

Relatório — Resolução de Fork

Aluno: Leonardo Pacheco

Testei localmente com Docker (2 containers: `blockchain_node_a` e `blockchain_node_b`) e tbm em outro computador, após conseguir acesso de uma empresa.

1) Causa do erro

O problema é um fork: dois nós produziram blocos válidos com o mesmo índice quase ao mesmo tempo. Em redes P2P descentralizadas isso é esperado sempre que há latência ou mineração concorrente. Nome técnico da pesquisa: "bloqueio concorrente" ou "blockchain fork".

Por que acontece aqui:

- Cada nó valida e aceita blocos localmente antes de sincronizar com peers.
 - Quando dois blocos válidos (mesmo `prev_hash` e mesmo índice) são gerados quase simultaneamente, cada nó passa a ter uma cadeia diferente.
-

2) Métodos comuns para resolver forks

- Longest chain rule (vencedora = cadeia mais longa) — simples, comum em blockchains educacionais.
 - Proof-of-Work cumulative difficulty — usado aqui; soma o trabalho total, não só comprimento.
 - GHOST / heaviest-subtree — alternativa que usa topologia da árvore.
 - Finalidade probabilística — aguardar N confirmações antes de confiar na transação.
-

3) Solução escolhida e detalhes

Escolhi dificuldade cumulativa (simples de implementar).

- `consensus.py`: nova lógica para calcular dificuldade cumulativa e comparar cadeias. Funções principais: - `calculate_cumulative_difficulty(chain)` — soma $2^{\text{leading_zeros}}$ por bloco; - `compare_chains(a,b)` — retorna qual cadeia tem mais trabalho; - `should_reorganize(current, candidate)` — decide se a reorganização é necessária.
 - `network.py`: ao receber bloco que não encaixa (possível fork), o nó aciona `sync_with_peers()` que pede a cadeia do peer e usa `should_reorganize()` para decidir se adota a cadeia do peer.
 - `chain.py`: integra chamadas de broadcast e sincronização;
 - `mine_block()` continua gerando coinbase automaticamente.
-

4) Resultados novos

Após aplicar/usar a implementação por dificuldade cumulativa, os testes demonstram:

- Forks ainda podem ocorrer, mas são resolvidos automaticamente quando os nós sincronizam.
- Exemplo:

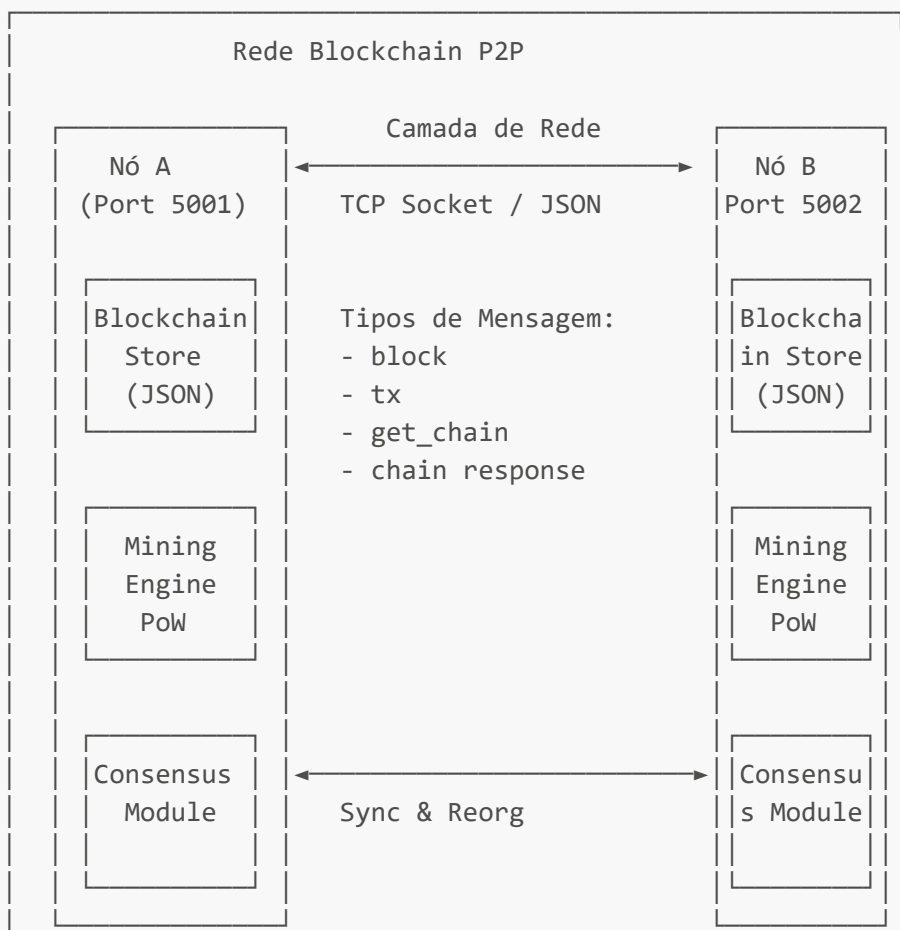
```
[FORK] Potential fork detected from ('172.18.0.3', 36572)
[SYNC] Sent chain to ('172.18.0.3', 36586)
[REORG] Reorganizing chain from peer blockchain_node_X
[REORG] Old chain length: 7, New chain length: 8
```

- Teste real, outra maquina:

```
5. Exit
> 4
[SERVER] Listening on 0.0.0.0:6004
[FORK] Potential fork detected from ('172.30.234.208', 63504). Block index: 20, Expected: 18
[SYNC] Synchronizing with peers to resolve fork...
[SYNC] Sent chain to ('172.30.234.208', 63505)
[REORG] Reorganizing chain from peer 172.30.234.208
[REORG] Old chain length: 18, New chain length: 21
Node ID: 1
```

5) Arquitetura (sistema e software)

- Diagrama de sistema: dois nós (containers/hosts) conectados por rede P2P, portas TCP expostas.
- Diagrama de software: módulos `main.py` (CLI + servidor), `chain.py` (blockchain), `network.py` (comunicação), `consensus.py` (decisão).



```
Opções:  
└─ Usando zerotier | docker (containers)
```

6) Links

- Repositório (público): https://github.com/ThreeOneOneZero/blockchain_example

Fiz o trabalho em Docker para facilitar testes sem precisar de dupla física — cada container representa um nó. Também testei em outro computador, com uma rede separada.

executar `docker-compose up; attach nos dois nós; executar python main.py configs/node_a_config.json 2>&1 | node_b respectivamente, configurar IP no arquivo peers_a.json e peers_b.json.`

pasta db - `histórico de mineração correspondente`

Transações sendo recebidas ao fazer alguma mineração:

```
2. Mine block  
3. View blockchain  
4. Get balance  
5. Exit  
> 4  
[+] Transaction received from ('172.30.234.208', 58859)  
[✓] New valid block 8 added from ('172.30.234.208', 58868)  
[SYNC] Sent chain to ('172.30.234.208', 58869)  
Node ID: █
```

```
> 4  
[+] Transaction received from ('172.30.234.208', 58859)  
[✓] New valid block 8 added from ('172.30.234.208', 58868)  
[SYNC] Sent chain to ('172.30.234.208', 58869)  
Node ID: Node_B  
[i] The balance of Node_B is 200.0.  
  
1. Add transaction  
2. Mine block  
3. View blockchain  
4. Get balance  
5. Exit  
> 4  
Node ID: Node_A  
[i] The balance of Node_A is 201.0.  
  
1. Add transaction  
2. Mine block  
3. View blockchain  
4. Get balance  
5. Exit  
> █
```

Internet do outro computador caiu, recebi informando que deu time out.

```
1. Add transaction
2. Mine block
3. View blockchain
4. Get balance
5. Exit
> 2
[✓] Mining block...
[✓] Block mined: 005f4962892b58ea2e9c242738e72c137ad57b2604d2c1044f7f15612cb3e15e
[BROADCAST] Broadcasting block...
[BROADCAST_BLOCK] Failed to send to 172.30.234.208: timed out
[✓] Block 9 mined and broadcasted.
[SYNC] Failed to get chain from 172.30.234.208: timed out

1. Add transaction
2. Mine block
3. View blockchain
4. Get balance
5. Exit
> █
```