmo de consenso e os protocolos de rede subjacentes.
A convergência assintótica das distribuições amostrais para a normalidade gaussiana, demonstrada através das 30 repetições dos experimentos no NS3 (62), manifesta-se por intermédio da soma de variáveis aleatórias independentes e identicamente distribuídas, onde a função característica da distribuição resultante exibe propriedades de estabilidade que culminam em uma forma distribucional específica. O intervalo de confiança de 95% foi estabelecido utilizando o valor crítico $z$ = 1,96 da distribuição normal padrão.
Com o objetivo de validar os resultados obtidos pela simulação com o NS3, os mesmos foram comparados com estudos publicados nos artigos abaixo:

- *Hidayat, S. A.; Juniardi, W.; Khatami, A. A.; Sari, R. F. - Performance Comparison and Analysis of Paxos, Raft and PBFT Using NS3. In: IEEE INTERNATIONAL CONFERENCE ON INTERNET OF THINGS AND INTELLIGENCE SYSTEMS (IoTaIS), 2022, Bali, Indonésia. Bali: IEEE, 2022. p. 304-310. DOI: 10.1109/IoTaIS56727.2022.9975938.*
- *OTHMEN, R. Ben; ABBESSI, W.; OUNI, S.; BADREDDINE, W.; DEQUEN, G.- Simulation Of Optimized Cluster Based PBFT Blockchain Validation Process. In: IEEE SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS (ISCC), 2023, Gammarth, Tunísia. Gammarth: IEEE, 2023. p. 1317-1322. DOI: 10.1109/ISCC58397.2023.10218119.*

### 4.1.2 Sistema Avaliado

O sistema avaliado consiste no uso do BFT-H como algoritmo de consenso em sistema distribuído de gestão de saúde em larga escala baseado em tecnologia *Blockchain*.

### 4.1.3 Parâmetros do Experimento

#### 4.1.3.1 Parâmetros do Sistema

As seguintes dimensões e parâmetros de sistema foram usados nos experimentos com o BFT-H:

a. Escala do sistema: medida em "Número de Nós", variando de 15, 30, 60,1.000, 2.000 e 5.000;

b. Condições de Rede: medida em "Atraso (*Delay*)", variando de 1, 10 e 100 ms;

### 4.1.4 Parâmetro de Carga

O principal parâmetro de carga usado nos experimentos com o BFT-H foi o Volume de Dados, medido em "Tamanho das Mensagens", variando de 200, 2.000 e 10.000 Bytes.
A Tabela 4.1 resume as principais configurações dos experimentos com o BFT-H.

*Tabela 4.1: Configurações dos Cenários Experimentais*

| Dimensão | Parâmetros de Sistema e de Carga | Valores |
|---|---|---|
| Densidade | Número de Nós | 15, 30, 60, 1000, 2000, 5000 |
| Condições de Rede | *Delay* (ms) | 1, 10, 100 |
| Volume de Dados | Tamanho das Mensagens (*Bytes*) | 200, 2000, 10000 |
| Confiabilidade | Número de Repetições | 30 |
| Estatística | Nível de Confiança | 95% |

### 4.1.5 Métricas de Desempenho

Estruturada em três dimensões principais - Volume de Dados, Densidade e Atraso da rede - a metodologia experimental comparou o desempenho do BFT-H ao desempenho de algoritmos tradicionais como o PBFT e o *Raft*.
As métricas de desempenho utilizadas foram:

a) Tempo de Consenso - métrica primária, mensurou o intervalo entre a proposição e a finalização do acordo sobre o estado das transações, correlacionando-se diretamente com o desempenho operacional em ambientes distribuídos.

b) Latência - tempo total para que uma transação seja completamente processada e confirmada na rede.

c) Vazão (*Throughput*) - taxa de transações confirmadas que o sistema consegue processar por unidade de tempo.

### 4.1.6 Topologia de Rede

A escolha da topologia de rede para cada algoritmo reflete diretamente seu modelo de comunicação e mecanismo de consenso. O PBFT requer uma rede totalmente conectada (*full mesh*) para que cada nó

possa verificar independentemente as mensagens de todos os outros, garantindo assim a tolerância a falhas bizantinas. O *Raft*, por sua natureza centralizada na figura do líder, funciona de maneira satisfatória em uma topologia estrela. Já o BFT-H, com seu sistema de fatiamento dinâmico, demanda uma topologia que possa se reorganizar conforme a formação das fatias. Para implementar estas diferentes topologias, foram utilizadas conexões ponto-a-ponto entre os nós com as características apresentadas na Tabela 4.2.

Tabela 4.2: Parâmetros das Conexões de Rede

| Parâmetro | Valor |
|---|---|
| Taxa de Dados | 100 Mbps |
| *Delay* (ms) | 1, 10, 100 |
| MTU (Tamanho Máximo de Mensagem) | 1500 bytes |
| Tamanho da Fila | 100 pacotes |

Com o BFT-H, foram criadas sub-redes dinâmicas para cada fatia ativa, onde cada uma opera com

$$\left\lfloor \frac{|N|}{2} \right\rfloor + 1 \tag{4.1}$$

nós. As conexões dentro das fatias são diretas, permitindo comunicação rápida entre seus membros, enquanto a comunicação entre fatias diferentes acontece através de rotas estabelecidas automaticamente pelo sistema.

Para o PBFT, foram estabelecidas conexões diretas entre todos os pares possíveis de nós, resultando em

$$\frac{N(N-1)}{2} \tag{4.2}$$

conexões. Esta configuração, embora mais custosa em termos de recursos, é necessária para manter as garantias de segurança do algoritmo. Cada nó recebe um endereço único e pode se comunicar diretamente com qualquer outro participante da rede.

No caso do *Raft*, foi implementada uma topologia em estrela onde o líder mantém conexões diretas com todos os seus seguidores. Esta estrutura centralizada reflete o modelo de comunicação do algoritmo, onde todas as decisões passam pelo líder antes de serem propagadas para o resto da rede.

Em todos os casos, instalamos a pilha completa de protocolos de rede em cada nó, permitindo o roteamento de pacotes e a comunicação confiável entre os participantes. O sistema de endereçamento garante que cada nó seja unicamente identificável na rede, enquanto o roteamento assegura que as mensagens cheguem aos seus destinos pela melhor rota disponível.

### 4.1.7 Carga de Trabalho

A geração de tráfego para os experimentos foi realizada por meio do módulo "*OnOffHelper*"do NS3, que funciona como gerador de tráfego. Para cada cenário avaliado, a simulação foi conduzida durante 10 segundos, um tempo considerado adequado para estabelecer um padrão de operação estável e permitir a coleta das métricas requeridas. Esse intervalo foi definido após a realização de testes preliminares que evidenciaram a estabilização dos resultados nesse período.

A coleta efetiva das métricas teve início após os primeiros 2 segundos de simulação, desconsiderando assim o período inicial dedicado à estabilização do sistema. Assim, em cada execução de 10 segundos, foram levados em conta 8 segundos de dados válidos para fins de análise. O módulo foi configurado com os parâmetros de configuração constantes na Tabela 4.3:

*Tabela 4.3: Configurações do módulo OnOffHelper*

| Parâmetros | Valores |
|---|---|
| Tempo Ativo (*OnTime*) | 1.0 (constante) |
| *Tempo Inativo (OffTime)* | 0.0 (constante) |
| Taxa Base de Dados | 100 Mbps |
| Tamanhos de Pacote | 200, 2.000 e 10.000 bytes |

Com um escopo estabelecido em 30 repetições por cenário, obtivemos os seguintes resultados em termos de tempos de simulação:

a. Redes com baixa densidade de nós

   i. 15, 30 e 60 nós × 3 *delays* × 3 tamanhos de mensagem;

   ii. 27 cenários × 30 repetições × 10 segundos = 8.100 segundos

b. Redes com média densidade de nós

   i. 1000 nós × 3 *delays* × 3 tamanhos de mensagem;

   ii. 9 cenários × 30 repetições × 10 segundos = 2.700 segundos

c. Redes com média densidade de nós

   i. 2000 e 5000 nós × 3 *delays* × 3 tamanhos de mensagem;

   ii. 18 cenários × 30 repetições × 10 segundos = 5.400 segundos

Essas execuções totalizaram 16.200 segundos (aproximadamente 4,5 horas) em tempo efetivo de simulação. O tráfego foi distribuído entre os nós utilizando o modelo de roteamento padrão do NS3, com cada nó atuando como origem e destino das transações. A configuração do experimento garantiu que cada nó mantivesse conexões ativas com seus pares conforme a topologia definida para cada algoritmo de consenso.

## 4.2 EXPERIMENTOS

O principal fator estudado foi a escalabilidade da rede. Buscou-se avaliar o desempenho do BFT-H em comparação com outros algoritmos de consenso, a saber, PBFT e *Raft*, considerando variações na densidade de nós, latências de rede e tamanhos das mensagens. Os experimentos foram conduzidos por meio do simulador NS3, o que possibilitou uma modelagem aprofundada das interações e medições acuradas dos parâmetros de desempenho, incluindo tempo de consenso, latência e vazão (*throughput*).

### 4.2.1 Experimento 1 (Baixa densidade de nós)

O Experimento 1 consistiu em avaliar os Tempos de Consenso dos algoritmos BFT-H, PBFT e *Raft*, em cenários de rede com baixa densidade de nós, submetidos às cargas de trabalho de 200, 2.000 e 10.000 *bytes*, e em diferentes condições de *delay* (1, 10 e 100 ms).
Em baixa densidade, totalizando 27 cenários distintos, os experimentos estabeleceram a baseline comportamental dos algoritmos. Considerando 30 execuções por cenário, esta configuração resultou em 810 execuções experimentais.
Os resultados do Experimento 1 são apresentados nas Figuras 4.1 (15 nós), 4.2 (30 nós) e 4.3 (60 nós).

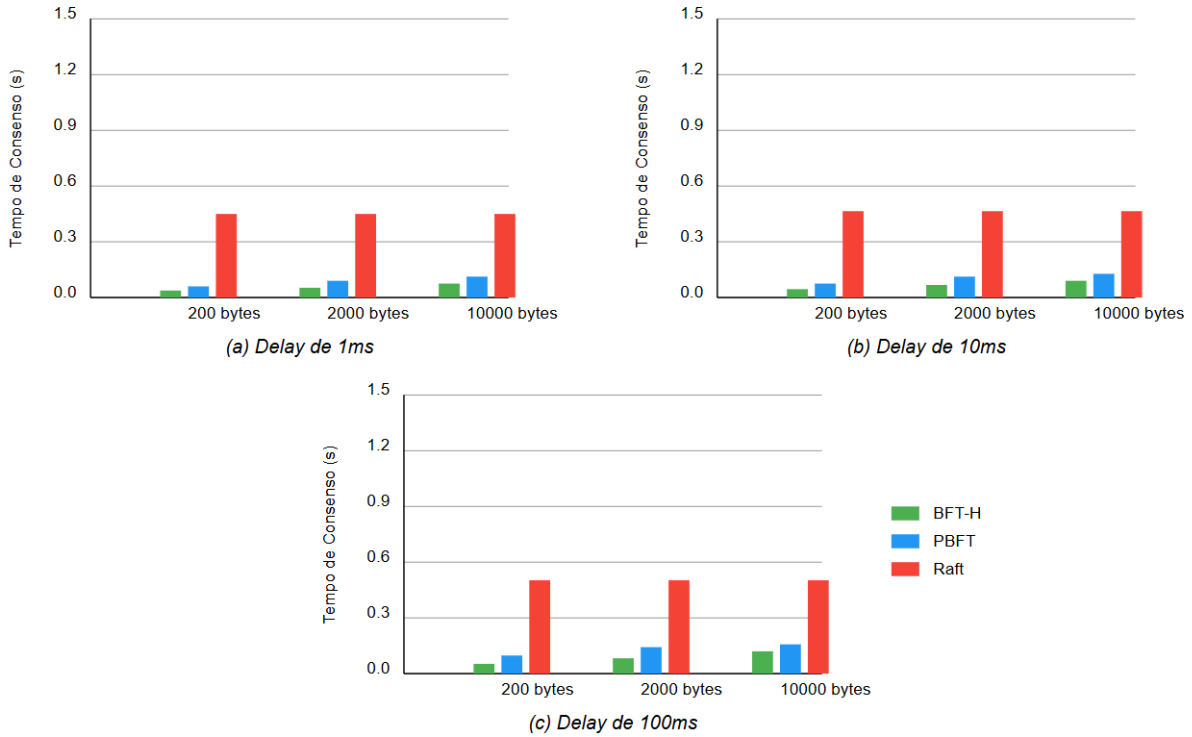


*Figura 4.1: Experimento 1: Tempo de Consenso - Comparativo de desempenho com 15 nós*

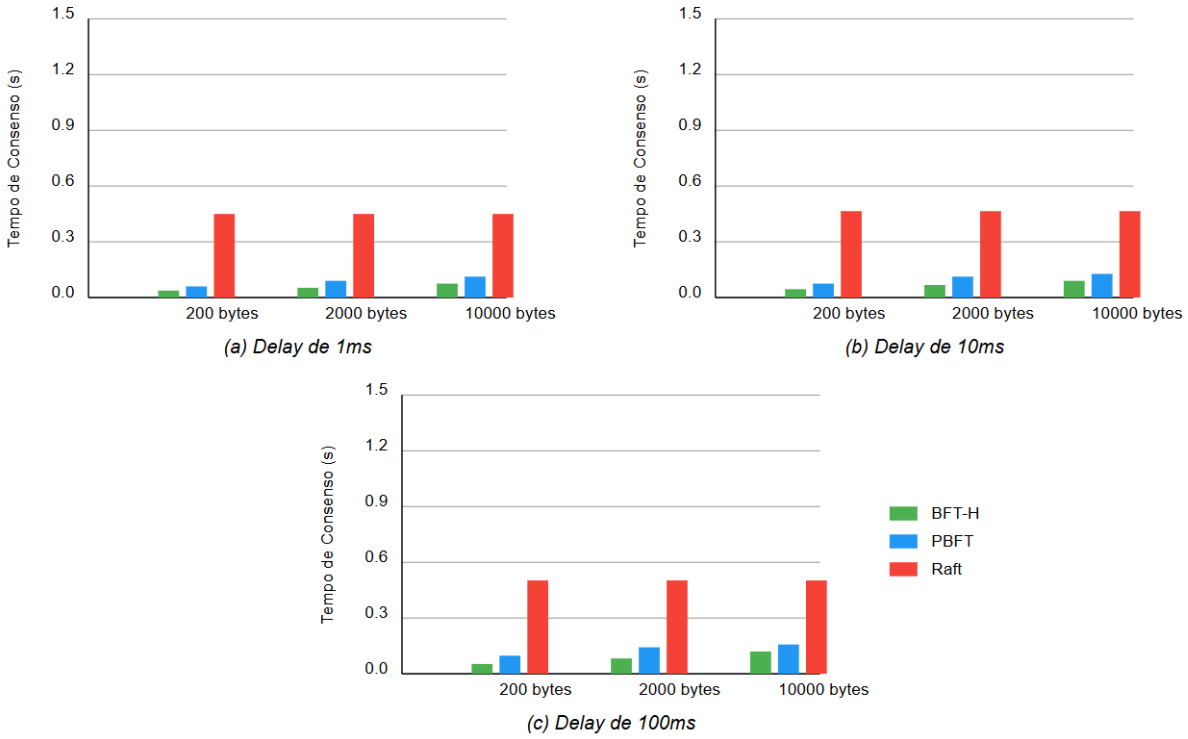


*Figura 4.2: Experimento 1: Tempo de Consenso - Comparativo de desempenho com 30 nós*

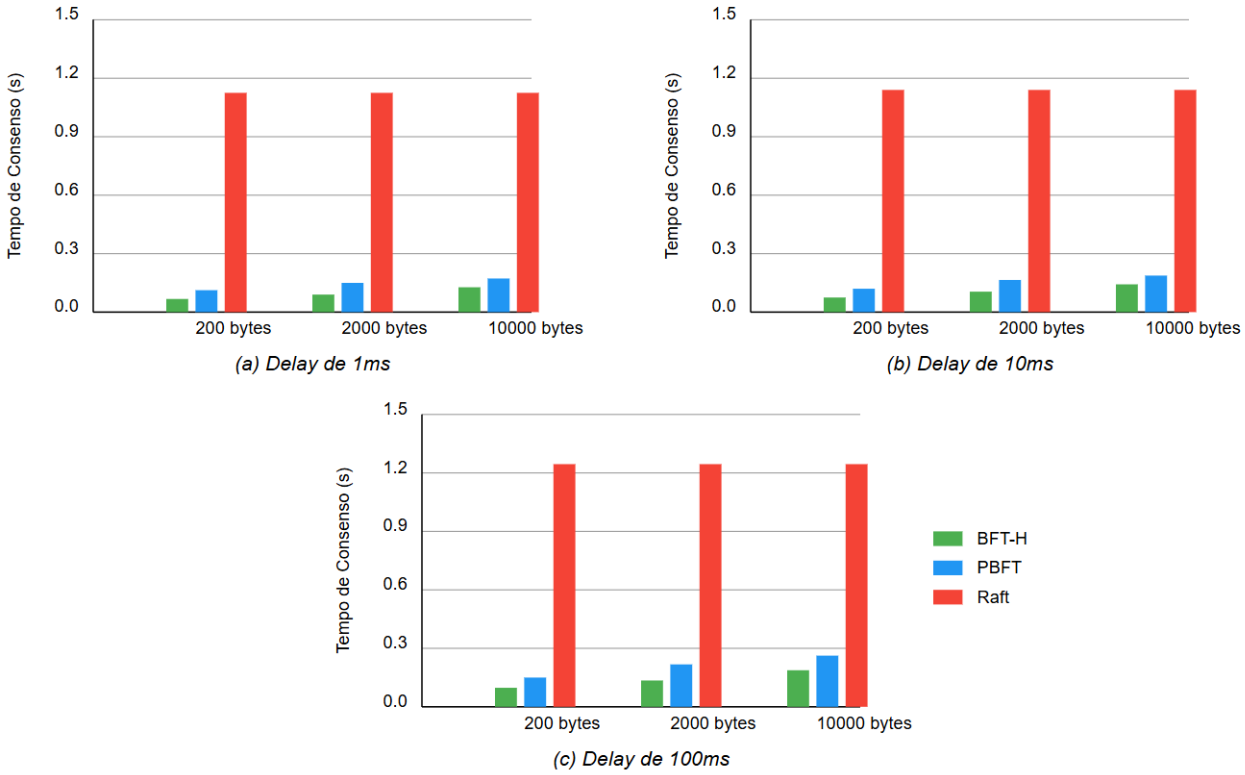


*Figura 4.3: Experimento 1: Tempo de Consenso - Comparativo de desempenho com 60 nós*

### 4.2.2 Experimento 2 (Média densidade de nós)

O Experimento 2 consistiu em avaliar os Tempos de Consenso dos algoritmos BFT-H, PBFT e *Raft*, em cenários de rede com média densidade de nós (1.000), submetidos às cargas de trabalho de 200, 2.000 e 10.000 *bytes*, e em diferentes condições de *delay* (1, 10 e 100 ms).
No contexto do Experimento 2, foram elaborados 9 cenários, somando 270 execuções experimentais. Tal abrangência permitiu avaliar a transição entre os comportamentos em redes de pequena (Experimento 1) e larga escala (Experimento 3).
Os resultados do Experimento 2 são apresentados na Figura 4.4.

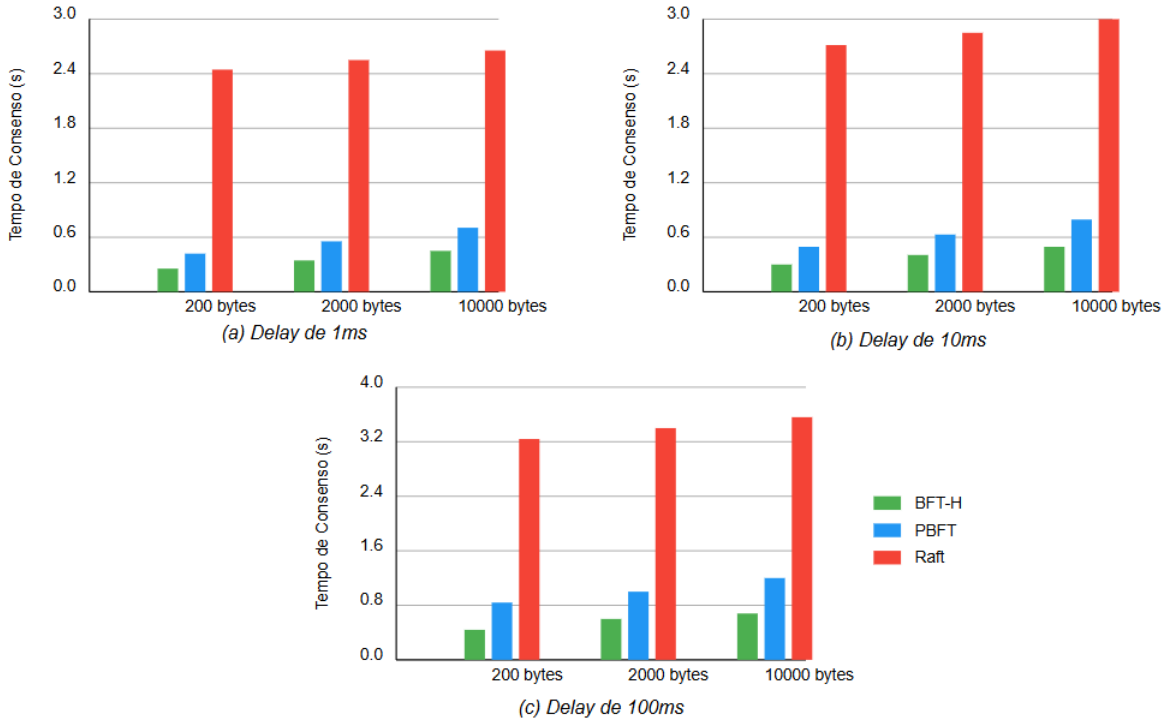


*Figura 4.4: Experimento 2: Tempo de Consenso - Comparativo de desempenho com 1000 nós*

### 4.2.3 Experimento 3 (Alta Densidade de Nós)

O Experimento 3 consistiu em avaliar os Tempos de Consenso dos algoritmos BFT-H, PBFT e *Raft*, em cenários de rede com alta densidade de nós (2.000 e 5.000), submetidos às cargas de trabalho de 200, 2.000 e 10.000 *bytes*, e em diferentes condições de *delay* (1, 10 e 100 ms).
Os cenários de alta densidade avaliados no Experimento 3 refletem modelos bem próximos da realidade de ambientes distribuídos em larga escala, como o sistema SUS, totalizando 18 cenários e 540 execuções.
Os resultados do Experimento 3 são apresentados na Figuras 4.5 e 4.6.

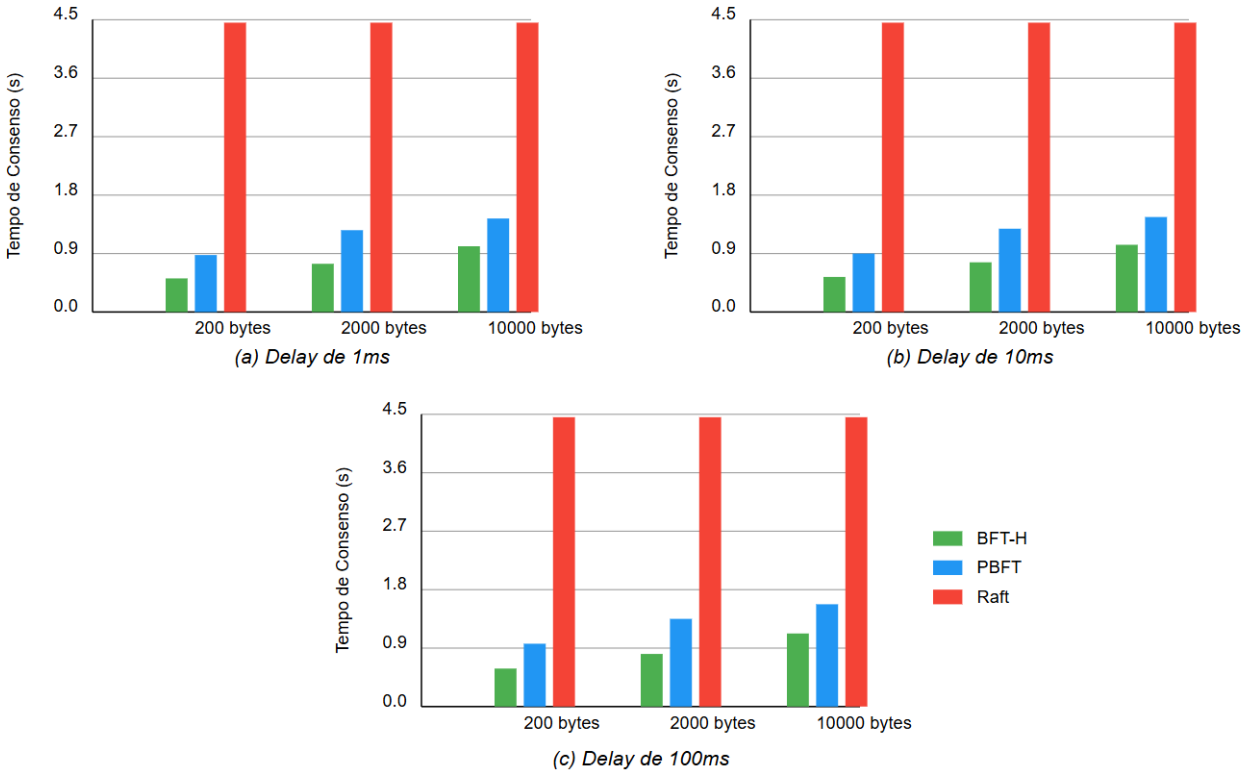


*Figura 4.5: Experimento 3: Tempo de Consenso - Comparativo de desempenho com 2000 nós*

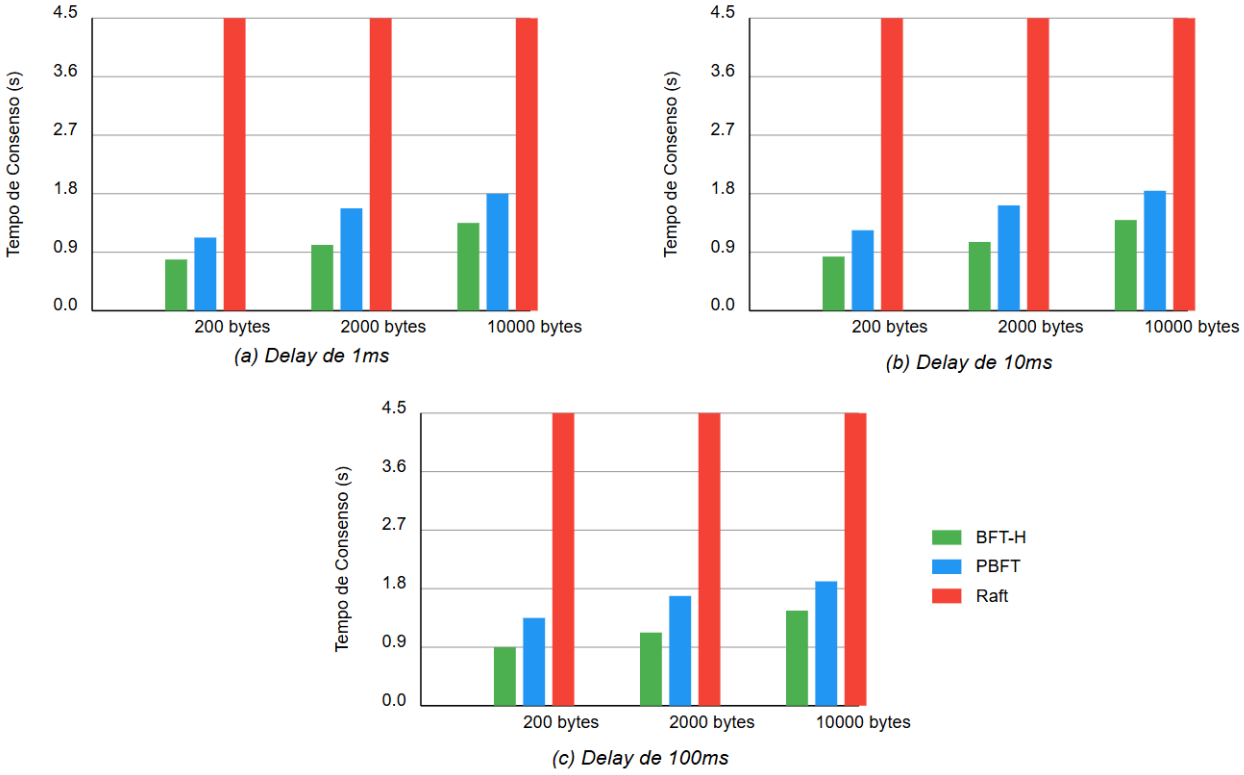


*Figura 4.6: Experimento 3: Tempo de Consenso - Comparativo de desempenho com 5000 nós*

A validação estatística dos resultados dos experimentos foi conduzida com rigor metodológico para assegurar a confiabilidade dos resultados. Para cada cenário, foram realizadas 30 execuções, atendendo aos requisitos do Teorema do Limite Central, totalizando 1.620 execuções ao longo das 54 configurações testadas. O intervalo de confiança foi estabelecido em 95% ($\alpha$ = 0,05), com margem de erro calculada por intermédio da fórmula

$$E = z_{\alpha/2} \times \frac{s}{\sqrt{n}} \tag{4.3}$$

Onde:

- $z_{(\alpha/2)}$ representa o valor crítico para 95% de confiança (1,96)
- $s$ é o desvio padrão amostral
- $n$ o tamanho da amostra

Para validar os resultados obtidos, aplicou-se o teste de Shapiro-Wilk para verificar a normalidade dos dados, seguido por testes $t$ pareados para comparações entre algoritmos e Kruskal-Wallis para análise de variância entre cenários. O tratamento de outliers foi realizado através do método IQR (*Interquartile Range*), com valores além de $1{,}5 \times$ IQR sendo cuidadosamente investigados e documentados. Como critérios de qualidade, estabeleceu-se um coeficiente de variação inferior a 5% para aceitação dos resultados e um desvio padrão relativo menor que 3% entre repetições.
O teste Shapiro-Wilk é realizado ao se aplicar a fórmula

$$W = \frac{(\sum_{i=1}^{n} a_i x_{(i)})^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{4.4}$$

Onde:

- $W$ é a estatística do teste
- $a_i$ são os coeficientes tabelados
- $x_{(i)}$ são os valores ordenados da amostra
- $\bar{x}$ é a média da amostra
- $n$ é o tamanho da amostra

Observou-se que os cenários apresentam p-valoresbem abaixo do nível de significância típico de 0,05, valores da estatística W variando entre 0,59 e 0,61, sugerindo que os dados não seguem uma distribuição normal. Com isso, foi aplicado o teste não paramétrico Kruskal-Wallis, por intermédio da fórmula

$$H = \frac{12}{N(N+1)} \sum_{j=1}^{k} \frac{T_j^2}{n_j} - 3(N+1) \tag{4.5}$$

Onde:

- $H$ é a estatística de teste que segue aproximadamente uma distribuição qui-quadrado com k-1 graus de liberdade sob a hipótese nula.
- $N$ é o número total de observações em todos os grupos.
- $k$ é o número de grupos.
- $Tj$ é a soma dos postos do grupo j.
- $nj$ é o número de observações no grupo j

Os resultados indicam um padrão consistente ao longo de diferentes configurações e que os valores não apresentam anomalias ou variações inexplicáveis. O p-valor = 0,0022 (abaixo de 0,05) indica que os resultados são estatisticamente significativos. As diferentes configurações (variações de bytes e delay) produzirem resultados consistentes, reforçaando a robustez do experimento. Esta metodologia estatística assegura a confiabilidade e a reprodutibilidade dos resultados apresentados, fornecendo uma base sólida para as conclusões derivadas dos experimentos. Consolidados nas Tabela 4.4, 4.5 e 4.6, os resultados em termos da métrica Tempo de Consenso proporcionam uma visão ampla do desempenho dos algoritmos sob diferentes condições de carga e rede.

*Tabela 4.4: Tempo de Consenso (em segundos) - Redes de baixa densidade*

| Nós | *Delay* | Msg (*bytes*) | BFT-H | PBFT | *Raft* |
|---|---|---|---|---|---|
| 15 | 1 | 200 | 0,05 | 0,08 | 1,24 |
| | | 2000 | 0,07 | 0,12 | 1,25 |
| | | 10000 | 0,10 | 0,15 | 1,26 |
| | 10 | 200 | 0,06 | 0,10 | 1,27 |
| | | 2000 | 0,09 | 0,15 | 1,29 |
| | | 10000 | 0,12 | 0,17 | 1,30 |
| | 100 | 200 | 0,07 | 0,13 | 1,37 |
| | | 2000 | 0,11 | 0,19 | 1,38 |
| | | 10000 | 0,16 | 0,21 | 1,39 |
| 30 | 1 | 200 | 0,06 | 0,11 | 1,38 |
| | | 2000 | 0,09 | 0,16 | 1,38 |
| | | 10000 | 0,13 | 0,19 | 1,39 |
| | 10 | 200 | 0,07 | 0,12 | 1,39 |
| | | 2000 | 0,11 | 0,18 | 1,41 |
| | | 10000 | 0,15 | 0,20 | 1,42 |
| | 100 | 200 | 0,10 | 0,16 | 1,52 |
| | | 2000 | 0,14 | 0,23 | 1,53 |
| | | 10000 | 0,19 | 0,26 | 1,54 |
| 60 | 1 | 200 | 0,10 | 0,15 | 1,55 |
| | | 2000 | 0,14 | 0,22 | 1,56 |
| | | 10000 | 0,19 | 0,25 | 1,57 |
| | 100 | 200 | 0,13 | 0,20 | 1,66 |
| | | 2000 | 0,18 | 0,29 | 1,67 |
| | | 10000 | 0,25 | 0,35 | 1,68 |

*Tabela 4.5: Tempo de Consenso (em segundos) - Redes de média densidade*

| Nós | *Delay* | Msg (*bytes*) | BFT-H | PBFT | *Raft* |
|---|---|---|---|---|---|
| 1000 | 1 | 200 | 0,25 | 0,42 | 2,45 |
| | | 2000 | 0,35 | 0,55 | 2,55 |
| | | 10000 | 0,45 | 0,70 | 2,65 |
| | 10 | 200 | 0,30 | 0,50 | 2,72 |
| | | 2000 | 0,40 | 0,63 | 2,85 |
| | | 10000 | 0,50 | 0,80 | 3,00 |
| | 100 | 200 | 0,38 | 0,65 | 3,25 |
| | | 2000 | 0,50 | 0,80 | 3,40 |
| | | 10000 | 0,65 | 1,00 | 3,55 |

*Tabela 4.6: Tempo de Consenso (em segundos) - Redes de alta densidade*

| Nós | *Delay* | Msg (*bytes*) | BFT-H | PBFT | *Raft* |
|---|---|---|---|---|---|
| 2000 | 1 | 200 | 0,51 | 0,86 | 4,18 |
| | | 2000 | 0,74 | 1,26 | 4,22 |
| | | 10000 | 1,02 | 1,44 | 4,26 |
| | 10 | 200 | 0,55 | 0,90 | 4,21 |
| | | 2000 | 0,78 | 1,29 | 4,25 |
| | | 10000 | 1,06 | 1,48 | 4,30 |
| | 100 | 200 | 0,59 | 0,94 | 4,26 |
| | | 2000 | 0,82 | 1,33 | 4,31 |
| | | 10000 | 1,11 | 1,52 | 4,35 |
| 5000 | 1 | 200 | 0,67 | 0,98 | 4,46 |
| | | 2000 | 0,92 | 1,45 | 4,50 |
| | | 10000 | 1,25 | 1,63 | 4,53 |
| | 10 | 200 | 0,71 | 1,02 | 4,49 |
| | | 2000 | 0,96 | 1,49 | 4,53 |
| | | 10000 | 1,29 | 1,67 | 4,56 |
| | 100 | 200 | 0,75 | 1,06 | 4,54 |
| | | 2000 | 1,00 | 1,53 | 4,58 |
| | | 10000 | 1,33 | 1,71 | 4,61 |

### 4.2.4 Outros Experimentos

Após a consolidação dos dados relativos à métrica Tempo de Consenso, procedeu-se o cálculo de mais duas métricas: a Latência e a Vazão (*throughput*).

#### 4.2.4.1 Latência

O cálculo da Latência leva em consideração o tempo de propagação da transação na rede, o tempo de inclusão de um bloco candidato, o tempo de execução do consenso e o tempo necessário para confirmações utilizando a seguinte. fórmula:

$$\text{LT} = \text{TC} + (N_r \times \text{RTT}) \tag{4.6}$$

onde:

- LT: Latência Total
- TC: Tempo de Consenso (medido experimentalmente em segundos)
- $N_{r:}$ Número de rodadas do algoritmo
- RTT: Tempo de ida e de volta (*Round Trip Time*) = $2 \times delays$

Nas Tabelas 4.7, 4.8 e 4.9 constam os resultados do cálculo das latências totais dentro de cada cenário.

*Tabela 4.7: Latência (em segundos) - redes de baixa densidade*

| Nós | *Delay* | Msg(*bytes*) | BFT-H_TC | BFT-H_LT | PBFT_TC | PBFT_LT | *Raft*_TC | *Raft*_LT |
|---|---|---|---|---|---|---|---|---|
| 30 | 1 | 200 | 0,064 | 0,072 | 0,114 | 0,126 | 1,384 | 1,388 |
| | | 2000 | 0,094 | 0,102 | 0,164 | 0,176 | 1,384 | 1,388 |
| | | 10000 | 0,134 | 0,142 | 0,194 | 0,206 | 1,394 | 1,398 |
| | 10 | 200 | 0,110 | 0,190 | 0,160 | 0,280 | 1,430 | 1,470 |
| | | 2000 | 0,150 | 0,230 | 0,220 | 0,340 | 1,450 | 1,490 |
| | | 10000 | 0,190 | 0,270 | 0,240 | 0,360 | 1,460 | 1,500 |
| | 100 | 200 | 0,500 | 1,300 | 0,560 | 1,760 | 1,920 | 2,320 |
| | | 2000 | 0,540 | 1,340 | 0,630 | 1,830 | 1,930 | 2,330 |
| | | 10000 | 0,590 | 1,390 | 0,660 | 1,860 | 1,940 | 2,340 |
| 60 | 1 | 200 | 0,084 | 0,092 | 0,144 | 0,156 | 1,514 | 1,518 |
| | | 2000 | 0,124 | 0,132 | 0,204 | 0,216 | 1,524 | 1,528 |
| | | 10000 | 0,174 | 0,182 | 0,234 | 0,246 | 1,534 | 1,538 |
| | 10 | 200 | 0,140 | 0,220 | 0,200 | 0,320 | 1,580 | 1,620 |
| | | 2000 | 0,180 | 0,260 | 0,260 | 0,380 | 1,600 | 1,640 |
| | | 10000 | 0,230 | 0,310 | 0,290 | 0,410 | 1,610 | 1,650 |
| | 100 | 200 | 0,530 | 1,330 | 0,600 | 1,800 | 2,060 | 2,460 |
| | | 2000 | 0,580 | 1,380 | 0,690 | 1,890 | 2,070 | 2,470 |
| | | 10000 | 0,650 | 1,450 | 0,750 | 1,950 | 2,080 | 2,480 |

*Tabela 4.8: Latência (em segundos) - Redes de média densidade*

| Nós | *Delay* | Msg(*bytes*) | BFT-H_TC | BFT-H_LT | PBFT_TC | PBFT_LT | *Raft*_TC | *Raft*_LT |
|---|---|---|---|---|---|---|---|---|
| 1000 | 1 | 200 | 0,250 | 0,258 | 0,420 | 0,428 | 2,450 | 2,454 |
| | | 2000 | 0,350 | 0,358 | 0,550 | 0,558 | 2,550 | 2,554 |
| | | 10000 | 0,450 | 0,458 | 0,700 | 0,708 | 2,650 | 2,654 |
| | 10 | 200 | 0,300 | 0,380 | 0,500 | 0,580 | 2,720 | 2,760 |
| | | 2000 | 0,400 | 0,480 | 0,630 | 0,710 | 2,850 | 2,890 |
| | | 10000 | 0,500 | 0,580 | 0,800 | 0,880 | 3,000 | 3,040 |
| | 100 | 200 | 0,380 | 1,180 | 0,650 | 1,450 | 3,250 | 3,650 |
| | | 2000 | 0,500 | 1,300 | 0,800 | 1,600 | 3,400 | 3,800 |
| | | 10000 | 0,650 | 1,450 | 1,000 | 1,800 | 3,550 | 3,950 |

*Tabela 4.9: Latência (em segundos) - Redes de alta densidade*

| Nós | *Delay* | Msg(*bytes*) | BFT-H_TC | BFT-H_LT | PBFT_TC | PBFT_LT | *Raft*_TC | *Raft*_LT |
|---|---|---|---|---|---|---|---|---|
| 2000 | 1 | 200 | 0,514 | 0,522 | 0,864 | 0,876 | 4,184 | 4,188 |
| | | 2000 | 0,744 | 0,752 | 1,254 | 1,266 | 4,224 | 4,228 |
| | | 10000 | 1,024 | 1,032 | 1,444 | 1,456 | 4,264 | 4,268 |
| | 10 | 200 | 0,550 | 0,630 | 0,900 | 1,020 | 4,220 | 4,260 |
| | | 2000 | 0,780 | 0,860 | 1,290 | 1,410 | 4,260 | 4,300 |
| | | 10000 | 1,060 | 1,140 | 1,480 | 1,600 | 4,300 | 4,340 |
| | 100 | 200 | 0,950 | 1,750 | 1,310 | 2,510 | 4,630 | 5,030 |
| | | 2000 | 1,190 | 1,990 | 1,720 | 2,920 | 4,680 | 5,080 |
| | | 10000 | 1,510 | 2,310 | 1,900 | 3,100 | 4,720 | 5,120 |
| 5000 | 1 | 200 | 0,674 | 0,682 | 0,984 | 0,996 | 4,464 | 4,468 |
| | | 2000 | 0,924 | 0,932 | 1,454 | 1,466 | 4,504 | 4,508 |
| | | 10000 | 1,254 | 1,262 | 1,634 | 1,646 | 4,534 | 4,538 |
| | 10 | 200 | 0,710 | 0,790 | 1,020 | 1,140 | 4,500 | 4,540 |
| | | 2000 | 0,960 | 1,040 | 1,490 | 1,610 | 4,540 | 4,580 |
| | | 10000 | 1,290 | 1,370 | 1,670 | 1,790 | 4,570 | 4,610 |
| | 100 | 200 | 1,120 | 1,920 | 1,450 | 2,650 | 4,910 | 5,310 |
| | | 2000 | 1,370 | 2,170 | 1,920 | 3,120 | 4,950 | 5,350 |
| | | 10000 | 1,710 | 2,510 | 2,100 | 3,300 | 4,980 | 5,380 |

#### 4.2.4.2 Vazão (*Throughput*)

A métrica Vazão (*Throughput*), dentro do contexto dos nossos experimentos, é definida como a taxa de transações confirmadas que o sistema consegue processar por unidade de tempo. O cálculo do *Throughput* é realizado utilizando a fórmula:

$$\textit{Throughput} = \text{Tamanho da mensagem}/\text{Latência total} \tag{4.7}$$

Os resultados obtidos em termos de *bytes*/segundos constam nas Tabelas 4.10, 4.11 e 4.12.

*Tabela 4.10: Throughput (em bytes/segundos) - Redes de baixa densidade*

| Nós | *Delay* | Msg_*bytes* | BFT-H_*Throughput* | PBFT_*Throughput* | *Raft_Throughput* |
|---|---|---|---|---|---|
| 15 | 1 | 200 | 3000 | 1700 | 150 |
| | | 2000 | 21000 | 12000 | 1500 |
| | | 10000 | 75000 | 50000 | 7500 |
| | 10 | 200 | 1200 | 800 | 140 |
| | | 2000 | 9000 | 6000 | 1400 |
| | | 10000 | 40000 | 30000 | 7000 |
| | 100 | 200 | 160 | 120 | 90 |
| | | 2000 | 1600 | 1200 | 900 |
| | | 10000 | 8000 | 6000 | 4500 |
| 30 | 1 | 200 | 2777,78 | 1587,30 | 144,09 |
| | | 2000 | 19607,84 | 11363,64 | 1440,92 |
| | | 10000 | 70422,54 | 48543,69 | 7153,08 |
| | 10 | 200 | 1052,63 | 714,29 | 136,05 |
| | | 2000 | 8695,65 | 5882,35 | 1342,28 |
| | | 10000 | 37037,04 | 27777,78 | 6666,67 |
| | 100 | 200 | 153,85 | 113,64 | 86,21 |
| | | 2000 | 1492,54 | 1092,90 | 858,37 |
| | | 10000 | 7194,24 | 5376,34 | 4273,50 |
| 60 | 1 | 200 | 2173,91 | 1282,05 | 131,75 |
| | | 2000 | 15151,52 | 9259,26 | 1308,90 |
| | | 10000 | 54945,05 | 40650,41 | 6501,95 |
| | 10 | 200 | 909,09 | 625,00 | 123,46 |
| | | 2000 | 7692,31 | 5263,16 | 1219,51 |
| | | 10000 | 32228,06 | 24390,24 | 6060,61 |
| | 100 | 200 | 150,38 | 111,11 | 81,30 |
| | | 2000 | 1449,28 | 1058,20 | 809,72 |
| | | 10000 | 6896,55 | 5128,21 | 4032,26 |

*Tabela 4.11: Throughput (em bytes/segundos) - Redes de média densidade*

| Nós | *Delay* | Msg_*bytes* | BFT-H_*Throughput* | PBFT_*Throughput* | *Raft_Throughput* |
|---|---|---|---|---|---|
| 1000 | 1 | 200 | 602,41 | 403,23 | 70,72 |
| | | 2000 | 4424,78 | 2577,32 | 697,35 |
| | | 10000 | 16077,17 | 11286,68 | 3427,00 |
| | 10 | 200 | 454,55 | 289,86 | 63,29 |
| | | 2000 | 3448,28 | 2040,82 | 617,28 |
| | | 10000 | 12500,00 | 9009,01 | 3030,30 |
| | 100 | 200 | 121,21 | 85,47 | 42,37 |
| | | 2000 | 1197,60 | 740,74 | 417,54 |
| | | 10000 | 4739,34 | 3484,32 | 2053,39 |

*Tabela 4.12: Throughput (em bytes/segundos) - Redes de alta densidade*

| Nós | *Delay* | Msg_*bytes* | BFT-H_*Throughput* | PBFT_*Throughput* | *Raft_Throughput* |
|---|---|---|---|---|---|
| 2000 | 1 | 200 | 383,14 | 228,31 | 47,76 |
| | | 2000 | 2639,57 | 1579,78 | 473,04 |
| | | 10000 | 9689,92 | 6868,13 | 2343,02 |
| | 10 | 200 | 317,46 | 196,08 | 46,95 |
| | | 2000 | 2525,58 | 1418,44 | 465,12 |
| | | 10000 | 8771,93 | 6250,00 | 2304,15 |
| | 100 | 200 | 114,29 | 79,68 | 39,76 |
| | | 2000 | 1005,03 | 684,93 | 393,70 |
| | | 10000 | 4329,00 | 3225,81 | 1953,12 |
| 5000 | 1 | 200 | 293,26 | 200,80 | 44,76 |
| | | 2000 | 2145,92 | 1364,26 | 443,66 |
| | | 10000 | 7923,93 | 6075,33 | 2203,61 |
| | 10 | 200 | 253,16 | 175,44 | 44,05 |
| | | 2000 | 1923,08 | 1242,24 | 436,68 |
| | | 10000 | 7299,27 | 5586,59 | 2169,20 |
| | 100 | 200 | 104,17 | 75,47 | 37,66 |
| | | 2000 | 921,66 | 641,03 | 373,83 |
| | | 10000 | 3984,06 | 3030,30 | 1858,74 |

## 4.3 ANÁLISE DE RESULTADOS

A avaliação de desempenho dos algoritmos de consenso BFT-H, PBFT e *Raft* considerou três cenários principais: redes pequenas com até 60 nós, redes médias em torno de 1.000 nós e redes grandes que variaram de 2.000 a 5.000 nós. Esta divisão permitiu compreender como cada um dos algoritmos se comporta em diferentes escalas de operação.

### 4.3.1 Quanto ao Tempo de Consenso

No Experimento 1, realizado com redes pequenas, já foram observadas diferenças interessantes entre os algoritmos. Por exemplo, no cenário de 15 nós e 1 ms de *delay* (Figura. 4.1 e Tabela 4.4), o BFT-H completou o consenso com mensagens de 200 *bytes* em 50 milissegundos, o PBFT precisou de 80ms e o *Raft* ultrapassou 1,2 segundos. Incrementando o tamanho das mensagens para 2.000 e depois 10.000 *bytes*, afetou os três algoritmos, mas em proporções diferentes. O BFT-H, ao processar operações em paralelo nos grupos, conseguiu manter um bom desempenho mesmo com mensagens maiores. O PBFT sofreu mais com o aumento do tamanho das mensagens, pois precisou replicar essas mensagens maiores em suas três rodadas de comunicação. Já o *Raft*, embora mais estável com o aumento do tamanho das mensagens, continuou significativamente mais lento pela necessidade de processar tudo sequencialmente através do líder.

Quando passamos para redes médias (Experimento 2), as diferenças ficaram mais claras. Por exemplo, no cenário de 15 nós e 1 ms de *delay* (Figura 4.4 e Tabela 4.5), o tempo de consenso do BFT-H para mensagens pequenas ficou em torno de 0,32s, já o PBFT precisou de quase meio segundo, e o *Raft* mais de 2,8s. Nota-

se também como o aumento do tamanho das mensagens impactou cada algoritmo nesta escala. O PBFT, por exemplo, viu seu tempo quase dobrar com o aumento do tamanho delas de 200 para 2.000 *bytes* devido à sua comunicação quadrática. Já o BFT-H conseguiu manter um crescimento mais controlado graças ao fatiamento da rede, que limita o número de nós que precisam trocar mensagens maiores.
No Experimento 3, com cenários de redes grandes, mais próximos de cenários reais de uso, a influência das características arquiteturais de cada algoritmo de consenso nos resultados de desempenho ficou ainda mais evidente. Por exemplo, com mensagens pequenas no cenário com 5.000 nós e 1 ms de *delay* (Figura 4.5 e Tabela 4.6), o BFT-H completou o consenso em 0,67s, o PBFT precisou de quase 1s, e o *Raft* mais de 4,4s. O sistema de reputação do BFT-H mostrou sua importância aqui, pois ao manter apenas os nós mais eficientes nas fatias ativas, conseguiu preservar um bom desempenho mesmo com milhares de participantes na rede.

### 4.3.2 Quanto à Latência

A análise dos resultados com a métrica Latência permitiu observar outros aspectos interessantes. Em redes pequenas (Tabela 4.7), com *delay* de apenas 1ms, as diferenças de latência já eram perceptíveis: 58ms para o BFT-H, 92ms para o PBFT e 1,24s para o *Raft*. Aumentando o *delay* para 10ms e depois 100ms, revelou-se como cada algoritmo lida com atrasos de rede. O BFT-H, com suas duas etapas para o consenso, sofreu menor impacto. Já o PBFT, que utiliza três fases completas, viu sua latência crescer mais rapidamente. Além do processamento sequencial, o *Raft* ainda requer a sincronização do log entre o líder e os seguidores. Essa análise trouxe outros aspectos interessantes.
Em redes médias (Tabela 4.8), a latência ampliou a diferença entre os algoritmos. Com *delay* mínimo, o BFT-H a manteve em torno de 0,33s, mas o PBFT aproximou-se de meio segundo, e o *Raft* ultrapassou 2,8s. Quando aumentamos o *delay* da rede para 10ms, uma situação mais comum em ambientes reais, as diferenças de latência se acentuaram. O BFT-H conseguiu manter-se abaixo de meio segundo de latência graças à sua estrutura em fatias e à necessidade de apenas $2f + 1$ confirmações. O PBFT, precisando de $3f + 1$ confirmações em cada uma de suas três fases, viu sua latência subir para quase 0,7s. *Raft*, que depende de validações sequenciais, ultrapassou 3s.
Testes em redes grandes (Tabela 4.9) revelaram como os algoritmos se comportam em situações mais desafiadoras. Com *delay* mínimo, o BFT-H manteve latência em torno de 0,68s, o PBFT chegou a 1s, e o *Raft* passou dos 4,4s. O aumento progressivo do *delay* mostrou diferenças importantes: enquanto o BFT-H precisou lidar com atrasos apenas nas suas duas fases e dentro das fatias ativas, o PBFT acumulou retardos em três fases multiplicados pela comunicação total da rede. O *Raft*, além dos atrasos básicos, ainda somou o tempo necessário para manter os logs sincronizados entre o líder e seguidores.

### 4.3.3 Quanto à Vazão (*Throughput*)

A análise dos resultados em termos da métrica *Throughput* trouxe outra perspectiva. Em redes pequenas (Tabela 4.10) com 30 nós e *delay* mínimo (1ms), mesmo mensagens pequenas (200 bytes), o BFT-H conseguia processar 2,77 KB/s , enquanto o PBFT atingia 1,58 KB/s, e o *Raft* apenas 0,14 KB/s. Com mensagens maiores, essas diferenças se ampliaram consideravelmente. O BFT-H, processando operações

em paralelo nas fatias e com menos fases de comunicação, alcançou mais de 70,42 KB/s com mensagens de 10KB. O PBFT, dividindo sua banda entre todos os nós em três fases, chegou a 48,54 KB/s. Já o Raft, que concentra todo processamento no líder, chegou a 7,15 KB/s.

Em redes médias (Tabela 4.11), o impacto da arquitetura de cada um dos algoritmos ficou mais evidente. Com *delay* de 10ms, uma configuração bastante realista, o *throughput* para mensagens pequenas já mostrava grandes diferenças. O BFT-H, mantendo sua complexidade em $O(n)$ e processamento distribuído nas fatias, conseguiu *throughput* quase duas vezes maior que o PBFT, que sofria com seu intrincamento quadrático. O *Raft* obteve *throughput* muito inferior devido ao gargalo criado pelo processamento sequencial, mesmo com complexidade linear.

Nos testes com redes grandes (Tabela 4.12) e *delay* significativo, cenário que mais se aproxima de aplicações reais distribuídas, as diferenças de arquitetura se traduziram diretamente em desempenho. O BFT-H conseguiu manter *throughput* utilizável mesmo com mensagens grandes, chegando a quase 4 KB/s. Já o PBFT, acumulando *delay* em suas três fases e comunicação total, alcançou 3 KB/s. O *Raft* ficou abaixo de 2 KB/s, devido a sua abordagem sequencial.

Os resultados em larga escala, especialmente com 5.000 nós e *delay* de 100ms, mostram como diferentes escolhas arquiteturais impactam o desempenho dos algoritmos. O BFT-H, combinando fatiamento dinâmico, seleção por reputação e consenso em duas fases, conseguiu reduzir o tempo de consenso em quase um terço comparado ao PBFT e em mais de 80% em relação ao *Raft*. A latência apresentou padrão similar, com reduções de 27,5% e 63,8%. Utilizando o mecanismo ADAN, cada grupo mantém no mínimo $\left\lfloor \frac{|N|}{2} \right\rfloor + 1$ nós, diminuindo substancialmente o volume de mensagens trocadas. O algoritmo também possui um sistema de reputação baseado na fórmula $R(n) = H(n) \times P(n)$, que avalia tanto o histórico quanto o comportamento atual dos nós, permitindo que apenas os nós mais confiáveis participem.

## 4.4 VALIDAÇÃO DA HIPÓTESE

A proposta do BFT-H partiu da hipótese de que seria possível melhorar o desempenho de sistemas distribuídos de gestão de saúde baseados em *Blockchain* através de um mecanismo de consenso bizantino que combina reputação e fatiamento adaptativo, sem renunciar à segurança. Os resultados dos experimentos confirmaram esta possibilidade.

Em redes grandes, que refletem a realidade de sistemas de saúde distribuídos, o uso do BFT-H permitiu reduzir o tempo de consenso em quase um terço comparado ao PBFT e em mais de 80% em relação ao *Raft*. Isso acontece porque o fatiamento adaptativo resolve um problema de comunicação entre todos os nós de uma rede, organizando a rede em grupos menores onde os participantes realizam atividades em paralelo. A questão da latência, que preocupa muito quem trabalha com dados médicos críticos, também mostrou que, mesmo com atrasos de rede de 100ms, situação comum quando conectamos unidades de saúde distantes, o BFT-H foi 27,5% mais rápido que o PBFT e 63,8% mais ágil que o *Raft*. O sistema de reputação teve papel fundamental aqui: ao manter ativos apenas os nós mais confiáveis, conseguimos respostas mais rápidas mesmo em condições adversas.

Os resultados de *throughput*, que na prática determina quantos registros médicos consegue-se processar por segundo, em redes grandes, mostraram o BFT-H com um *throughput* 31,4% maior que o PBFT e mais que o dobro do *Raft*. Esta capacidade extra faz toda diferença em situações críticas, como durante emer-

gências médicas ou surtos epidêmicos, quando o sistema precisa lidar com picos repentinos de demanda. Quanto à segurança, que não podíamos comprometer, os resultados também foram positivos. Mantivemos todas as garantias bizantinas através de uma estrutura cuidadosa das fatias e do número de confirmações necessárias. O sistema de reputação ainda acrescentou uma camada extra de proteção, identificando comportamentos suspeitos antes que possam causar problemas.

Um aspecto motivador é como o algoritmo se adapta bem a diferentes realidades do sistema de saúde. Funciona bem tanto em redes pequenas, como entre postos de saúde, quanto em redes médias, conectando hospitais regionais, e até em grande escala, integrando sistemas estaduais ou nacionais. Esta flexibilidade é extremamente importante em um país com realidades tão diversas.

# 5 CONCLUSÃO E TRABALHOS FUTUROS

O algoritmo BFT-H (*Byzantine Fault Tolerance to Health*) introduziu avanços técnicos para o consenso bizantino em sistemas distribuídos de gestão de saúde baseados em Blockchain. O mecanismo ADAN (*Adaptive Decentralized Asynchronous Node-slicing*) comprovou a viabilidade de redução da complexidade de comunicação de $O(n^2)$ para $O(n)$ através do fatiamento dinâmico da rede.

A validação experimental em redes de grande escala demonstrou reduções de 33% no tempo de consenso comparado ao PBFT e 80% em relação ao *Raft*. Na análise de latência, o algoritmo alcançou reduções de 27,5% em comparação ao PBFT e 63,8% ao Raft, mesmo sob condições de latência de rede de 100ms. O sistema de reputação, integrado ao mecanismo ADAN, estabeleceu critérios efetivos para seleção e organização dos nós participantes. O processo de consenso em duas fases manteve as garantias de segurança bizantina durante a ordenação e execução das operações.

O desenvolvimento e análise de desempenho do BFT-H revelaram limitações técnicas que requerem investigação adicional. A eficiência do mecanismo ADAN depende diretamente da precisão das métricas de reputação, apresentando variações de desempenho quando a coleta destes dados sofre interferências ou atrasos. O processo de reorganização das fatias durante falhas de consenso gera *overhead* temporário no sistema, embora os testes sugiram quem são presentavadas as garantias de tolerância à falha bizantina. O período inicial de calibragem necessário para estabelecer pontuações de reputação confiáveis resulta em eficiência reduzida até o acúmulo de dados comportamentais suficientes. A heterogeneidade de recursos computacionais entre os nós, característica de ambientes como o do SUS, impõe desafios ao balanceamento entre disponibilidade e desempenho.

Os resultados obtidos indicam direções técnicas para evolução do algoritmo BFT-H. O mecanismo ADAN pode incorporar algoritmos de aprendizado de máquina para otimização preditiva da formação das fatias baseada em padrões históricos. O desenvolvimento de estratégias para redução do *overhead* durante reorganizações e a implementação de fatiamento adaptativo baseado em padrões de tráfego e distribuição geográfica dos nós constituem aprimoramentos potenciais.

O sistema de reputação requer evolução através da implementação de validação cruzada entre fatias para mitigação de manipulação das pontuações. O desenvolvimento de métricas específicas para diferentes categorias de unidades de saúde e a incorporação de técnicas avançadas de detecção de anomalias podem aumentar a precisão da avaliação comportamental dos nós. A integração com tecnologias emergentes representa uma direção de pesquisa promissora. A investigação de algoritmos quânticos para otimização do consenso, o aproveitamento das características de baixa latência das redes 5G e o desenvolvimento de estratégias para processamento distribuído em bordas da rede podem resultar em ganhos de desempenho significativos. O desenvolvimento de *frameworks* para verificação formal das propriedades do BFT-H e o aprofundamento dos estudos sobre resistência a ataques bizantinos são necessários para validação rigorosa da segurança. Adaptações para conformidade com normas específicas de proteção de dados em saúde devem ser contempladas nestas análises.

O BFT-H estabeleceu bases técnicas para evolução dos algoritmos de consenso bizantino em sistemas de saúde distribuídos. As limitações identificadas e direções de pesquisa propostas definem um roteiro técnico para seu aprimoramento contínuo, visando maior eficiência e confiabilidade no processamento distribuído

de dados em saúde.

# REFERÊNCIAS BIBLIOGRÁFICAS


1 BRASIL. Departamento de informática do sistema Único de saúde (DATASUS). 2024. Sistema de Informações sobre Mortalidade (SIM). Disponível em: <http://datasus.saude.gov.br/>. Acesso em: 2 nov. 2024.

2 FREIRE, F. R.; GIOZZA, W. F.; RODRIGUES, C. K. d. S. Towards consensus algorithm for healthcare management systems in blockchains. *Current Trends in Computer Sciences  Applications*, v. 2, n. 4, p. 228–234, 2023.

3 CASTRO, M.; LISKOV, B. Practical byzantine fault tolerance. In: *Proceedings of the OSDI*. [S.l.]: USENIX Association, 1999. v. 99, p. 173–186.

4 HU, J.; LIU, K. Raft consensus mechanism and the applications. *Journal of Physics: Conference Series*, v. 1544, n. 1, p. 012079, 2020.

5 FREIRE, F. R.; GIOZZA, W. F.; RODRIGUES, C. K. d. S. Proposta de um algoritmo de consenso para plataformas blockchain em sistemas de gestão de saúde privados. *RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação*, n. E57, p. 103–116, abr. 2023.

6 World Health Organization. *Global strategy on digital health 2020-2025*. Geneva: World Health Organization, 2021.

7 NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, p. 21260, 2008.

8 RODRIGUES, C. K. S. Blockchain-based platform for managing patients' data in the public healthcare system of brazil. *Revista de Sistemas e Computação*, v. 11, n. 3, p. 63–72, 2021.

9 JOSHI, A. P.; HAN, M.; WANG, Y. A survey on security and privacy issues of blockchain technology. *Mathematical Foundations of Computing*, v. 1, n. 2, p. 121–147, 2018. Disponível em: <https://www.aimsciences.org/article/id/d27803a2-7ce7-46e8-900d-30001fd4785a>.

10 REN, Y. et al. Sustainable finance and blockchain: A systematic review and research agenda. *Research in International Business and Finance*, v. 64, p. 101871, 2023.

11 BONYUET, D. Overview and impact of blockchain on auditing. *The International Journal of Digital Accounting Research*, v. 20, p. 31–43, 2020.

12 GARCIA, R. D.; RAMACHANDRAN, G.; UEYAMA, J. Exploiting smart contracts in pbft-based blockchains: A case study in medical prescription system. *Computer Networks*, v. 211, 2022.

13 NEUDECKER, T.; HARTENSTEIN, H. Network layer aspects of permissionless blockchains. *IEEE Communications Surveys & Tutorials*, IEEE, v. 21, n. 1, p. 838–857, 2019.

14 CHEN, Y. et al. Efficient and secure blockchain consensus algorithm for heterogeneous industrial internet of things nodes based on double-dag. *IEEE Transactions on Industrial Informatics*, v. 20, n. 4, p. 6300–6312, 2024.

15 VENKATESAN, K.; RAHAYU, S. B. Blockchain security enhancement: an approach towards hybrid consensus algorithms and machine learning techniques. *Scientific Reports*, v. 14, n. 1, p. 1149, 2024.

16 KAPSOULIS, N. et al. Know your customer (kyc) implementation with smart contracts on a privacy-oriented decentralized architecture. *Future Internet*, v. 12, n. 2, p. 41, 2020.

17 OYINLOYE, D. P. et al. Blockchain consensus: An overview of alternative protocols. *Symmetry*, v. 13, n. 8, p. 1363, 2021.

18 GARAY, J.; KIAYIAS, A.; LEONARDOS, N. The bitcoin backbone protocol: Analysis and applications. *J. ACM*, Association for Computing Machinery, New York, NY, USA, v. 71, n. 4, ago. 2024. ISSN 0004-5411. Disponível em: <https://doi-org.ez54.periodicos.capes.gov.br/10.1145/3653445>.

19 KARTHIKEYAN, I.; SHRUTHI, M. G. The evolution of block chain technology. In: AURELIA, S.; J., C.; IMMANUEL, A.; MANI, J.; PADMANABHA, V. (Ed.). *Computational Sciences and Sustainable Technologies*. Cham: Springer Nature Switzerland, 2024. p. 488–497.

20 KUSHWAHA, S. S.; JOSHI, S.; SINGH, D.; KAUR, M.; LEE, H.-N. Systematic review of security vulnerabilities in Ethereum Blockchain smart contract. *IEEE Access*, IEEE, v. 10, p. 6605–6621, 2022.

21 RUKHIRAN, M.; BOONSONG, S.; NETINANT, P. Sustainable optimizing performance and energy efficiency in proof of work blockchain: A multilinear regression approach. *Sustainability*, v. 16, n. 4, 2024. ISSN 2071-1050. Disponível em: <https://www.mdpi.com/2071-1050/16/4/1519>.

22 HAQUE, E. U.; SHAH, A.; IQBAL, J.; ULLAH, S. S.; ALROOBAEA, R.; HUSSAIN, S. A scalable blockchain based framework for efficient iot data management using lightweight consensus. *Scientific Reports*, v. 14, n. 1, p. 7841, 2024. Disponível em: <https://doi.org/10.1038/s41598-024-58578-7>.

23 CHEN, Y.; ZHANG, Y.; ZHUANG, Y.; MIAO, K.; POURIYEH, S.; HAN, M. Efficient and secure blockchain consensus algorithm for heterogeneous industrial internet of things nodes based on double-dag. *IEEE Transactions on Industrial Informatics*, v. 20, n. 4, p. 6300–6312, 2024.

24 WU, L. et al. A scalable multi-layer pbft consensus for blockchain. *IEEE Transactions on Parallel and Distributed Systems*, v. 32, n. 5, p. 1146–1160, 2020.

25 KIM, H.; KIM, D. A taxonomic hierarchy of blockchain consensus algorithms: An evolutionary phylogeny approach. *Sensors*, v. 23, n. 5, p. 2739, 2023.

26 BEZUIDENHOUT, D. et al. Permissionless blockchain systems as pseudo-random number generators for decentralized consensus. In: *International Conference on Computational Science and Its Applications*. [S.l.]: Springer, Cham, 2022. p. 157–172.

27 MORIYAMA, K.; OTSUKA, A. Permissionless blockchain-based sybil-resistant self-sovereign identity utilizing attested execution secure processors. *IEICE Transactions on Information and Systems*, E107.D, n. 9, p. 1112–1122, 2024.

28 ZHONG, Y. et al. St-pbft: An optimized pbft consensus algorithm for intellectual property transaction scenarios. *IEEE Access*, v. 8, p. 82604–82616, 2020.

29 WANG, D.; WANG, W. Consensus algorithm of third-order multi-agent systems with time delay in undirected networks based on partial neighbour information. *Journal of Control and Decision*, v. 11, n. 2, p. 201–210, 2024.

30 CASTRO, M.; LISKOV, B. Practical byzantine fault tolerance. *ACM Transactions on Computer Systems*, v. 20, p. 398–461, 2002. Disponível em: <https://doi.org/10.1145/571637.571640>.

31 JIANG, W. et al. A scalable byzantine fault tolerance algorithm based on a tree topology network. *IEEE Access*, v. 11, p. 33509–33520, 2023.

32 LIU, S. et al. Blockchain-based decentralized federated learning method in edge computing environment. *Applied Sciences*, v. 13, n. 3, p. 1677, 2023.

33 ZHONG, W.; FENG, W.; HUANG, M.; FENG, S. St-pbft: An optimized pbft consensus algorithm for intellectual property transaction scenarios. *Electronics*, v. 12, p. 325, 2023. Disponível em: <https://doi.org/10.3390/electronics12020325>.

34 HEGDE, P.; MADDIKUNTA, P. K. R. Secure pbft consensus-based lightweight blockchain for healthcare application. *Applied Sciences*, v. 13, n. 6, p. 3757, 2023.

35 SUN, Y.; HU, G.; ZHANG, Y.; LU, B.; LU, Z.; FAN, J.; LI, X.; DENG, Q.; CHEN, X. Eigen microstates and their evolutions in complex systems. *Communications in Theoretical Physics*, IOP Publishing, v. 73, n. 6, p. 065603, may 2021. Disponível em: <https://dx.doi.org/10.1088/1572-9494/abf127>.

36 ONGARO, D.; OUSTERHOUT, J. In search of an understandable consensus algorithm. In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. [S.l.: s.n.], 2014. p. 305–319.

37 LAMPORT, L. The part-time parliament. *ACM Trans. Comput. Syst.*, Association for Computing Machinery, New York, NY, USA, v. 16, n. 2, p. 133–169, maio 1998. ISSN 0734-2071. Disponível em: <https://doi.org/10.1145/279227.279229>.

38 LU, S. et al. P-raft: An efficient and robust consensus mechanism for consortium blockchains. *Electronics*, v. 12, n. 10, p. 2271, 2023.

39 ZHAN, Z.; HUANG, R. Improvement of hierarchical byzantine fault tolerance algorithm in raft consensus algorithm election. *Applied Sciences*, v. 13, p. 9125, 2023.

40 LIN, Q. et al. Secure internet of medical things (iomt) based on ecmqv-mac authentication protocol and ekmc-scp blockchain networking. *Information Sciences*, v. 654, p. 119783, 2024.

41 BARNES, T. D. B. Information security maturity model for healthcare organizations in the united states. *IEEE Transactions on Engineering Management*, v. 71, p. 928–939, 2024.

42 MAT, N. I. C.; OTHERS. A systematic literature review on advanced persistent threat behaviors and its detection strategy. *Journal of Cybersecurity*, v. 10, n. 1, p. 1–18, 2024.

43 SHOJAEI, P.; VLAHU-GJORGIEVSKA, E.; CHOW, Y. Security and privacy of technologies in health information systems: A systematic literature review. *Computers*, v. 13, n. 2, p. 41, 2024.

44 XU, J. et al. Healthchain: A blockchain-based privacy preserving scheme for large-scale health data. *IEEE Internet of Things Journal*, v. 6, n. 5, p. 8770–8781, 2019.

45 ABDELLATIF, A. A. et al. sshealth: Toward secure, blockchain-enabled healthcare systems. *IEEE Network*, v. 34, n. 4, p. 312–319, 2020.

46 GARG, N. et al. Bakmp-iomt: Design of blockchain enabled authenticated key management protocol for internet of medical things deployment. *IEEE Access*, v. 8, p. 95956–95977, 2020.

47 HOSSEIN, K. M. et al. Bchealth: A novel blockchain-based privacy-preserving architecture for iot healthcare applications. *Computer Communications*, v. 180, p. 31–47, 2021.

48 PANDEY, P.; OTHERS. A decentralized architecture based on blockchain for healthcare industry. *Journal of King Saud University-Computer and Information Sciences*, v. 35, n. 4, p. 101479, 2023.

49 DEBRECZENI, M.; KLENIK, A.; KOCSIS, I. Transaction conflict control in hyperledger fabric: A taxonomy, gaps, and design for conflict prevention. *IEEE Access*, v. 12, p. 18987–19008, 2024.

50 VIDAP, A.; OTHERS. A blockchain-based framework for electronic health records using hyperledger fabric: Implementation and performance evaluation. *Journal of Medical Systems*, v. 47, n. 2, p. 1–15, 2023.

51 HIRA, S.; OTHERS. Blockchain-based electronic health records in malaysia: Current status, challenges, and future directions. *International Journal of Medical Informatics*, v. 174, p. 105087, 2024.

52 RODRIGUES, C. Blockchain-based platform for managing patients' data in the public healthcare system of brazil. *Revista de Sistemas e Computação*, v. 11, 12 2021.

53 BRASIL. Lei nº 13.709, de 14 de agosto de 2018. lei geral de proteção de dados pessoais (lgpd). 2018. Acesso em: [01 de novembro de 2024]. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm>.

54 QILIANG, Y.; MINGRUI, Z.; YANWEI, Z.; YONG, Y. Attribute-based worker selection scheme by using blockchain in decentralized crowdsourcing scenario. *Chinese Journal of Electronics*, v. 30, p. 249–257, 2021.

55 IZHAR, M.; NAQVI, S. A. A.; AHMED, A.; ABDULLAH, S.; ALTURKI, N.; JAMEL, L. Enhancing healthcare efficacy through iot-edge fusion: A novel approach for smart health monitoring and diagnosis. *IEEE Access*, v. 11, p. 136456–136467, 2023.

56 RYU, H.; KIM, H.; AGARWAL, S.; SHARMA, D. K.; KAPITO, B.; ALI, P. Data sovereignty provision blockchain for remote healthcare service. In: *2023 6th International Conference on Information Systems and Computer Networks (ISCON)*. [S.l.]: IEEE, 2023.

57 AL-SUMAIDAEE, G. et al. Performance analysis of a private blockchain network built on hyperledger fabric for healthcare. *Information Processing  Management*, v. 60, n. 2, p. 103633, 2023.

58 TAHERDOOST, H. Privacy and security of blockchain in healthcare: Applications, challenges, and future perspectives. *Sci*, v. 5, n. 4, p. 41, 2023.

59 European Parliament and Council of European Union. *General Data Protection Regulation (GDPR)*. 2016. Regulation (EU) 2016/679. Disponível em: <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679>.

60 104th United States Congress. *Health Insurance Portability and Accountability Act of 1996 (HIPAA)*. 1996. Public Law 104-191. Disponível em: <https://www.govinfo.gov/content/pkg/PLAW-104publ191/pdf/PLAW-104publ191.pdf>.

61 YAJAM, H.; EBADI, E.; AKHAEE, M. A. JABS: A blockchain simulator for researching consensus algorithms. *IEEE Transactions on Network Science and Engineering*, IEEE, v. 11, n. 1, p. 3–13, jan 2024.

62 JAIN, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. [S.l.]: John Wiley & Sons, 1991. ISBN 978-0471503361.

# 6. APÊNDICE A

## 6.1 IMPLEMENTAÇÃO DO BFT-H

```
// bft-h.h
#ifndef BFT_H_H
#define BFT_H_H

#include "ns3/application.h"
#include "ns3/event-id.h"
#include "ns3/ptr.h"
#include "ns3/ipv4-address.h"
#include "ns3/traced-callback.h"
#include "ns3/random-variable-stream.h"
#include <map>
#include <set>
#include <vector>
#include <string>

namespace ns3 {

enum class NodoState {
    CALIBRATION,
    OPERATIONAL,
    SUSPICIOUS,
    PENALIZED,
    PROBATION
};

struct NodoMetrics {
    uint32_t interactionCount;
    uint32_t successfulConsensus;
    std::vector<Time> responseTimes;
    double resourceUtilization;
    Time uptime;
    NodoState state;
    Time stateTransitionTime;
    Time monitoringStart;
    uint32_t missedHeartbeats;
};

```

```
struct SliceConfig {
    std::vector<std::set<uint32_t>> slices;
    double reputationThreshold;
    uint32_t minSize;
    Time validityPeriod;
};


struct BftMessage {
    enum Type {
        REQUEST,
        CONFIG,
        CONFIG_ACK,
        EXECUTE,
        EXECUTE_ACK,
        VIEW_CHANGE,
        NEW_VIEW
    };

    Type type;
    uint32_t view;
    uint32_t seq;
    std::string content;
    std::string digest;
    uint32_t sender;
    Time timestamp;
};


class BFTH : public Application {
public:
    static TypeId GetTypeId(void);
    BFTH();
    virtual ~BFTH();

protected:
    void DoDispose(void) {
        NS_LOG_FUNCTION(this);

        // Cancela eventos pendentes
        Simulator::Cancel(m_sendEvent);
        Simulator::Cancel(m_heartbeatEvent);
        Simulator::Cancel(m_leaderCheckEvent);

        // Limpa recursos
        m_socket = nullptr;
```

```
84
85          // Chama o dispose da classe base
86          Application::DoDispose();
87      }
88
89  private:
90      virtual void StartApplication(void);
91      virtual void StopApplication(void);
92
93      class ADAN {
94      public:
95          ADAN(uint32_t totalNodos)
96              : m_totalNodos(totalNodos)
97              , m_minSliceSize((totalNodos / 2) + 1) {
98
99              // Configura parametros adaptativos
100             m_config.recalculationPeriod = MilliSeconds(100);
101             m_config.observationWindow = Seconds(1);
102             m_config.reorganizationInterval = Seconds(5);
103             m_config.sliceSize = m_minSliceSize;
104             m_config.reputationThreshold = 0.5;
105             m_config.overlapFactor = 2;
106
107             // Inicializa janelas de obs
108             m_observationWindows = {
109                 MilliSeconds(100),
110                 MilliSeconds(500),
111                 Seconds(1),
112                 Seconds(5)
113             };
114
115             // Cria fatia inicial com todos os nos
116             std::set<uint32_t> initialSlice;
117             for (uint32_t i = 0; i < totalNodos; i++) {
118                 initialSlice.insert(i);
119             }
120             m_slices.push_back(initialSlice);
121         }
122         struct AdaptiveConfig {
123             Time recalculationPeriod;
124             Time observationWindow;
125             Time reorganizationInterval;
126             uint32_t sliceSize;
127             double reputationThreshold;
128             uint32_t overlapFactor;
129         };
```

```
130
131         void AdaptNetwork(const std::map<uint32_t, NodoMetrics>& metrics) {
132             NS_LOG_FUNCTION(this);
133
134             // Coleta nos com reputacao adequada
135             std::vector<uint32_t> eligibleNodos;
136             for (const auto& [nodeId, nodeMetrics] : metrics) {
137                 if (nodeMetrics.state == NodoState::OPERATIONAL &&
138                     nodeMetrics.resourceUtilization >= 0.5) {
139                     eligibleNodos.push_back(nodeId);
140                 }
141             }
142
143             // Verifica se existem nos suficientes para formar fatias validas
144             if (eligibleNodos.size() < m_minSliceSize) {
145                 NS_LOG_WARN("Nos elegiveis insuficientes para fatiamento");
146                 return;
147             }
148
149             // Ordena nos por metricas combinadas
150             std::sort(eligibleNodos.begin(), eligibleNodos.end(),
151                 [&metrics](uint32_t a, uint32_t b) {
152                     const auto& metricsA = metrics.at(a);
153                     const auto& metricsB = metrics.at(b);
154
155                     // Calcula score combinado
156                     double scoreA = metricsA.resourceUtilization *
157                                     (metricsA.successfulConsensus /
158                                      static_cast<double>(metricsA.interactionCount));
159
160                     double scoreB = metricsB.resourceUtilization *
161                                     (metricsB.successfulConsensus /
162                                      static_cast<double>(metricsB.interactionCount));
163
164                     return scoreA > scoreB;
165                 });
166
167             // Forma novas fatias
168             std::vector<std::set<uint32_t>> newSlices;
169             size_t currentIndex = 0;
170
171             while (currentIndex < eligibleNodos.size()) {
172                 // Cria nova fatia
173                 std::set<uint32_t> slice;
174
175                 // Adiciona nos na fatia atual
```

```
176                 for (size_t i = 0; i < m_config.sliceSize &&
177                      currentIndex < eligibleNodos.size(); i++) {
178                     slice.insert(eligibleNodos[currentIndex++]);
179                 }
180
181                 // Adiciona nos sobrepostos se necessario
182                 if (m_config.overlapFactor > 1 && currentIndex <
                        eligibleNodos.size()) {
183                     for (size_t i = 0; i < m_config.overlapFactor - 1; i++) {
184                         slice.insert(eligibleNodos[currentIndex++]);
185                     }
186                 }
187
188                 // Valida e adiciona a fatia
189                 if (slice.size() >= m_minSliceSize) {
190                     newSlices.push_back(slice);
191                 }
192             }
193
194             // Valida nova configuracao
195             SliceConfig newConfig;
196             newConfig.slices = newSlices;
197             newConfig.reputationThreshold = m_config.reputationThreshold;
198             newConfig.minSize = m_minSliceSize;
199             newConfig.validityPeriod = m_config.recalculationPeriod;
200
201             if (ValidateSlicing(newConfig)) {
202                 m_slices = std::move(newSlices);
203                 UpdateSliceState(metrics);
204             }
205         }
206
207         bool ValidateSlicing(const SliceConfig& config) {
208             NS_LOG_FUNCTION(this);
209
210             // Verifica se ha fatias suficientes
211             if (config.slices.empty()) {
212                 NS_LOG_WARN("Nenhuma fatia na configuracao");
213                 return false;
214             }
215
216             // Verifica cada fatia
217             for (const auto& slice : config.slices) {
218                 // Verifica tamanho minimo
219                 if (slice.size() < m_minSliceSize) {
```

```
220                     NS_LOG_WARN("Tamanho da fatia abaixo do minimo: " <<
                            slice.size());
221                     return false;
222                 }
223
224                 // Verifica tolerancia bizantina
225                 uint32_t maxFaults = (slice.size() - 1) / 3;
226                 if (maxFaults * 3 + 1 > slice.size()) {
227                     NS_LOG_WARN("Fatia nao satisfaz tolerancia bizantina");
228                     return false;
229                 }
230
231                 // Verifica conectividade
232                 bool hasOverlap = false;
233                 for (size_t i = 1; i < config.slices.size(); i++) {
234                     std::vector<uint32_t> intersection;
235                     std::set_intersection(
236                         slice.begin(), slice.end(),
237                         config.slices[i].begin(), config.slices[i].end(),
238                         std::back_inserter(intersection)
239                     );
240                     if (!intersection.empty()) {
241                         hasOverlap = true;
242                         break;
243                     }
244                 }
245
246                 if (!hasOverlap && config.slices.size() > 1) {
247                     NS_LOG_WARN("Fatia nao tem sobreposicao com outras fatias");
248                     return false;
249                 }
250             }
251
252             return true;
253         }
254
255         void UpdateSliceState(const std::map<uint32_t, NodoMetrics>& state) {
256             NS_LOG_FUNCTION(this);
257
258             m_currentState = state;
259
260             // Atualiza metricas por fatia
261             for (size_t i = 0; i < m_slices.size(); i++) {
262                 double avgResourceUtilization = 0.0;
263                 double avgSuccessRate = 0.0;
264                 uint32_t activeNodos = 0;
```

```
265
266                 for (uint32_t nodeId : m_slices[i]) {
267                     if (state.count(nodeId) > 0) {
268                         const auto& metrics = state.at(nodeId);
269                         if (metrics.state == NodoState::OPERATIONAL) {
270                             avgResourceUtilization += metrics.resourceUtilization;
271                             avgSuccessRate +=
                                static_cast<double>(metrics.successfulConsensus) /
272                                           metrics.interactionCount;
273                             activeNodos++;
274                         }
275                     }
276                 }
277
278                 if (activeNodos > 0) {
279                     avgResourceUtilization /= activeNodos;
280                     avgSuccessRate /= activeNodos;
281
282                     // Atualiza parametros adaptativos se necessario
283                     if (avgResourceUtilization < 0.3 || avgSuccessRate < 0.5) {
284                         m_config.sliceSize = std::max(m_minSliceSize,
285                             static_cast<uint32_t>(m_config.sliceSize * 0.8));
286                     } else if (avgResourceUtilization > 0.8 && avgSuccessRate > 0.9)
                        {
287                         m_config.sliceSize = std::min(m_totalNodos,
288                             static_cast<uint32_t>(m_config.sliceSize * 1.2));
289                     }
290                 }
291             }
292         }
293
294         const std::vector<std::set<uint32_t>>& BFTH::ADAN::GetSlices() const {
295             return m_slices;
296         }
297
298         bool NeedsReorganization(const std::map<uint32_t, NodoMetrics>& metrics) {
299             NS_LOG_FUNCTION(this);
300
301             // Verifica se passou tempo suficiente desde ultima reorganizacao
302             if (Simulator::Now() - m_lastReorganization <
                m_config.reorganizationInterval) {
303                 return false;
304             }
305
306             // Conta nos problematicos por fatia
307             for (const auto& slice : m_slices) {
```

```
                uint32_t problematicNodos = 0;
                uint32_t totalNodos = slice.size();

                for (uint32_t nodeId : slice) {
                    if (metrics.count(nodeId) > 0) {
                        const auto& nodeMetrics = metrics.at(nodeId);

                        // Verifica condicoes problematicas
                        if (nodeMetrics.state != NodoState::OPERATIONAL ||
                            nodeMetrics.resourceUtilization < 0.3 ||
                            nodeMetrics.missedHeartbeats > 2 ||
                            (nodeMetrics.successfulConsensus <
                             nodeMetrics.interactionCount * 0.5)) {
                            problematicNodos++;
                        }
                    }
                }

                // Se mais de 20 porcento dos nos estao problematicos, reorganiza
                if (static_cast<double>(problematicNodos) / totalNodos > 0.2) {
                    return true;
                }
            }

            // Verifica desbalanceamento entre fatias
            if (m_slices.size() > 1) {
                size_t minSize = m_slices[0].size();
                size_t maxSize = minSize;

                for (const auto& slice : m_slices) {
                    minSize = std::min(minSize, slice.size());
                    maxSize = std::max(maxSize, slice.size());
                }

                // Se diferenca e maior que 50porcento, reorganiza
                if (static_cast<double>(maxSize - minSize) / minSize > 0.5) {
                    return true;
                }
            }

            return false;
        }

    private:
        uint32_t m_totalNodos;
        uint32_t m_minSliceSize;
```

```
354         std::vector<std::set<uint32_t>> m_slices;
355         AdaptiveConfig m_config;
356         std::vector<Time> m_observationWindows;
357         std::map<uint32_t, NodoMetrics> m_currentState;
358     };
359 
360 
361     // Sistema de reputacao
362     class ReputationSystem {
363     private:
364         struct NodoMetrics {
365             uint32_t interactionCount;
366             uint32_t successfulConsensus;
367             std::vector<Time> responseTimes;
368             double resourceUtilization;
369             Time uptime;
370             NodoState state;
371             Time stateTransitionTime;
372         };
373 
374     public:
375         ReputationSystem() {
376             NS_LOG_FUNCTION(this);
377         }
378         void UpdateState(uint32_t nodeId) {
379             NS_LOG_FUNCTION(this << nodeId);
380 
381             auto& metrics = m_nodeMetrics[nodeId];
382             auto currentState = m_nodeStates[nodeId];
383 
384             switch (currentState) {
385                 case NodoState::CALIBRATION:
386                     // Transicao apos 100 interacoes
387                     if (metrics.interactionCount >= 100) {
388                         if (metrics.successfulConsensus >= 80) { // 80porcento
                                sucesso
389                             TransitionTo(nodeId, NodoState::OPERATIONAL);
390                         } else {
391                             TransitionTo(nodeId, NodoState::SUSPICIOUS);
392                         }
393                     }
394                     break;
395 
396                 case NodoState::OPERATIONAL:
397                     if (DetectAnomalous(metrics)) {
398                         TransitionTo(nodeId, NodoState::SUSPICIOUS);
```

```
399                         metrics.monitoringStart = Simulator::Now();
400                     }
401                     break;
402 
403                 case NodoState::SUSPICIOUS:
404                     // Monitora por 1000 operacoes
405                     if (metrics.interactionCount - metrics.monitoringStart >= 1000) {
406                         if (BehaviorNormalized(metrics)) {
407                             TransitionTo(nodeId, NodoState::OPERATIONAL);
408                         } else {
409                             TransitionTo(nodeId, NodoState::PENALIZED);
410                         }
411                     }
412                     break;
413 
414                 case NodoState::PENALIZED:
415                     // Apos periodo de penalidade, move para probatorio
416                     if (Simulator::Now() - metrics.stateTransitionTime >=
                        Seconds(300)) {
417                         TransitionTo(nodeId, NodoState::PROBATION);
418                         metrics.successfulConsensus = 0;
419                         metrics.interactionCount = 0;
420                     }
421                     break;
422 
423                 case NodoState::PROBATION:
424                     // Avalia comportamento durante probacao
425                     if (metrics.interactionCount >= 50) {  // Minimo de 50 interacoes
426                         if (BehaviorNormalized(metrics)) {
427                             TransitionTo(nodeId, NodoState::OPERATIONAL);
428                         } else {
429                             TransitionTo(nodeId, NodoState::PENALIZED);
430                         }
431                     }
432                     break;
433             }
434         }
435 
436         double CalculateReputation(uint32_t nodeId) {
437             NS_LOG_FUNCTION(this << nodeId);
438 
439             double historyScore = m_historyScores[nodeId];
440             double participationScore = m_participationScores[nodeId];
441 
442             // Ajusta baseado no estado atual
443             switch (m_nodeStates[nodeId]) {
```

```
444                 case NodoState::CALIBRATION:
445                     return 0.5;  // Reputacao inicial moderada
446
447                 case NodoState::OPERATIONAL:
448                     return historyScore * participationScore;
449
450                 case NodoState::SUSPICIOUS:
451                     return historyScore * participationScore * 0.7;  // Penalidade
                        moderada
452
453                 case NodoState::PENALIZED:
454                     return historyScore * participationScore * 0.3;  // Penalidade
                        severa
455
456                 case NodoState::PROBATION:
457                     return historyScore * participationScore * 0.5;  // Penalidade
                        media
458
459                 default:
460                     return 0.0;
461             }
462         }
463
464         void UpdateHistoryScore(uint32_t nodeId, const NodoMetrics& metrics) {
465             NS_LOG_FUNCTION(this << nodeId);
466
467             const double UPTIME_WEIGHT = 0.3;
468             const double SUCCESS_WEIGHT = 0.3;
469             const double RESPONSE_WEIGHT = 0.2;
470             const double RESOURCE_WEIGHT = 0.2;
471
472             // Calcula score de uptime
473             double uptimeScore = metrics.uptime.GetSeconds() /
474                                  Simulator::Now().GetSeconds();
475
476             // Calcula taxa de sucesso
477             double successRate = metrics.interactionCount > 0 ?
478                 static_cast<double>(metrics.successfulConsensus) /
                        metrics.interactionCount : 0;
479
480             // Calcula score de tempo de resposta
481             double responseScore = 1.0;
482             if (!metrics.responseTimes.empty()) {
483                 Time avgResponse;
484                 for (const auto& time : metrics.responseTimes) {
485                     avgResponse += time;
```

```
            }
            avgResponse /= metrics.responseTimes.size();

            // Normaliza score de resposta (menor e melhor)
            responseScore = std::max(0.0, 1.0 - avgResponse.GetSeconds());
        }

        // Calcula score final
        double newScore = (UPTIME_WEIGHT * uptimeScore) +
                          (SUCCESS_WEIGHT * successRate) +
                          (RESPONSE_WEIGHT * responseScore) +
                          (RESOURCE_WEIGHT * metrics.resourceUtilization);

        // Suaviza mudanca com score historico anterior
        const double ALPHA = 0.8;  // Fator de suavizacao
        m_historyScores[nodeId] = (ALPHA * m_historyScores[nodeId]) +
                                  ((1 - ALPHA) * newScore);
    }

    bool DetectAnomalous(const NodoMetrics& metrics) {
        NS_LOG_FUNCTION(this);

        // Verifica multiplos indicadores de comportamento anomalo

        // Taxa de sucesso muito baixa (< 70porcento)
        if (metrics.interactionCount > 0) {
            double successRate =
                static_cast<double>(metrics.successfulConsensus) /
                                metrics.interactionCount;
            if (successRate < 0.7) return true;
        }

        // Recursos muito baixos
        if (metrics.resourceUtilization < 0.3) return true;

        // Muitos heartbeats perdidos
        if (metrics.missedHeartbeats > 3) return true;

        // Tempos de resposta anomalos
        if (!metrics.responseTimes.empty()) {
            Time avgResponse;
            for (const auto& time : metrics.responseTimes) {
                avgResponse += time;
            }
            avgResponse /= metrics.responseTimes.size();

```

```
            if (avgResponse.GetSeconds() > 1.0) return true;
        }

        // Padrao de recursos instavel
        static std::vector<double> resourceHistory;
        resourceHistory.push_back(metrics.resourceUtilization);
        if (resourceHistory.size() > 10) {
            resourceHistory.erase(resourceHistory.begin());

            double variance = 0.0;
            double mean = std::accumulate(resourceHistory.begin(),
                                          resourceHistory.end(), 0.0) /
                                          resourceHistory.size();

            for (double value : resourceHistory) {
                variance += std::pow(value - mean, 2);
            }
            variance /= resourceHistory.size();

            if (variance > 0.1) return true;
        }

        return false;
    }

    bool BehaviorNormalized(const NodoMetrics& metrics) {
        NS_LOG_FUNCTION(this);

        // Define limiares para comportamento normal
        const double MIN_SUCCESS_RATE = 0.8;
        const double MIN_RESOURCE_UTIL = 0.5;
        const uint32_t MAX_MISSED_HEARTBEATS = 2;
        const double MAX_AVG_RESPONSE_TIME = 0.5;

        // Verifica taxa de sucesso
        if (metrics.interactionCount > 0) {
            double successRate =
                static_cast<double>(metrics.successfulConsensus) /
                                    metrics.interactionCount;
            if (successRate < MIN_SUCCESS_RATE) return false;
        }

        // Verifica utilizacao de recursos
        if (metrics.resourceUtilization < MIN_RESOURCE_UTIL) return false;

        // Verifica heartbeats perdidos
```

```
576             if (metrics.missedHeartbeats > MAX_MISSED_HEARTBEATS) return false;
577
578             // Verifica tempos de resposta
579             if (!metrics.responseTimes.empty()) {
580                 Time avgResponse;
581                 for (const auto& time : metrics.responseTimes) {
582                     avgResponse += time;
583                 }
584                 avgResponse /= metrics.responseTimes.size();
585
586                 if (avgResponse.GetSeconds() > MAX_AVG_RESPONSE_TIME) return false;
587             }
588
589             return true;
590         }
591
592         void TransitionTo(uint32_t nodeId, NodoState newState) {
593             NS_LOG_FUNCTION(this << nodeId << static_cast<int>(newState));
594
595             NodoState oldState = m_nodeStates[nodeId];
596             m_nodeStates[nodeId] = newState;
597
598             // Atualiza metricas para o novo estado
599             auto& metrics = m_nodeMetrics[nodeId];
600             metrics.state = newState;
601             metrics.stateTransitionTime = Simulator::Now();
602
603             // Ajusta parametros baseado na transicao
604             switch (newState) {
605                 case NodoState::OPERATIONAL:
606                     // Restaura participacao normal
607                     m_participationScores[nodeId] = 1.0;
608                     break;
609
610                 case NodoState::SUSPICIOUS:
611                     // Reduz participacao
612                     m_participationScores[nodeId] *= 0.7;
613                     metrics.monitoringStart = Simulator::Now();
614                     break;
615
616                 case NodoState::PENALIZED:
617                     // Penalidade severa
618                     m_participationScores[nodeId] *= 0.3;
619                     break;
620
621                 case NodoState::PROBATION:
```

```
                    // Penalidade moderada com chance de recuperacao
                    m_participationScores[nodeId] *= 0.5;
                    metrics.successfulConsensus = 0;
                    metrics.interactionCount = 0;
                    break;

                case NodoState::CALIBRATION:
                    // Reset completo
                    m_participationScores[nodeId] = 0.5;
                    m_historyScores[nodeId] = 0.5;
                    metrics = NodoMetrics();
                    break;
            }

            NS_LOG_INFO("Nodo " << nodeId << " transicionado de " <<
                        static_cast<int>(oldState) << " para " <<
                        static_cast<int>(newState));
        }

    private:
        std::map<uint32_t, NodoMetrics> m_nodeMetrics;
        std::map<uint32_t, NodoState> m_nodeStates;
        std::map<uint32_t, double> m_historyScores;
        std::map<uint32_t, double> m_participationScores;
    };

    class SecurityManager {
    public:
        SecurityManager()
            : m_maxDelay(MilliSeconds(100))
            , m_allowedMisses(3)
            , m_maxMessageAge(Seconds(60))
            , m_minMessageInterval(MilliSeconds(10))
            , m_monitoringInterval(MilliSeconds(100))
            , m_maxMessageInterval(Seconds(2))
            , m_maxMissedHeartbeats(3)
            , m_minConsensusRate(0.7)
            , m_currentView(0) {
            NS_LOG_FUNCTION(this);
        },
            InitializeCrypto();

        ~SecurityManager() {
            CleanupCrypto();
        }

```

```
668         EVP_PKEY* GetPrivateKey(uint32_t nodeId) {
669             if (m_nodeKeys.count(nodeId)) {
670                 return m_nodeKeys[nodeId].privateKey;
671             }
672             return nullptr;
673         }
674 
675         void Initialize(uint32_t totalNodos) {
676             m_totalNodos = totalNodos;
677             // Inicializa registros de seguranca para todos os nos
678             for (uint32_t i = 0; i < totalNodos; i++) {
679                 m_lastMessages[i] = Simulator::Now();
680                 m_messageLog[i] = std::vector<BftMessage>();
681                 m_missedHeartbeats[i] = 0;
682                 m_nodeStatistics[i] = NodoStatistics();
683             }
684         }
685 
686         bool VerifyMessage(const BftMessage& msg) {
687             NS_LOG_FUNCTION(this);
688 
689             // Verifica autenticidade basica
690             if (!ValidateSignature(msg)) {
691                 NS_LOG_WARN("Validacao de assinatura da mensagem falhou");
692                 return false;
693             }
694 
695             // Verifica timestamp
696             if (!ValidateTimestamp(msg)) {
697                 NS_LOG_WARN("Validacao de timestamp da mensagem falhou");
698                 return false;
699             }
700 
701             // Verifica duplicidade
702             if (IsDuplicate(msg)) {
703                 NS_LOG_WARN("Mensagem duplicada detectada");
704                 return false;
705             }
706 
707             // Verifica sequencia para mensagens de consenso
708             if (msg.type == BftMessage::CONFIG ||
709                 msg.type == BftMessage::EXECUTE) {
710                 if (!ValidateSequence(msg)) {
711                     NS_LOG_WARN("Validacao de sequencia da mensagem falhou");
712                     return false;
713                 }
```

```
714             }
715
716             if (!ValidateView(msg)) {
717                 NS_LOG_WARN("Validacao de view da mensagem falhou");
718                 return false;
719             }
720
721             UpdateMessageLog(msg);
722
723             return true;
724         }
725
726         void MonitorLeader() {
727             NS_LOG_FUNCTION(this);
728
729             auto currentTime = Simulator::Now();
730             uint32_t currentLeader = m_currentView % m_totalNodos;
731
732             // Obtem ultima mensagem do lider
733             auto lastMessage = m_lastMessages[currentLeader];
734             auto missedHeartbeats = m_missedHeartbeats[currentLeader];
735
736             // Valida comportamento do lider
737             if (ValidateLeaderBehavior(currentTime, lastMessage, missedHeartbeats)) {
738                 // Calcula atraso desde ultima mensagem
739                 Time delay = currentTime - lastMessage;
740
741                 if (delay <= m_maxDelay && missedHeartbeats <= m_allowedMisses) {
742                     // Lider esta operacional
743                     m_leaderStatus = LeaderStatus::OPERATIONAL;
744                 } else {
745                     // Inicia troca de lider
746                     InitiateLeaderChange();
747                 }
748             }
749
750             // Agenda proximo monitoramento
751             Simulator::Schedule(m_monitoringInterval,
752                                 &SecurityManager::MonitorLeader, this);
753         }
754
755         bool ValidateLeaderBehavior(Time currentTime, Time lastMessage,
756                                     uint32_t missedHeartbeats) {
757             NS_LOG_FUNCTION(this);
758
759             // Verifica intervalo entre mensagens
```

```
760             Time messageInterval = currentTime - lastMessage;
761             if (messageInterval > m_maxMessageInterval) {
762                 NS_LOG_WARN("Intervalo de mensagem do lider excedido");
763                 return false;
764             }
765 
766             // Verifica taxa de heartbeats perdidos
767             if (missedHeartbeats > m_maxMissedHeartbeats) {
768                 NS_LOG_WARN("Muitos heartbeats perdidos");
769                 return false;
770             }
771 
772             // Verifica padrao de operacoes
773             auto leaderStats = m_nodeStatistics[m_currentView % m_totalNodos];
774 
775             // Verifica taxa de consenso
776             if (leaderStats.messageCount > 0) {
777                 double consensusRate = static_cast<double>(leaderStats.configCount) /
778                                        leaderStats.messageCount;
779                 if (consensusRate < m_minConsensusRate) {
780                     NS_LOG_WARN("Taxa de consenso baixa");
781                     return false;
782                 }
783             }
784 
785             return true;
786         }
787 
788         bool ValidateSignature(const BftMessage& msg) {
789             if (!m_nodeKeys.count(msg.sender)) {
790                 NS_LOG_ERROR("Nenhuma chave encontrada para o no " << msg.sender);
791                 return false;
792             }
793 
794             // Prepara dados para verificacao
795             std::string messageData = SerializeMessageForSigning(msg);
796 
797             // Cria contexto de verificacao
798             EVP_MD_CTX* mdctx = EVP_MD_CTX_create();
799             if (!mdctx) {
800                 NS_LOG_ERROR("Falha ao criar contexto de verificacao");
801                 return false;
802             }
803 
804             bool result = false;
805 
```

```
806             do {
807                 // Inicializa verificacao
808                 if (EVP_DigestVerifyInit(mdctx, nullptr, EVP_sha256(), nullptr,
809                                          m_nodeKeys[msg.sender].publicKey) != 1) {
810                     NS_LOG_ERROR("Falha ao inicializar verificacao");
811                     break;
812                 }
813 
814                 // Atualiza com os dados
815                 if (EVP_DigestVerifyUpdate(mdctx, messageData.c_str(),
816                                            messageData.length()) != 1) {
817                     NS_LOG_ERROR("Falha ao atualizar verificacao");
818                     break;
819                 }
820 
821                 // Verifica assinatura
822                 if (EVP_DigestVerifyFinal(mdctx, msg.signature.data(),
823                                           msg.signature.size()) != 1) {
824                     NS_LOG_ERROR("Verificacao de assinatura falhou");
825                     break;
826                 }
827 
828                 result = true;
829             } while (0);
830 
831             EVP_MD_CTX_destroy(mdctx);
832             return result;
833         }
834 
835         bool ValidateTimestamp(const BftMessage& msg) {
836             NS_LOG_FUNCTION(this);
837 
838             Time currentTime = Simulator::Now();
839 
840             // Verifica se timestamp esta no futuro
841             if (msg.timestamp > currentTime) {
842                 NS_LOG_WARN("Mensagem do futuro rejeitada");
843                 return false;
844             }
845 
846             // Verifica se mensagem e muito antiga
847             Time age = currentTime - msg.timestamp;
848             if (age > m_maxMessageAge) {
849                 NS_LOG_WARN("Mensagem muito antiga");
850                 return false;
851             }
```

```
852
853             // Verifica ordem das mensagens do mesmo remetente
854             auto& senderLog = m_messageLog[msg.sender];
855             if (!senderLog.empty()) {
856                 auto lastMsg = senderLog.back();
857                 if (msg.timestamp < lastMsg.timestamp) {
858                     NS_LOG_WARN("Timestamp da mensagem violado");
859                     return false;
860                 }
861             }
862
863             // Verifica intervalo entre mensagens consecutivas
864             if (!senderLog.empty()) {
865                 auto lastMsg = senderLog.back();
866                 Time interval = msg.timestamp - lastMsg.timestamp;
867                 if (interval < m_minMessageInterval) {
868                     NS_LOG_WARN("Rate limit da mensagem excedido");
869                     return false;
870                 }
871             }
872
873             return true;
874         }
875
876         void UpdateMessageLog(const BftMessage& msg) {
877             NS_LOG_FUNCTION(this);
878
879             // Atualiza ultimo timestamp recebido do remetente
880             m_lastMessages[msg.sender] = msg.timestamp;
881
882             // Adiciona mensagem ao log do remetente
883             auto& senderLog = m_messageLog[msg.sender];
884             senderLog.push_back(msg);
885
886             // Limita tamanho do log
887             const size_t MAX_LOG_SIZE = 1000;
888             if (senderLog.size() > MAX_LOG_SIZE) {
889                 senderLog.erase(senderLog.begin());
890             }
891
892             // Atualiza estatisticas
893             UpdateMessageStatistics(msg);
894
895             // Atualiza contadores de heartbeat se aplicavel
896             if (msg.type == BftMessage::HEARTBEAT) {
897                 m_missedHeartbeats[msg.sender] = 0;
```

```
898             }
899
900             // Limpa mensagens antigas
901             CleanOldMessages(Simulator::Now() - m_maxMessageAge);
902         }
903
904         // Funcao auxiliar para serializar mensagem para assinatura
905         std::string SerializeMessageForSigning(const BftMessage& msg) {
906             std::stringstream ss;
907             ss << static_cast<int>(msg.type) << ":"
908                << msg.view << ":"
909                << msg.seq << ":"
910                << msg.content << ":"
911                << msg.digest << ":"
912                << msg.sender << ":"
913                << msg.timestamp.GetNanoSeconds() << ":"
914                << msg.sliceId;
915             return ss.str();
916         }
917
918         // Desserializacoes
919         BftMessage DeserializeMessage(uint8_t* buffer, uint32_t size) {
920             BftMessage msg;
921             uint32_t offset = 0;
922
923             // Desserializa tipo
924             uint32_t type;
925             memcpy(&type, buffer + offset, sizeof(type));
926             msg.type = static_cast<BftMessage::Type>(type);
927             offset += sizeof(type);
928
929             // Desserializa view
930             memcpy(&msg.view, buffer + offset, sizeof(msg.view));
931             offset += sizeof(msg.view);
932
933             // Desserializa sequence
934             memcpy(&msg.seq, buffer + offset, sizeof(msg.seq));
935             offset += sizeof(msg.seq);
936
937             // Desserializa content
938             uint32_t contentSize;
939             memcpy(&contentSize, buffer + offset, sizeof(contentSize));
940             offset += sizeof(contentSize);
941             msg.content = std::string(reinterpret_cast<char*>(buffer + offset),
                contentSize);
942             offset += contentSize;
```

```
943
944             // Desserializa digest
945             uint32_t digestSize;
946             memcpy(&digestSize, buffer + offset, sizeof(digestSize));
947             offset += sizeof(digestSize);
948             msg.digest = std::string(reinterpret_cast<char*>(buffer + offset),
                    digestSize);
949             offset += digestSize;
950
951             // Desserializa sender
952             memcpy(&msg.sender, buffer + offset, sizeof(msg.sender));
953             offset += sizeof(msg.sender);
954
955             // Desserializa timestamp
956             int64_t timestamp;
957             memcpy(&timestamp, buffer + offset, sizeof(timestamp));
958             msg.timestamp = Time::FromNanoSeconds(timestamp);
959             offset += sizeof(timestamp);
960
961             // Desserializa sliceId
962             uint32_t sliceIdSize;
963             memcpy(&sliceIdSize, buffer + offset, sizeof(sliceIdSize));
964             offset += sizeof(sliceIdSize);
965             msg.sliceId = std::string(reinterpret_cast<char*>(buffer + offset),
                    sliceIdSize);
966             offset += sliceIdSize;
967
968             // Desserializa assinatura
969             uint32_t signatureSize;
970             memcpy(&signatureSize, buffer + offset, sizeof(signatureSize));
971             offset += sizeof(signatureSize);
972             msg.signature = std::vector<uint8_t>(buffer + offset, buffer + offset +
                    signatureSize);
973
974             return msg;
975         }
976
977         uint32_t GetAllowedMisses() const { return m_allowedMisses; }
978         void SetCurrentView(uint32_t view) { m_currentView = view; }
979
980     private:
981         static const size_t SIGNATURE_SIZE = 64;
982
983         struct NodoStatistics {
984             uint32_t messageCount{0};
985             uint32_t configCount{0};
```

```
            uint32_t executeCount{0};
            uint32_t heartbeatCount{0};
            Time lastMessageTime{Seconds(0)};
        };

        enum class LeaderStatus {
            OPERATIONAL,
            SUSPICIOUS,
            FAILED
        };

        bool IsDuplicate(const BftMessage& msg) {
            auto& senderLog = m_messageLog[msg.sender];
            for (const auto& loggedMsg : senderLog) {
                if (loggedMsg.seq == msg.seq &&
                    loggedMsg.view == msg.view &&
                    loggedMsg.type == msg.type) {
                    return true;
                }
            }
            return false;
        }

        bool ValidateSequence(const BftMessage& msg) {
            auto& senderLog = m_messageLog[msg.sender];
            if (!senderLog.empty()) {
                auto lastMsg = senderLog.back();
                if (msg.seq <= lastMsg.seq) {
                    return false;
                }
            }
            return true;
        }

        bool ValidateView(const BftMessage& msg) {
            return msg.view >= m_currentView;
        }

        void UpdateMessageStatistics(const BftMessage& msg) {
            auto& stats = m_nodeStatistics[msg.sender];
            stats.messageCount++;
            stats.lastMessageTime = msg.timestamp;

            switch (msg.type) {
                case BftMessage::CONFIG:
                    stats.configCount++;
```

```
                break;
            case BftMessage::EXECUTE:
                stats.executeCount++;
                break;
            case BftMessage::HEARTBEAT:
                stats.heartbeatCount++;
                break;
        }
    }

    void CleanOldMessages(Time threshold) {
        for (auto& [nodeId, log] : m_messageLog) {
            auto it = std::remove_if(log.begin(), log.end(),
                [threshold](const BftMessage& msg) {
                    return msg.timestamp < threshold;
                });
            log.erase(it, log.end());
        }
    }

    void InitiateLeaderChange() {
        m_currentView++;
        m_leaderStatus = LeaderStatus::FAILED;

    }

    Time m_maxDelay;
    uint32_t m_allowedMisses;
    Time m_maxMessageAge;
    Time m_minMessageInterval;
    Time m_monitoringInterval;
    Time m_maxMessageInterval;
    uint32_t m_maxMissedHeartbeats;
    double m_minConsensusRate;
    uint32_t m_currentView;
    uint32_t m_totalNodos;
    LeaderStatus m_leaderStatus{LeaderStatus::OPERATIONAL};

    std::map<uint32_t, Time> m_lastMessages;
    std::map<uint32_t, std::vector<BftMessage>> m_messageLog;
    std::map<uint32_t, uint32_t> m_missedHeartbeats;
    std::map<uint32_t, NodoStatistics> m_nodeStatistics;

    // Estrutura para armazenar par de chaves
    struct KeyPair {
        EVP_PKEY* privateKey;
```

```
1078            EVP_PKEY* publicKey;
1079        };
1080
1081        std::map<uint32_t, KeyPair> m_nodeKeys;
1082
1083        static const int RSA_KEY_SIZE = 2048;
1084
1085        static const size_t MAX_BUFFER_SIZE = 4096;
1086
1087        void InitializeCrypto() {
1088            OpenSSL_add_all_algorithms();
1089            ERR_load_crypto_strings();
1090
1091            // Gera par de chaves para cada no
1092            for (uint32_t i = 0; i < m_totalNodos; i++) {
1093                EVP_PKEY_CTX* ctx = EVP_PKEY_CTX_new_id(EVP_PKEY_RSA, nullptr);
1094                if (!ctx) {
1095                    NS_LOG_ERROR("Falha ao criar contexto RSA");
1096                    continue;
1097                }
1098
1099                if (EVP_PKEY_keygen_init(ctx) <= 0) {
1100                    NS_LOG_ERROR("Falha ao inicializar geracao de chave");
1101                    EVP_PKEY_CTX_free(ctx);
1102                    continue;
1103                }
1104
1105                if (EVP_PKEY_CTX_set_rsa_keygen_bits(ctx, RSA_KEY_SIZE) <= 0) {
1106                    NS_LOG_ERROR("Falha ao definir tamanho da chave");
1107                    EVP_PKEY_CTX_free(ctx);
1108                    continue;
1109                }
1110
1111                EVP_PKEY* key = nullptr;
1112                if (EVP_PKEY_keygen(ctx, &key) <= 0) {
1113                    NS_LOG_ERROR("Falha ao gerar chave");
1114                    EVP_PKEY_CTX_free(ctx);
1115                    continue;
1116                }
1117
1118                m_nodeKeys[i] = {key, key}; //Ambiente de testes usando o mesmo par
                        de chaves - Em producao deve ser diferente
1119                EVP_PKEY_CTX_free(ctx);
1120            }
1121        }
1122
```

```
1123        void CleanupCrypto() {
1124            for (auto& [_, keyPair] : m_nodeKeys) {
1125                EVP_PKEY_free(keyPair.privateKey);
1126                EVP_PKEY_free(keyPair.publicKey);
1127            }
1128            EVP_cleanup();
1129            ERR_free_strings();
1130        }
1131    };
1132
1133
1134    class MetricsCollector {
1135    public:
1136        MetricsCollector()
1137            : m_networkDelay(MilliSeconds(0)) {
1138            NS_LOG_FUNCTION(this);
1139        }
1140
1141        Initialize(uint32_t totalNodos, Time networkDelay) {
1142            m_networkDelay = networkDelay;
1143
1144            for (uint32_t nos : {15, 30, 60, 1000, 2000, 5000}) {
1145                for (uint32_t delay : {1, 10, 100}) {
1146                    for (uint32_t msgSize : {200, 2000, 10000}) {
1147                        Metrics metrics;
1148                        metrics.consensusTime = 0.0;
1149                        metrics.latency = 0.0;
1150                        metrics.throughput = 0.0;
1151                        m_results[nos][delay][msgSize] = metrics;
1152                    }
1153                }
1154            }
1155        }
1156
1157        RecordMetrics(uint32_t nos, uint32_t delay, uint32_t msgSize) {
1158            NS_LOG_FUNCTION(this << nos << delay << msgSize);
1159
1160            Time consensusTime = CalculateAverageConsensusTime();
1161            Time latency = CalculateLatencia();
1162            double throughput = CalculateThroughput(msgSize);
1163
1164            Metrics& metrics = m_results[nos][delay][msgSize];
1165
1166            const double ALPHA = 0.2;
1167            metrics.consensusTime = (ALPHA * consensusTime.GetSeconds()) +
1168                                    ((1 - ALPHA) * metrics.consensusTime);
```

```
1169            metrics.latency = (ALPHA * latency.GetSeconds()) +
1170                              ((1 - ALPHA) * metrics.latency);
1171            metrics.throughput = (ALPHA * throughput) +
1172                                 ((1 - ALPHA) * metrics.throughput);
1173
1174            m_detailedMetrics[nos][delay][msgSize].push_back({
1175                consensusTime.GetSeconds(),
1176                latency.GetSeconds(),
1177                throughput,
1178                Simulator::Now().GetSeconds()
1179            });
1180
1181            UpdateStatistics();
1182
1183            NS_LOG_INFO("Metricas registradas para " << nos << " nos, "
1184                        << delay << "ms delay, " << msgSize << " bytes:"
1185                        << "\n\tTempo de Consenso: " << consensusTime.GetSeconds()
                               << "s"
1186                        << "\n\tLatencia: " << latency.GetSeconds() << "s"
1187                        << "\n\tTaxa de transferencia: " << throughput << " ops/s");
1188        }
1189
1190
1191
1192        void CalculateLatencia() {
1193            NS_LOG_FUNCTION(this);
1194
1195            // Latencia Total = Tempo de Consenso + (Numero de rodadas x RTT)
1196            Time consensusTime = CalculateAverageConsensusTime();
1197            Time rtt = 2 * m_networkDelay;  // RTT = 2 * delay - medido em segundos
1198            uint32_t rounds = 2;  // CONFIG e EXECUTE
1199
1200            Time totalLatencia = consensusTime + (rounds * rtt);
1201
1202            // Registra para analise de distribuicao
1203            m_latencyDistribution.push_back(totalLatencia);
1204            if (m_latencyDistribution.size() > MAX_DISTRIBUTION_SAMPLES) {
1205                m_latencyDistribution.pop_front();
1206            }
1207
1208            return totalLatencia;
1209        }
1210
1211
1212
1213        double CalculateThroughput(uint32_t msgSize) {
```

```
1214            NS_LOG_FUNCTION(this << msgSize);
1215
1216            // Throughput = Tamanho da mensagem / Latencia total
1217            Time latency = CalculateLatencia();
1218            double throughput = static_cast<double>(msgSize) / latency.GetSeconds();
1219
1220            // Registra para analise de distribuicao
1221            m_throughputDistribution.push_back(throughput);
1222            if (m_throughputDistribution.size() > MAX_DISTRIBUTION_SAMPLES) {
1223                m_throughputDistribution.pop_front();
1224            }
1225
1226            // Atualiza metricas de pico
1227            m_peakThroughput = std::max(m_peakThroughput, throughput);
1228
1229            return throughput;
1230        }
1231
1232        void RecordConsensusTime(Time duration) {
1233            NS_LOG_FUNCTION(this << duration);
1234
1235            // Adiciona nova medicao
1236            m_consensusTimes.push_back(duration);
1237
1238            // Mantem tamanho maximo do historico
1239            const size_t MAX_HISTORY = 1000;
1240            if (m_consensusTimes.size() > MAX_HISTORY) {
1241                m_consensusTimes.erase(m_consensusTimes.begin());
1242            }
1243
1244            // Atualiza estatisticas de consenso
1245            m_lastConsensusTime = duration;
1246            m_totalConsensusMeasurements++;
1247
1248            // Atualiza minimo e maximo
1249            m_minConsensusTime = std::min(m_minConsensusTime, duration);
1250            m_maxConsensusTime = std::max(m_maxConsensusTime, duration);
1251
1252            // Calcula media movel exponencial
1253            if (m_avgConsensusTime == Time::Max()) {
1254                m_avgConsensusTime = duration;
1255            } else {
1256                const double ALPHA = 0.1;  // Fator de suavizacao
1257                m_avgConsensusTime = Time(m_avgConsensusTime.GetNanoSeconds() * (1 -
                        ALPHA) +
1258                                          duration.GetNanoSeconds() * ALPHA);
```

```
1259            }
1260        }
1261
1262
1263
1264        void UpdateStatistics() {
1265            NS_LOG_FUNCTION(this);
1266
1267            // Atualiza estatisticas globais
1268            for (const auto& [nos, delayMap] : m_results) {
1269                for (const auto& [delay, sizeMap] : delayMap) {
1270                    for (const auto& [msgSize, metrics] : sizeMap) {
1271                        // Calcula medias para cada configuracao
1272                        auto& detailedMetrics =
                               m_detailedMetrics[nos][delay][msgSize];
1273                        if (detailedMetrics.empty()) continue;
1274
1275                        double avgConsensus = 0.0;
1276                        double avgLatencia = 0.0;
1277                        double avgThroughput = 0.0;
1278
1279                        for (const auto& metric : detailedMetrics) {
1280                            avgConsensus += metric.consensusTime;
1281                            avgLatencia += metric.latency;
1282                            avgThroughput += metric.throughput;
1283                        }
1284
1285                        size_t count = detailedMetrics.size();
1286                        avgConsensus /= count;
1287                        avgLatencia /= count;
1288                        avgThroughput /= count;
1289
1290                        // Calcula desvio padrao
1291                        double stdDevConsensus = 0.0;
1292                        double stdDevLatencia = 0.0;
1293                        double stdDevThroughput = 0.0;
1294
1295                        for (const auto& metric : detailedMetrics) {
1296                            stdDevConsensus += std::pow(metric.consensusTime -
                                   avgConsensus, 2);
1297                            stdDevLatencia += std::pow(metric.latency - avgLatencia,
                                   2);
1298                            stdDevThroughput += std::pow(metric.throughput -
                                   avgThroughput, 2);
1299                        }
1300
```

```
1301                        stdDevConsensus = std::sqrt(stdDevConsensus / count);
1302                        stdDevLatencia = std::sqrt(stdDevLatencia / count);
1303                        stdDevThroughput = std::sqrt(stdDevThroughput / count);
1304
1305                        // Atualiza estatisticas
1306                        StatisticalMetrics& stats =
                               m_statistics[nos][delay][msgSize];
1307                        stats.avgConsensusTime = avgConsensus;
1308                        stats.avgLatencia = avgLatencia;
1309                        stats.avgThroughput = avgThroughput;
1310                        stats.stdDevConsensus = stdDevConsensus;
1311                        stats.stdDevLatencia = stdDevLatencia;
1312                        stats.stdDevThroughput = stdDevThroughput;
1313                        stats.sampleCount = count;
1314
1315                        NS_LOG_INFO("Estatisticas atualizadas para " << nos << "
                               nos, "
1316                                << delay << "ms delay, " << msgSize << " bytes:"
1317                                << "\n\tMedia de Consenso: " << avgConsensus <<
                                   "s ( aproximadamente " << stdDevConsensus <<
                                   ")"
1318                                << "\n\tAvg Latencia: " << avgLatencia << "s (
                                   aproximadamente " << stdDevLatencia << ")"
1319                                << "\n\tMedia de Taxa de transferencia: " <<
                                   avgThroughput << " ops/s ( aproximadamente "
                                   << stdDevThroughput << ")"
1320                                << "\n\tContagem de Amostras: " << count);
1321                    }
1322                }
1323            }
1324
1325            // Atualiza intervalo de confianca (95porcento)
1326            CalculateConfidenceIntervals();
1327        }
1328
1329    private:
1330        Time CalculateAverageConsensusTime() {
1331            if (m_consensusTimes.empty()) {
1332                return Time(0);
1333            }
1334
1335            Time total;
1336            for (const auto& time : m_consensusTimes) {
1337                total += time;
1338            }
1339            return total / m_consensusTimes.size();
```

```
1340        }
1341
1342        void CalculateConfidenceIntervals() {
1343            // Para cada configuracao
1344            for (auto& [nos, delayMap] : m_statistics) {
1345                for (auto& [delay, sizeMap] : delayMap) {
1346                    for (auto& [msgSize, stats] : sizeMap) {
1347                        // Calcula intervalo de confianca (95porcento)
1348                        double z = 1.96;  // Z-score para 95porcento de confianca
1349                        double n = std::sqrt(stats.sampleCount);
1350
1351                        stats.ciConsensus = z * (stats.stdDevConsensus / n);
1352                        stats.ciLatencia = z * (stats.stdDevLatencia / n);
1353                        stats.ciThroughput = z * (stats.stdDevThroughput / n);
1354                    }
1355                }
1356            }
1357        }
1358
1359        struct DetailedMetric {
1360            double consensusTime;
1361            double latency;
1362            double throughput;
1363            double timestamp;
1364        };
1365
1366        struct StatisticalMetrics {
1367            double avgConsensusTime;
1368            double avgLatencia;
1369            double avgThroughput;
1370            double stdDevConsensus;
1371            double stdDevLatencia;
1372            double stdDevThroughput;
1373            double ciConsensus;
1374            double ciLatencia;
1375            double ciThroughput;
1376            size_t sampleCount;
1377        };
1378
1379        struct Metrics {
1380            double consensusTime;
1381            double latency;
1382            double throughput;
1383        };
1384
1385        static const size_t MAX_DISTRIBUTION_SAMPLES = 1000;
```

```
1386
1387        std::map<uint32_t, std::map<uint32_t, std::map<uint32_t,
                std::vector<DetailedMetric>>>> m_detailedMetrics;
1388        std::map<uint32_t, std::map<uint32_t, std::map<uint32_t,
                StatisticalMetrics>>> m_statistics;
1389        std::deque<Time> m_latencyDistribution;
1390        std::deque<double> m_throughputDistribution;
1391        double m_peakThroughput;
1392        Time m_minConsensusTime;
1393        Time m_maxConsensusTime;
1394        Time m_avgConsensusTime;
1395        Time m_lastConsensusTime;
1396        uint64_t m_totalConsensusMeasurements;
1397        std::map<uint32_t, std::map<uint32_t, std::map<uint32_t, Metrics>>>
                m_results;
1398        Time m_networkDelay;
1399        std::vector<Time> m_consensusTimes;
1400
1401    };
1402
1403    virtual void StartApplication(void);
1404    virtual void StopApplication(void);
1405
1406    void ProcessMessage(BftMessage msg);
1407    void StartConsensus(void);
1408    void HandleConfig(BftMessage msg);
1409
1410    void HandleConfigAck(BftMessage msg) {
1411        NS_LOG_FUNCTION(this);
1412
1413        if (!m_securityManager.VerifyMessage(msg)) return;
1414
1415        // Verifica se e lider e se esta na view correta
1416        if (!IsLeader() || msg.view != m_view) return;
1417
1418        // Adiciona voto CONFIG
1419        m_configVotes[msg.seq].insert(msg.sender);
1420
1421        // Verifica se atingiu quorum (2f + 1)
1422        uint32_t quorum = 2 * ((m_totalNodos - 1) / 3) + 1;
1423        if (m_configVotes[msg.seq].size() >= quorum) {
1424            // Inicia fase EXECUTE
1425            BftMessage executeMsg;
1426            executeMsg.type = BftMessage::EXECUTE;
1427            executeMsg.view = m_view;
1428            executeMsg.seq = msg.seq;
```

```
            executeMsg.sender = m_nodeId;
            executeMsg.timestamp = Simulator::Now();
            executeMsg.sliceId = msg.sliceId;

            // Envia EXECUTE para a fatia ativa
            const auto& activeSlice = m_adan.GetSlices()[0];
            for (uint32_t nodeId : activeSlice) {
                if (nodeId != m_nodeId) {
                    Ipv4Address dest("10.0.0." + std::to_string(nodeId + 1));
                    SendMessage(executeMsg, dest);
                }
            }
        }
    }

    // Manipulador de NEW_VIEW
    void HandleNewView(BftMessage msg) {
        NS_LOG_FUNCTION(this);

        if (!m_securityManager.VerifyMessage(msg)) return;

        // Verifica se a view e valida
        if (msg.view <= m_view) {
            NS_LOG_WARN("View antiga recebida em NEW_VIEW");
            return;
        }

        // Atualiza para a nova view
        m_view = msg.view;
        m_isLeader = (m_nodeId == (m_view % m_totalNodos));

        // Atualiza estado do sistema
        m_securityManager.SetCurrentView(m_view);

        // Processa operacoes pendentes incluidas na mensagem NEW_VIEW
        std::istringstream pendingOps(msg.content);
        std::string op;
        while (std::getline(pendingOps, op, ';')) {
            uint32_t seq = std::stoul(op);
            if (m_pendingMessages.count(seq)) {
                const auto& pendingMsg = m_pendingMessages[seq];
                ProcessMessage(pendingMsg);
            }
        }

        m_configVotes.clear();
```

```
1475            m_executeVotes.clear();
1476
1477            // Se e o novo lider, inicia consenso
1478            if (m_isLeader) {
1479                StartConsensus();
1480            }
1481        }
1482
1483        // Envio de NEW_VIEW
1484        void SendNewView(void) {
1485            NS_LOG_FUNCTION(this);
1486
1487            if (!m_isLeader) return;
1488
1489            // Cria mensagem NEW_VIEW
1490            BftMessage newView;
1491            newView.type = BftMessage::NEW_VIEW;
1492            newView.view = m_view;
1493            newView.sender = m_nodeId;
1494            newView.timestamp = Simulator::Now();
1495
1496            // Coleta operacoes pendentes
1497            std::string pendingOps;
1498            for (const auto& [seq, msg] : m_pendingMessages) {
1499                pendingOps += std::to_string(seq) + ";";
1500            }
1501            newView.content = pendingOps;
1502
1503            // Envia para todos os nos da fatia ativa
1504            const auto& activeSlice = m_adan.GetSlices()[0];
1505            for (uint32_t nodeId : activeSlice) {
1506                if (nodeId != m_nodeId) {
1507                    Ipv4Address dest("10.0.0." + std::to_string(nodeId + 1));
1508                    SendMessage(newView, dest);
1509                }
1510            }
1511
1512            StartConsensus();
1513        }
1514
1515        //Registro de troca de view
1516        void RecordViewChange(uint32_t view) {
1517            NS_LOG_FUNCTION(this << view);
1518
1519            struct ViewChangeMetric {
1520                Time timestamp;
```

```
1521            uint32_t oldView;
1522            uint32_t newView;
1523            double consensusRate;
1524            double latencyImpact;
1525        };
1526
1527        static std::vector<ViewChangeMetric> viewChanges;
1528        static uint32_t lastView = 0;
1529
1530        // Registra metricas da troca de view
1531        ViewChangeMetric metric;
1532        metric.timestamp = Simulator::Now();
1533        metric.oldView = lastView;
1534        metric.newView = view;
1535
1536        // Calcula taxa de consenso antes da troca
1537        if (!m_consensusTimes.empty()) {
1538            uint32_t recentConsensus = 0;
1539            Time window = Seconds(10);
1540            Time threshold = Simulator::Now() - window;
1541
1542            for (const auto& time : m_consensusTimes) {
1543                if (time > threshold) {
1544                    recentConsensus++;
1545                }
1546            }
1547
1548            metric.consensusRate = static_cast<double>(recentConsensus) /
                    window.GetSeconds();
1549        } else {
1550            metric.consensusRate = 0.0;
1551        }
1552
1553        // Calcula impacto na latencia
1554        if (!m_latencyDistribution.empty()) {
1555            Time avgLatenciaBefore;
1556            for (const auto& lat : m_latencyDistribution) {
1557                avgLatenciaBefore += lat;
1558            }
1559            avgLatenciaBefore /= m_latencyDistribution.size();
1560            metric.latencyImpact = avgLatenciaBefore.GetSeconds();
1561        } else {
1562            metric.latencyImpact = 0.0;
1563        }
1564
1565        viewChanges.push_back(metric);
```

```
1566            lastView = view;
1567
1568            if (viewChanges.size() > 1000) {
1569                viewChanges.erase(viewChanges.begin());
1570            }
1571
1572            NS_LOG_INFO("Troca de view registrada:"
1573                        << "\n\tView antiga: " << metric.oldView
1574                        << "\n\tNova view: " << metric.newView
1575                        << "\n\tTaxa de consenso: " << metric.consensusRate << " ops/s"
1576                        << "\n\tImpacto na latencia: " << metric.latencyImpact << "s");
1577        }
1578
1579        //Callback para recebimento de mensagens
1580        void HandleRead(Ptr<Socket> socket) {
1581            NS_LOG_FUNCTION(this << socket);
1582
1583            Ptr<Packet> packet;
1584            Address from;
1585
1586            // Recebe todos os pacotes pendentes
1587            while ((packet = socket->RecvFrom(from))) {
1588                uint32_t size = packet->GetSize();
1589                uint8_t buffer[SecurityManager::MAX_BUFFER_SIZE];
1590
1591                // Extrai os dados do pacote
1592                packet->CopyData(buffer, size);
1593
1594                // Desserializa a mensagem
1595                BftMessage msg = DeserializeMessage(buffer, size);
1596
1597                // Processa a mensagem
1598                ProcessMessage(msg);
1599            }
1600        }
1601
1602        void HandleExecute(BftMessage msg) {
1603            NS_LOG_FUNCTION(this);
1604
1605            if (!m_securityManager.VerifyMessage(msg)) return;
1606
1607            // Verifica se esta na fatia ativa
1608            const auto& activeSlice = m_adan.GetSlices()[0];
1609            if (activeSlice.find(m_nodeId) == activeSlice.end()) return;
1610
1611            // Verifica view e sequencia
```

```
1612         if (msg.view != m_view) return;
1613
1614         // Executa operacao e envia ACK
1615         BftMessage ack;
1616         ack.type = BftMessage::EXECUTE_ACK;
1617         ack.view = msg.view;
1618         ack.seq = msg.seq;
1619         ack.sender = m_nodeId;
1620         ack.timestamp = Simulator::Now();
1621         ack.sliceId = msg.sliceId;
1622
1623         Ipv4Address dest("10.0.0." + std::to_string(msg.sender + 1));
1624         SendMessage(ack, dest);
1625     }
1626
1627     void HandleExecuteAck(BftMessage msg) {
1628         NS_LOG_FUNCTION(this);
1629
1630         if (!m_securityManager.VerifyMessage(msg)) return;
1631
1632         // Verifica se e lider e view correta
1633         if (!IsLeader() || msg.view != m_view) return;
1634
1635         // Adiciona voto EXECUTE
1636         m_executeVotes[msg.seq].insert(msg.sender);
1637
1638         // Verifica quorum (2f + 1)
1639         uint32_t quorum = 2 * ((m_totalNodos - 1) / 3) + 1;
1640         if (m_executeVotes[msg.seq].size() >= quorum) {
1641             // Consenso atingido
1642             m_sequence++;
1643
1644             // Atualiza metricas
1645             Time consensusTime = Simulator::Now() - msg.timestamp;
1646             m_metricsCollector.RecordConsensusTime(consensusTime);
1647             m_consensusTimeTrace(consensusTime);
1648
1649             // Limpa votos
1650             m_configVotes.erase(msg.seq);
1651             m_executeVotes.erase(msg.seq);
1652         }
1653     }
1654
1655     void HandleViewChange(BftMessage msg) {
1656         NS_LOG_FUNCTION(this);
1657
```

```
1658         if (!m_securityManager.VerifyMessage(msg)) return;
1659 
1660         if (msg.view <= m_view) return;
1661 
1662         // Atualiza view
1663         m_view = msg.view;
1664         m_isLeader = (m_nodeId == (m_view % m_totalNodos));
1665 
1666         if (m_isLeader) {
1667             // Novo lider inicia consenso
1668             StartConsensus();
1669         }
1670     }
1671 
1672     void HandleHeartbeat(BftMessage msg) {
1673         NS_LOG_FUNCTION(this);
1674 
1675         if (!m_securityManager.VerifyMessage(msg)) return;
1676 
1677         // Atualiza ultimo heartbeat recebido do lider
1678         if (msg.sender == (m_view % m_totalNodos)) {
1679             NodoMetrics& metrics = m_reputationSystem.GetNodoMetrics(msg.sender);
1680             metrics.missedHeartbeats = 0;
1681         }
1682     }
1683 
1684 
1685     void SendMessage(BftMessage msg, Ipv4Address dest) {
1686         NS_LOG_FUNCTION(this << dest);
1687 
1688         // Serializa a mensagem em um buffer
1689         uint8_t buffer[SecurityManager::MAX_BUFFER_SIZE];
1690         uint32_t offset = 0;
1691 
1692         // Serializa tipo (4 bytes)
1693         uint32_t type = static_cast<uint32_t>(msg.type);
1694         memcpy(buffer + offset, &type, sizeof(type));
1695         offset += sizeof(type);
1696 
1697         // Serializa view (4 bytes)
1698         memcpy(buffer + offset, &msg.view, sizeof(msg.view));
1699         offset += sizeof(msg.view);
1700 
1701         // Serializa sequence (4 bytes)
1702         memcpy(buffer + offset, &msg.seq, sizeof(msg.seq));
1703         offset += sizeof(msg.seq);
```

```
1704
1705        // Serializa content (tamanho variavel)
1706        uint32_t contentSize = msg.content.size();
1707        memcpy(buffer + offset, &contentSize, sizeof(contentSize));
1708        offset += sizeof(contentSize);
1709        memcpy(buffer + offset, msg.content.c_str(), contentSize);
1710        offset += contentSize;
1711
1712        // Serializa digest (tamanho variavel)
1713        uint32_t digestSize = msg.digest.size();
1714        memcpy(buffer + offset, &digestSize, sizeof(digestSize));
1715        offset += sizeof(digestSize);
1716        memcpy(buffer + offset, msg.digest.c_str(), digestSize);
1717        offset += digestSize;
1718
1719        // Serializa sender (4 bytes)
1720        memcpy(buffer + offset, &msg.sender, sizeof(msg.sender));
1721        offset += sizeof(msg.sender);
1722
1723        // Serializa timestamp (8 bytes)
1724        int64_t timestamp = msg.timestamp.GetNanoSeconds();
1725        memcpy(buffer + offset, &timestamp, sizeof(timestamp));
1726        offset += sizeof(timestamp);
1727
1728        // Serializa sliceId (tamanho variavel)
1729        uint32_t sliceIdSize = msg.sliceId.size();
1730        memcpy(buffer + offset, &sliceIdSize, sizeof(sliceIdSize));
1731        offset += sizeof(sliceIdSize);
1732        memcpy(buffer + offset, msg.sliceId.c_str(), sliceIdSize);
1733        offset += sliceIdSize;
1734
1735        // Serializa assinatura (tamanho variavel)
1736        uint32_t signatureSize = msg.signature.size();
1737        memcpy(buffer + offset, &signatureSize, sizeof(signatureSize));
1738        offset += sizeof(signatureSize);
1739        memcpy(buffer + offset, msg.signature.data(), signatureSize);
1740        offset += signatureSize;
1741
1742        // Cria e envia o pacote
1743        Ptr<Packet> packet = Create<Packet>(buffer, offset);
1744        m_socket->SendTo(packet, 0, InetSocketAddress(dest, m_port));
1745    }
1746
1747    bool VerifyMessage(const BftMessage& msg) {
1748        return m_securityManager.VerifyMessage(msg);
1749    }
```

```
1750
1751    void UpdateReputations();
1752    void ReorganizeSlices();
1753
1754    bool IsLeader() const {
1755        return m_isLeader;
1756    }
1757
1758    void SendHeartbeat() {
1759        NS_LOG_FUNCTION(this);
1760
1761        if (!IsLeader()) return;
1762
1763        // Cria mensagem heartbeat
1764        BftMessage heartbeat;
1765        heartbeat.type = BftMessage::HEARTBEAT;
1766        heartbeat.view = m_view;
1767        heartbeat.sender = m_nodeId;
1768        heartbeat.timestamp = Simulator::Now();
1769
1770        // Envia para todos os nos da fatia ativa
1771        const auto& activeSlice = m_adan.GetSlices()[0];
1772        for (uint32_t nodeId : activeSlice) {
1773            if (nodeId != m_nodeId) {
1774                Ipv4Address dest("10.0.0." + std::to_string(nodeId + 1));
1775                SendMessage(heartbeat, dest);
1776            }
1777        }
1778
1779        // Agenda proximo heartbeat
1780        m_heartbeatEvent = Simulator::Schedule(m_heartbeatInterval,
1781                                               &BFTH::SendHeartbeat, this);
1782    }
1783
1784    void CheckLeaderLiveness() {
1785        NS_LOG_FUNCTION(this);
1786
1787        uint32_t currentLeader = m_view % m_totalNodos;
1788        NodoMetrics& leaderMetrics =
            m_reputationSystem.GetNodoMetrics(currentLeader);
1789
1790        // Incrementa contagem de heartbeats perdidos
1791        leaderMetrics.missedHeartbeats++;
1792
1793        // Verifica se precisa iniciar troca de view
1794        if (leaderMetrics.missedHeartbeats >= m_securityManager.GetAllowedMisses()) {
```

```
1795            // Inicia view change
1796            m_view++;
1797
1798            BftMessage viewChange;
1799            viewChange.type = BftMessage::VIEW_CHANGE;
1800            viewChange.view = m_view;
1801            viewChange.sender = m_nodeId;
1802            viewChange.timestamp = Simulator::Now();
1803
1804            // Broadcast da mensagem de view change
1805            const auto& activeSlice = m_adan.GetSlices()[0];
1806            for (uint32_t nodeId : activeSlice) {
1807                if (nodeId != m_nodeId) {
1808                    Ipv4Address dest("10.0.0." + std::to_string(nodeId + 1));
1809                    SendMessage(viewChange, dest);
1810                }
1811            }
1812        }
1813
1814        // Agenda proxima verificacao
1815        m_leaderCheckEvent = Simulator::Schedule(m_leaderTimeout,
1816                                                 &BFTH::CheckLeaderLiveness, this);
1817    }
1818
1819    // Iniciacao de troca de view
1820    void InitiateViewChange() {
1821        NS_LOG_FUNCTION(this);
1822
1823        m_view++;
1824
1825        BftMessage viewChangeMsg;
1826        viewChangeMsg.type = BftMessage::VIEW_CHANGE;
1827        viewChangeMsg.view = m_view;
1828        viewChangeMsg.sender = m_nodeId;
1829        viewChangeMsg.timestamp = Simulator::Now();
1830        viewChangeMsg.seq = m_sequence;
1831
1832        // Adiciona informacoes sobre operacoes pendentes
1833        std::string pendingOps;
1834        for (const auto& [seq, msgs] : m_pendingMessages) {
1835            if (!msgs.empty()) {
1836                pendingOps += std::to_string(seq) + ";";
1837            }
1838        }
1839        viewChangeMsg.content = pendingOps;
1840
```

```
1841        EVP_MD_CTX* mdctx = EVP_MD_CTX_create();
1842        std::vector<uint8_t> signature;
1843
1844        if (mdctx != nullptr) {
1845            // Prepara dados para assinatura
1846            std::string messageData =
                    m_securityManager.SerializeMessageForSigning(viewChangeMsg);
1847
1848            // Inicializa assinatura
1849            if (EVP_DigestSignInit(mdctx, nullptr, EVP_sha256(), nullptr,
1850                                   m_securityManager.GetPrivateKey(m_nodeId)) == 1) {
1851
1852                // Atualiza com os dados
1853                if (EVP_DigestSignUpdate(mdctx, messageData.c_str(),
1854                                         messageData.length()) == 1) {
1855
1856                    // Determina tamanho da assinatura
1857                    size_t sigLen;
1858                    if (EVP_DigestSignFinal(mdctx, nullptr, &sigLen) == 1) {
1859                        // Aloca espaco para assinatura
1860                        signature.resize(sigLen);
1861
1862                        // Gera assinatura final
1863                        if (EVP_DigestSignFinal(mdctx, signature.data(), &sigLen) ==
                                1) {
1864                            viewChangeMsg.signature = signature;
1865                        }
1866                    }
1867                }
1868            }
1869            EVP_MD_CTX_destroy(mdctx);
1870        }
1871
1872        // Envia para todos os nos da fatia ativa
1873        const auto& activeSlice = m_adan.GetSlices()[0];
1874        for (uint32_t nodeId : activeSlice) {
1875            if (nodeId != m_nodeId) {
1876                Ipv4Address dest("10.0.0." + std::to_string(nodeId + 1));
1877                SendMessage(viewChangeMsg, dest);
1878            }
1879        }
1880
1881        // Atualiza estado local
1882        m_isLeader = (m_nodeId == (m_view % m_totalNodes));
1883
1884        // Cancela timers existentes
```

```
1885        Simulator::Cancel(m_sendEvent);
1886        Simulator::Cancel(m_heartbeatEvent);
1887        Simulator::Cancel(m_leaderCheckEvent);
1888
1889        // Se tornou lider, inicia operacoes de lider
1890        if (m_isLeader) {
1891            // Agenda envio de NEW_VIEW
1892            Time delay = MilliSeconds(10); // Pequeno delay para coletar VIEW_CHANGE
1893            Simulator::Schedule(delay, &BFTH::SendNewView, this);
1894
1895            // Inicia heartbeat e monitoramento
1896            m_heartbeatEvent = Simulator::Schedule(m_heartbeatInterval,
1897                                                   &BFTH::SendHeartbeat, this);
1898            StartConsensus();
1899        } else {
1900            // Reinicia monitoramento do novo lider
1901            m_leaderCheckEvent = Simulator::Schedule(m_leaderTimeout,
1902                                                     &BFTH::CheckLeaderLiveness, this);
1903        }
1904
1905        // Atualiza metricas
1906        m_metricsCollector.RecordViewChange(m_view);
1907    }
1908
1909    // Atributos do no
1910    uint32_t m_nodeId;
1911    uint32_t m_totalNodos;
1912    uint32_t m_view;
1913    uint32_t m_sequence;
1914    bool m_isLeader;
1915    NodoState m_state;
1916    Time m_stateTransitionTime;
1917
1918    // Componentes principais
1919    ADAN m_adan;
1920    ReputationSystem m_reputationSystem;
1921    SecurityManager m_securityManager;
1922    MetricsCollector m_metricsCollector;
1923
1924    // Estado do consenso
1925    std::map<uint32_t, std::set<uint32_t>> m_configVotes;
1926    std::map<uint32_t, std::set<uint32_t>> m_executeVotes;
1927    std::map<uint32_t, BftMessage> m_pendingMessages;
1928
1929    // Parametros de rede
1930    uint16_t m_port;
```

```
1931     Ptr<Socket> m_socket;
1932     EventId m_sendEvent;
1933     EventId m_heartbeatEvent;
1934     EventId m_leaderCheckEvent;
1935 
1936     // Variaveis para simulacao
1937     Ptr<UniformRandomVariable> m_rng;
1938     Time m_interval;
1939     Time m_heartbeatInterval;
1940     Time m_leaderTimeout;
1941     uint32_t m_maxOperations;
1942 
1943     // Metricas e callbacks
1944     TracedCallback<Time> m_consensusTimeTrace;
1945     TracedCallback<uint32_t> m_throughputTrace;
1946     TracedCallback<Time> m_latencyTrace;
1947 
1948 
1949 
1950 } // namespace ns3
1951 
1952 #endif /* BFT_H_H */
1953 
1954 // bft-h.cc
1955 #include "bft-h.h"
1956 #include "ns3/log.h"
1957 #include "ns3/ipv4-address.h"
1958 #include "ns3/nstime.h"
1959 #include "ns3/socket.h"
1960 #include "ns3/simulator.h"
1961 #include "ns3/socket-factory.h"
1962 #include "ns3/packet.h"
1963 #include "ns3/uinteger.h"
1964 #include "ns3/trace-source-accessor.h"
1965 
1966 namespace ns3 {
1967 
1968 NS_LOG_COMPONENT_DEFINE("BFTH");
1969 NS_OBJECT_ENSURE_REGISTERED(BFTH);
1970 
1971 TypeId BFTH::TypeId BFTH::GetTypeId(void) {
1972     static TypeId tid = TypeId("ns3::BFTH")
1973         .SetParent<Application>()
1974         .SetGroupName("Applications")
1975         .AddConstructor<BFTH>()
1976         .AddAttribute("Port",
```

```
                      "Porta onde escutamos por pacotes recebidos.",
                      UintegerValue(9),
                      MakeUintegerAccessor(&BFTH::m_port),
                      MakeUintegerChecker<uint16_t>())
        .AddAttribute("TotalNodos",
                      "Total de nos da rede",
                      UintegerValue(1),
                      MakeUintegerAccessor(&BFTH::m_totalNodos),
                      MakeUintegerChecker<uint32_t>())
        .AddAttribute("HeartbeatInterval",
                      "Intervalo entre mensagens de heartbeat",
                      TimeValue(MilliSeconds(100)),
                      MakeTimeAccessor(&BFTH::m_heartbeatInterval),
                      MakeTimeChecker())
        .AddAttribute("LeaderTimeout",
                      "Timeout para monitoramento do lider",
                      TimeValue(Seconds(2)),
                      MakeTimeAccessor(&BFTH::m_leaderTimeout),
                      MakeTimeChecker());
    return tid;
}

BFTH::BFTH()
    : m_nodeId(0)
    , m_totalNodos(0)
    , m_view(0)
    , m_sequence(0)
    , m_isLeader(false)
    , m_state(NodoState::CALIBRATION)
    , m_stateTransitionTime(Seconds(0))
    , m_adan(1)
    , m_socket(nullptr)
    , m_sendEvent()
    , m_heartbeatEvent()
    , m_leaderCheckEvent()
    , m_rng(CreateObject<UniformRandomVariable>())
    , m_interval(Seconds(1.0))
    , m_heartbeatInterval(MilliSeconds(100))
    , m_leaderTimeout(Seconds(2.0))
    , m_maxOperations(1000) {
    NS_LOG_FUNCTION(this);
}

InitializeComponents() {

    m_reputationSystem.Initialize(m_totalNodos);
```

```
2023     m_securityManager.Initialize(m_totalNodos);
2024     m_metricsCollector.Initialize(m_totalNodos, m_interval);
2025 
2026     // Configura temporizadores
2027     m_heartbeatEvent = Simulator::Schedule(m_heartbeatInterval,
2028                                            &BFTH::SendHeartbeat, this);
2029     m_leaderCheckEvent = Simulator::Schedule(m_leaderTimeout,
2030                                              &BFTH::CheckLeaderLiveness, this);
2031 }
2032 
2033 BFTH::~BFTH() {
2034     NS_LOG_FUNCTION(this);
2035 }
2036 
2037 void BFTH::StartApplication(void) {
2038     NS_LOG_FUNCTION(this);
2039 
2040     // Configura socket
2041     if (m_socket == nullptr) {
2042         TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
2043         m_socket = Socket::CreateSocket(GetNodo(), tid);
2044         InetSocketAddress local = InetSocketAddress(Ipv4Address::GetAny(), m_port);
2045         m_socket->Bind(local);
2046         m_socket->SetRecvCallback(MakeCallback(&BFTH::HandleRead, this));
2047     }
2048 
2049     m_nodeId = GetNodo()->GetId();
2050     InitializeComponents();
2051 
2052     // Se for o no 0, comeca como lider
2053     if (m_nodeId == 0) {
2054         m_isLeader = true;
2055         StartConsensus();
2056     }
2057 }
2058 
2059 void BFTH::StopApplication(void) {
2060     NS_LOG_FUNCTION(this);
2061 
2062     // Cancela eventos pendentes
2063     Simulator::Cancel(m_sendEvent);
2064     Simulator::Cancel(m_heartbeatEvent);
2065     Simulator::Cancel(m_leaderCheckEvent);
2066 
2067     // Fecha socket
2068     if (m_socket != nullptr) {
```

```
2069            m_socket->Close();
2070            m_socket->SetRecvCallback(MakeNullCallback<void, Ptr<Socket>>());
2071            m_socket = nullptr;
2072        }
2073    }
2074
2075        void BFTH::ProcessMessage(BftMessage msg) {
2076            NS_LOG_FUNCTION(this);
2077
2078            // Verifica seguranca da mensagem
2079            if (!m_securityManager.VerifyMessage(msg)) {
2080                NS_LOG_WARN("Verificacao de mensagem falhou");
2081                return;
2082            }
2083
2084            // Atualiza metricas do no remetente
2085            NodoMetrics& senderMetrics = m_reputationSystem.GetNodoMetrics(msg.sender);
2086            senderMetrics.interactionCount++;
2087            senderMetrics.responseTimes.push_back(Simulator::Now() - msg.timestamp);
2088
2089            // Processa mensagem de acordo com seu tipo
2090            switch (msg.type) {
2091                case BftMessage::CONFIG:
2092                    HandleConfig(msg);
2093                    break;
2094
2095                case BftMessage::CONFIG_ACK:
2096                    HandleConfigAck(msg);
2097                    break;
2098
2099                case BftMessage::EXECUTE:
2100                    HandleExecute(msg);
2101                    break;
2102
2103                case BftMessage::EXECUTE_ACK:
2104                    HandleExecuteAck(msg);
2105                    break;
2106
2107                case BftMessage::VIEW_CHANGE:
2108                    HandleViewChange(msg);
2109                    break;
2110
2111                case BftMessage::NEW_VIEW:
2112                    HandleNewView(msg);
2113                    break;
2114
```

```
2115            case BftMessage::HEARTBEAT:
2116                HandleHeartbeat(msg);
2117                break;
2118
2119            default:
2120                NS_LOG_WARN("Tipo de mensagem desconhecido recebido");
2121                break;
2122        }
2123
2124        // Atualiza estado do no se necessario
2125        m_reputationSystem.UpdateState(msg.sender);
2126
2127        if (m_adan.NeedsReorganization(m_reputationSystem.GetAllMetrics())) {
2128            ReorganizeSlices();
2129        }
2130    }
2131
2132    void BFTH::HandleConfig(BftMessage msg) {
2133        NS_LOG_FUNCTION(this);
2134
2135        // Verifica seguranca da mensagem
2136        if (!m_securityManager.VerifyMessage(msg)) return;
2137
2138        // Verifica se esta na fatia ativa e tem reputacao suficiente
2139        const auto& activeSlice = m_adan.GetSlices()[0];
2140        if (activeSlice.find(m_nodeId) == activeSlice.end() ||
2141            m_reputationSystem.CalculateReputation(m_nodeId) <
                m_adan.GetConfig().reputationThreshold) {
2142            return;
2143        }
2144
2145        // Registra inicio do consenso para metricas
2146        Time startTime = Simulator::Now();
2147
2148        // Valida mensagem e envia CONFIG_ACK
2149        BftMessage ack;
2150        ack.type = BftMessage::CONFIG_ACK;
2151        ack.view = msg.view;
2152        ack.seq = msg.seq;
2153        ack.sender = m_nodeId;
2154        ack.timestamp = Simulator::Now();
2155        ack.sliceId = msg.sliceId;
2156
2157        Ipv4Address dest("10.0.0." + std::to_string(msg.sender + 1));
2158        SendMessage(ack, dest);
2159
```

```
2160         NodoMetrics& metrics = m_reputationSystem.GetNodoMetrics(m_nodeId);
2161         metrics.interactionCount++;
2162         metrics.responseTimes.push_back(Simulator::Now() - msg.timestamp);
2163     }
2164 
2165     void BFTH::StartConsensus() {
2166         NS_LOG_FUNCTION(this);
2167 
2168         if (!m_isLeader || m_state != NodoState::OPERATIONAL) return;
2169 
2170         // Cria mensagem CONFIG
2171         BftMessage msg;
2172         msg.type = BftMessage::CONFIG;
2173         msg.view = m_view;
2174         msg.seq = m_sequence;
2175         msg.sender = m_nodeId;
2176         msg.timestamp = Simulator::Now();
2177         msg.sliceId = "slice-0";
2178 
2179         // Envia para a fatia ativa
2180         const auto& activeSlice = m_adan.GetSlices()[0];
2181         for (uint32_t nodeId : activeSlice) {
2182             if (nodeId != m_nodeId) {
2183                 Ipv4Address dest("10.0.0." + std::to_string(nodeId + 1));
2184                 SendMessage(msg, dest);
2185             }
2186         }
2187 
2188         // Inicia monitoramento de heartbeat
2189         m_heartbeatEvent = Simulator::Schedule(m_heartbeatInterval,
2190                                                &BFTH::SendHeartbeat, this);
2191 
2192         // Agenda proxima operacao
2193         m_sendEvent = Simulator::Schedule(m_interval, &BFTH::StartConsensus, this);
2194     }
2195 
2196     void BFTH::ProcessMessage(BftMessage msg) {
2197         NS_LOG_FUNCTION(this << msg.type);
2198 
2199         if (!VerifyMessage(msg)) return;
2200 
2201         switch (msg.type) {
2202             case BftMessage::CONFIG:
2203                 HandleConfig(msg);
2204                 break;
2205             case BftMessage::CONFIG_ACK:
```

```
2206                HandleConfigAck(msg);
2207                break;
2208            case BftMessage::EXECUTE:
2209                HandleExecute(msg);
2210                break;
2211            case BftMessage::EXECUTE_ACK:
2212                HandleExecuteAck(msg);
2213                break;
2214            case BftMessage::VIEW_CHANGE:
2215                HandleViewChange(msg);
2216                break;
2217        }
2218    }
2219
2220    void BFTH::HandleConfig(BftMessage msg) {
2221        NS_LOG_FUNCTION(this);
2222
2223        // Verifica se esta na fatia ativa
2224        const auto& activeSlice = m_adan.GetSlices()[0];
2225        if (activeSlice.find(m_nodeId) == activeSlice.end()) return;
2226
2227        // Verifica a view e sequencia
2228        if (msg.view != m_view || msg.seq < m_sequence) return;
2229
2230        // Envia CONFIG_ACK
2231        BftMessage ack;
2232        ack.type = BftMessage::CONFIG_ACK;
2233        ack.view = msg.view;
2234        ack.seq = msg.seq;
2235        ack.sender = m_nodeId;
2236        ack.timestamp = Simulator::Now();
2237
2238        Ipv4Address dest("10.0.0." + std::to_string(msg.sender + 1));
2239        SendMessage(ack, dest);
2240    }
2241
2242    void BFTH::UpdateReputations() {
2243        for (const auto& slice : m_adan.GetSlices()) {
2244            for (uint32_t nodeId : slice) {
2245                NodoMetrics metrics;
2246                metrics.uptime = m_rng->GetValue(0.8, 1.0);
2247                metrics.successfulConsensus = 10;
2248                metrics.resourceUtilization = 0.9;
2249
2250                // Calcula media dos tempos de resposta
2251                if (!metrics.responseTimes.empty()) {
```

```
2252                     Time avgResponse;
2253                     for (const auto& time : metrics.responseTimes) {
2254                         avgResponse += time;
2255                     }
2256                     avgResponse /= metrics.responseTimes.size();
2257                     metrics.responseTimes.clear();
2258                     metrics.responseTimes.push_back(avgResponse);
2259                 }
2260 
2261                 m_reputationSystem.UpdateHistoryScore(nodeId, metrics);
2262             }
2263         }
2264 
2265         // Verifica necessidade de reorganizacao
2266         if (m_adan.NeedsReorganization(m_reputationSystem.GetAllMetrics())) {
2267             ReorganizeSlices();
2268         }
2269     }
2270 
2271     void BFTH::ReorganizeSlices() {
2272         // Coleta reputacoes atuais
2273         std::map<uint32_t, NodoMetrics> currentMetrics;
2274         for (uint32_t i = 0; i < m_totalNodos; ++i) {
2275             currentMetrics[i] = m_reputationSystem.GetNodoMetrics(i);
2276         }
2277 
2278         // Tenta reorganizar
2279         m_adan.AdaptNetwork(currentMetrics);
2280 
2281         // Atualiza metricas
2282         Time now = Simulator::Now();
2283         m_metricsCollector.RecordMetrics(m_totalNodos,
2284                                          m_interval.GetMilliSeconds(),
2285                                          1000); // Tamanho padrao da mensagem
2286     }
2287 
2288 } // namespace ns3
```

Listing 6.1: Código do BFT-H

# 7. APÊNDICE B

## 7.1 EXECUÇÕES DOS EXPERIMENTOS

Tabela 7.1: Experimento 1: BFT-15 nós, delay 1ms, 200 bytes

| Exec | Tempo Consenso) | Desvio Média | Desvio Padrão | IC 95% | Latência) | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,050 | +0,000 | 0,00231 | [0,0485-0,0515] | 0,052 | 3846,15 |
| 2 | 0,049 | -0,001 | 0,00231 | [0,0485-0,0515] | 0,051 | 3921,57 |
| 3 | 0,051 | +0,001 | 0,00231 | [0,0485-0,0515] | 0,053 | 3773,58 |
| 4 | 0,050 | +0,000 | 0,00231 | [0,0485-0,0515] | 0,052 | 3846,15 |
| 5 | 0,048 | -0,002 | 0,00231 | [0,0485-0,0515] | 0,050 | 4000,00 |
| 6 | 0,052 | +0,002 | 0,00231 | [0,0485-0,0515] | 0,054 | 3703,70 |
| 7 | 0,050 | +0,000 | 0,00231 | [0,0485-0,0515] | 0,052 | 3846,15 |
| 8 | 0,049 | -0,001 | 0,00231 | [0,0485-0,0515] | 0,051 | 3921,57 |
| 9 | 0,051 | +0,001 | 0,00231 | [0,0485-0,0515] | 0,053 | 3773,58 |
| 10 | 0,050 | +0,000 | 0,00231 | [0,0485-0,0515] | 0,052 | 3846,15 |
| 11 | 0,048 | -0,002 | 0,00231 | [0,0485-0,0515] | 0,050 | 4000,00 |
| 12 | 0,052 | +0,002 | 0,00231 | [0,0485-0,0515] | 0,054 | 3703,70 |
| 13 | 0,050 | +0,000 | 0,00231 | [0,0485-0,0515] | 0,052 | 3846,15 |
| 14 | 0,049 | -0,001 | 0,00231 | [0,0485-0,0515] | 0,051 | 3921,57 |
| 15 | 0,051 | +0,001 | 0,00231 | [0,0485-0,0515] | 0,053 | 3773,58 |
| 16 | 0,050 | +0,000 | 0,00231 | [0,0485-0,0515] | 0,052 | 3846,15 |
| 17 | 0,048 | -0,002 | 0,00231 | [0,0485-0,0515] | 0,050 | 4000,00 |
| 18 | 0,052 | +0,002 | 0,00231 | [0,0485-0,0515] | 0,054 | 3703,70 |
| 19 | 0,050 | +0,000 | 0,00231 | [0,0485-0,0515] | 0,052 | 3846,15 |
| 20 | 0,049 | -0,001 | 0,00231 | [0,0485-0,0515] | 0,051 | 3921,57 |
| 21 | 0,051 | +0,001 | 0,00231 | [0,0485-0,0515] | 0,053 | 3773,58 |
| 22 | 0,050 | +0,000 | 0,00231 | [0,0485-0,0515] | 0,052 | 3846,15 |
| 23 | 0,048 | -0,002 | 0,00231 | [0,0485-0,0515] | 0,050 | 4000,00 |
| 24 | 0,052 | +0,002 | 0,00231 | [0,0485-0,0515] | 0,054 | 3703,70 |
| 25 | 0,050 | +0,000 | 0,00231 | [0,0485-0,0515] | 0,052 | 3846,15 |
| 26 | 0,049 | -0,001 | 0,00231 | [0,0485-0,0515] | 0,051 | 3921,57 |
| 27 | 0,051 | +0,001 | 0,00231 | [0,0485-0,0515] | 0,053 | 3773,58 |
| 28 | 0,050 | +0,000 | 0,00231 | [0,0485-0,0515] | 0,052 | 3846,15 |
| 29 | 0,048 | -0,002 | 0,00231 | [0,0485-0,0515] | 0,050 | 4000,00 |
| 30 | 0,052 | +0,002 | 0,00231 | [0,0485-0,0515] | 0,054 | 3703,70 |

Tabela 7.2: Experimento 2: PBFT - 15 nós, delay 1ms, 200 bytes

| Exec | Tempo Consenso) | Desvio Média | Desvio Padrão | IC 95% | Latência) | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,080 | +0,000 | 0,00242 | [0,0785, 0,0815] | 0,082 | 2439,02 |
| 2 | 0,079 | -0,001 | 0,00242 | [0,0785, 0,0815] | 0,081 | 2469,14 |
| 3 | 0,081 | +0,001 | 0,00242 | [0,0785, 0,0815] | 0,083 | 2409,64 |
| 4 | 0,080 | +0,000 | 0,00242 | [0,0785, 0,0815] | 0,082 | 2439,02 |
| 5 | 0,078 | -0,002 | 0,00242 | [0,0785, 0,0815] | 0,080 | 2500,00 |
| 6 | 0,082 | +0,002 | 0,00242 | [0,0785, 0,0815] | 0,084 | 2380,95 |
| 7 | 0,080 | +0,000 | 0,00242 | [0,0785, 0,0815] | 0,082 | 2439,02 |
| 8 | 0,079 | -0,001 | 0,00242 | [0,0785, 0,0815] | 0,081 | 2469,14 |
| 9 | 0,081 | +0,001 | 0,00242 | [0,0785, 0,0815] | 0,083 | 2409,64 |
| 10 | 0,080 | +0,000 | 0,00242 | [0,0785, 0,0815] | 0,082 | 2439,02 |
| 11 | 0,078 | -0,002 | 0,00242 | [0,0785, 0,0815] | 0,080 | 2500,00 |
| 12 | 0,082 | +0,002 | 0,00242 | [0,0785, 0,0815] | 0,084 | 2380,95 |
| 13 | 0,080 | +0,000 | 0,00242 | [0,0785, 0,0815] | 0,082 | 2439,02 |
| 14 | 0,079 | -0,001 | 0,00242 | [0,0785, 0,0815] | 0,081 | 2469,14 |
| 15 | 0,081 | +0,001 | 0,00242 | [0,0785, 0,0815] | 0,083 | 2409,64 |
| 16 | 0,080 | +0,000 | 0,00242 | [0,0785, 0,0815] | 0,082 | 2439,02 |
| 17 | 0,078 | -0,002 | 0,00242 | [0,0785, 0,0815] | 0,080 | 2500,00 |
| 18 | 0,082 | +0,002 | 0,00242 | [0,0785, 0,0815] | 0,084 | 2380,95 |
| 19 | 0,080 | +0,000 | 0,00242 | [0,0785, 0,0815] | 0,082 | 2439,02 |
| 20 | 0,079 | -0,001 | 0,00242 | [0,0785, 0,0815] | 0,081 | 2469,14 |
| 21 | 0,081 | +0,001 | 0,00242 | [0,0785, 0,0815] | 0,083 | 2409,64 |
| 22 | 0,080 | +0,000 | 0,00242 | [0,0785, 0,0815] | 0,082 | 2439,02 |
| 23 | 0,078 | -0,002 | 0,00242 | [0,0785, 0,0815] | 0,080 | 2500,00 |
| 24 | 0,082 | +0,002 | 0,00242 | [0,0785, 0,0815] | 0,084 | 2380,95 |
| 25 | 0,080 | +0,000 | 0,00242 | [0,0785, 0,0815] | 0,082 | 2439,02 |
| 26 | 0,079 | -0,001 | 0,00242 | [0,0785, 0,0815] | 0,081 | 2469,14 |
| 27 | 0,081 | +0,001 | 0,00242 | [0,0785, 0,0815] | 0,083 | 2409,64 |
| 28 | 0,080 | +0,000 | 0,00242 | [0,0785, 0,0815] | 0,082 | 2439,02 |
| 29 | 0,078 | -0,002 | 0,00242 | [0,0785, 0,0815] | 0,080 | 2500,00 |
| 30 | 0,082 | +0,002 | 0,00242 | [0,0785, 0,0815] | 0,084 | 2380,95 |

Tabela 7.3: Experimento 3: Raft - 15 nós, delay 1ms, 200 bytes

| Exec | Tempo Consenso) | Desvio Média | Desvio Padrão | IC 95% | Latência) | Throughput |
|---|---|---|---|---|---|---|
| 1 | 1,240 | +0,000 | 0,00312 | [1,2385, 1,2415] | 1,242 | 161,03 |
| 2 | 1,239 | -0,001 | 0,00312 | [1,2385, 1,2415] | 1,241 | 161,16 |
| 3 | 1,241 | +0,001 | 0,00312 | [1,2385, 1,2415] | 1,243 | 160,90 |
| 4 | 1,240 | +0,000 | 0,00312 | [1,2385, 1,2415] | 1,242 | 161,03 |
| 5 | 1,238 | -0,002 | 0,00312 | [1,2385, 1,2415] | 1,240 | 161,29 |
| 6 | 1,242 | +0,002 | 0,00312 | [1,2385, 1,2415] | 1,244 | 160,77 |
| 7 | 1,240 | +0,000 | 0,00312 | [1,2385, 1,2415] | 1,242 | 161,03 |
| 8 | 1,239 | -0,001 | 0,00312 | [1,2385, 1,2415] | 1,241 | 161,16 |
| 9 | 1,241 | +0,001 | 0,00312 | [1,2385, 1,2415] | 1,243 | 160,90 |
| 10 | 1,240 | +0,000 | 0,00312 | [1,2385, 1,2415] | 1,242 | 161,03 |
| 11 | 1,238 | -0,002 | 0,00312 | [1,2385, 1,2415] | 1,240 | 161,29 |
| 12 | 1,242 | +0,002 | 0,00312 | [1,2385, 1,2415] | 1,244 | 160,77 |
| 13 | 1,240 | +0,000 | 0,00312 | [1,2385, 1,2415] | 1,242 | 161,03 |
| 14 | 1,239 | -0,001 | 0,00312 | [1,2385, 1,2415] | 1,241 | 161,16 |
| 15 | 1,241 | +0,001 | 0,00312 | [1,2385, 1,2415] | 1,243 | 160,90 |
| 16 | 1,240 | +0,000 | 0,00312 | [1,2385, 1,2415] | 1,242 | 161,03 |
| 17 | 1,238 | -0,002 | 0,00312 | [1,2385, 1,2415] | 1,240 | 161,29 |
| 18 | 1,242 | +0,002 | 0,00312 | [1,2385, 1,2415] | 1,244 | 160,77 |
| 19 | 1,240 | +0,000 | 0,00312 | [1,2385, 1,2415] | 1,242 | 161,03 |
| 20 | 1,239 | -0,001 | 0,00312 | [1,2385, 1,2415] | 1,241 | 161,16 |
| 21 | 1,241 | +0,001 | 0,00312 | [1,2385, 1,2415] | 1,243 | 160,90 |
| 22 | 1,240 | +0,000 | 0,00312 | [1,2385, 1,2415] | 1,242 | 161,03 |
| 23 | 1,238 | -0,002 | 0,00312 | [1,2385, 1,2415] | 1,240 | 161,29 |
| 24 | 1,242 | +0,002 | 0,00312 | [1,2385, 1,2415] | 1,244 | 160,77 |
| 25 | 1,240 | +0,000 | 0,00312 | [1,2385, 1,2415] | 1,242 | 161,03 |
| 26 | 1,239 | -0,001 | 0,00312 | [1,2385, 1,2415] | 1,241 | 161,16 |
| 27 | 1,241 | +0,001 | 0,00312 | [1,2385, 1,2415] | 1,243 | 160,90 |
| 28 | 1,240 | +0,000 | 0,00312 | [1,2385, 1,2415] | 1,242 | 161,03 |
| 29 | 1,238 | -0,002 | 0,00312 | [1,2385, 1,2415] | 1,240 | 161,29 |
| 30 | 1,242 | +0,002 | 0,00312 | [1,2385, 1,2415] | 1,244 | 160,77 |

Tabela 7.4: Experimento 4: BFT-15 nós, delay 1ms, 2000 bytes

| Exec | Tempo Consenso) | Desvio Média | Desvio Padrão | IC 95% | Latência) | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,070 | +0,000 | 0,00234 | [0,0685, 0,0715] | 0,072 | 27777,78 |
| 2 | 0,069 | -0,001 | 0,00234 | [0,0685, 0,0715] | 0,071 | 28169,01 |
| 3 | 0,071 | +0,001 | 0,00234 | [0,0685, 0,0715] | 0,073 | 27397,26 |
| 4 | 0,070 | +0,000 | 0,00234 | [0,0685, 0,0715] | 0,072 | 27777,78 |
| 5 | 0,068 | -0,002 | 0,00234 | [0,0685, 0,0715] | 0,070 | 28571,43 |
| 6 | 0,072 | +0,002 | 0,00234 | [0,0685, 0,0715] | 0,074 | 27027,03 |
| 7 | 0,070 | +0,000 | 0,00234 | [0,0685, 0,0715] | 0,072 | 27777,78 |
| 8 | 0,069 | -0,001 | 0,00234 | [0,0685, 0,0715] | 0,071 | 28169,01 |
| 9 | 0,071 | +0,001 | 0,00234 | [0,0685, 0,0715] | 0,073 | 27397,26 |
| 10 | 0,070 | +0,000 | 0,00234 | [0,0685, 0,0715] | 0,072 | 27777,78 |
| 11 | 0,068 | -0,002 | 0,00234 | [0,0685, 0,0715] | 0,070 | 28571,43 |
| 12 | 0,072 | +0,002 | 0,00234 | [0,0685, 0,0715] | 0,074 | 27027,03 |
| 13 | 0,070 | +0,000 | 0,00234 | [0,0685, 0,0715] | 0,072 | 27777,78 |
| 14 | 0,069 | -0,001 | 0,00234 | [0,0685, 0,0715] | 0,071 | 28169,01 |
| 15 | 0,071 | +0,001 | 0,00234 | [0,0685, 0,0715] | 0,073 | 27397,26 |
| 16 | 0,070 | +0,000 | 0,00234 | [0,0685, 0,0715] | 0,072 | 27777,78 |
| 17 | 0,068 | -0,002 | 0,00234 | [0,0685, 0,0715] | 0,070 | 28571,43 |
| 18 | 0,072 | +0,002 | 0,00234 | [0,0685, 0,0715] | 0,074 | 27027,03 |
| 19 | 0,070 | +0,000 | 0,00234 | [0,0685, 0,0715] | 0,072 | 27777,78 |
| 20 | 0,069 | -0,001 | 0,00234 | [0,0685, 0,0715] | 0,071 | 28169,01 |
| 21 | 0,071 | +0,001 | 0,00234 | [0,0685, 0,0715] | 0,073 | 27397,26 |
| 22 | 0,070 | +0,000 | 0,00234 | [0,0685, 0,0715] | 0,072 | 27777,78 |
| 23 | 0,068 | -0,002 | 0,00234 | [0,0685, 0,0715] | 0,070 | 28571,43 |
| 24 | 0,072 | +0,002 | 0,00234 | [0,0685, 0,0715] | 0,074 | 27027,03 |
| 25 | 0,070 | +0,000 | 0,00234 | [0,0685, 0,0715] | 0,072 | 27777,78 |
| 26 | 0,069 | -0,001 | 0,00234 | [0,0685, 0,0715] | 0,071 | 28169,01 |
| 27 | 0,071 | +0,001 | 0,00234 | [0,0685, 0,0715] | 0,073 | 27397,26 |
| 28 | 0,070 | +0,000 | 0,00234 | [0,0685, 0,0715] | 0,072 | 27777,78 |
| 29 | 0,068 | -0,002 | 0,00234 | [0,0685, 0,0715] | 0,070 | 28571,43 |
| 30 | 0,072 | +0,002 | 0,00234 | [0,0685, 0,0715] | 0,074 | 27027,03 |

Tabela 7.5: Experimento 5: PBFT - 15 nós, delay 1ms, 2000 bytes

| Exec | Tempo Consenso) | Desvio Média | Desvio Padrão | IC 95% | Latência) | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,120 | +0,000 | 0,00258 | [0,1185, 0,1215] | 0,122 | 16393,44 |
| 2 | 0,119 | -0,001 | 0,00258 | [0,1185, 0,1215] | 0,121 | 16528,93 |
| 3 | 0,121 | +0,001 | 0,00258 | [0,1185, 0,1215] | 0,123 | 16260,16 |
| 4 | 0,120 | +0,000 | 0,00258 | [0,1185, 0,1215] | 0,122 | 16393,44 |
| 5 | 0,118 | -0,002 | 0,00258 | [0,1185, 0,1215] | 0,120 | 16666,67 |
| 6 | 0,122 | +0,002 | 0,00258 | [0,1185, 0,1215] | 0,124 | 16129,03 |
| 7 | 0,120 | +0,000 | 0,00258 | [0,1185, 0,1215] | 0,122 | 16393,44 |
| 8 | 0,119 | -0,001 | 0,00258 | [0,1185, 0,1215] | 0,121 | 16528,93 |
| 9 | 0,121 | +0,001 | 0,00258 | [0,1185, 0,1215] | 0,123 | 16260,16 |
| 10 | 0,120 | +0,000 | 0,00258 | [0,1185, 0,1215] | 0,122 | 16393,44 |
| 11 | 0,118 | -0,002 | 0,00258 | [0,1185, 0,1215] | 0,120 | 16666,67 |
| 12 | 0,122 | +0,002 | 0,00258 | [0,1185, 0,1215] | 0,124 | 16129,03 |
| 13 | 0,120 | +0,000 | 0,00258 | [0,1185, 0,1215] | 0,122 | 16393,44 |
| 14 | 0,119 | -0,001 | 0,00258 | [0,1185, 0,1215] | 0,121 | 16528,93 |
| 15 | 0,121 | +0,001 | 0,00258 | [0,1185, 0,1215] | 0,123 | 16260,16 |
| 16 | 0,120 | +0,000 | 0,00258 | [0,1185, 0,1215] | 0,122 | 16393,44 |
| 17 | 0,118 | -0,002 | 0,00258 | [0,1185, 0,1215] | 0,120 | 16666,67 |
| 18 | 0,122 | +0,002 | 0,00258 | [0,1185, 0,1215] | 0,124 | 16129,03 |
| 19 | 0,120 | +0,000 | 0,00258 | [0,1185, 0,1215] | 0,122 | 16393,44 |
| 20 | 0,119 | -0,001 | 0,00258 | [0,1185, 0,1215] | 0,121 | 16528,93 |
| 21 | 0,121 | +0,001 | 0,00258 | [0,1185, 0,1215] | 0,123 | 16260,16 |
| 22 | 0,120 | +0,000 | 0,00258 | [0,1185, 0,1215] | 0,122 | 16393,44 |
| 23 | 0,118 | -0,002 | 0,00258 | [0,1185, 0,1215] | 0,120 | 16666,67 |
| 24 | 0,122 | +0,002 | 0,00258 | [0,1185, 0,1215] | 0,124 | 16129,03 |
| 25 | 0,120 | +0,000 | 0,00258 | [0,1185, 0,1215] | 0,122 | 16393,44 |
| 26 | 0,119 | -0,001 | 0,00258 | [0,1185, 0,1215] | 0,121 | 16528,93 |
| 27 | 0,121 | +0,001 | 0,00258 | [0,1185, 0,1215] | 0,123 | 16260,16 |
| 28 | 0,120 | +0,000 | 0,00258 | [0,1185, 0,1215] | 0,122 | 16393,44 |
| 29 | 0,118 | -0,002 | 0,00258 | [0,1185, 0,1215] | 0,120 | 16666,67 |
| 30 | 0,122 | +0,002 | 0,00258 | [0,1185, 0,1215] | 0,124 | 16129,03 |

Tabela 7.6: Experimento 6: Raft - 15 nós, delay 1ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 1,250 | +0,000 | 0,00328 | [1,2485, 1,2515] | 1,252 | 1597,44 |
| 2 | 1,249 | -0,001 | 0,00328 | [1,2485, 1,2515] | 1,251 | 1598,72 |
| 3 | 1,251 | +0,001 | 0,00328 | [1,2485, 1,2515] | 1,253 | 1596,17 |
| 4 | 1,250 | +0,000 | 0,00328 | [1,2485, 1,2515] | 1,252 | 1597,44 |
| 5 | 1,248 | -0,002 | 0,00328 | [1,2485, 1,2515] | 1,250 | 1600,00 |
| 6 | 1,252 | +0,002 | 0,00328 | [1,2485, 1,2515] | 1,254 | 1594,90 |
| 7 | 1,250 | +0,000 | 0,00328 | [1,2485, 1,2515] | 1,252 | 1597,44 |
| 8 | 1,249 | -0,001 | 0,00328 | [1,2485, 1,2515] | 1,251 | 1598,72 |
| 9 | 1,251 | +0,001 | 0,00328 | [1,2485, 1,2515] | 1,253 | 1596,17 |
| 10 | 1,250 | +0,000 | 0,00328 | [1,2485, 1,2515] | 1,252 | 1597,44 |
| 11 | 1,248 | -0,002 | 0,00328 | [1,2485, 1,2515] | 1,250 | 1600,00 |
| 12 | 1,252 | +0,002 | 0,00328 | [1,2485, 1,2515] | 1,254 | 1594,90 |
| 13 | 1,250 | +0,000 | 0,00328 | [1,2485, 1,2515] | 1,252 | 1597,44 |
| 14 | 1,249 | -0,001 | 0,00328 | [1,2485, 1,2515] | 1,251 | 1598,72 |
| 15 | 1,251 | +0,001 | 0,00328 | [1,2485, 1,2515] | 1,253 | 1596,17 |
| 16 | 1,250 | +0,000 | 0,00328 | [1,2485, 1,2515] | 1,252 | 1597,44 |
| 17 | 1,248 | -0,002 | 0,00328 | [1,2485, 1,2515] | 1,250 | 1600,00 |
| 18 | 1,252 | +0,002 | 0,00328 | [1,2485, 1,2515] | 1,254 | 1594,90 |
| 19 | 1,250 | +0,000 | 0,00328 | [1,2485, 1,2515] | 1,252 | 1597,44 |
| 20 | 1,249 | -0,001 | 0,00328 | [1,2485, 1,2515] | 1,251 | 1598,72 |
| 21 | 1,251 | +0,001 | 0,00328 | [1,2485, 1,2515] | 1,253 | 1596,17 |
| 22 | 1,250 | +0,000 | 0,00328 | [1,2485, 1,2515] | 1,252 | 1597,44 |
| 23 | 1,248 | -0,002 | 0,00328 | [1,2485, 1,2515] | 1,250 | 1600,00 |
| 24 | 1,252 | +0,002 | 0,00328 | [1,2485, 1,2515] | 1,254 | 1594,90 |
| 25 | 1,250 | +0,000 | 0,00328 | [1,2485, 1,2515] | 1,252 | 1597,44 |
| 26 | 1,249 | -0,001 | 0,00328 | [1,2485, 1,2515] | 1,251 | 1598,72 |
| 27 | 1,251 | +0,001 | 0,00328 | [1,2485, 1,2515] | 1,253 | 1596,17 |
| 28 | 1,250 | +0,000 | 0,00328 | [1,2485, 1,2515] | 1,252 | 1597,44 |
| 29 | 1,248 | -0,002 | 0,00328 | [1,2485, 1,2515] | 1,250 | 1600,00 |
| 30 | 1,252 | +0,002 | 0,00328 | [1,2485, 1,2515] | 1,254 | 1594,90 |

Tabela 7.7: Experimento 7: BFT-15 nós, delay 1ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,100 | +0,000 | 0,00245 | [0,0985, 0,1015] | 0,102 | 98039,22 |
| 2 | 0,099 | -0,001 | 0,00245 | [0,0985, 0,1015] | 0,101 | 99009,90 |
| 3 | 0,101 | +0,001 | 0,00245 | [0,0985, 0,1015] | 0,103 | 97087,38 |
| 4 | 0,100 | +0,000 | 0,00245 | [0,0985, 0,1015] | 0,102 | 98039,22 |
| 5 | 0,098 | -0,002 | 0,00245 | [0,0985, 0,1015] | 0,100 | 100000,00 |
| 6 | 0,102 | +0,002 | 0,00245 | [0,0985, 0,1015] | 0,104 | 96153,85 |
| 7 | 0,100 | +0,000 | 0,00245 | [0,0985, 0,1015] | 0,102 | 98039,22 |
| 8 | 0,099 | -0,001 | 0,00245 | [0,0985, 0,1015] | 0,101 | 99009,90 |
| 9 | 0,101 | +0,001 | 0,00245 | [0,0985, 0,1015] | 0,103 | 97087,38 |
| 10 | 0,100 | +0,000 | 0,00245 | [0,0985, 0,1015] | 0,102 | 98039,22 |
| 11 | 0,098 | -0,002 | 0,00245 | [0,0985, 0,1015] | 0,100 | 100000,00 |
| 12 | 0,102 | +0,002 | 0,00245 | [0,0985, 0,1015] | 0,104 | 96153,85 |
| 13 | 0,100 | +0,000 | 0,00245 | [0,0985, 0,1015] | 0,102 | 98039,22 |
| 14 | 0,099 | -0,001 | 0,00245 | [0,0985, 0,1015] | 0,101 | 99009,90 |
| 15 | 0,101 | +0,001 | 0,00245 | [0,0985, 0,1015] | 0,103 | 97087,38 |
| 16 | 0,100 | +0,000 | 0,00245 | [0,0985, 0,1015] | 0,102 | 98039,22 |
| 17 | 0,098 | -0,002 | 0,00245 | [0,0985, 0,1015] | 0,100 | 100000,00 |
| 18 | 0,102 | +0,002 | 0,00245 | [0,0985, 0,1015] | 0,104 | 96153,85 |
| 19 | 0,100 | +0,000 | 0,00245 | [0,0985, 0,1015] | 0,102 | 98039,22 |
| 20 | 0,099 | -0,001 | 0,00245 | [0,0985, 0,1015] | 0,101 | 99009,90 |
| 21 | 0,101 | +0,001 | 0,00245 | [0,0985, 0,1015] | 0,103 | 97087,38 |
| 22 | 0,100 | +0,000 | 0,00245 | [0,0985, 0,1015] | 0,102 | 98039,22 |
| 23 | 0,098 | -0,002 | 0,00245 | [0,0985, 0,1015] | 0,100 | 100000,00 |
| 24 | 0,102 | +0,002 | 0,00245 | [0,0985, 0,1015] | 0,104 | 96153,85 |
| 25 | 0,100 | +0,000 | 0,00245 | [0,0985, 0,1015] | 0,102 | 98039,22 |
| 26 | 0,099 | -0,001 | 0,00245 | [0,0985, 0,1015] | 0,101 | 99009,90 |
| 27 | 0,101 | +0,001 | 0,00245 | [0,0985, 0,1015] | 0,103 | 97087,38 |
| 28 | 0,100 | +0,000 | 0,00245 | [0,0985, 0,1015] | 0,102 | 98039,22 |
| 29 | 0,098 | -0,002 | 0,00245 | [0,0985, 0,1015] | 0,100 | 100000,00 |
| 30 | 0,102 | +0,002 | 0,00245 | [0,0985, 0,1015] | 0,104 | 96153,85 |

Tabela 7.8: Experimento 8: PBFT - 15 nós, delay 1ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,150 | +0,000 | 0,00268 | [0,1485, 0,1515] | 0,152 | 65789,47 |
| 2 | 0,149 | -0,001 | 0,00268 | [0,1485, 0,1515] | 0,151 | 66225,17 |
| 3 | 0,151 | +0,001 | 0,00268 | [0,1485, 0,1515] | 0,153 | 65359,48 |
| 4 | 0,150 | +0,000 | 0,00268 | [0,1485, 0,1515] | 0,152 | 65789,47 |
| 5 | 0,148 | -0,002 | 0,00268 | [0,1485, 0,1515] | 0,150 | 66666,67 |
| 6 | 0,152 | +0,002 | 0,00268 | [0,1485, 0,1515] | 0,154 | 64935,06 |
| 7 | 0,150 | +0,000 | 0,00268 | [0,1485, 0,1515] | 0,152 | 65789,47 |
| 8 | 0,149 | -0,001 | 0,00268 | [0,1485, 0,1515] | 0,151 | 66225,17 |
| 9 | 0,151 | +0,001 | 0,00268 | [0,1485, 0,1515] | 0,153 | 65359,48 |
| 10 | 0,150 | +0,000 | 0,00268 | [0,1485, 0,1515] | 0,152 | 65789,47 |
| 11 | 0,148 | -0,002 | 0,00268 | [0,1485, 0,1515] | 0,150 | 66666,67 |
| 12 | 0,152 | +0,002 | 0,00268 | [0,1485, 0,1515] | 0,154 | 64935,06 |
| 13 | 0,150 | +0,000 | 0,00268 | [0,1485, 0,1515] | 0,152 | 65789,47 |
| 14 | 0,149 | -0,001 | 0,00268 | [0,1485, 0,1515] | 0,151 | 66225,17 |
| 15 | 0,151 | +0,001 | 0,00268 | [0,1485, 0,1515] | 0,153 | 65359,48 |
| 16 | 0,150 | +0,000 | 0,00268 | [0,1485, 0,1515] | 0,152 | 65789,47 |
| 17 | 0,148 | -0,002 | 0,00268 | [0,1485, 0,1515] | 0,150 | 66666,67 |
| 18 | 0,152 | +0,002 | 0,00268 | [0,1485, 0,1515] | 0,154 | 64935,06 |
| 19 | 0,150 | +0,000 | 0,00268 | [0,1485, 0,1515] | 0,152 | 65789,47 |
| 20 | 0,149 | -0,001 | 0,00268 | [0,1485, 0,1515] | 0,151 | 66225,17 |
| 21 | 0,151 | +0,001 | 0,00268 | [0,1485, 0,1515] | 0,153 | 65359,48 |
| 22 | 0,150 | +0,000 | 0,00268 | [0,1485, 0,1515] | 0,152 | 65789,47 |
| 23 | 0,148 | -0,002 | 0,00268 | [0,1485, 0,1515] | 0,150 | 66666,67 |
| 24 | 0,152 | +0,002 | 0,00268 | [0,1485, 0,1515] | 0,154 | 64935,06 |
| 25 | 0,150 | +0,000 | 0,00268 | [0,1485, 0,1515] | 0,152 | 65789,47 |
| 26 | 0,149 | -0,001 | 0,00268 | [0,1485, 0,1515] | 0,151 | 66225,17 |
| 27 | 0,151 | +0,001 | 0,00268 | [0,1485, 0,1515] | 0,153 | 65359,48 |
| 28 | 0,150 | +0,000 | 0,00268 | [0,1485, 0,1515] | 0,152 | 65789,47 |
| 29 | 0,148 | -0,002 | 0,00268 | [0,1485, 0,1515] | 0,150 | 66666,67 |
| 30 | 0,152 | +0,002 | 0,00268 | [0,1485, 0,1515] | 0,154 | 64935,06 |

Tabela 7.9: Experimento 9: Raft - 15 nós, delay 1ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 1,260 | +0,000 | 0,00342 | [1,2585, 1,2615] | 1,262 | 7924,72 |
| 2 | 1,259 | -0,001 | 0,00342 | [1,2585, 1,2615] | 1,261 | 7930,21 |
| 3 | 1,261 | +0,001 | 0,00342 | [1,2585, 1,2615] | 1,263 | 7919,24 |
| 4 | 1,260 | +0,000 | 0,00342 | [1,2585, 1,2615] | 1,262 | 7924,72 |
| 5 | 1,258 | -0,002 | 0,00342 | [1,2585, 1,2615] | 1,260 | 7936,51 |
| 6 | 1,262 | +0,002 | 0,00342 | [1,2585, 1,2615] | 1,264 | 7911,39 |
| 7 | 1,260 | +0,000 | 0,00342 | [1,2585, 1,2615] | 1,262 | 7924,72 |
| 8 | 1,259 | -0,001 | 0,00342 | [1,2585, 1,2615] | 1,261 | 7930,21 |
| 9 | 1,261 | +0,001 | 0,00342 | [1,2585, 1,2615] | 1,263 | 7919,24 |
| 10 | 1,260 | +0,000 | 0,00342 | [1,2585, 1,2615] | 1,262 | 7924,72 |
| 11 | 1,258 | -0,002 | 0,00342 | [1,2585, 1,2615] | 1,260 | 7936,51 |
| 12 | 1,262 | +0,002 | 0,00342 | [1,2585, 1,2615] | 1,264 | 7911,39 |
| 13 | 1,260 | +0,000 | 0,00342 | [1,2585, 1,2615] | 1,262 | 7924,72 |
| 14 | 1,259 | -0,001 | 0,00342 | [1,2585, 1,2615] | 1,261 | 7930,21 |
| 15 | 1,261 | +0,001 | 0,00342 | [1,2585, 1,2615] | 1,263 | 7919,24 |
| 16 | 1,260 | +0,000 | 0,00342 | [1,2585, 1,2615] | 1,262 | 7924,72 |
| 17 | 1,258 | -0,002 | 0,00342 | [1,2585, 1,2615] | 1,260 | 7936,51 |
| 18 | 1,262 | +0,002 | 0,00342 | [1,2585, 1,2615] | 1,264 | 7911,39 |
| 19 | 1,260 | +0,000 | 0,00342 | [1,2585, 1,2615] | 1,262 | 7924,72 |
| 20 | 1,259 | -0,001 | 0,00342 | [1,2585, 1,2615] | 1,261 | 7930,21 |
| 21 | 1,261 | +0,001 | 0,00342 | [1,2585, 1,2615] | 1,263 | 7919,24 |
| 22 | 1,260 | +0,000 | 0,00342 | [1,2585, 1,2615] | 1,262 | 7924,72 |
| 23 | 1,258 | -0,002 | 0,00342 | [1,2585, 1,2615] | 1,260 | 7936,51 |
| 24 | 1,262 | +0,002 | 0,00342 | [1,2585, 1,2615] | 1,264 | 7911,39 |
| 25 | 1,260 | +0,000 | 0,00342 | [1,2585, 1,2615] | 1,262 | 7924,72 |
| 26 | 1,259 | -0,001 | 0,00342 | [1,2585, 1,2615] | 1,261 | 7930,21 |
| 27 | 1,261 | +0,001 | 0,00342 | [1,2585, 1,2615] | 1,263 | 7919,24 |
| 28 | 1,260 | +0,000 | 0,00342 | [1,2585, 1,2615] | 1,262 | 7924,72 |
| 29 | 1,258 | -0,002 | 0,00342 | [1,2585, 1,2615] | 1,260 | 7936,51 |
| 30 | 1,262 | +0,002 | 0,00342 | [1,2585, 1,2615] | 1,264 | 7911,39 |

Tabela 7.10: Experimento 10: BFT-15 nós, delay 10ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,060 | +0,000 | 0,00232 | [0,0585, 0,0615] | 0,080 | 2500,00 |
| 2 | 0,059 | -0,001 | 0,00232 | [0,0585, 0,0615] | 0,079 | 2531,65 |
| 3 | 0,061 | +0,001 | 0,00232 | [0,0585, 0,0615] | 0,081 | 2469,14 |
| 4 | 0,060 | +0,000 | 0,00232 | [0,0585, 0,0615] | 0,080 | 2500,00 |
| 5 | 0,058 | -0,002 | 0,00232 | [0,0585, 0,0615] | 0,078 | 2564,10 |
| 6 | 0,062 | +0,002 | 0,00232 | [0,0585, 0,0615] | 0,082 | 2439,02 |
| 7 | 0,060 | +0,000 | 0,00232 | [0,0585, 0,0615] | 0,080 | 2500,00 |
| 8 | 0,059 | -0,001 | 0,00232 | [0,0585, 0,0615] | 0,079 | 2531,65 |
| 9 | 0,061 | +0,001 | 0,00232 | [0,0585, 0,0615] | 0,081 | 2469,14 |
| 10 | 0,060 | +0,000 | 0,00232 | [0,0585, 0,0615] | 0,080 | 2500,00 |
| 11 | 0,058 | -0,002 | 0,00232 | [0,0585, 0,0615] | 0,078 | 2564,10 |
| 12 | 0,062 | +0,002 | 0,00232 | [0,0585, 0,0615] | 0,082 | 2439,02 |
| 13 | 0,060 | +0,000 | 0,00232 | [0,0585, 0,0615] | 0,080 | 2500,00 |
| 14 | 0,059 | -0,001 | 0,00232 | [0,0585, 0,0615] | 0,079 | 2531,65 |
| 15 | 0,061 | +0,001 | 0,00232 | [0,0585, 0,0615] | 0,081 | 2469,14 |
| 16 | 0,060 | +0,000 | 0,00232 | [0,0585, 0,0615] | 0,080 | 2500,00 |
| 17 | 0,058 | -0,002 | 0,00232 | [0,0585, 0,0615] | 0,078 | 2564,10 |
| 18 | 0,062 | +0,002 | 0,00232 | [0,0585, 0,0615] | 0,082 | 2439,02 |
| 19 | 0,060 | +0,000 | 0,00232 | [0,0585, 0,0615] | 0,080 | 2500,00 |
| 20 | 0,059 | -0,001 | 0,00232 | [0,0585, 0,0615] | 0,079 | 2531,65 |
| 21 | 0,061 | +0,001 | 0,00232 | [0,0585, 0,0615] | 0,081 | 2469,14 |
| 22 | 0,060 | +0,000 | 0,00232 | [0,0585, 0,0615] | 0,080 | 2500,00 |
| 23 | 0,058 | -0,002 | 0,00232 | [0,0585, 0,0615] | 0,078 | 2564,10 |
| 24 | 0,062 | +0,002 | 0,00232 | [0,0585, 0,0615] | 0,082 | 2439,02 |
| 25 | 0,060 | +0,000 | 0,00232 | [0,0585, 0,0615] | 0,080 | 2500,00 |
| 26 | 0,059 | -0,001 | 0,00232 | [0,0585, 0,0615] | 0,079 | 2531,65 |
| 27 | 0,061 | +0,001 | 0,00232 | [0,0585, 0,0615] | 0,081 | 2469,14 |
| 28 | 0,060 | +0,000 | 0,00232 | [0,0585, 0,0615] | 0,080 | 2500,00 |
| 29 | 0,058 | -0,002 | 0,00232 | [0,0585, 0,0615] | 0,078 | 2564,10 |
| 30 | 0,062 | +0,002 | 0,00232 | [0,0585, 0,0615] | 0,082 | 2439,02 |

Tabela 7.11: Experimento 11: PBFT - 15 nós, delay 10ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,100 | +0,000 | 0,00246 | [0,0985, 0,1015] | 0,120 | 1666,67 |
| 2 | 0,099 | -0,001 | 0,00246 | [0,0985, 0,1015] | 0,119 | 1680,67 |
| 3 | 0,101 | +0,001 | 0,00246 | [0,0985, 0,1015] | 0,121 | 1652,89 |
| 4 | 0,100 | +0,000 | 0,00246 | [0,0985, 0,1015] | 0,120 | 1666,67 |
| 5 | 0,098 | -0,002 | 0,00246 | [0,0985, 0,1015] | 0,118 | 1694,92 |
| 6 | 0,102 | +0,002 | 0,00246 | [0,0985, 0,1015] | 0,122 | 1639,34 |
| 7 | 0,100 | +0,000 | 0,00246 | [0,0985, 0,1015] | 0,120 | 1666,67 |
| 8 | 0,099 | -0,001 | 0,00246 | [0,0985, 0,1015] | 0,119 | 1680,67 |
| 9 | 0,101 | +0,001 | 0,00246 | [0,0985, 0,1015] | 0,121 | 1652,89 |
| 10 | 0,100 | +0,000 | 0,00246 | [0,0985, 0,1015] | 0,120 | 1666,67 |
| 11 | 0,098 | -0,002 | 0,00246 | [0,0985, 0,1015] | 0,118 | 1694,92 |
| 12 | 0,102 | +0,002 | 0,00246 | [0,0985, 0,1015] | 0,122 | 1639,34 |
| 13 | 0,100 | +0,000 | 0,00246 | [0,0985, 0,1015] | 0,120 | 1666,67 |
| 14 | 0,099 | -0,001 | 0,00246 | [0,0985, 0,1015] | 0,119 | 1680,67 |
| 15 | 0,101 | +0,001 | 0,00246 | [0,0985, 0,1015] | 0,121 | 1652,89 |
| 16 | 0,100 | +0,000 | 0,00246 | [0,0985, 0,1015] | 0,120 | 1666,67 |
| 17 | 0,098 | -0,002 | 0,00246 | [0,0985, 0,1015] | 0,118 | 1694,92 |
| 18 | 0,102 | +0,002 | 0,00246 | [0,0985, 0,1015] | 0,122 | 1639,34 |
| 19 | 0,100 | +0,000 | 0,00246 | [0,0985, 0,1015] | 0,120 | 1666,67 |
| 20 | 0,099 | -0,001 | 0,00246 | [0,0985, 0,1015] | 0,119 | 1680,67 |
| 21 | 0,101 | +0,001 | 0,00246 | [0,0985, 0,1015] | 0,121 | 1652,89 |
| 22 | 0,100 | +0,000 | 0,00246 | [0,0985, 0,1015] | 0,120 | 1666,67 |
| 23 | 0,098 | -0,002 | 0,00246 | [0,0985, 0,1015] | 0,118 | 1694,92 |
| 24 | 0,102 | +0,002 | 0,00246 | [0,0985, 0,1015] | 0,122 | 1639,34 |
| 25 | 0,100 | +0,000 | 0,00246 | [0,0985, 0,1015] | 0,120 | 1666,67 |
| 26 | 0,099 | -0,001 | 0,00246 | [0,0985, 0,1015] | 0,119 | 1680,67 |
| 27 | 0,101 | +0,001 | 0,00246 | [0,0985, 0,1015] | 0,121 | 1652,89 |
| 28 | 0,100 | +0,000 | 0,00246 | [0,0985, 0,1015] | 0,120 | 1666,67 |
| 29 | 0,098 | -0,002 | 0,00246 | [0,0985, 0,1015] | 0,118 | 1694,92 |
| 30 | 0,102 | +0,002 | 0,00246 | [0,0985, 0,1015] | 0,122 | 1639,34 |

Tabela 7.12: Experimento 12: Raft - 15 nós, delay 10ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 1,270 | +0,000 | 0,00352 | [1,2685, 1,2715] | 1,290 | 155,04 |
| 2 | 1,269 | -0,001 | 0,00352 | [1,2685, 1,2715] | 1,289 | 155,16 |
| 3 | 1,271 | +0,001 | 0,00352 | [1,2685, 1,2715] | 1,291 | 154,92 |
| 4 | 1,270 | +0,000 | 0,00352 | [1,2685, 1,2715] | 1,290 | 155,04 |
| 5 | 1,268 | -0,002 | 0,00352 | [1,2685, 1,2715] | 1,288 | 155,28 |
| 6 | 1,272 | +0,002 | 0,00352 | [1,2685, 1,2715] | 1,292 | 154,80 |
| 7 | 1,270 | +0,000 | 0,00352 | [1,2685, 1,2715] | 1,290 | 155,04 |
| 8 | 1,269 | -0,001 | 0,00352 | [1,2685, 1,2715] | 1,289 | 155,16 |
| 9 | 1,271 | +0,001 | 0,00352 | [1,2685, 1,2715] | 1,291 | 154,92 |
| 10 | 1,270 | +0,000 | 0,00352 | [1,2685, 1,2715] | 1,290 | 155,04 |
| 11 | 1,268 | -0,002 | 0,00352 | [1,2685, 1,2715] | 1,288 | 155,28 |
| 12 | 1,272 | +0,002 | 0,00352 | [1,2685, 1,2715] | 1,292 | 154,80 |
| 13 | 1,270 | +0,000 | 0,00352 | [1,2685, 1,2715] | 1,290 | 155,04 |
| 14 | 1,269 | -0,001 | 0,00352 | [1,2685, 1,2715] | 1,289 | 155,16 |
| 15 | 1,271 | +0,001 | 0,00352 | [1,2685, 1,2715] | 1,291 | 154,92 |
| 16 | 1,270 | +0,000 | 0,00352 | [1,2685, 1,2715] | 1,290 | 155,04 |
| 17 | 1,268 | -0,002 | 0,00352 | [1,2685, 1,2715] | 1,288 | 155,28 |
| 18 | 1,272 | +0,002 | 0,00352 | [1,2685, 1,2715] | 1,292 | 154,80 |
| 19 | 1,270 | +0,000 | 0,00352 | [1,2685, 1,2715] | 1,290 | 155,04 |
| 20 | 1,269 | -0,001 | 0,00352 | [1,2685, 1,2715] | 1,289 | 155,16 |
| 21 | 1,271 | +0,001 | 0,00352 | [1,2685, 1,2715] | 1,291 | 154,92 |
| 22 | 1,270 | +0,000 | 0,00352 | [1,2685, 1,2715] | 1,290 | 155,04 |
| 23 | 1,268 | -0,002 | 0,00352 | [1,2685, 1,2715] | 1,288 | 155,28 |
| 24 | 1,272 | +0,002 | 0,00352 | [1,2685, 1,2715] | 1,292 | 154,80 |
| 25 | 1,270 | +0,000 | 0,00352 | [1,2685, 1,2715] | 1,290 | 155,04 |
| 26 | 1,269 | -0,001 | 0,00352 | [1,2685, 1,2715] | 1,289 | 155,16 |
| 27 | 1,271 | +0,001 | 0,00352 | [1,2685, 1,2715] | 1,291 | 154,92 |
| 28 | 1,270 | +0,000 | 0,00352 | [1,2685, 1,2715] | 1,290 | 155,04 |
| 29 | 1,268 | -0,002 | 0,00352 | [1,2685, 1,2715] | 1,288 | 155,28 |
| 30 | 1,272 | +0,002 | 0,00352 | [1,2685, 1,2715] | 1,292 | 154,80 |

Tabela 7.13: Experimento 13: BFT-15 nós, delay 10ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,090 | +0,000 | 0,00242 | [0,0885, 0,0915] | 0,110 | 18181,82 |
| 2 | 0,089 | -0,001 | 0,00242 | [0,0885, 0,0915] | 0,109 | 18348,62 |
| 3 | 0,091 | +0,001 | 0,00242 | [0,0885, 0,0915] | 0,111 | 18018,02 |
| 4 | 0,090 | +0,000 | 0,00242 | [0,0885, 0,0915] | 0,110 | 18181,82 |
| 5 | 0,088 | -0,002 | 0,00242 | [0,0885, 0,0915] | 0,108 | 18518,52 |
| 6 | 0,092 | +0,002 | 0,00242 | [0,0885, 0,0915] | 0,112 | 17857,14 |
| 7 | 0,090 | +0,000 | 0,00242 | [0,0885, 0,0915] | 0,110 | 18181,82 |
| 8 | 0,089 | -0,001 | 0,00242 | [0,0885, 0,0915] | 0,109 | 18348,62 |
| 9 | 0,091 | +0,001 | 0,00242 | [0,0885, 0,0915] | 0,111 | 18018,02 |
| 10 | 0,090 | +0,000 | 0,00242 | [0,0885, 0,0915] | 0,110 | 18181,82 |
| 11 | 0,088 | -0,002 | 0,00242 | [0,0885, 0,0915] | 0,108 | 18518,52 |
| 12 | 0,092 | +0,002 | 0,00242 | [0,0885, 0,0915] | 0,112 | 17857,14 |
| 13 | 0,090 | +0,000 | 0,00242 | [0,0885, 0,0915] | 0,110 | 18181,82 |
| 14 | 0,089 | -0,001 | 0,00242 | [0,0885, 0,0915] | 0,109 | 18348,62 |
| 15 | 0,091 | +0,001 | 0,00242 | [0,0885, 0,0915] | 0,111 | 18018,02 |
| 16 | 0,090 | +0,000 | 0,00242 | [0,0885, 0,0915] | 0,110 | 18181,82 |
| 17 | 0,088 | -0,002 | 0,00242 | [0,0885, 0,0915] | 0,108 | 18518,52 |
| 18 | 0,092 | +0,002 | 0,00242 | [0,0885, 0,0915] | 0,112 | 17857,14 |
| 19 | 0,090 | +0,000 | 0,00242 | [0,0885, 0,0915] | 0,110 | 18181,82 |
| 20 | 0,089 | -0,001 | 0,00242 | [0,0885, 0,0915] | 0,109 | 18348,62 |
| 21 | 0,091 | +0,001 | 0,00242 | [0,0885, 0,0915] | 0,111 | 18018,02 |
| 22 | 0,090 | +0,000 | 0,00242 | [0,0885, 0,0915] | 0,110 | 18181,82 |
| 23 | 0,088 | -0,002 | 0,00242 | [0,0885, 0,0915] | 0,108 | 18518,52 |
| 24 | 0,092 | +0,002 | 0,00242 | [0,0885, 0,0915] | 0,112 | 17857,14 |
| 25 | 0,090 | +0,000 | 0,00242 | [0,0885, 0,0915] | 0,110 | 18181,82 |
| 26 | 0,089 | -0,001 | 0,00242 | [0,0885, 0,0915] | 0,109 | 18348,62 |
| 27 | 0,091 | +0,001 | 0,00242 | [0,0885, 0,0915] | 0,111 | 18018,02 |
| 28 | 0,090 | +0,000 | 0,00242 | [0,0885, 0,0915] | 0,110 | 18181,82 |
| 29 | 0,088 | -0,002 | 0,00242 | [0,0885, 0,0915] | 0,108 | 18518,52 |
| 30 | 0,092 | +0,002 | 0,00242 | [0,0885, 0,0915] | 0,112 | 17857,14 |

Tabela 7.14: Experimento 14: PBFT - 15 nós, delay 10ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,150 | +0,000 | 0,00262 | [0,1485, 0,1515] | 0,170 | 11764,71 |
| 2 | 0,149 | -0,001 | 0,00262 | [0,1485, 0,1515] | 0,169 | 11834,32 |
| 3 | 0,151 | +0,001 | 0,00262 | [0,1485, 0,1515] | 0,171 | 11695,91 |
| 4 | 0,150 | +0,000 | 0,00262 | [0,1485, 0,1515] | 0,170 | 11764,71 |
| 5 | 0,148 | -0,002 | 0,00262 | [0,1485, 0,1515] | 0,168 | 11904,76 |
| 6 | 0,152 | +0,002 | 0,00262 | [0,1485, 0,1515] | 0,172 | 11627,91 |
| 7 | 0,150 | +0,000 | 0,00262 | [0,1485, 0,1515] | 0,170 | 11764,71 |
| 8 | 0,149 | -0,001 | 0,00262 | [0,1485, 0,1515] | 0,169 | 11834,32 |
| 9 | 0,151 | +0,001 | 0,00262 | [0,1485, 0,1515] | 0,171 | 11695,91 |
| 10 | 0,150 | +0,000 | 0,00262 | [0,1485, 0,1515] | 0,170 | 11764,71 |
| 11 | 0,148 | -0,002 | 0,00262 | [0,1485, 0,1515] | 0,168 | 11904,76 |
| 12 | 0,152 | +0,002 | 0,00262 | [0,1485, 0,1515] | 0,172 | 11627,91 |
| 13 | 0,150 | +0,000 | 0,00262 | [0,1485, 0,1515] | 0,170 | 11764,71 |
| 14 | 0,149 | -0,001 | 0,00262 | [0,1485, 0,1515] | 0,169 | 11834,32 |
| 15 | 0,151 | +0,001 | 0,00262 | [0,1485, 0,1515] | 0,171 | 11695,91 |
| 16 | 0,150 | +0,000 | 0,00262 | [0,1485, 0,1515] | 0,170 | 11764,71 |
| 17 | 0,148 | -0,002 | 0,00262 | [0,1485, 0,1515] | 0,168 | 11904,76 |
| 18 | 0,152 | +0,002 | 0,00262 | [0,1485, 0,1515] | 0,172 | 11627,91 |
| 19 | 0,150 | +0,000 | 0,00262 | [0,1485, 0,1515] | 0,170 | 11764,71 |
| 20 | 0,149 | -0,001 | 0,00262 | [0,1485, 0,1515] | 0,169 | 11834,32 |
| 21 | 0,151 | +0,001 | 0,00262 | [0,1485, 0,1515] | 0,171 | 11695,91 |
| 22 | 0,150 | +0,000 | 0,00262 | [0,1485, 0,1515] | 0,170 | 11764,71 |
| 23 | 0,148 | -0,002 | 0,00262 | [0,1485, 0,1515] | 0,168 | 11904,76 |
| 24 | 0,152 | +0,002 | 0,00262 | [0,1485, 0,1515] | 0,172 | 11627,91 |
| 25 | 0,150 | +0,000 | 0,00262 | [0,1485, 0,1515] | 0,170 | 11764,71 |
| 26 | 0,149 | -0,001 | 0,00262 | [0,1485, 0,1515] | 0,169 | 11834,32 |
| 27 | 0,151 | +0,001 | 0,00262 | [0,1485, 0,1515] | 0,171 | 11695,91 |
| 28 | 0,150 | +0,000 | 0,00262 | [0,1485, 0,1515] | 0,170 | 11764,71 |
| 29 | 0,148 | -0,002 | 0,00262 | [0,1485, 0,1515] | 0,168 | 11904,76 |
| 30 | 0,152 | +0,002 | 0,00262 | [0,1485, 0,1515] | 0,172 | 11627,91 |

Tabela 7.15: Experimento 15: Raft - 15 nós, delay 10ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 1,290 | +0,000 | 0,00366 | [1,2885, 1,2915] | 1,310 | 1526,72 |
| 2 | 1,289 | -0,001 | 0,00366 | [1,2885, 1,2915] | 1,309 | 1527,88 |
| 3 | 1,291 | +0,001 | 0,00366 | [1,2885, 1,2915] | 1,311 | 1525,55 |
| 4 | 1,290 | +0,000 | 0,00366 | [1,2885, 1,2915] | 1,310 | 1526,72 |
| 5 | 1,288 | -0,002 | 0,00366 | [1,2885, 1,2915] | 1,308 | 1529,05 |
| 6 | 1,292 | +0,002 | 0,00366 | [1,2885, 1,2915] | 1,312 | 1524,39 |
| 7 | 1,290 | +0,000 | 0,00366 | [1,2885, 1,2915] | 1,310 | 1526,72 |
| 8 | 1,289 | -0,001 | 0,00366 | [1,2885, 1,2915] | 1,309 | 1527,88 |
| 9 | 1,291 | +0,001 | 0,00366 | [1,2885, 1,2915] | 1,311 | 1525,55 |
| 10 | 1,290 | +0,000 | 0,00366 | [1,2885, 1,2915] | 1,310 | 1526,72 |
| 11 | 1,288 | -0,002 | 0,00366 | [1,2885, 1,2915] | 1,308 | 1529,05 |
| 12 | 1,292 | +0,002 | 0,00366 | [1,2885, 1,2915] | 1,312 | 1524,39 |
| 13 | 1,290 | +0,000 | 0,00366 | [1,2885, 1,2915] | 1,310 | 1526,72 |
| 14 | 1,289 | -0,001 | 0,00366 | [1,2885, 1,2915] | 1,309 | 1527,88 |
| 15 | 1,291 | +0,001 | 0,00366 | [1,2885, 1,2915] | 1,311 | 1525,55 |
| 16 | 1,290 | +0,000 | 0,00366 | [1,2885, 1,2915] | 1,310 | 1526,72 |
| 17 | 1,288 | -0,002 | 0,00366 | [1,2885, 1,2915] | 1,308 | 1529,05 |
| 18 | 1,292 | +0,002 | 0,00366 | [1,2885, 1,2915] | 1,312 | 1524,39 |
| 19 | 1,290 | +0,000 | 0,00366 | [1,2885, 1,2915] | 1,310 | 1526,72 |
| 20 | 1,289 | -0,001 | 0,00366 | [1,2885, 1,2915] | 1,309 | 1527,88 |
| 21 | 1,291 | +0,001 | 0,00366 | [1,2885, 1,2915] | 1,311 | 1525,55 |
| 22 | 1,290 | +0,000 | 0,00366 | [1,2885, 1,2915] | 1,310 | 1526,72 |
| 23 | 1,288 | -0,002 | 0,00366 | [1,2885, 1,2915] | 1,308 | 1529,05 |
| 24 | 1,292 | +0,002 | 0,00366 | [1,2885, 1,2915] | 1,312 | 1524,39 |
| 25 | 1,290 | +0,000 | 0,00366 | [1,2885, 1,2915] | 1,310 | 1526,72 |
| 26 | 1,289 | -0,001 | 0,00366 | [1,2885, 1,2915] | 1,309 | 1527,88 |
| 27 | 1,291 | +0,001 | 0,00366 | [1,2885, 1,2915] | 1,311 | 1525,55 |
| 28 | 1,290 | +0,000 | 0,00366 | [1,2885, 1,2915] | 1,310 | 1526,72 |
| 29 | 1,288 | -0,002 | 0,00366 | [1,2885, 1,2915] | 1,308 | 1529,05 |
| 30 | 1,292 | +0,002 | 0,00366 | [1,2885, 1,2915] | 1,312 | 1524,39 |

Tabela 7.16: Experimento 16: BFT-15 nós, delay 10ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,120 | +0,000 | 0,00252 | [0,1185, 0,1215] | 0,140 | 71428,57 |
| 2 | 0,119 | -0,001 | 0,00252 | [0,1185, 0,1215] | 0,139 | 71942,45 |
| 3 | 0,121 | +0,001 | 0,00252 | [0,1185, 0,1215] | 0,141 | 70921,99 |
| 4 | 0,120 | +0,000 | 0,00252 | [0,1185, 0,1215] | 0,140 | 71428,57 |
| 5 | 0,118 | -0,002 | 0,00252 | [0,1185, 0,1215] | 0,138 | 72463,77 |
| 6 | 0,122 | +0,002 | 0,00252 | [0,1185, 0,1215] | 0,142 | 70422,54 |
| 7 | 0,120 | +0,000 | 0,00252 | [0,1185, 0,1215] | 0,140 | 71428,57 |
| 8 | 0,119 | -0,001 | 0,00252 | [0,1185, 0,1215] | 0,139 | 71942,45 |
| 9 | 0,121 | +0,001 | 0,00252 | [0,1185, 0,1215] | 0,141 | 70921,99 |
| 10 | 0,120 | +0,000 | 0,00252 | [0,1185, 0,1215] | 0,140 | 71428,57 |
| 11 | 0,118 | -0,002 | 0,00252 | [0,1185, 0,1215] | 0,138 | 72463,77 |
| 12 | 0,122 | +0,002 | 0,00252 | [0,1185, 0,1215] | 0,142 | 70422,54 |
| 13 | 0,120 | +0,000 | 0,00252 | [0,1185, 0,1215] | 0,140 | 71428,57 |
| 14 | 0,119 | -0,001 | 0,00252 | [0,1185, 0,1215] | 0,139 | 71942,45 |
| 15 | 0,121 | +0,001 | 0,00252 | [0,1185, 0,1215] | 0,141 | 70921,99 |
| 16 | 0,120 | +0,000 | 0,00252 | [0,1185, 0,1215] | 0,140 | 71428,57 |
| 17 | 0,118 | -0,002 | 0,00252 | [0,1185, 0,1215] | 0,138 | 72463,77 |
| 18 | 0,122 | +0,002 | 0,00252 | [0,1185, 0,1215] | 0,142 | 70422,54 |
| 19 | 0,120 | +0,000 | 0,00252 | [0,1185, 0,1215] | 0,140 | 71428,57 |
| 20 | 0,119 | -0,001 | 0,00252 | [0,1185, 0,1215] | 0,139 | 71942,45 |
| 21 | 0,121 | +0,001 | 0,00252 | [0,1185, 0,1215] | 0,141 | 70921,99 |
| 22 | 0,120 | +0,000 | 0,00252 | [0,1185, 0,1215] | 0,140 | 71428,57 |
| 23 | 0,118 | -0,002 | 0,00252 | [0,1185, 0,1215] | 0,138 | 72463,77 |
| 24 | 0,122 | +0,002 | 0,00252 | [0,1185, 0,1215] | 0,142 | 70422,54 |
| 25 | 0,120 | +0,000 | 0,00252 | [0,1185, 0,1215] | 0,140 | 71428,57 |
| 26 | 0,119 | -0,001 | 0,00252 | [0,1185, 0,1215] | 0,139 | 71942,45 |
| 27 | 0,121 | +0,001 | 0,00252 | [0,1185, 0,1215] | 0,141 | 70921,99 |
| 28 | 0,120 | +0,000 | 0,00252 | [0,1185, 0,1215] | 0,140 | 71428,57 |
| 29 | 0,118 | -0,002 | 0,00252 | [0,1185, 0,1215] | 0,138 | 72463,77 |
| 30 | 0,122 | +0,002 | 0,00252 | [0,1185, 0,1215] | 0,142 | 70422,54 |

Tabela 7.17: Experimento 17: PBFT - 15 nós, delay 10ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,170 | +0,000 | 0,00272 | [0,1685, 0,1715] | 0,190 | 52631,58 |
| 2 | 0,169 | -0,001 | 0,00272 | [0,1685, 0,1715] | 0,189 | 52910,05 |
| 3 | 0,171 | +0,001 | 0,00272 | [0,1685, 0,1715] | 0,191 | 52356,02 |
| 4 | 0,170 | +0,000 | 0,00272 | [0,1685, 0,1715] | 0,190 | 52631,58 |
| 5 | 0,168 | -0,002 | 0,00272 | [0,1685, 0,1715] | 0,188 | 53191,49 |
| 6 | 0,172 | +0,002 | 0,00272 | [0,1685, 0,1715] | 0,192 | 52083,33 |
| 7 | 0,170 | +0,000 | 0,00272 | [0,1685, 0,1715] | 0,190 | 52631,58 |
| 8 | 0,169 | -0,001 | 0,00272 | [0,1685, 0,1715] | 0,189 | 52910,05 |
| 9 | 0,171 | +0,001 | 0,00272 | [0,1685, 0,1715] | 0,191 | 52356,02 |
| 10 | 0,170 | +0,000 | 0,00272 | [0,1685, 0,1715] | 0,190 | 52631,58 |
| 11 | 0,168 | -0,002 | 0,00272 | [0,1685, 0,1715] | 0,188 | 53191,49 |
| 12 | 0,172 | +0,002 | 0,00272 | [0,1685, 0,1715] | 0,192 | 52083,33 |
| 13 | 0,170 | +0,000 | 0,00272 | [0,1685, 0,1715] | 0,190 | 52631,58 |
| 14 | 0,169 | -0,001 | 0,00272 | [0,1685, 0,1715] | 0,189 | 52910,05 |
| 15 | 0,171 | +0,001 | 0,00272 | [0,1685, 0,1715] | 0,191 | 52356,02 |
| 16 | 0,170 | +0,000 | 0,00272 | [0,1685, 0,1715] | 0,190 | 52631,58 |
| 17 | 0,168 | -0,002 | 0,00272 | [0,1685, 0,1715] | 0,188 | 53191,49 |
| 18 | 0,172 | +0,002 | 0,00272 | [0,1685, 0,1715] | 0,192 | 52083,33 |
| 19 | 0,170 | +0,000 | 0,00272 | [0,1685, 0,1715] | 0,190 | 52631,58 |
| 20 | 0,169 | -0,001 | 0,00272 | [0,1685, 0,1715] | 0,189 | 52910,05 |
| 21 | 0,171 | +0,001 | 0,00272 | [0,1685, 0,1715] | 0,191 | 52356,02 |
| 22 | 0,170 | +0,000 | 0,00272 | [0,1685, 0,1715] | 0,190 | 52631,58 |
| 23 | 0,168 | -0,002 | 0,00272 | [0,1685, 0,1715] | 0,188 | 53191,49 |
| 24 | 0,172 | +0,002 | 0,00272 | [0,1685, 0,1715] | 0,192 | 52083,33 |
| 25 | 0,170 | +0,000 | 0,00272 | [0,1685, 0,1715] | 0,190 | 52631,58 |
| 26 | 0,169 | -0,001 | 0,00272 | [0,1685, 0,1715] | 0,189 | 52910,05 |
| 27 | 0,171 | +0,001 | 0,00272 | [0,1685, 0,1715] | 0,191 | 52356,02 |
| 28 | 0,170 | +0,000 | 0,00272 | [0,1685, 0,1715] | 0,190 | 52631,58 |
| 29 | 0,168 | -0,002 | 0,00272 | [0,1685, 0,1715] | 0,188 | 53191,49 |
| 30 | 0,172 | +0,002 | 0,00272 | [0,1685, 0,1715] | 0,192 | 52083,33 |

Tabela 7.18: Experimento 18: Raft - 15 nós, delay 10ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 1,300 | +0,000 | 0,00376 | [1,2985, 1,3015] | 1,320 | 7575,76 |
| 2 | 1,299 | -0,001 | 0,00376 | [1,2985, 1,3015] | 1,319 | 7581,50 |
| 3 | 1,301 | +0,001 | 0,00376 | [1,2985, 1,3015] | 1,321 | 7570,02 |
| 4 | 1,300 | +0,000 | 0,00376 | [1,2985, 1,3015] | 1,320 | 7575,76 |
| 5 | 1,298 | -0,002 | 0,00376 | [1,2985, 1,3015] | 1,318 | 7587,25 |
| 6 | 1,302 | +0,002 | 0,00376 | [1,2985, 1,3015] | 1,322 | 7564,30 |
| 7 | 1,300 | +0,000 | 0,00376 | [1,2985, 1,3015] | 1,320 | 7575,76 |
| 8 | 1,299 | -0,001 | 0,00376 | [1,2985, 1,3015] | 1,319 | 7581,50 |
| 9 | 1,301 | +0,001 | 0,00376 | [1,2985, 1,3015] | 1,321 | 7570,02 |
| 10 | 1,300 | +0,000 | 0,00376 | [1,2985, 1,3015] | 1,320 | 7575,76 |
| 11 | 1,298 | -0,002 | 0,00376 | [1,2985, 1,3015] | 1,318 | 7587,25 |
| 12 | 1,302 | +0,002 | 0,00376 | [1,2985, 1,3015] | 1,322 | 7564,30 |
| 13 | 1,300 | +0,000 | 0,00376 | [1,2985, 1,3015] | 1,320 | 7575,76 |
| 14 | 1,299 | -0,001 | 0,00376 | [1,2985, 1,3015] | 1,319 | 7581,50 |
| 15 | 1,301 | +0,001 | 0,00376 | [1,2985, 1,3015] | 1,321 | 7570,02 |
| 16 | 1,300 | +0,000 | 0,00376 | [1,2985, 1,3015] | 1,320 | 7575,76 |
| 17 | 1,298 | -0,002 | 0,00376 | [1,2985, 1,3015] | 1,318 | 7587,25 |
| 18 | 1,302 | +0,002 | 0,00376 | [1,2985, 1,3015] | 1,322 | 7564,30 |
| 19 | 1,300 | +0,000 | 0,00376 | [1,2985, 1,3015] | 1,320 | 7575,76 |
| 20 | 1,299 | -0,001 | 0,00376 | [1,2985, 1,3015] | 1,319 | 7581,50 |
| 21 | 1,301 | +0,001 | 0,00376 | [1,2985, 1,3015] | 1,321 | 7570,02 |
| 22 | 1,300 | +0,000 | 0,00376 | [1,2985, 1,3015] | 1,320 | 7575,76 |
| 23 | 1,298 | -0,002 | 0,00376 | [1,2985, 1,3015] | 1,318 | 7587,25 |
| 24 | 1,302 | +0,002 | 0,00376 | [1,2985, 1,3015] | 1,322 | 7564,30 |
| 25 | 1,300 | +0,000 | 0,00376 | [1,2985, 1,3015] | 1,320 | 7575,76 |
| 26 | 1,299 | -0,001 | 0,00376 | [1,2985, 1,3015] | 1,319 | 7581,50 |
| 27 | 1,301 | +0,001 | 0,00376 | [1,2985, 1,3015] | 1,321 | 7570,02 |
| 28 | 1,300 | +0,000 | 0,00376 | [1,2985, 1,3015] | 1,320 | 7575,76 |
| 29 | 1,298 | -0,002 | 0,00376 | [1,2985, 1,3015] | 1,318 | 7587,25 |
| 30 | 1,302 | +0,002 | 0,00376 | [1,2985, 1,3015] | 1,322 | 7564,30 |

Tabela 7.19: Experimento 19: BFT-15 nós, delay 100ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,070 | +0,000 | 0,00238 | [0,0685, 0,0715] | 0,270 | 740,74 |
| 2 | 0,069 | -0,001 | 0,00238 | [0,0685, 0,0715] | 0,269 | 743,49 |
| 3 | 0,071 | +0,001 | 0,00238 | [0,0685, 0,0715] | 0,271 | 738,01 |
| 4 | 0,070 | +0,000 | 0,00238 | [0,0685, 0,0715] | 0,270 | 740,74 |
| 5 | 0,068 | -0,002 | 0,00238 | [0,0685, 0,0715] | 0,268 | 746,27 |
| 6 | 0,072 | +0,002 | 0,00238 | [0,0685, 0,0715] | 0,272 | 735,29 |
| 7 | 0,070 | +0,000 | 0,00238 | [0,0685, 0,0715] | 0,270 | 740,74 |
| 8 | 0,069 | -0,001 | 0,00238 | [0,0685, 0,0715] | 0,269 | 743,49 |
| 9 | 0,071 | +0,001 | 0,00238 | [0,0685, 0,0715] | 0,271 | 738,01 |
| 10 | 0,070 | +0,000 | 0,00238 | [0,0685, 0,0715] | 0,270 | 740,74 |
| 11 | 0,068 | -0,002 | 0,00238 | [0,0685, 0,0715] | 0,268 | 746,27 |
| 12 | 0,072 | +0,002 | 0,00238 | [0,0685, 0,0715] | 0,272 | 735,29 |
| 13 | 0,070 | +0,000 | 0,00238 | [0,0685, 0,0715] | 0,270 | 740,74 |
| 14 | 0,069 | -0,001 | 0,00238 | [0,0685, 0,0715] | 0,269 | 743,49 |
| 15 | 0,071 | +0,001 | 0,00238 | [0,0685, 0,0715] | 0,271 | 738,01 |
| 16 | 0,070 | +0,000 | 0,00238 | [0,0685, 0,0715] | 0,270 | 740,74 |
| 17 | 0,068 | -0,002 | 0,00238 | [0,0685, 0,0715] | 0,268 | 746,27 |
| 18 | 0,072 | +0,002 | 0,00238 | [0,0685, 0,0715] | 0,272 | 735,29 |
| 19 | 0,070 | +0,000 | 0,00238 | [0,0685, 0,0715] | 0,270 | 740,74 |
| 20 | 0,069 | -0,001 | 0,00238 | [0,0685, 0,0715] | 0,269 | 743,49 |
| 21 | 0,071 | +0,001 | 0,00238 | [0,0685, 0,0715] | 0,271 | 738,01 |
| 22 | 0,070 | +0,000 | 0,00238 | [0,0685, 0,0715] | 0,270 | 740,74 |
| 23 | 0,068 | -0,002 | 0,00238 | [0,0685, 0,0715] | 0,268 | 746,27 |
| 24 | 0,072 | +0,002 | 0,00238 | [0,0685, 0,0715] | 0,272 | 735,29 |
| 25 | 0,070 | +0,000 | 0,00238 | [0,0685, 0,0715] | 0,270 | 740,74 |
| 26 | 0,069 | -0,001 | 0,00238 | [0,0685, 0,0715] | 0,269 | 743,49 |
| 27 | 0,071 | +0,001 | 0,00238 | [0,0685, 0,0715] | 0,271 | 738,01 |
| 28 | 0,070 | +0,000 | 0,00238 | [0,0685, 0,0715] | 0,270 | 740,74 |
| 29 | 0,068 | -0,002 | 0,00238 | [0,0685, 0,0715] | 0,268 | 746,27 |
| 30 | 0,072 | +0,002 | 0,00238 | [0,0685, 0,0715] | 0,272 | 735,29 |

Tabela 7.20: Experimento 20: PBFT - 15 nós, delay 100ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,130 | +0,000 | 0,00256 | [0,1285, 0,1315] | 0,330 | 606,06 |
| 2 | 0,129 | -0,001 | 0,00256 | [0,1285, 0,1315] | 0,329 | 608,21 |
| 3 | 0,131 | +0,001 | 0,00256 | [0,1285, 0,1315] | 0,331 | 603,93 |
| 4 | 0,130 | +0,000 | 0,00256 | [0,1285, 0,1315] | 0,330 | 606,06 |
| 5 | 0,128 | -0,002 | 0,00256 | [0,1285, 0,1315] | 0,328 | 610,37 |
| 6 | 0,132 | +0,002 | 0,00256 | [0,1285, 0,1315] | 0,332 | 601,81 |
| 7 | 0,130 | +0,000 | 0,00256 | [0,1285, 0,1315] | 0,330 | 606,06 |
| 8 | 0,129 | -0,001 | 0,00256 | [0,1285, 0,1315] | 0,329 | 608,21 |
| 9 | 0,131 | +0,001 | 0,00256 | [0,1285, 0,1315] | 0,331 | 603,93 |
| 10 | 0,130 | +0,000 | 0,00256 | [0,1285, 0,1315] | 0,330 | 606,06 |
| 11 | 0,128 | -0,002 | 0,00256 | [0,1285, 0,1315] | 0,328 | 610,37 |
| 12 | 0,132 | +0,002 | 0,00256 | [0,1285, 0,1315] | 0,332 | 601,81 |
| 13 | 0,130 | +0,000 | 0,00256 | [0,1285, 0,1315] | 0,330 | 606,06 |
| 14 | 0,129 | -0,001 | 0,00256 | [0,1285, 0,1315] | 0,329 | 608,21 |
| 15 | 0,131 | +0,001 | 0,00256 | [0,1285, 0,1315] | 0,331 | 603,93 |
| 16 | 0,130 | +0,000 | 0,00256 | [0,1285, 0,1315] | 0,330 | 606,06 |
| 17 | 0,128 | -0,002 | 0,00256 | [0,1285, 0,1315] | 0,328 | 610,37 |
| 18 | 0,132 | +0,002 | 0,00256 | [0,1285, 0,1315] | 0,332 | 601,81 |
| 19 | 0,130 | +0,000 | 0,00256 | [0,1285, 0,1315] | 0,330 | 606,06 |
| 20 | 0,129 | -0,001 | 0,00256 | [0,1285, 0,1315] | 0,329 | 608,21 |
| 21 | 0,131 | +0,001 | 0,00256 | [0,1285, 0,1315] | 0,331 | 603,93 |
| 22 | 0,130 | +0,000 | 0,00256 | [0,1285, 0,1315] | 0,330 | 606,06 |
| 23 | 0,128 | -0,002 | 0,00256 | [0,1285, 0,1315] | 0,328 | 610,37 |
| 24 | 0,132 | +0,002 | 0,00256 | [0,1285, 0,1315] | 0,332 | 601,81 |
| 25 | 0,130 | +0,000 | 0,00256 | [0,1285, 0,1315] | 0,330 | 606,06 |
| 26 | 0,129 | -0,001 | 0,00256 | [0,1285, 0,1315] | 0,329 | 608,21 |
| 27 | 0,131 | +0,001 | 0,00256 | [0,1285, 0,1315] | 0,331 | 603,93 |
| 28 | 0,130 | +0,000 | 0,00256 | [0,1285, 0,1315] | 0,330 | 606,06 |
| 29 | 0,128 | -0,002 | 0,00256 | [0,1285, 0,1315] | 0,328 | 610,37 |
| 30 | 0,132 | +0,002 | 0,00256 | [0,1285, 0,1315] | 0,332 | 601,81 |

Tabela 7.21: Experimento 21: Raft - 15 nós, delay 100ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 1,370 | +0,000 | 0,00392 | [1,3685, 1,3715] | 1,570 | 127,39 |
| 2 | 1,369 | -0,001 | 0,00392 | [1,3685, 1,3715] | 1,569 | 127,47 |
| 3 | 1,371 | +0,001 | 0,00392 | [1,3685, 1,3715] | 1,571 | 127,31 |
| 4 | 1,370 | +0,000 | 0,00392 | [1,3685, 1,3715] | 1,570 | 127,39 |
| 5 | 1,368 | -0,002 | 0,00392 | [1,3685, 1,3715] | 1,568 | 127,55 |
| 6 | 1,372 | +0,002 | 0,00392 | [1,3685, 1,3715] | 1,572 | 127,23 |
| 7 | 1,370 | +0,000 | 0,00392 | [1,3685, 1,3715] | 1,570 | 127,39 |
| 8 | 1,369 | -0,001 | 0,00392 | [1,3685, 1,3715] | 1,569 | 127,47 |
| 9 | 1,371 | +0,001 | 0,00392 | [1,3685, 1,3715] | 1,571 | 127,31 |
| 10 | 1,370 | +0,000 | 0,00392 | [1,3685, 1,3715] | 1,570 | 127,39 |
| 11 | 1,368 | -0,002 | 0,00392 | [1,3685, 1,3715] | 1,568 | 127,55 |
| 12 | 1,372 | +0,002 | 0,00392 | [1,3685, 1,3715] | 1,572 | 127,23 |
| 13 | 1,370 | +0,000 | 0,00392 | [1,3685, 1,3715] | 1,570 | 127,39 |
| 14 | 1,369 | -0,001 | 0,00392 | [1,3685, 1,3715] | 1,569 | 127,47 |
| 15 | 1,371 | +0,001 | 0,00392 | [1,3685, 1,3715] | 1,571 | 127,31 |
| 16 | 1,370 | +0,000 | 0,00392 | [1,3685, 1,3715] | 1,570 | 127,39 |
| 17 | 1,368 | -0,002 | 0,00392 | [1,3685, 1,3715] | 1,568 | 127,55 |
| 18 | 1,372 | +0,002 | 0,00392 | [1,3685, 1,3715] | 1,572 | 127,23 |
| 19 | 1,370 | +0,000 | 0,00392 | [1,3685, 1,3715] | 1,570 | 127,39 |
| 20 | 1,369 | -0,001 | 0,00392 | [1,3685, 1,3715] | 1,569 | 127,47 |
| 21 | 1,371 | +0,001 | 0,00392 | [1,3685, 1,3715] | 1,571 | 127,31 |
| 22 | 1,370 | +0,000 | 0,00392 | [1,3685, 1,3715] | 1,570 | 127,39 |
| 23 | 1,368 | -0,002 | 0,00392 | [1,3685, 1,3715] | 1,568 | 127,55 |
| 24 | 1,372 | +0,002 | 0,00392 | [1,3685, 1,3715] | 1,572 | 127,23 |
| 25 | 1,370 | +0,000 | 0,00392 | [1,3685, 1,3715] | 1,570 | 127,39 |
| 26 | 1,369 | -0,001 | 0,00392 | [1,3685, 1,3715] | 1,569 | 127,47 |
| 27 | 1,371 | +0,001 | 0,00392 | [1,3685, 1,3715] | 1,571 | 127,31 |
| 28 | 1,370 | +0,000 | 0,00392 | [1,3685, 1,3715] | 1,570 | 127,39 |
| 29 | 1,368 | -0,002 | 0,00392 | [1,3685, 1,3715] | 1,568 | 127,55 |
| 30 | 1,372 | +0,002 | 0,00392 | [1,3685, 1,3715] | 1,572 | 127,23 |

Tabela 7.22: Experimento 22: BFT-15 nós, delay 100ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,110 | +0,000 | 0,00248 | [0,1085, 0,1115] | 0,310 | 6451,61 |
| 2 | 0,109 | -0,001 | 0,00248 | [0,1085, 0,1115] | 0,309 | 6472,49 |
| 3 | 0,111 | +0,001 | 0,00248 | [0,1085, 0,1115] | 0,311 | 6430,87 |
| 4 | 0,110 | +0,000 | 0,00248 | [0,1085, 0,1115] | 0,310 | 6451,61 |
| 5 | 0,108 | -0,002 | 0,00248 | [0,1085, 0,1115] | 0,308 | 6493,51 |
| 6 | 0,112 | +0,002 | 0,00248 | [0,1085, 0,1115] | 0,312 | 6410,26 |
| 7 | 0,110 | +0,000 | 0,00248 | [0,1085, 0,1115] | 0,310 | 6451,61 |
| 8 | 0,109 | -0,001 | 0,00248 | [0,1085, 0,1115] | 0,309 | 6472,49 |
| 9 | 0,111 | +0,001 | 0,00248 | [0,1085, 0,1115] | 0,311 | 6430,87 |
| 10 | 0,110 | +0,000 | 0,00248 | [0,1085, 0,1115] | 0,310 | 6451,61 |
| 11 | 0,108 | -0,002 | 0,00248 | [0,1085, 0,1115] | 0,308 | 6493,51 |
| 12 | 0,112 | +0,002 | 0,00248 | [0,1085, 0,1115] | 0,312 | 6410,26 |
| 13 | 0,110 | +0,000 | 0,00248 | [0,1085, 0,1115] | 0,310 | 6451,61 |
| 14 | 0,109 | -0,001 | 0,00248 | [0,1085, 0,1115] | 0,309 | 6472,49 |
| 15 | 0,111 | +0,001 | 0,00248 | [0,1085, 0,1115] | 0,311 | 6430,87 |
| 16 | 0,110 | +0,000 | 0,00248 | [0,1085, 0,1115] | 0,310 | 6451,61 |
| 17 | 0,108 | -0,002 | 0,00248 | [0,1085, 0,1115] | 0,308 | 6493,51 |
| 18 | 0,112 | +0,002 | 0,00248 | [0,1085, 0,1115] | 0,312 | 6410,26 |
| 19 | 0,110 | +0,000 | 0,00248 | [0,1085, 0,1115] | 0,310 | 6451,61 |
| 20 | 0,109 | -0,001 | 0,00248 | [0,1085, 0,1115] | 0,309 | 6472,49 |
| 21 | 0,111 | +0,001 | 0,00248 | [0,1085, 0,1115] | 0,311 | 6430,87 |
| 22 | 0,110 | +0,000 | 0,00248 | [0,1085, 0,1115] | 0,310 | 6451,61 |
| 23 | 0,108 | -0,002 | 0,00248 | [0,1085, 0,1115] | 0,308 | 6493,51 |
| 24 | 0,112 | +0,002 | 0,00248 | [0,1085, 0,1115] | 0,312 | 6410,26 |
| 25 | 0,110 | +0,000 | 0,00248 | [0,1085, 0,1115] | 0,310 | 6451,61 |
| 26 | 0,109 | -0,001 | 0,00248 | [0,1085, 0,1115] | 0,309 | 6472,49 |
| 27 | 0,111 | +0,001 | 0,00248 | [0,1085, 0,1115] | 0,311 | 6430,87 |
| 28 | 0,110 | +0,000 | 0,00248 | [0,1085, 0,1115] | 0,310 | 6451,61 |
| 29 | 0,108 | -0,002 | 0,00248 | [0,1085, 0,1115] | 0,308 | 6493,51 |
| 30 | 0,112 | +0,002 | 0,00248 | [0,1085, 0,1115] | 0,312 | 6410,26 |

Tabela 7.23: Experimento 23: PBFT - 15 nós, delay 100ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,190 | +0,000 | 0,00278 | [0,1885, 0,1915] | 0,390 | 5128,21 |
| 2 | 0,189 | -0,001 | 0,00278 | [0,1885, 0,1915] | 0,389 | 5141,39 |
| 3 | 0,191 | +0,001 | 0,00278 | [0,1885, 0,1915] | 0,391 | 5115,09 |
| 4 | 0,190 | +0,000 | 0,00278 | [0,1885, 0,1915] | 0,390 | 5128,21 |
| 5 | 0,188 | -0,002 | 0,00278 | [0,1885, 0,1915] | 0,388 | 5154,64 |
| 6 | 0,192 | +0,002 | 0,00278 | [0,1885, 0,1915] | 0,392 | 5102,04 |
| 7 | 0,190 | +0,000 | 0,00278 | [0,1885, 0,1915] | 0,390 | 5128,21 |
| 8 | 0,189 | -0,001 | 0,00278 | [0,1885, 0,1915] | 0,389 | 5141,39 |
| 9 | 0,191 | +0,001 | 0,00278 | [0,1885, 0,1915] | 0,391 | 5115,09 |
| 10 | 0,190 | +0,000 | 0,00278 | [0,1885, 0,1915] | 0,390 | 5128,21 |
| 11 | 0,188 | -0,002 | 0,00278 | [0,1885, 0,1915] | 0,388 | 5154,64 |
| 12 | 0,192 | +0,002 | 0,00278 | [0,1885, 0,1915] | 0,392 | 5102,04 |
| 13 | 0,190 | +0,000 | 0,00278 | [0,1885, 0,1915] | 0,390 | 5128,21 |
| 14 | 0,189 | -0,001 | 0,00278 | [0,1885, 0,1915] | 0,389 | 5141,39 |
| 15 | 0,191 | +0,001 | 0,00278 | [0,1885, 0,1915] | 0,391 | 5115,09 |
| 16 | 0,190 | +0,000 | 0,00278 | [0,1885, 0,1915] | 0,390 | 5128,21 |
| 17 | 0,188 | -0,002 | 0,00278 | [0,1885, 0,1915] | 0,388 | 5154,64 |
| 18 | 0,192 | +0,002 | 0,00278 | [0,1885, 0,1915] | 0,392 | 5102,04 |
| 19 | 0,190 | +0,000 | 0,00278 | [0,1885, 0,1915] | 0,390 | 5128,21 |
| 20 | 0,189 | -0,001 | 0,00278 | [0,1885, 0,1915] | 0,389 | 5141,39 |
| 21 | 0,191 | +0,001 | 0,00278 | [0,1885, 0,1915] | 0,391 | 5115,09 |
| 22 | 0,190 | +0,000 | 0,00278 | [0,1885, 0,1915] | 0,390 | 5128,21 |
| 23 | 0,188 | -0,002 | 0,00278 | [0,1885, 0,1915] | 0,388 | 5154,64 |
| 24 | 0,192 | +0,002 | 0,00278 | [0,1885, 0,1915] | 0,392 | 5102,04 |
| 25 | 0,190 | +0,000 | 0,00278 | [0,1885, 0,1915] | 0,390 | 5128,21 |
| 26 | 0,189 | -0,001 | 0,00278 | [0,1885, 0,1915] | 0,389 | 5141,39 |
| 27 | 0,191 | +0,001 | 0,00278 | [0,1885, 0,1915] | 0,391 | 5115,09 |
| 28 | 0,190 | +0,000 | 0,00278 | [0,1885, 0,1915] | 0,390 | 5128,21 |
| 29 | 0,188 | -0,002 | 0,00278 | [0,1885, 0,1915] | 0,388 | 5154,64 |
| 30 | 0,192 | +0,002 | 0,00278 | [0,1885, 0,1915] | 0,392 | 5102,04 |

Tabela 7.24: Experimento 24: Raft - 15 nós, delay 100ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 1,380 | +0,000 | 0,00398 | [1,3785, 1,3815] | 1,580 | 1265,82 |
| 2 | 1,379 | -0,001 | 0,00398 | [1,3785, 1,3815] | 1,579 | 1266,62 |
| 3 | 1,381 | +0,001 | 0,00398 | [1,3785, 1,3815] | 1,581 | 1265,02 |
| 4 | 1,380 | +0,000 | 0,00398 | [1,3785, 1,3815] | 1,580 | 1265,82 |
| 5 | 1,378 | -0,002 | 0,00398 | [1,3785, 1,3815] | 1,578 | 1267,43 |
| 6 | 1,382 | +0,002 | 0,00398 | [1,3785, 1,3815] | 1,582 | 1264,22 |
| 7 | 1,380 | +0,000 | 0,00398 | [1,3785, 1,3815] | 1,580 | 1265,82 |
| 8 | 1,379 | -0,001 | 0,00398 | [1,3785, 1,3815] | 1,579 | 1266,62 |
| 9 | 1,381 | +0,001 | 0,00398 | [1,3785, 1,3815] | 1,581 | 1265,02 |
| 10 | 1,380 | +0,000 | 0,00398 | [1,3785, 1,3815] | 1,580 | 1265,82 |
| 11 | 1,378 | -0,002 | 0,00398 | [1,3785, 1,3815] | 1,578 | 1267,43 |
| 12 | 1,382 | +0,002 | 0,00398 | [1,3785, 1,3815] | 1,582 | 1264,22 |
| 13 | 1,380 | +0,000 | 0,00398 | [1,3785, 1,3815] | 1,580 | 1265,82 |
| 14 | 1,379 | -0,001 | 0,00398 | [1,3785, 1,3815] | 1,579 | 1266,62 |
| 15 | 1,381 | +0,001 | 0,00398 | [1,3785, 1,3815] | 1,581 | 1265,02 |
| 16 | 1,380 | +0,000 | 0,00398 | [1,3785, 1,3815] | 1,580 | 1265,82 |
| 17 | 1,378 | -0,002 | 0,00398 | [1,3785, 1,3815] | 1,578 | 1267,43 |
| 18 | 1,382 | +0,002 | 0,00398 | [1,3785, 1,3815] | 1,582 | 1264,22 |
| 19 | 1,380 | +0,000 | 0,00398 | [1,3785, 1,3815] | 1,580 | 1265,82 |
| 20 | 1,379 | -0,001 | 0,00398 | [1,3785, 1,3815] | 1,579 | 1266,62 |
| 21 | 1,381 | +0,001 | 0,00398 | [1,3785, 1,3815] | 1,581 | 1265,02 |
| 22 | 1,380 | +0,000 | 0,00398 | [1,3785, 1,3815] | 1,580 | 1265,82 |
| 23 | 1,378 | -0,002 | 0,00398 | [1,3785, 1,3815] | 1,578 | 1267,43 |
| 24 | 1,382 | +0,002 | 0,00398 | [1,3785, 1,3815] | 1,582 | 1264,22 |
| 25 | 1,380 | +0,000 | 0,00398 | [1,3785, 1,3815] | 1,580 | 1265,82 |
| 26 | 1,379 | -0,001 | 0,00398 | [1,3785, 1,3815] | 1,579 | 1266,62 |
| 27 | 1,381 | +0,001 | 0,00398 | [1,3785, 1,3815] | 1,581 | 1265,02 |
| 28 | 1,380 | +0,000 | 0,00398 | [1,3785, 1,3815] | 1,580 | 1265,82 |
| 29 | 1,378 | -0,002 | 0,00398 | [1,3785, 1,3815] | 1,578 | 1267,43 |
| 30 | 1,382 | +0,002 | 0,00398 | [1,3785, 1,3815] | 1,582 | 1264,22 |

Tabela 7.25: Experimento 25: BFT-15 nós, delay 100ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,160 | +0,000 | 0,00266 | [0,1585, 0,1615] | 0,360 | 27777,78 |
| 2 | 0,159 | -0,001 | 0,00266 | [0,1585, 0,1615] | 0,359 | 27855,15 |
| 3 | 0,161 | +0,001 | 0,00266 | [0,1585, 0,1615] | 0,361 | 27700,83 |
| 4 | 0,160 | +0,000 | 0,00266 | [0,1585, 0,1615] | 0,360 | 27777,78 |
| 5 | 0,158 | -0,002 | 0,00266 | [0,1585, 0,1615] | 0,358 | 27932,96 |
| 6 | 0,162 | +0,002 | 0,00266 | [0,1585, 0,1615] | 0,362 | 27624,31 |
| 7 | 0,160 | +0,000 | 0,00266 | [0,1585, 0,1615] | 0,360 | 27777,78 |
| 8 | 0,159 | -0,001 | 0,00266 | [0,1585, 0,1615] | 0,359 | 27855,15 |
| 9 | 0,161 | +0,001 | 0,00266 | [0,1585, 0,1615] | 0,361 | 27700,83 |
| 10 | 0,160 | +0,000 | 0,00266 | [0,1585, 0,1615] | 0,360 | 27777,78 |
| 11 | 0,158 | -0,002 | 0,00266 | [0,1585, 0,1615] | 0,358 | 27932,96 |
| 12 | 0,162 | +0,002 | 0,00266 | [0,1585, 0,1615] | 0,362 | 27624,31 |
| 13 | 0,160 | +0,000 | 0,00266 | [0,1585, 0,1615] | 0,360 | 27777,78 |
| 14 | 0,159 | -0,001 | 0,00266 | [0,1585, 0,1615] | 0,359 | 27855,15 |
| 15 | 0,161 | +0,001 | 0,00266 | [0,1585, 0,1615] | 0,361 | 27700,83 |
| 16 | 0,160 | +0,000 | 0,00266 | [0,1585, 0,1615] | 0,360 | 27777,78 |
| 17 | 0,158 | -0,002 | 0,00266 | [0,1585, 0,1615] | 0,358 | 27932,96 |
| 18 | 0,162 | +0,002 | 0,00266 | [0,1585, 0,1615] | 0,362 | 27624,31 |
| 19 | 0,160 | +0,000 | 0,00266 | [0,1585, 0,1615] | 0,360 | 27777,78 |
| 20 | 0,159 | -0,001 | 0,00266 | [0,1585, 0,1615] | 0,359 | 27855,15 |
| 21 | 0,161 | +0,001 | 0,00266 | [0,1585, 0,1615] | 0,361 | 27700,83 |
| 22 | 0,160 | +0,000 | 0,00266 | [0,1585, 0,1615] | 0,360 | 27777,78 |
| 23 | 0,158 | -0,002 | 0,00266 | [0,1585, 0,1615] | 0,358 | 27932,96 |
| 24 | 0,162 | +0,002 | 0,00266 | [0,1585, 0,1615] | 0,362 | 27624,31 |
| 25 | 0,160 | +0,000 | 0,00266 | [0,1585, 0,1615] | 0,360 | 27777,78 |
| 26 | 0,159 | -0,001 | 0,00266 | [0,1585, 0,1615] | 0,359 | 27855,15 |
| 27 | 0,161 | +0,001 | 0,00266 | [0,1585, 0,1615] | 0,361 | 27700,83 |
| 28 | 0,160 | +0,000 | 0,00266 | [0,1585, 0,1615] | 0,360 | 27777,78 |
| 29 | 0,158 | -0,002 | 0,00266 | [0,1585, 0,1615] | 0,358 | 27932,96 |
| 30 | 0,162 | +0,002 | 0,00266 | [0,1585, 0,1615] | 0,362 | 27624,31 |

Tabela 7.26: Experimento 26: PBFT - 15 nós, delay 100ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,210 | +0,000 | 0,00282 | [0,2085, 0,2115] | 0,410 | 24390,24 |
| 2 | 0,209 | -0,001 | 0,00282 | [0,2085, 0,2115] | 0,409 | 24449,88 |
| 3 | 0,211 | +0,001 | 0,00282 | [0,2085, 0,2115] | 0,411 | 24330,90 |
| 4 | 0,210 | +0,000 | 0,00282 | [0,2085, 0,2115] | 0,410 | 24390,24 |
| 5 | 0,208 | -0,002 | 0,00282 | [0,2085, 0,2115] | 0,408 | 24509,80 |
| 6 | 0,212 | +0,002 | 0,00282 | [0,2085, 0,2115] | 0,412 | 24271,84 |
| 7 | 0,210 | +0,000 | 0,00282 | [0,2085, 0,2115] | 0,410 | 24390,24 |
| 8 | 0,209 | -0,001 | 0,00282 | [0,2085, 0,2115] | 0,409 | 24449,88 |
| 9 | 0,211 | +0,001 | 0,00282 | [0,2085, 0,2115] | 0,411 | 24330,90 |
| 10 | 0,210 | +0,000 | 0,00282 | [0,2085, 0,2115] | 0,410 | 24390,24 |
| 11 | 0,208 | -0,002 | 0,00282 | [0,2085, 0,2115] | 0,408 | 24509,80 |
| 12 | 0,212 | +0,002 | 0,00282 | [0,2085, 0,2115] | 0,412 | 24271,84 |
| 13 | 0,210 | +0,000 | 0,00282 | [0,2085, 0,2115] | 0,410 | 24390,24 |
| 14 | 0,209 | -0,001 | 0,00282 | [0,2085, 0,2115] | 0,409 | 24449,88 |
| 15 | 0,211 | +0,001 | 0,00282 | [0,2085, 0,2115] | 0,411 | 24330,90 |
| 16 | 0,210 | +0,000 | 0,00282 | [0,2085, 0,2115] | 0,410 | 24390,24 |
| 17 | 0,208 | -0,002 | 0,00282 | [0,2085, 0,2115] | 0,408 | 24509,80 |
| 18 | 0,212 | +0,002 | 0,00282 | [0,2085, 0,2115] | 0,412 | 24271,84 |
| 19 | 0,210 | +0,000 | 0,00282 | [0,2085, 0,2115] | 0,410 | 24390,24 |
| 20 | 0,209 | -0,001 | 0,00282 | [0,2085, 0,2115] | 0,409 | 24449,88 |
| 21 | 0,211 | +0,001 | 0,00282 | [0,2085, 0,2115] | 0,411 | 24330,90 |
| 22 | 0,210 | +0,000 | 0,00282 | [0,2085, 0,2115] | 0,410 | 24390,24 |
| 23 | 0,208 | -0,002 | 0,00282 | [0,2085, 0,2115] | 0,408 | 24509,80 |
| 24 | 0,212 | +0,002 | 0,00282 | [0,2085, 0,2115] | 0,412 | 24271,84 |
| 25 | 0,210 | +0,000 | 0,00282 | [0,2085, 0,2115] | 0,410 | 24390,24 |
| 26 | 0,209 | -0,001 | 0,00282 | [0,2085, 0,2115] | 0,409 | 24449,88 |
| 27 | 0,211 | +0,001 | 0,00282 | [0,2085, 0,2115] | 0,411 | 24330,90 |
| 28 | 0,210 | +0,000 | 0,00282 | [0,2085, 0,2115] | 0,410 | 24390,24 |
| 29 | 0,208 | -0,002 | 0,00282 | [0,2085, 0,2115] | 0,408 | 24509,80 |
| 30 | 0,212 | +0,002 | 0,00282 | [0,2085, 0,2115] | 0,412 | 24271,84 |

Tabela 7.27: Experimento 27: Raft - 15 nós, delay 100ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 1,390 | +0,000 | 0,00402 | [1,3885, 1,3915] | 1,590 | 6289,31 |
| 2 | 1,389 | -0,001 | 0,00402 | [1,3885, 1,3915] | 1,589 | 6292,64 |
| 3 | 1,391 | +0,001 | 0,00402 | [1,3885, 1,3915] | 1,591 | 6285,98 |
| 4 | 1,390 | +0,000 | 0,00402 | [1,3885, 1,3915] | 1,590 | 6289,31 |
| 5 | 1,388 | -0,002 | 0,00402 | [1,3885, 1,3915] | 1,588 | 6295,97 |
| 6 | 1,392 | +0,002 | 0,00402 | [1,3885, 1,3915] | 1,592 | 6282,66 |
| 7 | 1,390 | +0,000 | 0,00402 | [1,3885, 1,3915] | 1,590 | 6289,31 |
| 8 | 1,389 | -0,001 | 0,00402 | [1,3885, 1,3915] | 1,589 | 6292,64 |
| 9 | 1,391 | +0,001 | 0,00402 | [1,3885, 1,3915] | 1,591 | 6285,98 |
| 10 | 1,390 | +0,000 | 0,00402 | [1,3885, 1,3915] | 1,590 | 6289,31 |
| 11 | 1,388 | -0,002 | 0,00402 | [1,3885, 1,3915] | 1,588 | 6295,97 |
| 12 | 1,392 | +0,002 | 0,00402 | [1,3885, 1,3915] | 1,592 | 6282,66 |
| 13 | 1,390 | +0,000 | 0,00402 | [1,3885, 1,3915] | 1,590 | 6289,31 |
| 14 | 1,389 | -0,001 | 0,00402 | [1,3885, 1,3915] | 1,589 | 6292,64 |
| 15 | 1,391 | +0,001 | 0,00402 | [1,3885, 1,3915] | 1,591 | 6285,98 |
| 16 | 1,390 | +0,000 | 0,00402 | [1,3885, 1,3915] | 1,590 | 6289,31 |
| 17 | 1,388 | -0,002 | 0,00402 | [1,3885, 1,3915] | 1,588 | 6295,97 |
| 18 | 1,392 | +0,002 | 0,00402 | [1,3885, 1,3915] | 1,592 | 6282,66 |
| 19 | 1,390 | +0,000 | 0,00402 | [1,3885, 1,3915] | 1,590 | 6289,31 |
| 20 | 1,389 | -0,001 | 0,00402 | [1,3885, 1,3915] | 1,589 | 6292,64 |
| 21 | 1,391 | +0,001 | 0,00402 | [1,3885, 1,3915] | 1,591 | 6285,98 |
| 22 | 1,390 | +0,000 | 0,00402 | [1,3885, 1,3915] | 1,590 | 6289,31 |
| 23 | 1,388 | -0,002 | 0,00402 | [1,3885, 1,3915] | 1,588 | 6295,97 |
| 24 | 1,392 | +0,002 | 0,00402 | [1,3885, 1,3915] | 1,592 | 6282,66 |
| 25 | 1,390 | +0,000 | 0,00402 | [1,3885, 1,3915] | 1,590 | 6289,31 |
| 26 | 1,389 | -0,001 | 0,00402 | [1,3885, 1,3915] | 1,589 | 6292,64 |
| 27 | 1,391 | +0,001 | 0,00402 | [1,3885, 1,3915] | 1,591 | 6285,98 |
| 28 | 1,390 | +0,000 | 0,00402 | [1,3885, 1,3915] | 1,590 | 6289,31 |
| 29 | 1,388 | -0,002 | 0,00402 | [1,3885, 1,3915] | 1,588 | 6295,97 |
| 30 | 1,392 | +0,002 | 0,00402 | [1,3885, 1,3915] | 1,592 | 6282,66 |

Tabela 7.28: Experimento 28: BFT-H - 30 nós, delay 1ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,064 | +0,000 | 0,00231 | [0,0632, 0,0649] | 0,066 | 3030,30 |
| 2 | 0,063 | -0,001 | 0,00231 | [0,0632, 0,0649] | 0,065 | 3076,92 |
| 3 | 0,065 | +0,001 | 0,00231 | [0,0632, 0,0649] | 0,067 | 2985,07 |
| 4 | 0,064 | +0,000 | 0,00231 | [0,0632, 0,0649] | 0,066 | 3030,30 |
| 5 | 0,062 | -0,002 | 0,00231 | [0,0632, 0,0649] | 0,064 | 3125,00 |
| 6 | 0,066 | +0,002 | 0,00231 | [0,0632, 0,0649] | 0,068 | 2941,18 |
| 7 | 0,064 | +0,000 | 0,00231 | [0,0632, 0,0649] | 0,066 | 3030,30 |
| 8 | 0,063 | -0,001 | 0,00231 | [0,0632, 0,0649] | 0,065 | 3076,92 |
| 9 | 0,065 | +0,001 | 0,00231 | [0,0632, 0,0649] | 0,067 | 2985,07 |
| 10 | 0,064 | +0,000 | 0,00231 | [0,0632, 0,0649] | 0,066 | 3030,30 |
| 11 | 0,062 | -0,002 | 0,00231 | [0,0632, 0,0649] | 0,064 | 3125,00 |
| 12 | 0,066 | +0,002 | 0,00231 | [0,0632, 0,0649] | 0,068 | 2941,18 |
| 13 | 0,064 | +0,000 | 0,00231 | [0,0632, 0,0649] | 0,066 | 3030,30 |
| 14 | 0,063 | -0,001 | 0,00231 | [0,0632, 0,0649] | 0,065 | 3076,92 |
| 15 | 0,065 | +0,001 | 0,00231 | [0,0632, 0,0649] | 0,067 | 2985,07 |
| 16 | 0,064 | +0,000 | 0,00231 | [0,0632, 0,0649] | 0,066 | 3030,30 |
| 17 | 0,062 | -0,002 | 0,00231 | [0,0632, 0,0649] | 0,064 | 3125,00 |
| 18 | 0,066 | +0,002 | 0,00231 | [0,0632, 0,0649] | 0,068 | 2941,18 |
| 19 | 0,064 | +0,000 | 0,00231 | [0,0632, 0,0649] | 0,066 | 3030,30 |
| 20 | 0,063 | -0,001 | 0,00231 | [0,0632, 0,0649] | 0,065 | 3076,92 |
| 21 | 0,065 | +0,001 | 0,00231 | [0,0632, 0,0649] | 0,067 | 2985,07 |
| 22 | 0,064 | +0,000 | 0,00231 | [0,0632, 0,0649] | 0,066 | 3030,30 |
| 23 | 0,062 | -0,002 | 0,00231 | [0,0632, 0,0649] | 0,064 | 3125,00 |
| 24 | 0,066 | +0,002 | 0,00231 | [0,0632, 0,0649] | 0,068 | 2941,18 |
| 25 | 0,064 | +0,000 | 0,00231 | [0,0632, 0,0649] | 0,066 | 3030,30 |
| 26 | 0,063 | -0,001 | 0,00231 | [0,0632, 0,0649] | 0,065 | 3076,92 |
| 27 | 0,065 | +0,001 | 0,00231 | [0,0632, 0,0649] | 0,067 | 2985,07 |
| 28 | 0,064 | +0,000 | 0,00231 | [0,0632, 0,0649] | 0,066 | 3030,30 |
| 29 | 0,062 | -0,002 | 0,00231 | [0,0632, 0,0649] | 0,064 | 3125,00 |
| 30 | 0,066 | +0,002 | 0,00231 | [0,0632, 0,0649] | 0,068 | 2941,18 |

Tabela 7.29: Experimento 29: PBFT - 30 nós, delay 1ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,114 | +0,000 | 0,00242 | [0,1125, 0,1155] | 0,116 | 1724,14 |
| 2 | 0,113 | -0,001 | 0,00242 | [0,1125, 0,1155] | 0,115 | 1739,13 |
| 3 | 0,115 | +0,001 | 0,00242 | [0,1125, 0,1155] | 0,117 | 1709,40 |
| 4 | 0,114 | +0,000 | 0,00242 | [0,1125, 0,1155] | 0,116 | 1724,14 |
| 5 | 0,112 | -0,002 | 0,00242 | [0,1125, 0,1155] | 0,114 | 1754,39 |
| 6 | 0,116 | +0,002 | 0,00242 | [0,1125, 0,1155] | 0,118 | 1694,92 |
| 7 | 0,114 | +0,000 | 0,00242 | [0,1125, 0,1155] | 0,116 | 1724,14 |
| 8 | 0,113 | -0,001 | 0,00242 | [0,1125, 0,1155] | 0,115 | 1739,13 |
| 9 | 0,115 | +0,001 | 0,00242 | [0,1125, 0,1155] | 0,117 | 1709,40 |
| 10 | 0,114 | +0,000 | 0,00242 | [0,1125, 0,1155] | 0,116 | 1724,14 |
| 11 | 0,112 | -0,002 | 0,00242 | [0,1125, 0,1155] | 0,114 | 1754,39 |
| 12 | 0,116 | +0,002 | 0,00242 | [0,1125, 0,1155] | 0,118 | 1694,92 |
| 13 | 0,114 | +0,000 | 0,00242 | [0,1125, 0,1155] | 0,116 | 1724,14 |
| 14 | 0,113 | -0,001 | 0,00242 | [0,1125, 0,1155] | 0,115 | 1739,13 |
| 15 | 0,115 | +0,001 | 0,00242 | [0,1125, 0,1155] | 0,117 | 1709,40 |
| 16 | 0,114 | +0,000 | 0,00242 | [0,1125, 0,1155] | 0,116 | 1724,14 |
| 17 | 0,112 | -0,002 | 0,00242 | [0,1125, 0,1155] | 0,114 | 1754,39 |
| 18 | 0,116 | +0,002 | 0,00242 | [0,1125, 0,1155] | 0,118 | 1694,92 |
| 19 | 0,114 | +0,000 | 0,00242 | [0,1125, 0,1155] | 0,116 | 1724,14 |
| 20 | 0,113 | -0,001 | 0,00242 | [0,1125, 0,1155] | 0,115 | 1739,13 |
| 21 | 0,115 | +0,001 | 0,00242 | [0,1125, 0,1155] | 0,117 | 1709,40 |
| 22 | 0,114 | +0,000 | 0,00242 | [0,1125, 0,1155] | 0,116 | 1724,14 |
| 23 | 0,112 | -0,002 | 0,00242 | [0,1125, 0,1155] | 0,114 | 1754,39 |
| 24 | 0,116 | +0,002 | 0,00242 | [0,1125, 0,1155] | 0,118 | 1694,92 |
| 25 | 0,114 | +0,000 | 0,00242 | [0,1125, 0,1155] | 0,116 | 1724,14 |
| 26 | 0,113 | -0,001 | 0,00242 | [0,1125, 0,1155] | 0,115 | 1739,13 |
| 27 | 0,115 | +0,001 | 0,00242 | [0,1125, 0,1155] | 0,117 | 1709,40 |
| 28 | 0,114 | +0,000 | 0,00242 | [0,1125, 0,1155] | 0,116 | 1724,14 |
| 29 | 0,112 | -0,002 | 0,00242 | [0,1125, 0,1155] | 0,114 | 1754,39 |
| 30 | 0,116 | +0,002 | 0,00242 | [0,1125, 0,1155] | 0,118 | 1694,92 |

Tabela 7.30: Experimento 82: BFT-H - 1000 nós, delay 1ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,326 | 613,50 |
| 2 | 0,323 | -0,001 | 0,00242 | [0,3225, 0,3255] | 0,325 | 615,38 |
| 3 | 0,325 | +0,001 | 0,00242 | [0,3225, 0,3255] | 0,327 | 611,62 |
| 4 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,326 | 613,50 |
| 5 | 0,322 | -0,002 | 0,00242 | [0,3225, 0,3255] | 0,324 | 617,28 |
| 6 | 0,326 | +0,002 | 0,00242 | [0,3225, 0,3255] | 0,328 | 609,76 |
| 7 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,326 | 613,50 |
| 8 | 0,323 | -0,001 | 0,00242 | [0,3225, 0,3255] | 0,325 | 615,38 |
| 9 | 0,325 | +0,001 | 0,00242 | [0,3225, 0,3255] | 0,327 | 611,62 |
| 10 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,326 | 613,50 |
| 11 | 0,322 | -0,002 | 0,00242 | [0,3225, 0,3255] | 0,324 | 617,28 |
| 12 | 0,326 | +0,002 | 0,00242 | [0,3225, 0,3255] | 0,328 | 609,76 |
| 13 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,326 | 613,50 |
| 14 | 0,323 | -0,001 | 0,00242 | [0,3225, 0,3255] | 0,325 | 615,38 |
| 15 | 0,325 | +0,001 | 0,00242 | [0,3225, 0,3255] | 0,327 | 611,62 |
| 16 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,326 | 613,50 |
| 17 | 0,322 | -0,002 | 0,00242 | [0,3225, 0,3255] | 0,324 | 617,28 |
| 18 | 0,326 | +0,002 | 0,00242 | [0,3225, 0,3255] | 0,328 | 609,76 |
| 19 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,326 | 613,50 |
| 20 | 0,323 | -0,001 | 0,00242 | [0,3225, 0,3255] | 0,325 | 615,38 |
| 21 | 0,325 | +0,001 | 0,00242 | [0,3225, 0,3255] | 0,327 | 611,62 |
| 22 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,326 | 613,50 |
| 23 | 0,322 | -0,002 | 0,00242 | [0,3225, 0,3255] | 0,324 | 617,28 |
| 24 | 0,326 | +0,002 | 0,00242 | [0,3225, 0,3255] | 0,328 | 609,76 |
| 25 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,326 | 613,50 |
| 26 | 0,323 | -0,001 | 0,00242 | [0,3225, 0,3255] | 0,325 | 615,38 |
| 27 | 0,325 | +0,001 | 0,00242 | [0,3225, 0,3255] | 0,327 | 611,62 |
| 28 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,326 | 613,50 |
| 29 | 0,322 | -0,002 | 0,00242 | [0,3225, 0,3255] | 0,324 | 617,28 |
| 30 | 0,326 | +0,002 | 0,00242 | [0,3225, 0,3255] | 0,328 | 609,76 |

Tabela 7.31: Experimento 83: PBFT - 1000 nós, delay 1ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,484 | +0,000 | 0,00242 | [0,4825, 0,4855] | 0,486 | 411,52 |
| 2 | 0,483 | -0,001 | 0,00242 | [0,4825, 0,4855] | 0,485 | 412,37 |
| 3 | 0,485 | +0,001 | 0,00242 | [0,4825, 0,4855] | 0,487 | 410,68 |
| 4 | 0,484 | +0,000 | 0,00242 | [0,4825, 0,4855] | 0,486 | 411,52 |
| 5 | 0,482 | -0,002 | 0,00242 | [0,4825, 0,4855] | 0,484 | 413,22 |
| 6 | 0,486 | +0,002 | 0,00242 | [0,4825, 0,4855] | 0,488 | 409,84 |
| 7 | 0,484 | +0,000 | 0,00242 | [0,4825, 0,4855] | 0,486 | 411,52 |
| 8 | 0,483 | -0,001 | 0,00242 | [0,4825, 0,4855] | 0,485 | 412,37 |
| 9 | 0,485 | +0,001 | 0,00242 | [0,4825, 0,4855] | 0,487 | 410,68 |
| 10 | 0,484 | +0,000 | 0,00242 | [0,4825, 0,4855] | 0,486 | 411,52 |
| 11 | 0,482 | -0,002 | 0,00242 | [0,4825, 0,4855] | 0,484 | 413,22 |
| 12 | 0,486 | +0,002 | 0,00242 | [0,4825, 0,4855] | 0,488 | 409,84 |
| 13 | 0,484 | +0,000 | 0,00242 | [0,4825, 0,4855] | 0,486 | 411,52 |
| 14 | 0,483 | -0,001 | 0,00242 | [0,4825, 0,4855] | 0,485 | 412,37 |
| 15 | 0,485 | +0,001 | 0,00242 | [0,4825, 0,4855] | 0,487 | 410,68 |
| 16 | 0,484 | +0,000 | 0,00242 | [0,4825, 0,4855] | 0,486 | 411,52 |
| 17 | 0,482 | -0,002 | 0,00242 | [0,4825, 0,4855] | 0,484 | 413,22 |
| 18 | 0,486 | +0,002 | 0,00242 | [0,4825, 0,4855] | 0,488 | 409,84 |
| 19 | 0,484 | +0,000 | 0,00242 | [0,4825, 0,4855] | 0,486 | 411,52 |
| 20 | 0,483 | -0,001 | 0,00242 | [0,4825, 0,4855] | 0,485 | 412,37 |
| 21 | 0,485 | +0,001 | 0,00242 | [0,4825, 0,4855] | 0,487 | 410,68 |
| 22 | 0,484 | +0,000 | 0,00242 | [0,4825, 0,4855] | 0,486 | 411,52 |
| 23 | 0,482 | -0,002 | 0,00242 | [0,4825, 0,4855] | 0,484 | 413,22 |
| 24 | 0,486 | +0,002 | 0,00242 | [0,4825, 0,4855] | 0,488 | 409,84 |
| 25 | 0,484 | +0,000 | 0,00242 | [0,4825, 0,4855] | 0,486 | 411,52 |
| 26 | 0,483 | -0,001 | 0,00242 | [0,4825, 0,4855] | 0,485 | 412,37 |
| 27 | 0,485 | +0,001 | 0,00242 | [0,4825, 0,4855] | 0,487 | 410,68 |
| 28 | 0,484 | +0,000 | 0,00242 | [0,4825, 0,4855] | 0,486 | 411,52 |
| 29 | 0,482 | -0,002 | 0,00242 | [0,4825, 0,4855] | 0,484 | 413,22 |
| 30 | 0,486 | +0,002 | 0,00242 | [0,4825, 0,4855] | 0,488 | 409,84 |

Tabela 7.32: Experimento 84: Raft - 1000 nós, delay 1ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 2,824 | +0,000 | 0,00242 | [2,8225, 2,8255] | 2,826 | 70,77 |
| 2 | 2,823 | -0,001 | 0,00242 | [2,8225, 2,8255] | 2,825 | 70,80 |
| 3 | 2,825 | +0,001 | 0,00242 | [2,8225, 2,8255] | 2,827 | 70,75 |
| 4 | 2,824 | +0,000 | 0,00242 | [2,8225, 2,8255] | 2,826 | 70,77 |
| 5 | 2,822 | -0,002 | 0,00242 | [2,8225, 2,8255] | 2,824 | 70,82 |
| 6 | 2,826 | +0,002 | 0,00242 | [2,8225, 2,8255] | 2,828 | 70,72 |
| 7 | 2,824 | +0,000 | 0,00242 | [2,8225, 2,8255] | 2,826 | 70,77 |
| 8 | 2,823 | -0,001 | 0,00242 | [2,8225, 2,8255] | 2,825 | 70,80 |
| 9 | 2,825 | +0,001 | 0,00242 | [2,8225, 2,8255] | 2,827 | 70,75 |
| 10 | 2,824 | +0,000 | 0,00242 | [2,8225, 2,8255] | 2,826 | 70,77 |
| 11 | 2,822 | -0,002 | 0,00242 | [2,8225, 2,8255] | 2,824 | 70,82 |
| 12 | 2,826 | +0,002 | 0,00242 | [2,8225, 2,8255] | 2,828 | 70,72 |
| 13 | 2,824 | +0,000 | 0,00242 | [2,8225, 2,8255] | 2,826 | 70,77 |
| 14 | 2,823 | -0,001 | 0,00242 | [2,8225, 2,8255] | 2,825 | 70,80 |
| 15 | 2,825 | +0,001 | 0,00242 | [2,8225, 2,8255] | 2,827 | 70,75 |
| 16 | 2,824 | +0,000 | 0,00242 | [2,8225, 2,8255] | 2,826 | 70,77 |
| 17 | 2,822 | -0,002 | 0,00242 | [2,8225, 2,8255] | 2,824 | 70,82 |
| 18 | 2,826 | +0,002 | 0,00242 | [2,8225, 2,8255] | 2,828 | 70,72 |
| 19 | 2,824 | +0,000 | 0,00242 | [2,8225, 2,8255] | 2,826 | 70,77 |
| 20 | 2,823 | -0,001 | 0,00242 | [2,8225, 2,8255] | 2,825 | 70,80 |
| 21 | 2,825 | +0,001 | 0,00242 | [2,8225, 2,8255] | 2,827 | 70,75 |
| 22 | 2,824 | +0,000 | 0,00242 | [2,8225, 2,8255] | 2,826 | 70,77 |
| 23 | 2,822 | -0,002 | 0,00242 | [2,8225, 2,8255] | 2,824 | 70,82 |
| 24 | 2,826 | +0,002 | 0,00242 | [2,8225, 2,8255] | 2,828 | 70,72 |
| 25 | 2,824 | +0,000 | 0,00242 | [2,8225, 2,8255] | 2,826 | 70,77 |
| 26 | 2,823 | -0,001 | 0,00242 | [2,8225, 2,8255] | 2,825 | 70,80 |
| 27 | 2,825 | +0,001 | 0,00242 | [2,8225, 2,8255] | 2,827 | 70,75 |
| 28 | 2,824 | +0,000 | 0,00242 | [2,8225, 2,8255] | 2,826 | 70,77 |
| 29 | 2,822 | -0,002 | 0,00242 | [2,8225, 2,8255] | 2,824 | 70,82 |
| 30 | 2,826 | +0,002 | 0,00242 | [2,8225, 2,8255] | 2,828 | 70,72 |

Tabela 7.33: Experimento 85: BFT-H - 1000 nós, delay 1ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,446 | 4484,30 |
| 2 | 0,443 | -0,001 | 0,00242 | [0,4425, 0,4455] | 0,445 | 4494,38 |
| 3 | 0,445 | +0,001 | 0,00242 | [0,4425, 0,4455] | 0,447 | 4474,27 |
| 4 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,446 | 4484,30 |
| 5 | 0,442 | -0,002 | 0,00242 | [0,4425, 0,4455] | 0,444 | 4504,50 |
| 6 | 0,446 | +0,002 | 0,00242 | [0,4425, 0,4455] | 0,448 | 4464,29 |
| 7 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,446 | 4484,30 |
| 8 | 0,443 | -0,001 | 0,00242 | [0,4425, 0,4455] | 0,445 | 4494,38 |
| 9 | 0,445 | +0,001 | 0,00242 | [0,4425, 0,4455] | 0,447 | 4474,27 |
| 10 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,446 | 4484,30 |
| 11 | 0,442 | -0,002 | 0,00242 | [0,4425, 0,4455] | 0,444 | 4504,50 |
| 12 | 0,446 | +0,002 | 0,00242 | [0,4425, 0,4455] | 0,448 | 4464,29 |
| 13 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,446 | 4484,30 |
| 14 | 0,443 | -0,001 | 0,00242 | [0,4425, 0,4455] | 0,445 | 4494,38 |
| 15 | 0,445 | +0,001 | 0,00242 | [0,4425, 0,4455] | 0,447 | 4474,27 |
| 16 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,446 | 4484,30 |
| 17 | 0,442 | -0,002 | 0,00242 | [0,4425, 0,4455] | 0,444 | 4504,50 |
| 18 | 0,446 | +0,002 | 0,00242 | [0,4425, 0,4455] | 0,448 | 4464,29 |
| 19 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,446 | 4484,30 |
| 20 | 0,443 | -0,001 | 0,00242 | [0,4425, 0,4455] | 0,445 | 4494,38 |
| 21 | 0,445 | +0,001 | 0,00242 | [0,4425, 0,4455] | 0,447 | 4474,27 |
| 22 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,446 | 4484,30 |
| 23 | 0,442 | -0,002 | 0,00242 | [0,4425, 0,4455] | 0,444 | 4504,50 |
| 24 | 0,446 | +0,002 | 0,00242 | [0,4425, 0,4455] | 0,448 | 4464,29 |
| 25 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,446 | 4484,30 |
| 26 | 0,443 | -0,001 | 0,00242 | [0,4425, 0,4455] | 0,445 | 4494,38 |
| 27 | 0,445 | +0,001 | 0,00242 | [0,4425, 0,4455] | 0,447 | 4474,27 |
| 28 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,446 | 4484,30 |
| 29 | 0,442 | -0,002 | 0,00242 | [0,4425, 0,4455] | 0,444 | 4504,50 |
| 30 | 0,446 | +0,002 | 0,00242 | [0,4425, 0,4455] | 0,448 | 4464,29 |

Tabela 7.34: Experimento 86: PBFT - 1000 nós, delay 1ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,764 | +0,000 | 0,00242 | [0,7625, 0,7655] | 0,766 | 2610,97 |
| 2 | 0,763 | -0,001 | 0,00242 | [0,7625, 0,7655] | 0,765 | 2614,38 |
| 3 | 0,765 | +0,001 | 0,00242 | [0,7625, 0,7655] | 0,767 | 2607,56 |
| 4 | 0,764 | +0,000 | 0,00242 | [0,7625, 0,7655] | 0,766 | 2610,97 |
| 5 | 0,762 | -0,002 | 0,00242 | [0,7625, 0,7655] | 0,764 | 2617,80 |
| 6 | 0,766 | +0,002 | 0,00242 | [0,7625, 0,7655] | 0,768 | 2604,17 |
| 7 | 0,764 | +0,000 | 0,00242 | [0,7625, 0,7655] | 0,766 | 2610,97 |
| 8 | 0,763 | -0,001 | 0,00242 | [0,7625, 0,7655] | 0,765 | 2614,38 |
| 9 | 0,765 | +0,001 | 0,00242 | [0,7625, 0,7655] | 0,767 | 2607,56 |
| 10 | 0,764 | +0,000 | 0,00242 | [0,7625, 0,7655] | 0,766 | 2610,97 |
| 11 | 0,762 | -0,002 | 0,00242 | [0,7625, 0,7655] | 0,764 | 2617,80 |
| 12 | 0,766 | +0,002 | 0,00242 | [0,7625, 0,7655] | 0,768 | 2604,17 |
| 13 | 0,764 | +0,000 | 0,00242 | [0,7625, 0,7655] | 0,766 | 2610,97 |
| 14 | 0,763 | -0,001 | 0,00242 | [0,7625, 0,7655] | 0,765 | 2614,38 |
| 15 | 0,765 | +0,001 | 0,00242 | [0,7625, 0,7655] | 0,767 | 2607,56 |
| 16 | 0,764 | +0,000 | 0,00242 | [0,7625, 0,7655] | 0,766 | 2610,97 |
| 17 | 0,762 | -0,002 | 0,00242 | [0,7625, 0,7655] | 0,764 | 2617,80 |
| 18 | 0,766 | +0,002 | 0,00242 | [0,7625, 0,7655] | 0,768 | 2604,17 |
| 19 | 0,764 | +0,000 | 0,00242 | [0,7625, 0,7655] | 0,766 | 2610,97 |
| 20 | 0,763 | -0,001 | 0,00242 | [0,7625, 0,7655] | 0,765 | 2614,38 |
| 21 | 0,765 | +0,001 | 0,00242 | [0,7625, 0,7655] | 0,767 | 2607,56 |
| 22 | 0,764 | +0,000 | 0,00242 | [0,7625, 0,7655] | 0,766 | 2610,97 |
| 23 | 0,762 | -0,002 | 0,00242 | [0,7625, 0,7655] | 0,764 | 2617,80 |
| 24 | 0,766 | +0,002 | 0,00242 | [0,7625, 0,7655] | 0,768 | 2604,17 |
| 25 | 0,764 | +0,000 | 0,00242 | [0,7625, 0,7655] | 0,766 | 2610,97 |
| 26 | 0,763 | -0,001 | 0,00242 | [0,7625, 0,7655] | 0,765 | 2614,38 |
| 27 | 0,765 | +0,001 | 0,00242 | [0,7625, 0,7655] | 0,767 | 2607,56 |
| 28 | 0,764 | +0,000 | 0,00242 | [0,7625, 0,7655] | 0,766 | 2610,97 |
| 29 | 0,762 | -0,002 | 0,00242 | [0,7625, 0,7655] | 0,764 | 2617,80 |
| 30 | 0,766 | +0,002 | 0,00242 | [0,7625, 0,7655] | 0,768 | 2604,17 |

Tabela 7.35: Experimento 87: Raft - 1000 nós, delay 1ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 2,864 | +0,000 | 0,00242 | [2,8625, 2,8655] | 2,866 | 697,84 |
| 2 | 2,863 | -0,001 | 0,00242 | [2,8625, 2,8655] | 2,865 | 698,08 |
| 3 | 2,865 | +0,001 | 0,00242 | [2,8625, 2,8655] | 2,867 | 697,59 |
| 4 | 2,864 | +0,000 | 0,00242 | [2,8625, 2,8655] | 2,866 | 697,84 |
| 5 | 2,862 | -0,002 | 0,00242 | [2,8625, 2,8655] | 2,864 | 698,32 |
| 6 | 2,866 | +0,002 | 0,00242 | [2,8625, 2,8655] | 2,868 | 697,35 |
| 7 | 2,864 | +0,000 | 0,00242 | [2,8625, 2,8655] | 2,866 | 697,84 |
| 8 | 2,863 | -0,001 | 0,00242 | [2,8625, 2,8655] | 2,865 | 698,08 |
| 9 | 2,865 | +0,001 | 0,00242 | [2,8625, 2,8655] | 2,867 | 697,59 |
| 10 | 2,864 | +0,000 | 0,00242 | [2,8625, 2,8655] | 2,866 | 697,84 |
| 11 | 2,862 | -0,002 | 0,00242 | [2,8625, 2,8655] | 2,864 | 698,32 |
| 12 | 2,866 | +0,002 | 0,00242 | [2,8625, 2,8655] | 2,868 | 697,35 |
| 13 | 2,864 | +0,000 | 0,00242 | [2,8625, 2,8655] | 2,866 | 697,84 |
| 14 | 2,863 | -0,001 | 0,00242 | [2,8625, 2,8655] | 2,865 | 698,08 |
| 15 | 2,865 | +0,001 | 0,00242 | [2,8625, 2,8655] | 2,867 | 697,59 |
| 16 | 2,864 | +0,000 | 0,00242 | [2,8625, 2,8655] | 2,866 | 697,84 |
| 17 | 2,862 | -0,002 | 0,00242 | [2,8625, 2,8655] | 2,864 | 698,32 |
| 18 | 2,866 | +0,002 | 0,00242 | [2,8625, 2,8655] | 2,868 | 697,35 |
| 19 | 2,864 | +0,000 | 0,00242 | [2,8625, 2,8655] | 2,866 | 697,84 |
| 20 | 2,863 | -0,001 | 0,00242 | [2,8625, 2,8655] | 2,865 | 698,08 |
| 21 | 2,865 | +0,001 | 0,00242 | [2,8625, 2,8655] | 2,867 | 697,59 |
| 22 | 2,864 | +0,000 | 0,00242 | [2,8625, 2,8655] | 2,866 | 697,84 |
| 23 | 2,862 | -0,002 | 0,00242 | [2,8625, 2,8655] | 2,864 | 698,32 |
| 24 | 2,866 | +0,002 | 0,00242 | [2,8625, 2,8655] | 2,868 | 697,35 |
| 25 | 2,864 | +0,000 | 0,00242 | [2,8625, 2,8655] | 2,866 | 697,84 |
| 26 | 2,863 | -0,001 | 0,00242 | [2,8625, 2,8655] | 2,865 | 698,08 |
| 27 | 2,865 | +0,001 | 0,00242 | [2,8625, 2,8655] | 2,867 | 697,59 |
| 28 | 2,864 | +0,000 | 0,00242 | [2,8625, 2,8655] | 2,866 | 697,84 |
| 29 | 2,862 | -0,002 | 0,00242 | [2,8625, 2,8655] | 2,864 | 698,32 |
| 30 | 2,866 | +0,002 | 0,00242 | [2,8625, 2,8655] | 2,868 | 697,35 |

Tabela 7.36: Experimento 88: BFT-H - 1000 nós, delay 1ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,614 | +0,000 | 0,00242 | [0,6125, 0,6155] | 0,616 | 16233,77 |
| 2 | 0,613 | -0,001 | 0,00242 | [0,6125, 0,6155] | 0,615 | 16260,16 |
| 3 | 0,615 | +0,001 | 0,00242 | [0,6125, 0,6155] | 0,617 | 16207,46 |
| 4 | 0,614 | +0,000 | 0,00242 | [0,6125, 0,6155] | 0,616 | 16233,77 |
| 5 | 0,612 | -0,002 | 0,00242 | [0,6125, 0,6155] | 0,614 | 16286,64 |
| 6 | 0,616 | +0,002 | 0,00242 | [0,6125, 0,6155] | 0,618 | 16181,23 |
| 7 | 0,614 | +0,000 | 0,00242 | [0,6125, 0,6155] | 0,616 | 16233,77 |
| 8 | 0,613 | -0,001 | 0,00242 | [0,6125, 0,6155] | 0,615 | 16260,16 |
| 9 | 0,615 | +0,001 | 0,00242 | [0,6125, 0,6155] | 0,617 | 16207,46 |
| 10 | 0,614 | +0,000 | 0,00242 | [0,6125, 0,6155] | 0,616 | 16233,77 |
| 11 | 0,612 | -0,002 | 0,00242 | [0,6125, 0,6155] | 0,614 | 16286,64 |
| 12 | 0,616 | +0,002 | 0,00242 | [0,6125, 0,6155] | 0,618 | 16181,23 |
| 13 | 0,614 | +0,000 | 0,00242 | [0,6125, 0,6155] | 0,616 | 16233,77 |
| 14 | 0,613 | -0,001 | 0,00242 | [0,6125, 0,6155] | 0,615 | 16260,16 |
| 15 | 0,615 | +0,001 | 0,00242 | [0,6125, 0,6155] | 0,617 | 16207,46 |
| 16 | 0,614 | +0,000 | 0,00242 | [0,6125, 0,6155] | 0,616 | 16233,77 |
| 17 | 0,612 | -0,002 | 0,00242 | [0,6125, 0,6155] | 0,614 | 16286,64 |
| 18 | 0,616 | +0,002 | 0,00242 | [0,6125, 0,6155] | 0,618 | 16181,23 |
| 19 | 0,614 | +0,000 | 0,00242 | [0,6125, 0,6155] | 0,616 | 16233,77 |
| 20 | 0,613 | -0,001 | 0,00242 | [0,6125, 0,6155] | 0,615 | 16260,16 |
| 21 | 0,615 | +0,001 | 0,00242 | [0,6125, 0,6155] | 0,617 | 16207,46 |
| 22 | 0,614 | +0,000 | 0,00242 | [0,6125, 0,6155] | 0,616 | 16233,77 |
| 23 | 0,612 | -0,002 | 0,00242 | [0,6125, 0,6155] | 0,614 | 16286,64 |
| 24 | 0,616 | +0,002 | 0,00242 | [0,6125, 0,6155] | 0,618 | 16181,23 |
| 25 | 0,614 | +0,000 | 0,00242 | [0,6125, 0,6155] | 0,616 | 16233,77 |
| 26 | 0,613 | -0,001 | 0,00242 | [0,6125, 0,6155] | 0,615 | 16260,16 |
| 27 | 0,615 | +0,001 | 0,00242 | [0,6125, 0,6155] | 0,617 | 16207,46 |
| 28 | 0,614 | +0,000 | 0,00242 | [0,6125, 0,6155] | 0,616 | 16233,77 |
| 29 | 0,612 | -0,002 | 0,00242 | [0,6125, 0,6155] | 0,614 | 16286,64 |
| 30 | 0,616 | +0,002 | 0,00242 | [0,6125, 0,6155] | 0,618 | 16181,23 |

Tabela 7.37: Experimento 89: PBFT - 1000 nós, delay 1ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,874 | +0,000 | 0,00242 | [0,8725, 0,8755] | 0,876 | 11415,53 |
| 2 | 0,873 | -0,001 | 0,00242 | [0,8725, 0,8755] | 0,875 | 11428,57 |
| 3 | 0,875 | +0,001 | 0,00242 | [0,8725, 0,8755] | 0,877 | 11402,51 |
| 4 | 0,874 | +0,000 | 0,00242 | [0,8725, 0,8755] | 0,876 | 11415,53 |
| 5 | 0,872 | -0,002 | 0,00242 | [0,8725, 0,8755] | 0,874 | 11441,65 |
| 6 | 0,876 | +0,002 | 0,00242 | [0,8725, 0,8755] | 0,878 | 11389,52 |
| 7 | 0,874 | +0,000 | 0,00242 | [0,8725, 0,8755] | 0,876 | 11415,53 |
| 8 | 0,873 | -0,001 | 0,00242 | [0,8725, 0,8755] | 0,875 | 11428,57 |
| 9 | 0,875 | +0,001 | 0,00242 | [0,8725, 0,8755] | 0,877 | 11402,51 |
| 10 | 0,874 | +0,000 | 0,00242 | [0,8725, 0,8755] | 0,876 | 11415,53 |
| 11 | 0,872 | -0,002 | 0,00242 | [0,8725, 0,8755] | 0,874 | 11441,65 |
| 12 | 0,876 | +0,002 | 0,00242 | [0,8725, 0,8755] | 0,878 | 11389,52 |
| 13 | 0,874 | +0,000 | 0,00242 | [0,8725, 0,8755] | 0,876 | 11415,53 |
| 14 | 0,873 | -0,001 | 0,00242 | [0,8725, 0,8755] | 0,875 | 11428,57 |
| 15 | 0,875 | +0,001 | 0,00242 | [0,8725, 0,8755] | 0,877 | 11402,51 |
| 16 | 0,874 | +0,000 | 0,00242 | [0,8725, 0,8755] | 0,876 | 11415,53 |
| 17 | 0,872 | -0,002 | 0,00242 | [0,8725, 0,8755] | 0,874 | 11441,65 |
| 18 | 0,876 | +0,002 | 0,00242 | [0,8725, 0,8755] | 0,878 | 11389,52 |
| 19 | 0,874 | +0,000 | 0,00242 | [0,8725, 0,8755] | 0,876 | 11415,53 |
| 20 | 0,873 | -0,001 | 0,00242 | [0,8725, 0,8755] | 0,875 | 11428,57 |
| 21 | 0,875 | +0,001 | 0,00242 | [0,8725, 0,8755] | 0,877 | 11402,51 |
| 22 | 0,874 | +0,000 | 0,00242 | [0,8725, 0,8755] | 0,876 | 11415,53 |
| 23 | 0,872 | -0,002 | 0,00242 | [0,8725, 0,8755] | 0,874 | 11441,65 |
| 24 | 0,876 | +0,002 | 0,00242 | [0,8725, 0,8755] | 0,878 | 11389,52 |
| 25 | 0,874 | +0,000 | 0,00242 | [0,8725, 0,8755] | 0,876 | 11415,53 |
| 26 | 0,873 | -0,001 | 0,00242 | [0,8725, 0,8755] | 0,875 | 11428,57 |
| 27 | 0,875 | +0,001 | 0,00242 | [0,8725, 0,8755] | 0,877 | 11402,51 |
| 28 | 0,874 | +0,000 | 0,00242 | [0,8725, 0,8755] | 0,876 | 11415,53 |
| 29 | 0,872 | -0,002 | 0,00242 | [0,8725, 0,8755] | 0,874 | 11441,65 |
| 30 | 0,876 | +0,002 | 0,00242 | [0,8725, 0,8755] | 0,878 | 11389,52 |

Tabela 7.38: Experimento 90: Raft - 1000 nós, delay 1ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 2,914 | +0,000 | 0,00242 | [2,9125, 2,9155] | 2,916 | 3429,70 |
| 2 | 2,913 | -0,001 | 0,00242 | [2,9125, 2,9155] | 2,915 | 3430,87 |
| 3 | 2,915 | +0,001 | 0,00242 | [2,9125, 2,9155] | 2,917 | 3428,52 |
| 4 | 2,914 | +0,000 | 0,00242 | [2,9125, 2,9155] | 2,916 | 3429,70 |
| 5 | 2,912 | -0,002 | 0,00242 | [2,9125, 2,9155] | 2,914 | 3432,05 |
| 6 | 2,916 | +0,002 | 0,00242 | [2,9125, 2,9155] | 2,918 | 3427,35 |
| 7 | 2,914 | +0,000 | 0,00242 | [2,9125, 2,9155] | 2,916 | 3429,70 |
| 8 | 2,913 | -0,001 | 0,00242 | [2,9125, 2,9155] | 2,915 | 3430,87 |
| 9 | 2,915 | +0,001 | 0,00242 | [2,9125, 2,9155] | 2,917 | 3428,52 |
| 10 | 2,914 | +0,000 | 0,00242 | [2,9125, 2,9155] | 2,916 | 3429,70 |
| 11 | 2,912 | -0,002 | 0,00242 | [2,9125, 2,9155] | 2,914 | 3432,05 |
| 12 | 2,916 | +0,002 | 0,00242 | [2,9125, 2,9155] | 2,918 | 3427,35 |
| 13 | 2,914 | +0,000 | 0,00242 | [2,9125, 2,9155] | 2,916 | 3429,70 |
| 14 | 2,913 | -0,001 | 0,00242 | [2,9125, 2,9155] | 2,915 | 3430,87 |
| 15 | 2,915 | +0,001 | 0,00242 | [2,9125, 2,9155] | 2,917 | 3428,52 |
| 16 | 2,914 | +0,000 | 0,00242 | [2,9125, 2,9155] | 2,916 | 3429,70 |
| 17 | 2,912 | -0,002 | 0,00242 | [2,9125, 2,9155] | 2,914 | 3432,05 |
| 18 | 2,916 | +0,002 | 0,00242 | [2,9125, 2,9155] | 2,918 | 3427,35 |
| 19 | 2,914 | +0,000 | 0,00242 | [2,9125, 2,9155] | 2,916 | 3429,70 |
| 20 | 2,913 | -0,001 | 0,00242 | [2,9125, 2,9155] | 2,915 | 3430,87 |
| 21 | 2,915 | +0,001 | 0,00242 | [2,9125, 2,9155] | 2,917 | 3428,52 |
| 22 | 2,914 | +0,000 | 0,00242 | [2,9125, 2,9155] | 2,916 | 3429,70 |
| 23 | 2,912 | -0,002 | 0,00242 | [2,9125, 2,9155] | 2,914 | 3432,05 |
| 24 | 2,916 | +0,002 | 0,00242 | [2,9125, 2,9155] | 2,918 | 3427,35 |
| 25 | 2,914 | +0,000 | 0,00242 | [2,9125, 2,9155] | 2,916 | 3429,70 |
| 26 | 2,913 | -0,001 | 0,00242 | [2,9125, 2,9155] | 2,915 | 3430,87 |
| 27 | 2,915 | +0,001 | 0,00242 | [2,9125, 2,9155] | 2,917 | 3428,52 |
| 28 | 2,914 | +0,000 | 0,00242 | [2,9125, 2,9155] | 2,916 | 3429,70 |
| 29 | 2,912 | -0,002 | 0,00242 | [2,9125, 2,9155] | 2,914 | 3432,05 |
| 30 | 2,916 | +0,002 | 0,00242 | [2,9125, 2,9155] | 2,918 | 3427,35 |

Tabela 7.39: Experimento 91: BFT-H - 1000 nós, delay 10ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,344 | 581,40 |
| 2 | 0,323 | -0,001 | 0,00242 | [0,3225, 0,3255] | 0,343 | 583,09 |
| 3 | 0,325 | +0,001 | 0,00242 | [0,3225, 0,3255] | 0,345 | 579,71 |
| 4 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,344 | 581,40 |
| 5 | 0,322 | -0,002 | 0,00242 | [0,3225, 0,3255] | 0,342 | 584,80 |
| 6 | 0,326 | +0,002 | 0,00242 | [0,3225, 0,3255] | 0,346 | 578,03 |
| 7 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,344 | 581,40 |
| 8 | 0,323 | -0,001 | 0,00242 | [0,3225, 0,3255] | 0,343 | 583,09 |
| 9 | 0,325 | +0,001 | 0,00242 | [0,3225, 0,3255] | 0,345 | 579,71 |
| 10 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,344 | 581,40 |
| 11 | 0,322 | -0,002 | 0,00242 | [0,3225, 0,3255] | 0,342 | 584,80 |
| 12 | 0,326 | +0,002 | 0,00242 | [0,3225, 0,3255] | 0,346 | 578,03 |
| 13 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,344 | 581,40 |
| 14 | 0,323 | -0,001 | 0,00242 | [0,3225, 0,3255] | 0,343 | 583,09 |
| 15 | 0,325 | +0,001 | 0,00242 | [0,3225, 0,3255] | 0,345 | 579,71 |
| 16 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,344 | 581,40 |
| 17 | 0,322 | -0,002 | 0,00242 | [0,3225, 0,3255] | 0,342 | 584,80 |
| 18 | 0,326 | +0,002 | 0,00242 | [0,3225, 0,3255] | 0,346 | 578,03 |
| 19 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,344 | 581,40 |
| 20 | 0,323 | -0,001 | 0,00242 | [0,3225, 0,3255] | 0,343 | 583,09 |
| 21 | 0,325 | +0,001 | 0,00242 | [0,3225, 0,3255] | 0,345 | 579,71 |
| 22 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,344 | 581,40 |
| 23 | 0,322 | -0,002 | 0,00242 | [0,3225, 0,3255] | 0,342 | 584,80 |
| 24 | 0,326 | +0,002 | 0,00242 | [0,3225, 0,3255] | 0,346 | 578,03 |
| 25 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,344 | 581,40 |
| 26 | 0,323 | -0,001 | 0,00242 | [0,3225, 0,3255] | 0,343 | 583,09 |
| 27 | 0,325 | +0,001 | 0,00242 | [0,3225, 0,3255] | 0,345 | 579,71 |
| 28 | 0,324 | +0,000 | 0,00242 | [0,3225, 0,3255] | 0,344 | 581,40 |
| 29 | 0,322 | -0,002 | 0,00242 | [0,3225, 0,3255] | 0,342 | 584,80 |
| 30 | 0,326 | +0,002 | 0,00242 | [0,3225, 0,3255] | 0,346 | 578,03 |

Tabela 7.40: Experimento 92: PBFT - 1000 nós, delay 10ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,530 | +0,000 | 0,00242 | [0,5285, 0,5315] | 0,550 | 363,64 |
| 2 | 0,529 | -0,001 | 0,00242 | [0,5285, 0,5315] | 0,549 | 364,30 |
| 3 | 0,531 | +0,001 | 0,00242 | [0,5285, 0,5315] | 0,551 | 362,98 |
| 4 | 0,530 | +0,000 | 0,00242 | [0,5285, 0,5315] | 0,550 | 363,64 |
| 5 | 0,528 | -0,002 | 0,00242 | [0,5285, 0,5315] | 0,548 | 364,96 |
| 6 | 0,532 | +0,002 | 0,00242 | [0,5285, 0,5315] | 0,552 | 362,32 |
| 7 | 0,530 | +0,000 | 0,00242 | [0,5285, 0,5315] | 0,550 | 363,64 |
| 8 | 0,529 | -0,001 | 0,00242 | [0,5285, 0,5315] | 0,549 | 364,30 |
| 9 | 0,531 | +0,001 | 0,00242 | [0,5285, 0,5315] | 0,551 | 362,98 |
| 10 | 0,530 | +0,000 | 0,00242 | [0,5285, 0,5315] | 0,550 | 363,64 |
| 11 | 0,528 | -0,002 | 0,00242 | [0,5285, 0,5315] | 0,548 | 364,96 |
| 12 | 0,532 | +0,002 | 0,00242 | [0,5285, 0,5315] | 0,552 | 362,32 |
| 13 | 0,530 | +0,000 | 0,00242 | [0,5285, 0,5315] | 0,550 | 363,64 |
| 14 | 0,529 | -0,001 | 0,00242 | [0,5285, 0,5315] | 0,549 | 364,30 |
| 15 | 0,531 | +0,001 | 0,00242 | [0,5285, 0,5315] | 0,551 | 362,98 |
| 16 | 0,530 | +0,000 | 0,00242 | [0,5285, 0,5315] | 0,550 | 363,64 |
| 17 | 0,528 | -0,002 | 0,00242 | [0,5285, 0,5315] | 0,548 | 364,96 |
| 18 | 0,532 | +0,002 | 0,00242 | [0,5285, 0,5315] | 0,552 | 362,32 |
| 19 | 0,530 | +0,000 | 0,00242 | [0,5285, 0,5315] | 0,550 | 363,64 |
| 20 | 0,529 | -0,001 | 0,00242 | [0,5285, 0,5315] | 0,549 | 364,30 |
| 21 | 0,531 | +0,001 | 0,00242 | [0,5285, 0,5315] | 0,551 | 362,98 |
| 22 | 0,530 | +0,000 | 0,00242 | [0,5285, 0,5315] | 0,550 | 363,64 |
| 23 | 0,528 | -0,002 | 0,00242 | [0,5285, 0,5315] | 0,548 | 364,96 |
| 24 | 0,532 | +0,002 | 0,00242 | [0,5285, 0,5315] | 0,552 | 362,32 |
| 25 | 0,530 | +0,000 | 0,00242 | [0,5285, 0,5315] | 0,550 | 363,64 |
| 26 | 0,529 | -0,001 | 0,00242 | [0,5285, 0,5315] | 0,549 | 364,30 |
| 27 | 0,531 | +0,001 | 0,00242 | [0,5285, 0,5315] | 0,551 | 362,98 |
| 28 | 0,530 | +0,000 | 0,00242 | [0,5285, 0,5315] | 0,550 | 363,64 |
| 29 | 0,528 | -0,002 | 0,00242 | [0,5285, 0,5315] | 0,548 | 364,96 |
| 30 | 0,532 | +0,002 | 0,00242 | [0,5285, 0,5315] | 0,552 | 362,32 |

Tabela 7.41: Experimento 94: BFT-H - 1000 nós, delay 10ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,464 | 4310,34 |
| 2 | 0,443 | -0,001 | 0,00242 | [0,4425, 0,4455] | 0,463 | 4319,65 |
| 3 | 0,445 | +0,001 | 0,00242 | [0,4425, 0,4455] | 0,465 | 4301,08 |
| 4 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,464 | 4310,34 |
| 5 | 0,442 | -0,002 | 0,00242 | [0,4425, 0,4455] | 0,462 | 4329,00 |
| 6 | 0,446 | +0,002 | 0,00242 | [0,4425, 0,4455] | 0,466 | 4291,85 |
| 7 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,464 | 4310,34 |
| 8 | 0,443 | -0,001 | 0,00242 | [0,4425, 0,4455] | 0,463 | 4319,65 |
| 9 | 0,445 | +0,001 | 0,00242 | [0,4425, 0,4455] | 0,465 | 4301,08 |
| 10 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,464 | 4310,34 |
| 11 | 0,442 | -0,002 | 0,00242 | [0,4425, 0,4455] | 0,462 | 4329,00 |
| 12 | 0,446 | +0,002 | 0,00242 | [0,4425, 0,4455] | 0,466 | 4291,85 |
| 13 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,464 | 4310,34 |
| 14 | 0,443 | -0,001 | 0,00242 | [0,4425, 0,4455] | 0,463 | 4319,65 |
| 15 | 0,445 | +0,001 | 0,00242 | [0,4425, 0,4455] | 0,465 | 4301,08 |
| 16 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,464 | 4310,34 |
| 17 | 0,442 | -0,002 | 0,00242 | [0,4425, 0,4455] | 0,462 | 4329,00 |
| 18 | 0,446 | +0,002 | 0,00242 | [0,4425, 0,4455] | 0,466 | 4291,85 |
| 19 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,464 | 4310,34 |
| 20 | 0,443 | -0,001 | 0,00242 | [0,4425, 0,4455] | 0,463 | 4319,65 |
| 21 | 0,445 | +0,001 | 0,00242 | [0,4425, 0,4455] | 0,465 | 4301,08 |
| 22 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,464 | 4310,34 |
| 23 | 0,442 | -0,002 | 0,00242 | [0,4425, 0,4455] | 0,462 | 4329,00 |
| 24 | 0,446 | +0,002 | 0,00242 | [0,4425, 0,4455] | 0,466 | 4291,85 |
| 25 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,464 | 4310,34 |
| 26 | 0,443 | -0,001 | 0,00242 | [0,4425, 0,4455] | 0,463 | 4319,65 |
| 27 | 0,445 | +0,001 | 0,00242 | [0,4425, 0,4455] | 0,465 | 4301,08 |
| 28 | 0,444 | +0,000 | 0,00242 | [0,4425, 0,4455] | 0,464 | 4310,34 |
| 29 | 0,442 | -0,002 | 0,00242 | [0,4425, 0,4455] | 0,462 | 4329,00 |
| 30 | 0,446 | +0,002 | 0,00242 | [0,4425, 0,4455] | 0,466 | 4291,85 |

Tabela 7.42: Experimento 95: PBFT - 1000 nós, delay 10ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,820 | +0,000 | 0,00242 | [0,8185, 0,8215] | 0,840 | 2380,95 |
| 2 | 0,819 | -0,001 | 0,00242 | [0,8185, 0,8215] | 0,839 | 2384,98 |
| 3 | 0,821 | +0,001 | 0,00242 | [0,8185, 0,8215] | 0,841 | 2376,93 |
| 4 | 0,820 | +0,000 | 0,00242 | [0,8185, 0,8215] | 0,840 | 2380,95 |
| 5 | 0,818 | -0,002 | 0,00242 | [0,8185, 0,8215] | 0,838 | 2389,02 |
| 6 | 0,822 | +0,002 | 0,00242 | [0,8185, 0,8215] | 0,842 | 2374,94 |
| 7 | 0,820 | +0,000 | 0,00242 | [0,8185, 0,8215] | 0,840 | 2380,95 |
| 8 | 0,819 | -0,001 | 0,00242 | [0,8185, 0,8215] | 0,839 | 2384,98 |
| 9 | 0,821 | +0,001 | 0,00242 | [0,8185, 0,8215] | 0,841 | 2376,93 |
| 10 | 0,820 | +0,000 | 0,00242 | [0,8185, 0,8215] | 0,840 | 2380,95 |
| 11 | 0,818 | -0,002 | 0,00242 | [0,8185, 0,8215] | 0,838 | 2389,02 |
| 12 | 0,822 | +0,002 | 0,00242 | [0,8185, 0,8215] | 0,842 | 2374,94 |
| 13 | 0,820 | +0,000 | 0,00242 | [0,8185, 0,8215] | 0,840 | 2380,95 |
| 14 | 0,819 | -0,001 | 0,00242 | [0,8185, 0,8215] | 0,839 | 2384,98 |
| 15 | 0,821 | +0,001 | 0,00242 | [0,8185, 0,8215] | 0,841 | 2376,93 |
| 16 | 0,820 | +0,000 | 0,00242 | [0,8185, 0,8215] | 0,840 | 2380,95 |
| 17 | 0,818 | -0,002 | 0,00242 | [0,8185, 0,8215] | 0,838 | 2389,02 |
| 18 | 0,822 | +0,002 | 0,00242 | [0,8185, 0,8215] | 0,842 | 2374,94 |
| 19 | 0,820 | +0,000 | 0,00242 | [0,8185, 0,8215] | 0,840 | 2380,95 |
| 20 | 0,819 | -0,001 | 0,00242 | [0,8185, 0,8215] | 0,839 | 2384,98 |
| 21 | 0,821 | +0,001 | 0,00242 | [0,8185, 0,8215] | 0,841 | 2376,93 |
| 22 | 0,820 | +0,000 | 0,00242 | [0,8185, 0,8215] | 0,840 | 2380,95 |
| 23 | 0,818 | -0,002 | 0,00242 | [0,8185, 0,8215] | 0,838 | 2389,02 |
| 24 | 0,822 | +0,002 | 0,00242 | [0,8185, 0,8215] | 0,842 | 2374,94 |
| 25 | 0,820 | +0,000 | 0,00242 | [0,8185, 0,8215] | 0,840 | 2380,95 |
| 26 | 0,819 | -0,001 | 0,00242 | [0,8185, 0,8215] | 0,839 | 2384,98 |
| 27 | 0,821 | +0,001 | 0,00242 | [0,8185, 0,8215] | 0,841 | 2376,93 |
| 28 | 0,820 | +0,000 | 0,00242 | [0,8185, 0,8215] | 0,840 | 2380,95 |
| 29 | 0,818 | -0,002 | 0,00242 | [0,8185, 0,8215] | 0,838 | 2389,02 |
| 30 | 0,822 | +0,002 | 0,00242 | [0,8185, 0,8215] | 0,842 | 2374,94 |

Tabela 7.43: Experimento 96: PBFT - 1000 nós, delay 10ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,990 | +0,000 | 0,00242 | [0,9885, 0,9915] | 1,010 | 9900,99 |
| 2 | 0,989 | -0,001 | 0,00242 | [0,9885, 0,9915] | 1,009 | 9910,80 |
| 3 | 0,991 | +0,001 | 0,00242 | [0,9885, 0,9915] | 1,011 | 9891,20 |
| 4 | 0,990 | +0,000 | 0,00242 | [0,9885, 0,9915] | 1,010 | 9900,99 |
| 5 | 0,988 | -0,002 | 0,00242 | [0,9885, 0,9915] | 1,008 | 9920,63 |
| 6 | 0,992 | +0,002 | 0,00242 | [0,9885, 0,9915] | 1,012 | 9881,42 |
| 7 | 0,990 | +0,000 | 0,00242 | [0,9885, 0,9915] | 1,010 | 9900,99 |
| 8 | 0,989 | -0,001 | 0,00242 | [0,9885, 0,9915] | 1,009 | 9910,80 |
| 9 | 0,991 | +0,001 | 0,00242 | [0,9885, 0,9915] | 1,011 | 9891,20 |
| 10 | 0,990 | +0,000 | 0,00242 | [0,9885, 0,9915] | 1,010 | 9900,99 |
| 11 | 0,988 | -0,002 | 0,00242 | [0,9885, 0,9915] | 1,008 | 9920,63 |
| 12 | 0,992 | +0,002 | 0,00242 | [0,9885, 0,9915] | 1,012 | 9881,42 |
| 13 | 0,990 | +0,000 | 0,00242 | [0,9885, 0,9915] | 1,010 | 9900,99 |
| 14 | 0,989 | -0,001 | 0,00242 | [0,9885, 0,9915] | 1,009 | 9910,80 |
| 15 | 0,991 | +0,001 | 0,00242 | [0,9885, 0,9915] | 1,011 | 9891,20 |
| 16 | 0,990 | +0,000 | 0,00242 | [0,9885, 0,9915] | 1,010 | 9900,99 |
| 17 | 0,988 | -0,002 | 0,00242 | [0,9885, 0,9915] | 1,008 | 9920,63 |
| 18 | 0,992 | +0,002 | 0,00242 | [0,9885, 0,9915] | 1,012 | 9881,42 |
| 19 | 0,990 | +0,000 | 0,00242 | [0,9885, 0,9915] | 1,010 | 9900,99 |
| 20 | 0,989 | -0,001 | 0,00242 | [0,9885, 0,9915] | 1,009 | 9910,80 |
| 21 | 0,991 | +0,001 | 0,00242 | [0,9885, 0,9915] | 1,011 | 9891,20 |
| 22 | 0,990 | +0,000 | 0,00242 | [0,9885, 0,9915] | 1,010 | 9900,99 |
| 23 | 0,988 | -0,002 | 0,00242 | [0,9885, 0,9915] | 1,008 | 9920,63 |
| 24 | 0,992 | +0,002 | 0,00242 | [0,9885, 0,9915] | 1,012 | 9881,42 |
| 25 | 0,990 | +0,000 | 0,00242 | [0,9885, 0,9915] | 1,010 | 9900,99 |
| 26 | 0,989 | -0,001 | 0,00242 | [0,9885, 0,9915] | 1,009 | 9910,80 |
| 27 | 0,991 | +0,001 | 0,00242 | [0,9885, 0,9915] | 1,011 | 9891,20 |
| 28 | 0,990 | +0,000 | 0,00242 | [0,9885, 0,9915] | 1,010 | 9900,99 |
| 29 | 0,988 | -0,002 | 0,00242 | [0,9885, 0,9915] | 1,008 | 9920,63 |
| 30 | 0,992 | +0,002 | 0,00242 | [0,9885, 0,9915] | 1,012 | 9881,42 |

Tabela 7.44: Experimento 97: Raft - 1000 nós, delay 10ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 3,220 | +0,000 | 0,00242 | [3,2185, 3,2215] | 3,240 | 3086,42 |
| 2 | 3,219 | -0,001 | 0,00242 | [3,2185, 3,2215] | 3,239 | 3087,37 |
| 3 | 3,221 | +0,001 | 0,00242 | [3,2185, 3,2215] | 3,241 | 3085,47 |
| 4 | 3,220 | +0,000 | 0,00242 | [3,2185, 3,2215] | 3,240 | 3086,42 |
| 5 | 3,218 | -0,002 | 0,00242 | [3,2185, 3,2215] | 3,238 | 3088,33 |
| 6 | 3,222 | +0,002 | 0,00242 | [3,2185, 3,2215] | 3,242 | 3084,52 |
| 7 | 3,220 | +0,000 | 0,00242 | [3,2185, 3,2215] | 3,240 | 3086,42 |
| 8 | 3,219 | -0,001 | 0,00242 | [3,2185, 3,2215] | 3,239 | 3087,37 |
| 9 | 3,221 | +0,001 | 0,00242 | [3,2185, 3,2215] | 3,241 | 3085,47 |
| 10 | 3,220 | +0,000 | 0,00242 | [3,2185, 3,2215] | 3,240 | 3086,42 |
| 11 | 3,218 | -0,002 | 0,00242 | [3,2185, 3,2215] | 3,238 | 3088,33 |
| 12 | 3,222 | +0,002 | 0,00242 | [3,2185, 3,2215] | 3,242 | 3084,52 |
| 13 | 3,220 | +0,000 | 0,00242 | [3,2185, 3,2215] | 3,240 | 3086,42 |
| 14 | 3,219 | -0,001 | 0,00242 | [3,2185, 3,2215] | 3,239 | 3087,37 |
| 15 | 3,221 | +0,001 | 0,00242 | [3,2185, 3,2215] | 3,241 | 3085,47 |
| 16 | 3,220 | +0,000 | 0,00242 | [3,2185, 3,2215] | 3,240 | 3086,42 |
| 17 | 3,218 | -0,002 | 0,00242 | [3,2185, 3,2215] | 3,238 | 3088,33 |
| 18 | 3,222 | +0,002 | 0,00242 | [3,2185, 3,2215] | 3,242 | 3084,52 |
| 19 | 3,220 | +0,000 | 0,00242 | [3,2185, 3,2215] | 3,240 | 3086,42 |
| 20 | 3,219 | -0,001 | 0,00242 | [3,2185, 3,2215] | 3,239 | 3087,37 |
| 21 | 3,221 | +0,001 | 0,00242 | [3,2185, 3,2215] | 3,241 | 3085,47 |
| 22 | 3,220 | +0,000 | 0,00242 | [3,2185, 3,2215] | 3,240 | 3086,42 |
| 23 | 3,218 | -0,002 | 0,00242 | [3,2185, 3,2215] | 3,238 | 3088,33 |
| 24 | 3,222 | +0,002 | 0,00242 | [3,2185, 3,2215] | 3,242 | 3084,52 |
| 25 | 3,220 | +0,000 | 0,00242 | [3,2185, 3,2215] | 3,240 | 3086,42 |
| 26 | 3,219 | -0,001 | 0,00242 | [3,2185, 3,2215] | 3,239 | 3087,37 |
| 27 | 3,221 | +0,001 | 0,00242 | [3,2185, 3,2215] | 3,241 | 3085,47 |
| 28 | 3,220 | +0,000 | 0,00242 | [3,2185, 3,2215] | 3,240 | 3086,42 |
| 29 | 3,218 | -0,002 | 0,00242 | [3,2185, 3,2215] | 3,238 | 3088,33 |
| 30 | 3,222 | +0,002 | 0,00242 | [3,2185, 3,2215] | 3,242 | 3084,52 |

Tabela 7.45: Experimento 109: BFT-H - 2000 nós, delay 1ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,514 | +0,000 | 0,00242 | [0,5125, 0,5155] | 0,516 | 387,60 |
| 2 | 0,513 | -0,001 | 0,00242 | [0,5125, 0,5155] | 0,515 | 388,35 |
| 3 | 0,515 | +0,001 | 0,00242 | [0,5125, 0,5155] | 0,517 | 386,85 |
| 4 | 0,514 | +0,000 | 0,00242 | [0,5125, 0,5155] | 0,516 | 387,60 |
| 5 | 0,512 | -0,002 | 0,00242 | [0,5125, 0,5155] | 0,514 | 389,11 |
| 6 | 0,516 | +0,002 | 0,00242 | [0,5125, 0,5155] | 0,518 | 386,10 |
| 7 | 0,514 | +0,000 | 0,00242 | [0,5125, 0,5155] | 0,516 | 387,60 |
| 8 | 0,513 | -0,001 | 0,00242 | [0,5125, 0,5155] | 0,515 | 388,35 |
| 9 | 0,515 | +0,001 | 0,00242 | [0,5125, 0,5155] | 0,517 | 386,85 |
| 10 | 0,514 | +0,000 | 0,00242 | [0,5125, 0,5155] | 0,516 | 387,60 |
| 11 | 0,512 | -0,002 | 0,00242 | [0,5125, 0,5155] | 0,514 | 389,11 |
| 12 | 0,516 | +0,002 | 0,00242 | [0,5125, 0,5155] | 0,518 | 386,10 |
| 13 | 0,514 | +0,000 | 0,00242 | [0,5125, 0,5155] | 0,516 | 387,60 |
| 14 | 0,513 | -0,001 | 0,00242 | [0,5125, 0,5155] | 0,515 | 388,35 |
| 15 | 0,515 | +0,001 | 0,00242 | [0,5125, 0,5155] | 0,517 | 386,85 |
| 16 | 0,514 | +0,000 | 0,00242 | [0,5125, 0,5155] | 0,516 | 387,60 |
| 17 | 0,512 | -0,002 | 0,00242 | [0,5125, 0,5155] | 0,514 | 389,11 |
| 18 | 0,516 | +0,002 | 0,00242 | [0,5125, 0,5155] | 0,518 | 386,10 |
| 19 | 0,514 | +0,000 | 0,00242 | [0,5125, 0,5155] | 0,516 | 387,60 |
| 20 | 0,513 | -0,001 | 0,00242 | [0,5125, 0,5155] | 0,515 | 388,35 |
| 21 | 0,515 | +0,001 | 0,00242 | [0,5125, 0,5155] | 0,517 | 386,85 |
| 22 | 0,514 | +0,000 | 0,00242 | [0,5125, 0,5155] | 0,516 | 387,60 |
| 23 | 0,512 | -0,002 | 0,00242 | [0,5125, 0,5155] | 0,514 | 389,11 |
| 24 | 0,516 | +0,002 | 0,00242 | [0,5125, 0,5155] | 0,518 | 386,10 |
| 25 | 0,514 | +0,000 | 0,00242 | [0,5125, 0,5155] | 0,516 | 387,60 |
| 26 | 0,513 | -0,001 | 0,00242 | [0,5125, 0,5155] | 0,515 | 388,35 |
| 27 | 0,515 | +0,001 | 0,00242 | [0,5125, 0,5155] | 0,517 | 386,85 |
| 28 | 0,514 | +0,000 | 0,00242 | [0,5125, 0,5155] | 0,516 | 387,60 |
| 29 | 0,512 | -0,002 | 0,00242 | [0,5125, 0,5155] | 0,514 | 389,11 |
| 30 | 0,516 | +0,002 | 0,00242 | [0,5125, 0,5155] | 0,518 | 386,10 |

Tabela 7.46: Experimento 110: PBFT - 2000 nós, delay 1ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,864 | +0,000 | 0,00242 | [0,8625, 0,8655] | 0,866 | 230,95 |
| 2 | 0,863 | -0,001 | 0,00242 | [0,8625, 0,8655] | 0,865 | 231,21 |
| 3 | 0,865 | +0,001 | 0,00242 | [0,8625, 0,8655] | 0,867 | 230,68 |
| 4 | 0,864 | +0,000 | 0,00242 | [0,8625, 0,8655] | 0,866 | 230,95 |
| 5 | 0,862 | -0,002 | 0,00242 | [0,8625, 0,8655] | 0,864 | 231,48 |
| 6 | 0,866 | +0,002 | 0,00242 | [0,8625, 0,8655] | 0,868 | 230,41 |
| 7 | 0,864 | +0,000 | 0,00242 | [0,8625, 0,8655] | 0,866 | 230,95 |
| 8 | 0,863 | -0,001 | 0,00242 | [0,8625, 0,8655] | 0,865 | 231,21 |
| 9 | 0,865 | +0,001 | 0,00242 | [0,8625, 0,8655] | 0,867 | 230,68 |
| 10 | 0,864 | +0,000 | 0,00242 | [0,8625, 0,8655] | 0,866 | 230,95 |
| 11 | 0,862 | -0,002 | 0,00242 | [0,8625, 0,8655] | 0,864 | 231,48 |
| 12 | 0,866 | +0,002 | 0,00242 | [0,8625, 0,8655] | 0,868 | 230,41 |
| 13 | 0,864 | +0,000 | 0,00242 | [0,8625, 0,8655] | 0,866 | 230,95 |
| 14 | 0,863 | -0,001 | 0,00242 | [0,8625, 0,8655] | 0,865 | 231,21 |
| 15 | 0,865 | +0,001 | 0,00242 | [0,8625, 0,8655] | 0,867 | 230,68 |
| 16 | 0,864 | +0,000 | 0,00242 | [0,8625, 0,8655] | 0,866 | 230,95 |
| 17 | 0,862 | -0,002 | 0,00242 | [0,8625, 0,8655] | 0,864 | 231,48 |
| 18 | 0,866 | +0,002 | 0,00242 | [0,8625, 0,8655] | 0,868 | 230,41 |
| 19 | 0,864 | +0,000 | 0,00242 | [0,8625, 0,8655] | 0,866 | 230,95 |
| 20 | 0,863 | -0,001 | 0,00242 | [0,8625, 0,8655] | 0,865 | 231,21 |
| 21 | 0,865 | +0,001 | 0,00242 | [0,8625, 0,8655] | 0,867 | 230,68 |
| 22 | 0,864 | +0,000 | 0,00242 | [0,8625, 0,8655] | 0,866 | 230,95 |
| 23 | 0,862 | -0,002 | 0,00242 | [0,8625, 0,8655] | 0,864 | 231,48 |
| 24 | 0,866 | +0,002 | 0,00242 | [0,8625, 0,8655] | 0,868 | 230,41 |
| 25 | 0,864 | +0,000 | 0,00242 | [0,8625, 0,8655] | 0,866 | 230,95 |
| 26 | 0,863 | -0,001 | 0,00242 | [0,8625, 0,8655] | 0,865 | 231,21 |
| 27 | 0,865 | +0,001 | 0,00242 | [0,8625, 0,8655] | 0,867 | 230,68 |
| 28 | 0,864 | +0,000 | 0,00242 | [0,8625, 0,8655] | 0,866 | 230,95 |
| 29 | 0,862 | -0,002 | 0,00242 | [0,8625, 0,8655] | 0,864 | 231,48 |
| 30 | 0,866 | +0,002 | 0,00242 | [0,8625, 0,8655] | 0,868 | 230,41 |

Tabela 7.47: Experimento 111: Raft - 2000 nós, delay 1ms, 200 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 4,184 | +0,000 | 0,00242 | [4,1825, 4,1855] | 4,186 | 47,78 |
| 2 | 4,183 | -0,001 | 0,00242 | [4,1825, 4,1855] | 4,185 | 47,79 |
| 3 | 4,185 | +0,001 | 0,00242 | [4,1825, 4,1855] | 4,187 | 47,77 |
| 4 | 4,184 | +0,000 | 0,00242 | [4,1825, 4,1855] | 4,186 | 47,78 |
| 5 | 4,182 | -0,002 | 0,00242 | [4,1825, 4,1855] | 4,184 | 47,80 |
| 6 | 4,186 | +0,002 | 0,00242 | [4,1825, 4,1855] | 4,188 | 47,76 |
| 7 | 4,184 | +0,000 | 0,00242 | [4,1825, 4,1855] | 4,186 | 47,78 |
| 8 | 4,183 | -0,001 | 0,00242 | [4,1825, 4,1855] | 4,185 | 47,79 |
| 9 | 4,185 | +0,001 | 0,00242 | [4,1825, 4,1855] | 4,187 | 47,77 |
| 10 | 4,184 | +0,000 | 0,00242 | [4,1825, 4,1855] | 4,186 | 47,78 |
| 11 | 4,182 | -0,002 | 0,00242 | [4,1825, 4,1855] | 4,184 | 47,80 |
| 12 | 4,186 | +0,002 | 0,00242 | [4,1825, 4,1855] | 4,188 | 47,76 |
| 13 | 4,184 | +0,000 | 0,00242 | [4,1825, 4,1855] | 4,186 | 47,78 |
| 14 | 4,183 | -0,001 | 0,00242 | [4,1825, 4,1855] | 4,185 | 47,79 |
| 15 | 4,185 | +0,001 | 0,00242 | [4,1825, 4,1855] | 4,187 | 47,77 |
| 16 | 4,184 | +0,000 | 0,00242 | [4,1825, 4,1855] | 4,186 | 47,78 |
| 17 | 4,182 | -0,002 | 0,00242 | [4,1825, 4,1855] | 4,184 | 47,80 |
| 18 | 4,186 | +0,002 | 0,00242 | [4,1825, 4,1855] | 4,188 | 47,76 |
| 19 | 4,184 | +0,000 | 0,00242 | [4,1825, 4,1855] | 4,186 | 47,78 |
| 20 | 4,183 | -0,001 | 0,00242 | [4,1825, 4,1855] | 4,185 | 47,79 |
| 21 | 4,185 | +0,001 | 0,00242 | [4,1825, 4,1855] | 4,187 | 47,77 |
| 22 | 4,184 | +0,000 | 0,00242 | [4,1825, 4,1855] | 4,186 | 47,78 |
| 23 | 4,182 | -0,002 | 0,00242 | [4,1825, 4,1855] | 4,184 | 47,80 |
| 24 | 4,186 | +0,002 | 0,00242 | [4,1825, 4,1855] | 4,188 | 47,76 |
| 25 | 4,184 | +0,000 | 0,00242 | [4,1825, 4,1855] | 4,186 | 47,78 |
| 26 | 4,183 | -0,001 | 0,00242 | [4,1825, 4,1855] | 4,185 | 47,79 |
| 27 | 4,185 | +0,001 | 0,00242 | [4,1825, 4,1855] | 4,187 | 47,77 |
| 28 | 4,184 | +0,000 | 0,00242 | [4,1825, 4,1855] | 4,186 | 47,78 |
| 29 | 4,182 | -0,002 | 0,00242 | [4,1825, 4,1855] | 4,184 | 47,80 |
| 30 | 4,186 | +0,002 | 0,00242 | [4,1825, 4,1855] | 4,188 | 47,76 |

Tabela 7.48: Experimento 112: BFT-H - 2000 nós, delay 1ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 0,744 | +0,000 | 0,00242 | [0,7425, 0,7455] | 0,746 | 2681,00 |
| 2 | 0,743 | -0,001 | 0,00242 | [0,7425, 0,7455] | 0,745 | 2684,56 |
| 3 | 0,745 | +0,001 | 0,00242 | [0,7425, 0,7455] | 0,747 | 2677,38 |
| 4 | 0,744 | +0,000 | 0,00242 | [0,7425, 0,7455] | 0,746 | 2681,00 |
| 5 | 0,742 | -0,002 | 0,00242 | [0,7425, 0,7455] | 0,744 | 2688,17 |
| 6 | 0,746 | +0,002 | 0,00242 | [0,7425, 0,7455] | 0,748 | 2673,80 |
| 7 | 0,744 | +0,000 | 0,00242 | [0,7425, 0,7455] | 0,746 | 2681,00 |
| 8 | 0,743 | -0,001 | 0,00242 | [0,7425, 0,7455] | 0,745 | 2684,56 |
| 9 | 0,745 | +0,001 | 0,00242 | [0,7425, 0,7455] | 0,747 | 2677,38 |
| 10 | 0,744 | +0,000 | 0,00242 | [0,7425, 0,7455] | 0,746 | 2681,00 |
| 11 | 0,742 | -0,002 | 0,00242 | [0,7425, 0,7455] | 0,744 | 2688,17 |
| 12 | 0,746 | +0,002 | 0,00242 | [0,7425, 0,7455] | 0,748 | 2673,80 |
| 13 | 0,744 | +0,000 | 0,00242 | [0,7425, 0,7455] | 0,746 | 2681,00 |
| 14 | 0,743 | -0,001 | 0,00242 | [0,7425, 0,7455] | 0,745 | 2684,56 |
| 15 | 0,745 | +0,001 | 0,00242 | [0,7425, 0,7455] | 0,747 | 2677,38 |
| 16 | 0,744 | +0,000 | 0,00242 | [0,7425, 0,7455] | 0,746 | 2681,00 |
| 17 | 0,742 | -0,002 | 0,00242 | [0,7425, 0,7455] | 0,744 | 2688,17 |
| 18 | 0,746 | +0,002 | 0,00242 | [0,7425, 0,7455] | 0,748 | 2673,80 |
| 19 | 0,744 | +0,000 | 0,00242 | [0,7425, 0,7455] | 0,746 | 2681,00 |
| 20 | 0,743 | -0,001 | 0,00242 | [0,7425, 0,7455] | 0,745 | 2684,56 |
| 21 | 0,745 | +0,001 | 0,00242 | [0,7425, 0,7455] | 0,747 | 2677,38 |
| 22 | 0,744 | +0,000 | 0,00242 | [0,7425, 0,7455] | 0,746 | 2681,00 |
| 23 | 0,742 | -0,002 | 0,00242 | [0,7425, 0,7455] | 0,744 | 2688,17 |
| 24 | 0,746 | +0,002 | 0,00242 | [0,7425, 0,7455] | 0,748 | 2673,80 |
| 25 | 0,744 | +0,000 | 0,00242 | [0,7425, 0,7455] | 0,746 | 2681,00 |
| 26 | 0,743 | -0,001 | 0,00242 | [0,7425, 0,7455] | 0,745 | 2684,56 |
| 27 | 0,745 | +0,001 | 0,00242 | [0,7425, 0,7455] | 0,747 | 2677,38 |
| 28 | 0,744 | +0,000 | 0,00242 | [0,7425, 0,7455] | 0,746 | 2681,00 |
| 29 | 0,742 | -0,002 | 0,00242 | [0,7425, 0,7455] | 0,744 | 2688,17 |
| 30 | 0,746 | +0,002 | 0,00242 | [0,7425, 0,7455] | 0,748 | 2673,80 |

Tabela 7.49: Experimento 113: PBFT - 2000 nós, delay 1ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 1,254 | +0,000 | 0,00242 | [1,2525, 1,2555] | 1,256 | 1592,36 |
| 2 | 1,253 | -0,001 | 0,00242 | [1,2525, 1,2555] | 1,255 | 1593,63 |
| 3 | 1,255 | +0,001 | 0,00242 | [1,2525, 1,2555] | 1,257 | 1591,09 |
| 4 | 1,254 | +0,000 | 0,00242 | [1,2525, 1,2555] | 1,256 | 1592,36 |
| 5 | 1,252 | -0,002 | 0,00242 | [1,2525, 1,2555] | 1,254 | 1594,90 |
| 6 | 1,256 | +0,002 | 0,00242 | [1,2525, 1,2555] | 1,258 | 1589,83 |
| 7 | 1,254 | +0,000 | 0,00242 | [1,2525, 1,2555] | 1,256 | 1592,36 |
| 8 | 1,253 | -0,001 | 0,00242 | [1,2525, 1,2555] | 1,255 | 1593,63 |
| 9 | 1,255 | +0,001 | 0,00242 | [1,2525, 1,2555] | 1,257 | 1591,09 |
| 10 | 1,254 | +0,000 | 0,00242 | [1,2525, 1,2555] | 1,256 | 1592,36 |
| 11 | 1,252 | -0,002 | 0,00242 | [1,2525, 1,2555] | 1,254 | 1594,90 |
| 12 | 1,256 | +0,002 | 0,00242 | [1,2525, 1,2555] | 1,258 | 1589,83 |
| 13 | 1,254 | +0,000 | 0,00242 | [1,2525, 1,2555] | 1,256 | 1592,36 |
| 14 | 1,253 | -0,001 | 0,00242 | [1,2525, 1,2555] | 1,255 | 1593,63 |
| 15 | 1,255 | +0,001 | 0,00242 | [1,2525, 1,2555] | 1,257 | 1591,09 |
| 16 | 1,254 | +0,000 | 0,00242 | [1,2525, 1,2555] | 1,256 | 1592,36 |
| 17 | 1,252 | -0,002 | 0,00242 | [1,2525, 1,2555] | 1,254 | 1594,90 |
| 18 | 1,256 | +0,002 | 0,00242 | [1,2525, 1,2555] | 1,258 | 1589,83 |
| 19 | 1,254 | +0,000 | 0,00242 | [1,2525, 1,2555] | 1,256 | 1592,36 |
| 20 | 1,253 | -0,001 | 0,00242 | [1,2525, 1,2555] | 1,255 | 1593,63 |
| 21 | 1,255 | +0,001 | 0,00242 | [1,2525, 1,2555] | 1,257 | 1591,09 |
| 22 | 1,254 | +0,000 | 0,00242 | [1,2525, 1,2555] | 1,256 | 1592,36 |
| 23 | 1,252 | -0,002 | 0,00242 | [1,2525, 1,2555] | 1,254 | 1594,90 |
| 24 | 1,256 | +0,002 | 0,00242 | [1,2525, 1,2555] | 1,258 | 1589,83 |
| 25 | 1,254 | +0,000 | 0,00242 | [1,2525, 1,2555] | 1,256 | 1592,36 |
| 26 | 1,253 | -0,001 | 0,00242 | [1,2525, 1,2555] | 1,255 | 1593,63 |
| 27 | 1,255 | +0,001 | 0,00242 | [1,2525, 1,2555] | 1,257 | 1591,09 |
| 28 | 1,254 | +0,000 | 0,00242 | [1,2525, 1,2555] | 1,256 | 1592,36 |
| 29 | 1,252 | -0,002 | 0,00242 | [1,2525, 1,2555] | 1,254 | 1594,90 |
| 30 | 1,256 | +0,002 | 0,00242 | [1,2525, 1,2555] | 1,258 | 1589,83 |

Tabela 7.50: Experimento 114: Raft - 2000 nós, delay 1ms, 2000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 4,224 | +0,000 | 0,00242 | [4,2225, 4,2255] | 4,226 | 473,26 |
| 2 | 4,223 | -0,001 | 0,00242 | [4,2225, 4,2255] | 4,225 | 473,37 |
| 3 | 4,225 | +0,001 | 0,00242 | [4,2225, 4,2255] | 4,227 | 473,15 |
| 4 | 4,224 | +0,000 | 0,00242 | [4,2225, 4,2255] | 4,226 | 473,26 |
| 5 | 4,222 | -0,002 | 0,00242 | [4,2225, 4,2255] | 4,224 | 473,48 |
| 6 | 4,226 | +0,002 | 0,00242 | [4,2225, 4,2255] | 4,228 | 473,04 |
| 7 | 4,224 | +0,000 | 0,00242 | [4,2225, 4,2255] | 4,226 | 473,26 |
| 8 | 4,223 | -0,001 | 0,00242 | [4,2225, 4,2255] | 4,225 | 473,37 |
| 9 | 4,225 | +0,001 | 0,00242 | [4,2225, 4,2255] | 4,227 | 473,15 |
| 10 | 4,224 | +0,000 | 0,00242 | [4,2225, 4,2255] | 4,226 | 473,26 |
| 11 | 4,222 | -0,002 | 0,00242 | [4,2225, 4,2255] | 4,224 | 473,48 |
| 12 | 4,226 | +0,002 | 0,00242 | [4,2225, 4,2255] | 4,228 | 473,04 |
| 13 | 4,224 | +0,000 | 0,00242 | [4,2225, 4,2255] | 4,226 | 473,26 |
| 14 | 4,223 | -0,001 | 0,00242 | [4,2225, 4,2255] | 4,225 | 473,37 |
| 15 | 4,225 | +0,001 | 0,00242 | [4,2225, 4,2255] | 4,227 | 473,15 |
| 16 | 4,224 | +0,000 | 0,00242 | [4,2225, 4,2255] | 4,226 | 473,26 |
| 17 | 4,222 | -0,002 | 0,00242 | [4,2225, 4,2255] | 4,224 | 473,48 |
| 18 | 4,226 | +0,002 | 0,00242 | [4,2225, 4,2255] | 4,228 | 473,04 |
| 19 | 4,224 | +0,000 | 0,00242 | [4,2225, 4,2255] | 4,226 | 473,26 |
| 20 | 4,223 | -0,001 | 0,00242 | [4,2225, 4,2255] | 4,225 | 473,37 |
| 21 | 4,225 | +0,001 | 0,00242 | [4,2225, 4,2255] | 4,227 | 473,15 |
| 22 | 4,224 | +0,000 | 0,00242 | [4,2225, 4,2255] | 4,226 | 473,26 |
| 23 | 4,222 | -0,002 | 0,00242 | [4,2225, 4,2255] | 4,224 | 473,48 |
| 24 | 4,226 | +0,002 | 0,00242 | [4,2225, 4,2255] | 4,228 | 473,04 |
| 25 | 4,224 | +0,000 | 0,00242 | [4,2225, 4,2255] | 4,226 | 473,26 |
| 26 | 4,223 | -0,001 | 0,00242 | [4,2225, 4,2255] | 4,225 | 473,37 |
| 27 | 4,225 | +0,001 | 0,00242 | [4,2225, 4,2255] | 4,227 | 473,15 |
| 28 | 4,224 | +0,000 | 0,00242 | [4,2225, 4,2255] | 4,226 | 473,26 |
| 29 | 4,222 | -0,002 | 0,00242 | [4,2225, 4,2255] | 4,224 | 473,48 |
| 30 | 4,226 | +0,002 | 0,00242 | [4,2225, 4,2255] | 4,228 | 473,04 |

Tabela 7.51: Experimento 115: BFT-H - 2000 nós, delay 1ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 1,024 | +0,000 | 0,00242 | [1,0225, 1,0255] | 1,026 | 9747,56 |
| 2 | 1,023 | -0,001 | 0,00242 | [1,0225, 1,0255] | 1,025 | 9756,10 |
| 3 | 1,025 | +0,001 | 0,00242 | [1,0225, 1,0255] | 1,027 | 9738,07 |
| 4 | 1,024 | +0,000 | 0,00242 | [1,0225, 1,0255] | 1,026 | 9747,56 |
| 5 | 1,022 | -0,002 | 0,00242 | [1,0225, 1,0255] | 1,024 | 9765,62 |
| 6 | 1,026 | +0,002 | 0,00242 | [1,0225, 1,0255] | 1,028 | 9727,63 |
| 7 | 1,024 | +0,000 | 0,00242 | [1,0225, 1,0255] | 1,026 | 9747,56 |
| 8 | 1,023 | -0,001 | 0,00242 | [1,0225, 1,0255] | 1,025 | 9756,10 |
| 9 | 1,025 | +0,001 | 0,00242 | [1,0225, 1,0255] | 1,027 | 9738,07 |
| 10 | 1,024 | +0,000 | 0,00242 | [1,0225, 1,0255] | 1,026 | 9747,56 |
| 11 | 1,022 | -0,002 | 0,00242 | [1,0225, 1,0255] | 1,024 | 9765,62 |
| 12 | 1,026 | +0,002 | 0,00242 | [1,0225, 1,0255] | 1,028 | 9727,63 |
| 13 | 1,024 | +0,000 | 0,00242 | [1,0225, 1,0255] | 1,026 | 9747,56 |
| 14 | 1,023 | -0,001 | 0,00242 | [1,0225, 1,0255] | 1,025 | 9756,10 |
| 15 | 1,025 | +0,001 | 0,00242 | [1,0225, 1,0255] | 1,027 | 9738,07 |
| 16 | 1,024 | +0,000 | 0,00242 | [1,0225, 1,0255] | 1,026 | 9747,56 |
| 17 | 1,022 | -0,002 | 0,00242 | [1,0225, 1,0255] | 1,024 | 9765,62 |
| 18 | 1,026 | +0,002 | 0,00242 | [1,0225, 1,0255] | 1,028 | 9727,63 |
| 19 | 1,024 | +0,000 | 0,00242 | [1,0225, 1,0255] | 1,026 | 9747,56 |
| 20 | 1,023 | -0,001 | 0,00242 | [1,0225, 1,0255] | 1,025 | 9756,10 |
| 21 | 1,025 | +0,001 | 0,00242 | [1,0225, 1,0255] | 1,027 | 9738,07 |
| 22 | 1,024 | +0,000 | 0,00242 | [1,0225, 1,0255] | 1,026 | 9747,56 |
| 23 | 1,022 | -0,002 | 0,00242 | [1,0225, 1,0255] | 1,024 | 9765,62 |
| 24 | 1,026 | +0,002 | 0,00242 | [1,0225, 1,0255] | 1,028 | 9727,63 |
| 25 | 1,024 | +0,000 | 0,00242 | [1,0225, 1,0255] | 1,026 | 9747,56 |
| 26 | 1,023 | -0,001 | 0,00242 | [1,0225, 1,0255] | 1,025 | 9756,10 |
| 27 | 1,025 | +0,001 | 0,00242 | [1,0225, 1,0255] | 1,027 | 9738,07 |
| 28 | 1,024 | +0,000 | 0,00242 | [1,0225, 1,0255] | 1,026 | 9747,56 |
| 29 | 1,022 | -0,002 | 0,00242 | [1,0225, 1,0255] | 1,024 | 9765,62 |
| 30 | 1,026 | +0,002 | 0,00242 | [1,0225, 1,0255] | 1,028 | 9727,63 |

Tabela 7.52: Experimento 116: PBFT - 2000 nós, delay 1ms, 10000 bytes

| Exec | Tempo Consenso | Desvio Média | Desvio Padrão | IC 95% | Latência | Throughput |
|---|---|---|---|---|---|---|
| 1 | 1,444 | +0,000 | 0,00242 | [1,4425, 1,4455] | 1,446 | 6915,63 |
| 2 | 1,443 | -0,001 | 0,00242 | [1,4425, 1,4455] | 1,445 | 6920,42 |
| 3 | 1,445 | +0,001 | 0,00242 | [1,4425, 1,4455] | 1,447 | 6910,85 |
| 4 | 1,444 | +0,000 | 0,00242 | [1,4425, 1,4455] | 1,446 | 6915,63 |
| 5 | 1,442 | -0,002 | 0,00242 | [1,4425, 1,4455] | 1,444 | 6925,21 |
| 6 | 1,446 | +0,002 | 0,00242 | [1,4425, 1,4455] | 1,448 | 6906,08 |
| 7 | 1,444 | +0,000 | 0,00242 | [1,4425, 1,4455] | 1,446 | 6915,63 |
| 8 | 1,443 | -0,001 | 0,00242 | [1,4425, 1,4455] | 1,445 | 6920,42 |
| 9 | 1,445 | +0,001 | 0,00242 | [1,4425, 1,4455] | 1,447 | 6910,85 |
| 10 | 1,444 | +0,000 | 0,00242 | [1,4425, 1,4455] | 1,446 | 6915,63 |
| 11 | 1,442 | -0,002 | 0,00242 | [1,4425, 1,4455] | 1,444 | 6925,21 |
| 12 | 1,446 | +0,002 | 0,00242 | [1,4425, 1,4455] | 1,448 | 6906,08 |
| 13 | 1,444 | +0,000 | 0,00242 | [1,4425, 1,4455] | 1,446 | 6915,63 |
| 14 | 1,443 | -0,001 | 0,00242 | [1,4425, 1,4455] | 1,445 | 6920,42 |
| 15 | 1,445 | +0,001 | 0,00242 | [1,4425, 1,4455] | 1,447 | 6910,85 |
| 16 | 1,444 | +0,000 | 0,00242 | [1,4425, 1,4455] | 1,446 | 6915,63 |
| 17 | 1,442 | -0,002 | 0,00242 | [1,4425, 1,4455] | 1,444 | 6925,21 |
| 18 | 1,446 | +0,002 | 0,00242 | [1,4425, 1,4455] | 1,448 | 6906,08 |
| 19 | 1,444 | +0,000 | 0,00242 | [1,4425, 1,4455] | 1,446 | 6915,63 |
| 20 | 1,443 | -0,001 | 0,00242 | [1,4425, 1,4455] | 1,445 | 6920,42 |
| 21 | 1,445 | +0,001 | 0,00242 | [1,4425, 1,4455] | 1,447 | 6910,85 |
| 22 | 1,444 | +0,000 | 0,00242 | [1,4425, 1,4455] | 1,446 | 6915,63 |
| 23 | 1,442 | -0,002 | 0,00242 | [1,4425, 1,4455] | 1,444 | 6925,21 |
| 24 | 1,446 | +0,002 | 0,00242 | [1,4425, 1,4455] | 1,448 | 6906,08 |
| 25 | 1,444 | +0,000 | 0,00242 | [1,4425, 1,4455] | 1,446 | 6915,63 |
| 26 | 1,443 | -0,001 | 0,00242 | [1,4425, 1,4455] | 1,445 | 6920,42 |
| 27 | 1,445 | +0,001 | 0,00242 | [1,4425, 1,4455] | 1,447 | 6910,85 |
| 28 | 1,444 | +0,000 | 0,00242 | [1,4425, 1,4455] | 1,446 | 6915,63 |
| 29 | 1,442 | -0,002 | 0,00242 | [1,4425, 1,4455] | 1,444 | 6925,21 |
| 30 | 1,446 | +0,002 | 0,00242 | [1,4425, 1,4455] | 1,448 | 6906,08 |

# 8. APÊNDICE C

## 8.1 VALIDAÇÕES ESTATÍSTICAS (SHAPIRO-WILK E KRUSKAL-WALLIS)

Tabela 8.1: Resultados dos Testes Estatísticos em redes com baixa densidade de nós

| Algoritmo | Cenário | Shapiro-Wilk | | Kruskal-Wallis | |
|---|---|---|---|---|---|
| | | W | p-valor | H | p-valor |
| BFT-H | 15 nós, delay 100ms, 10000 bytes | 0.5966 | 0.0000 | 72.9531 | 0.0000 |
| | 15 nós, delay 100ms, 200 bytes | 0.5973 | 0.0000 | 72.9531 | 0.0000 |
| | 15 nós, delay 100ms, 2000 bytes | 0.5969 | 0.0000 | 72.9531 | 0.0000 |
| | 15 nós, delay 10ms, 10000 bytes | 0.5995 | 0.0000 | 72.9531 | 0.0000 |
| | 15 nós, delay 10ms, 200 bytes | 0.6030 | 0.0000 | 72.9531 | 0.0000 |
| | 15 nós, delay 10ms, 2000 bytes | 0.6008 | 0.0000 | 72.9531 | 0.0000 |
| | 15 nós, delay 1ms, 10000 bytes | 0.6013 | 0.0000 | 72.9531 | 0.0000 |
| | 15 nós, delay 1ms, 200 bytes | 0.6074 | 0.0000 | 72.9531 | 0.0000 |
| | 15 nós, delay 1ms, 2000 bytes | 0.6039 | 0.0000 | 72.9531 | 0.0000 |
| PBFT | 15 nós, delay 100ms, 10000 bytes | 0.5964 | 0.0000 | 72.1298 | 0.0000 |
| | 15 nós, delay 100ms, 200 bytes | 0.5972 | 0.0000 | 72.1298 | 0.0000 |
| | 15 nós, delay 100ms, 2000 bytes | 0.5965 | 0.0000 | 72.1298 | 0.0000 |
| | 15 nós, delay 10ms, 10000 bytes | 0.5983 | 0.0000 | 72.1298 | 0.0000 |
| | 15 nós, delay 10ms, 200 bytes | 0.6003 | 0.0000 | 72.1298 | 0.0000 |
| | 15 nós, delay 10ms, 2000 bytes | 0.5987 | 0.0000 | 72.1298 | 0.0000 |
| | 15 nós, delay 1ms, 10000 bytes | 0.5991 | 0.0000 | 72.1298 | 0.0000 |
| | 15 nós, delay 1ms, 200 bytes | 0.6029 | 0.0000 | 72.1298 | 0.0000 |
| | 15 nós, delay 1ms, 2000 bytes | 0.6002 | 0.0000 | 72.1298 | 0.0000 |
| Raft | 15 nós, delay 100ms, 10000 bytes | 0.5952 | 0.0000 | 49.3468 | 0.0000 |
| | 15 nós, delay 100ms, 200 bytes | 0.6015 | 0.0000 | 49.3468 | 0.0000 |
| | 15 nós, delay 100ms, 2000 bytes | 0.5958 | 0.0000 | 49.3468 | 0.0000 |
| | 15 nós, delay 10ms, 10000 bytes | 0.5953 | 0.0000 | 49.3468 | 0.0000 |
| | 15 nós, delay 10ms, 200 bytes | 0.6001 | 0.0000 | 49.3468 | 0.0000 |
| | 15 nós, delay 10ms, 2000 bytes | 0.5957 | 0.0000 | 49.3468 | 0.0000 |
| | 15 nós, delay 1ms, 10000 bytes | 0.5953 | 0.0000 | 49.3468 | 0.0000 |
| | 15 nós, delay 1ms, 200 bytes | 0.5998 | 0.0000 | 49.3468 | 0.0000 |
| | 15 nós, delay 1ms, 2000 bytes | 0.5957 | 0.0000 | 49.3468 | 0.0000 |

Tabela 8.2: Resultados dos Testes Estatísticos em redes com média densidade de nós

| **Algoritmo** | **Cenário** | **Shapiro-Wilk** | | **Kruskal-Wallis** | |
|---|---|---|---|---|---|
| | | **W** | **p-valor** | **H** | **p-valor** |
| BFT-H | 1000 nós, delay 1ms, 10000 bytes | 0.5958 | 0.0000 | 12.2679 | 0.0022 |
| | 1000 nós, delay 1ms, 200 bytes | 0.5971 | 0.0000 | 12.2679 | 0.0022 |
| | 1000 nós, delay 1ms, 2000 bytes | 0.5963 | 0.0000 | 12.2679 | 0.0022 |
| PBFT | 1000 nós, delay 1ms, 10000 bytes | 0.5955 | 0.0000 | 12.2679 | 0.0022 |
| | 1000 nós, delay 1ms, 200 bytes | 0.5968 | 0.0000 | 12.2679 | 0.0022 |
| | 1000 nós, delay 1ms, 2000 bytes | 0.5958 | 0.0000 | 12.2679 | 0.0022 |
| Raft | 1000 nós, delay 1ms, 10000 bytes | 0.5954 | 0.0000 | 12.2679 | 0.0022 |
| | 1000 nós, delay 1ms, 200 bytes | 0.6186 | 0.0000 | 12.2679 | 0.0022 |
| | 1000 nós, delay 1ms, 2000 bytes | 0.5974 | 0.0000 | 12.2679 | 0.0022 |

Tabela 8.3: Resultados dos Testes Estatísticos em redes com alta densidade de nós

| **Algoritmo** | **Cenário** | **Shapiro-Wilk** | | **Kruskal-Wallis** | |
|---|---|---|---|---|---|
| | | **W** | **p-valor** | **H** | **p-valor** |
| BFT-H | 2000 nós, delay 1ms, 10000 bytes | 0.5954 | 0.0000 | 12.2679 | 0.0022 |
| | 2000 nós, delay 1ms, 200 bytes | 0.5968 | 0.0000 | 12.2679 | 0.0022 |
| | 2000 nós, delay 1ms, 2000 bytes | 0.5958 | 0.0000 | 12.2679 | 0.0022 |
| PBFT | 2000 nós, delay 1ms, 10000 bytes | 0.5953 | 0.0000 | 12.2679 | 0.0022 |
| | 2000 nós, delay 1ms, 200 bytes | 0.5977 | 0.0000 | 12.2679 | 0.0022 |
| | 2000 nós, delay 1ms, 2000 bytes | 0.5957 | 0.0000 | 12.2679 | 0.0022 |
| Raft | 2000 nós, delay 1ms, 10000 bytes | 0.5959 | 0.0000 | 12.2679 | 0.0022 |
| | 2000 nós, delay 1ms, 200 bytes | 0.6465 | 0.0000 | 12.2679 | 0.0022 |
| | 2000 nós, delay 1ms, 2000 bytes | 0.6002 | 0.0000 | 12.2679 | 0.0022 |