Aula 008 - Algoritmos de Consenso

Uma Visão Geral

Prof. Rogério Aparecido Gonçalves[1]
rogerioag@utfpr.edu.br

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
Departamento de Computação (DACOM)
Campo Mourão - Paraná - Brasil

Programa de Pós Graduação em Ciência da Computação
Mestrado em Ciência da Computação
PPGCC17 - Tópicos em Redes de Computadores e Cibersegurança

Agenda i

Introdução

Objetivos

- Apresentação dos principais Algoritmos de Consenso.

O Problema de Consenso

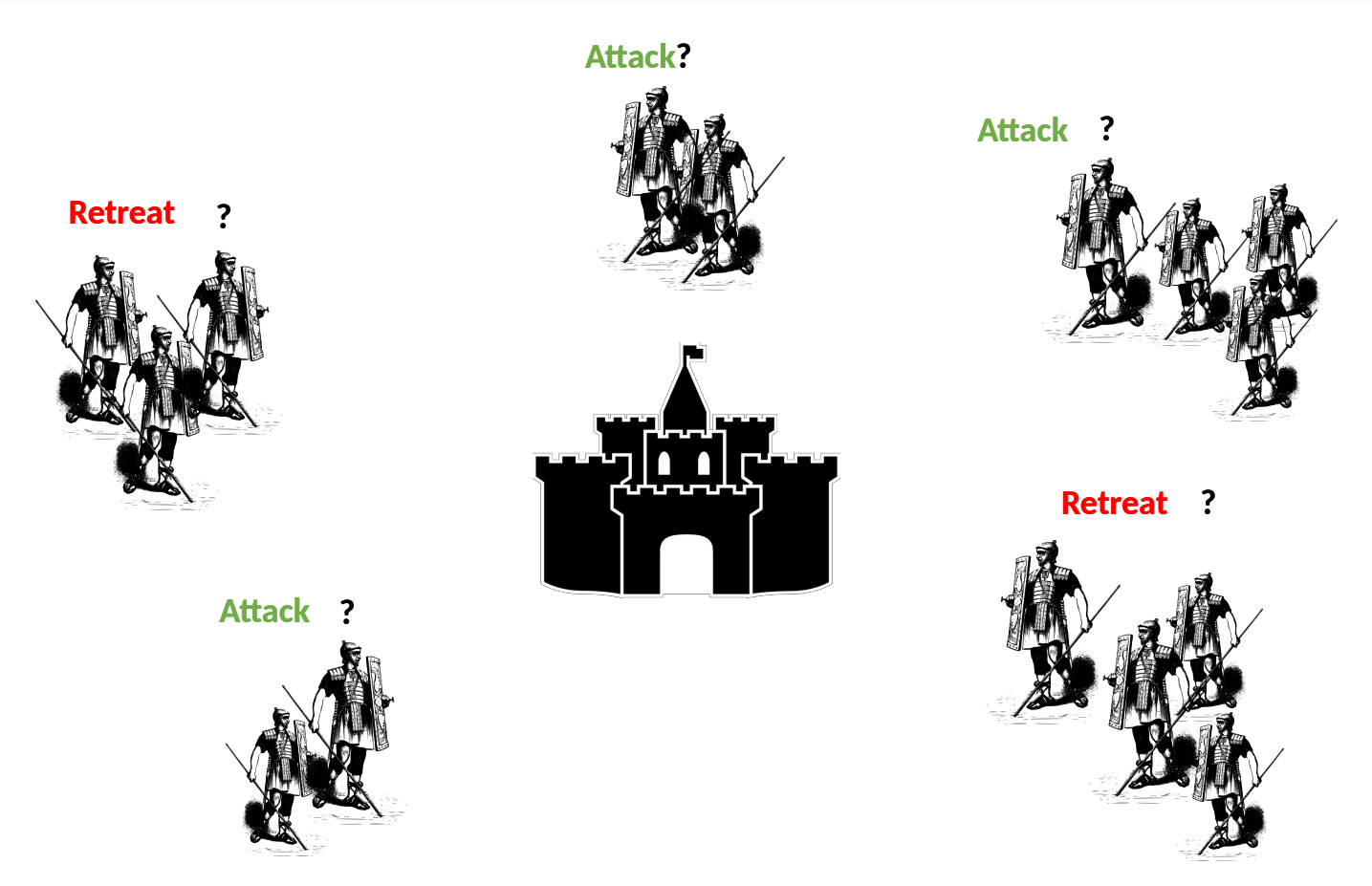
O Problema de Consenso i

- É um problema que tem sido estudado estensivamente na área de Sistemas Distribuídos desde o final de 1970.
- O que resultou na evolução rápida da tecnologia *blockchain* e em mais pesquisas em novos métodos de consenso.
- Ocorrendo uma conversão dos mecanismos de consenso distribuídos tradicionais (clássicos) em suas variantes para *blockchain*.

O Problema dos Generais Bizantinos i

O famoso *Byzantine generals problem* foi formulado por *Lamport et al.* no artigo: `Lamport, L., Shostak, R. and Pease, M., 1982. The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems, 4(3), pp.382-401.`

O Problema dos Generais Bizantinos ii



- Atacar ou recuar? O **Consenso** é necessário para vencer.

O Problema dos Generais Bizantinos iii

- Em 1982, um experimento foi proposto por Lamport e outros em um artigo (Lamport, Shostak, and Pease 1982), **The Byzantine Generals Problem**.
- Como analogia a sistemas distribuídos, os generais podem ser considerado os **nós**, os traidores como **nós bizantinos** (maliciosos), e o mensageiro pode ser pensado como um canal de comunicação entre os generais.
- O problema foi resolvido em 1999 por *Castro e Liskov*, apresentaram o algoritmo *Practical Byzantine Fault Tolerance (PBFT)* (Castro and Liskov 1999), onde o consenso é alcançado depois de um certo número de mensagens serem recebidas contendo o mesmo conteúdo assinado.

Tolerânica a Falhas i

Tipos de Tolerância a falhas:

- *Crash Fault Tolerance (CFT)*
- *Byzantine Fault Tolerance (BFT)*

Como alcançar um nível de Tolerância a Falhas:

- **Replicação:** Abordagem padrão para tornar um sistema tolerante a falhas. Cópias dos dados são sincronizadas entre todos os nós da rede.
  - Ativa: Cada réplica torna-se uma cópia da máquina de estados original.
  - Passiva: Existe somente uma cópia da máquina de estados no sistema, mantido como nó primário, e os outros nós ou réplicas somente mantem o estado.

Tolerânica a Falhas ii

- **State Machine Replication (SMR):** é uma técnica que é usada para a replicação determinística de serviços para alcançar os requisitos de tolerância a falhas em um Sistema Distribuído.
  - **Artigo:** *Time, Clocks and the Ordering of Events in a Distributed System*(Lamport 1978)

Tolerânica a Falhas iii

Limitações e Resultados:

- FLP impossibility (Fischer, Lynch, and Paterson 1985): Fischer, M.J., Lynch, N.A. and Paterson, M.S., 1982. Impossibility of distributed consensus with one faulty process (No. MIT/LCS/TR-282)
- Lower bounds
- Upper bounds

Limitações i

- Onde F = número de falhas:
  - No caso do CFT, no mínimo $2F + 1$ número de nós é necessário para alcançar o consenso.
  - No caso do BFT, no mínimo $3F + 1$ número de nós é necessário para alcançar o consenso.

Análise e Projeto i

Projetar e analisar mecanismos de consenso requer um modelo que possa ser usado para estudar e explorar as propriedades do sistema e tornar os projetos mais lógicos.

- O modelo envolve:
  - Processos
  - Timing assumptions
  - Sincronia
  - Assincronia
  - Sincronia parcial

Classificação i

Desde a invenção do *Bitcoin*, que introduziu os algoritmos de consenso baseados em loteria, existem dois tipos principais de consenso:

- Tradicional
  - Esta categoria inclui `Paxos`, Algoritmos `BFT`.
- Baseados em Loteria
  - Esta categoria inclui os modernos algoritmos de `Proof of Work (PoW)`

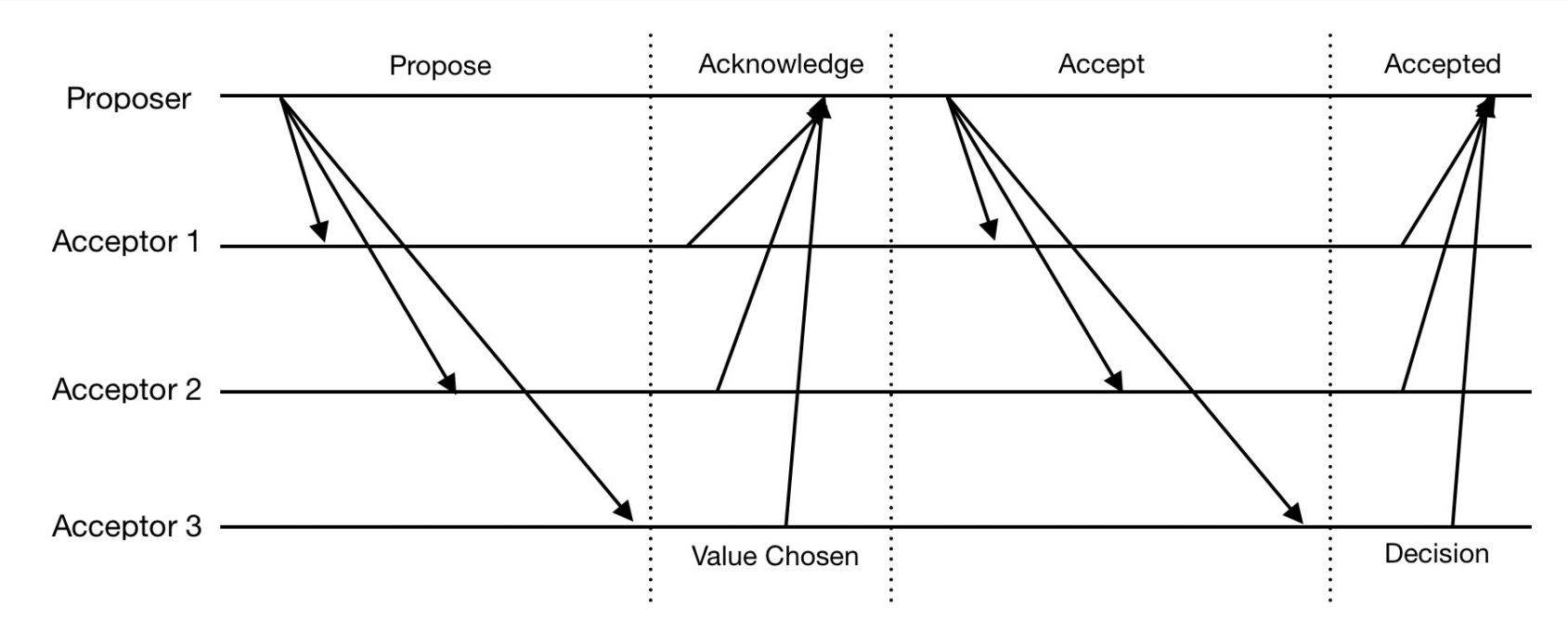
Propriedades de corretude do consenso i

- Segurança (Safety)
  - Concordância (Agreement)
  - Validade (Validity)
  - Integridade(Integrity)
- Liveness
  - Terminação (Termination)

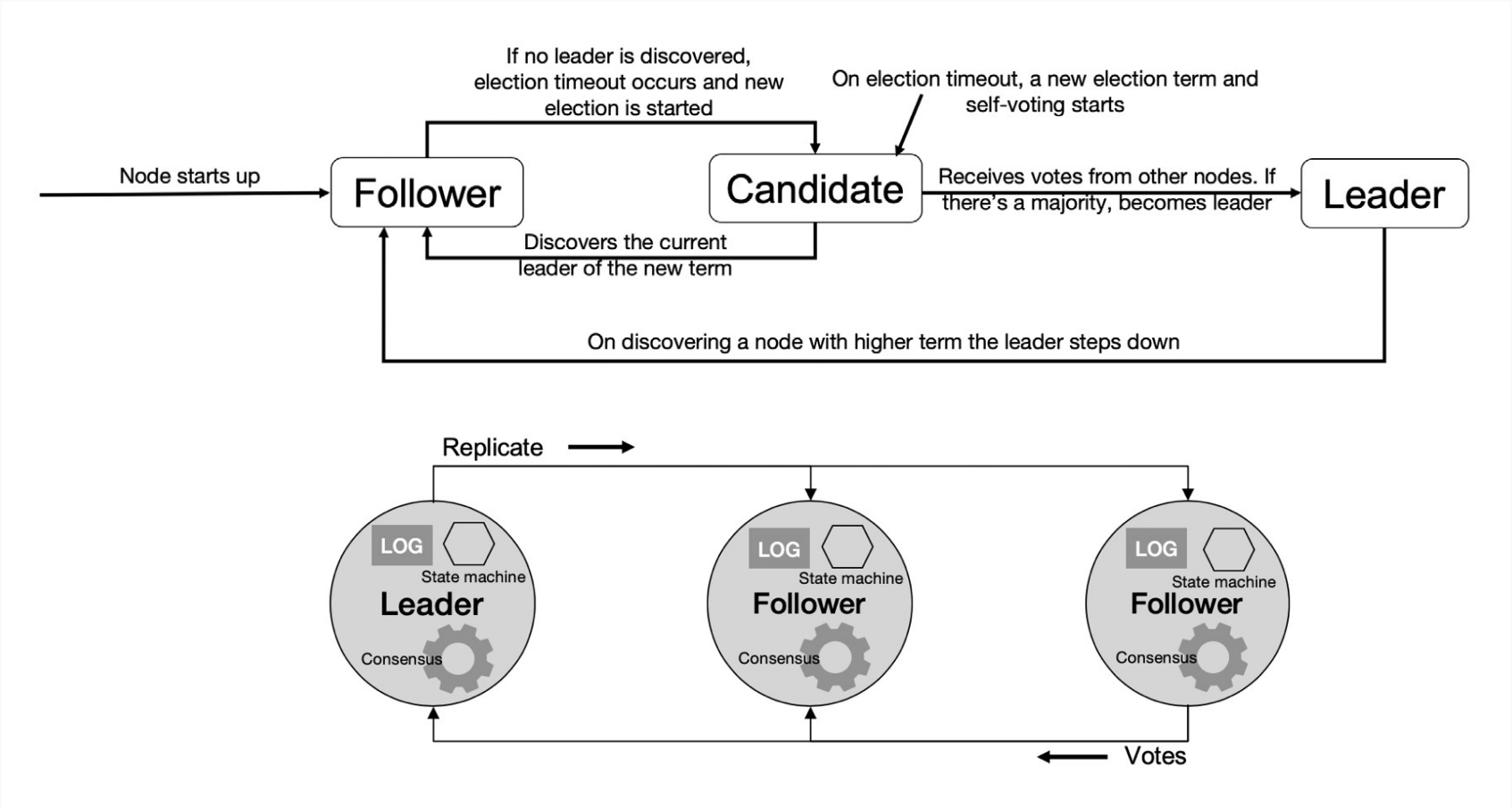
Algoritmos i

- CFT
  - Paxos
  - Raft
- BFT
  - PBFT
  - IBFT
  - Tendermint'
  - HotStuff
- Nakamoto and post-Nakamoto
  - PoW
  - Proof of Stake (PoS)

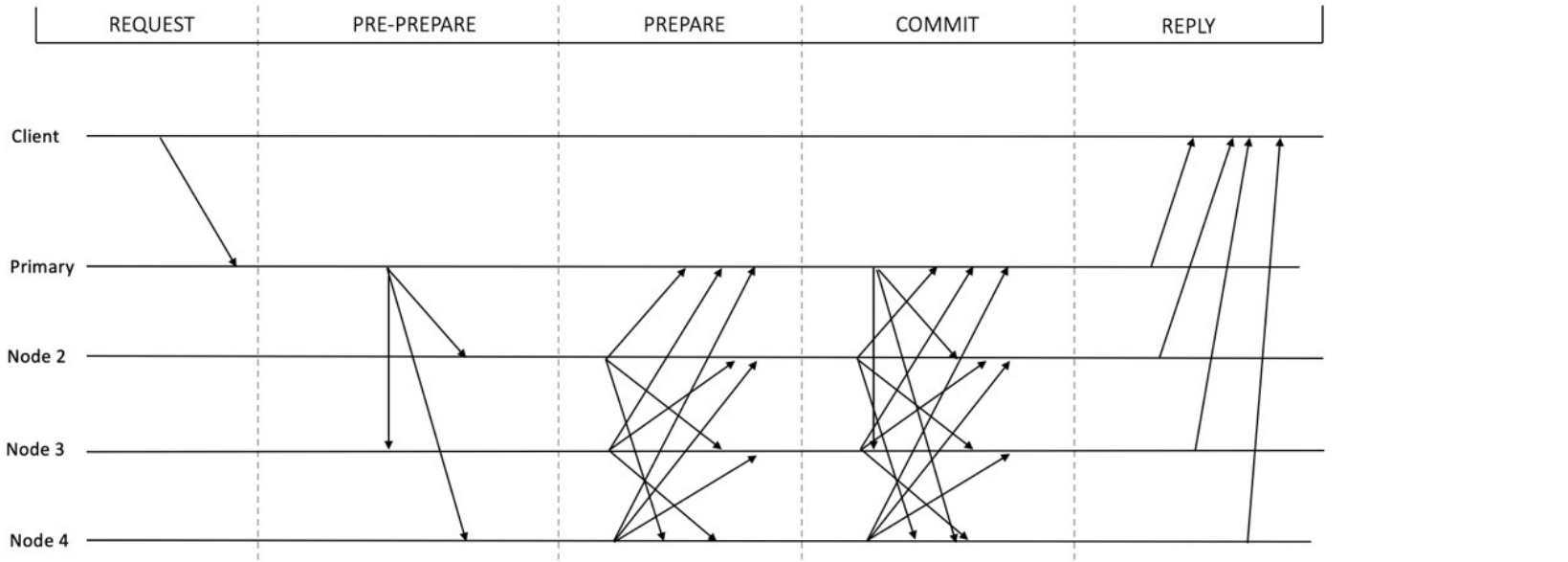
Paxos i


Propose
Acknowledge
Accept
Accepted
Proposer
Acceptor 1
Acceptor 2
Acceptor 3
Value Chosen
Decision

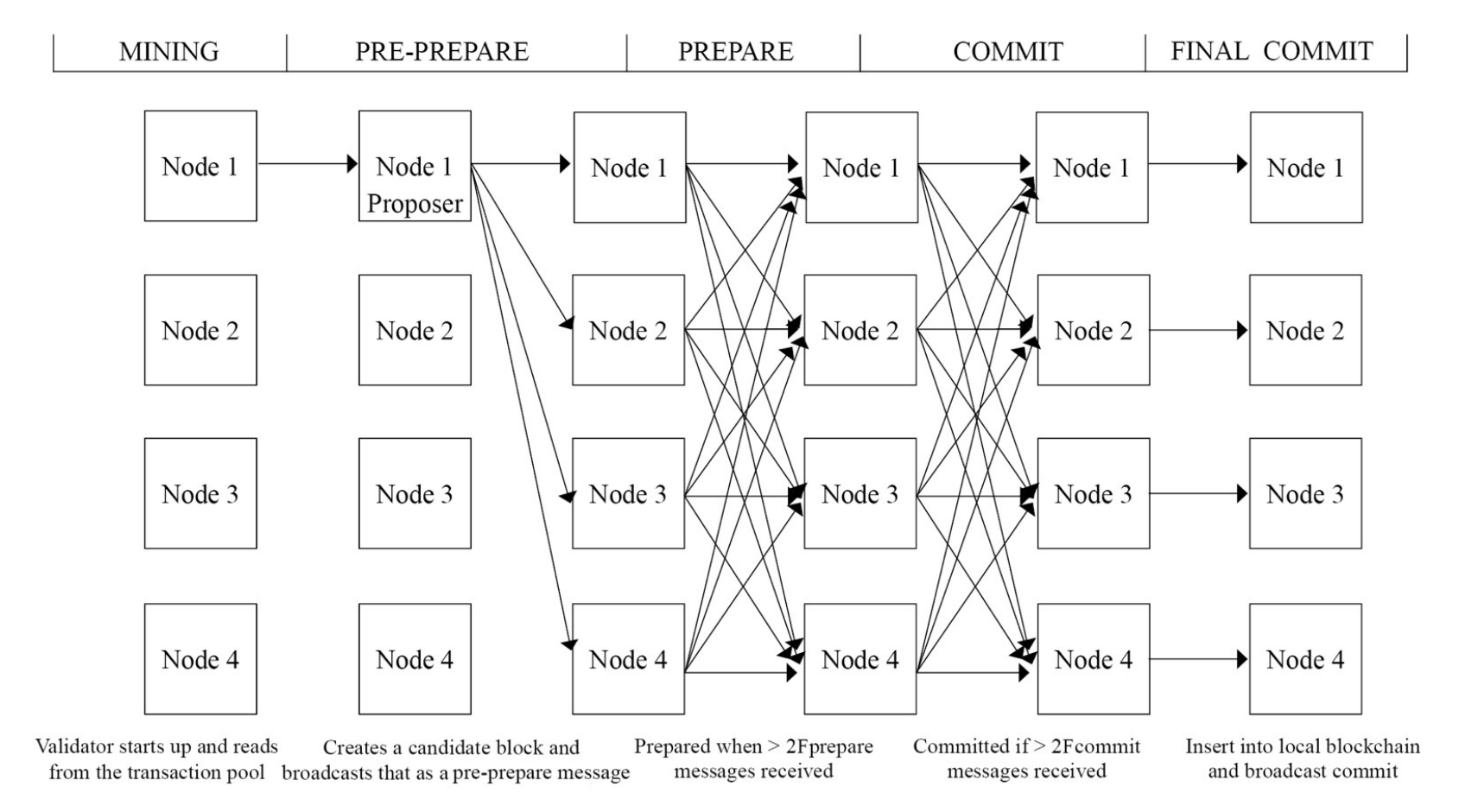
Raft i


If no leader is discovered,
election timeout occurs and new
election is started
On election timeout, a new election term and
self-voting starts
Node starts up
Follower
Candidate
Receives votes from other nodes. If
there's a majority, becomes leader
Leader
Discovers the current
leader of the new term
On discovering a node with higher term the leader steps down
Replicate
LOG
State machine
Leader
Consensus
LOG
State machine
Follower
Consensus
LOG
State machine
Follower
Consensus
Votes

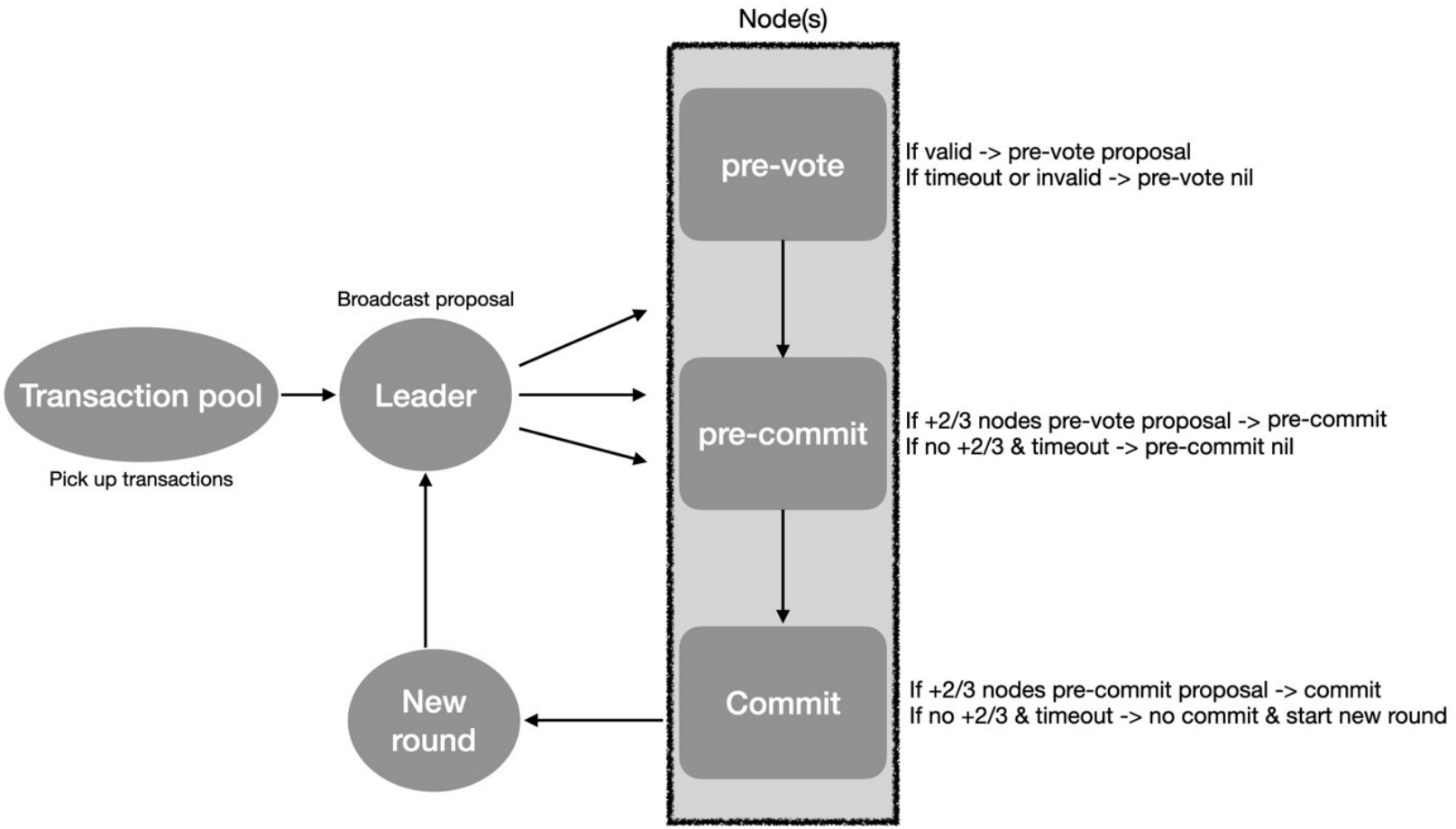
PBFT i


REQUEST
PRE-PREPARE
PREPARE
COMMIT
REPLY
Client
Primary
Node 2
Node 3
Node 4
VIEW-CHANGE
NEW VIEW
Primary
New primary
Node 3
Node 4

IBFT i


MINING
PRE-PREPARE
PREPARE
COMMIT
FINAL COMMIT
Node 1
Node 2
Node 3
Node 4
Node 1 Proposer
Node 1
Node 2
Node 3
Node 4
Node 1
Node 2
Node 3
Node 4
Node 1
Node 2
Node 3
Node 4
Node 1
Node 2
Node 3
Node 4
Validator starts up and reads from the transaction pool
Creates a candidate block and broadcasts that as a pre-prepare message
Prepared when > 2Fprepare messages received
Committed if > 2Fcommit messages received
Insert into local blockchain and broadcast commit

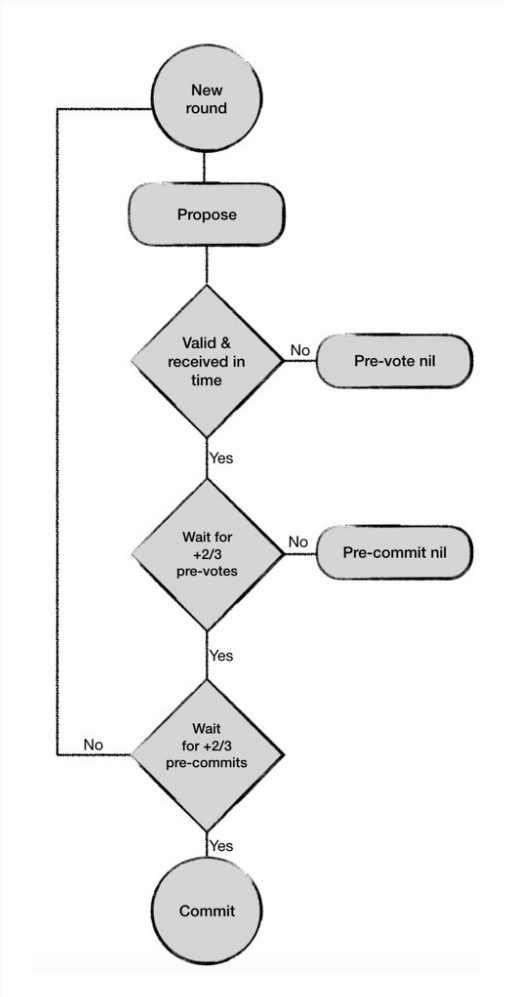
Tendermint - Visão de Alto Nível i



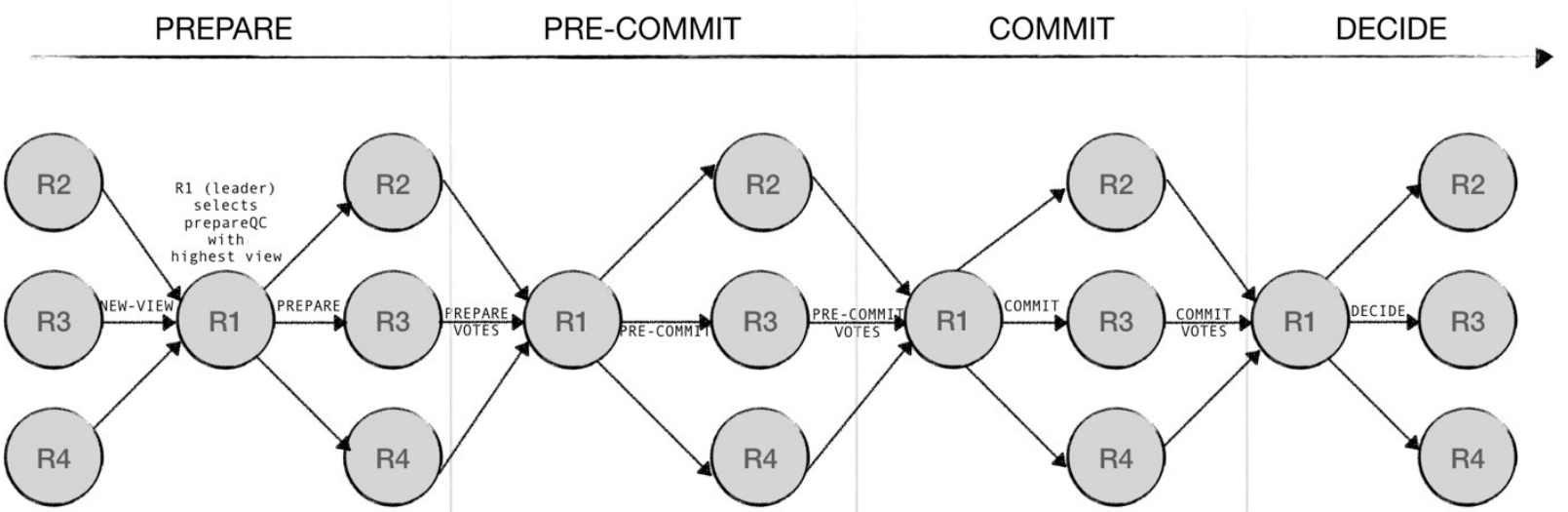
Tendermint - Visão de Alto Nível ii

O processo do **Tendermint** é simples:

$Proposal \longrightarrow Pre\text{-}vote \longrightarrow Pre\text{-}commit$



HotStuf i



R1 (leader) broadcasts Msg(prepare, curProposal , high QC) R2,R3,R4 send voteMsg(prepare, m.node , ⊥) to R1

R1 (leader) broadcast Msg(pre-commit, ⊥, prepareQC) R2,R3,R4 (replicas) send voteMsg(pre-commit, m.justify .node,⊥) to R1

R1(leader) broadcast Msg(commit, ⊥, pre-commit QC) R2,R3,R4 (replicas) send voteMsg(commit, m.justify .node , ⊥) to R1

R1 (leader) broadcast Msg(decide, ⊥, commit QC) R2,R3,R4 (replicas) execute & respond to clients

Once a new leader has collected new-view messages from N - F nodes, the protocol for new leader starts

As soon the leader receives N - F prepare votes, it creates a quorum certificate (QC) called 'prepare quorum certificate'

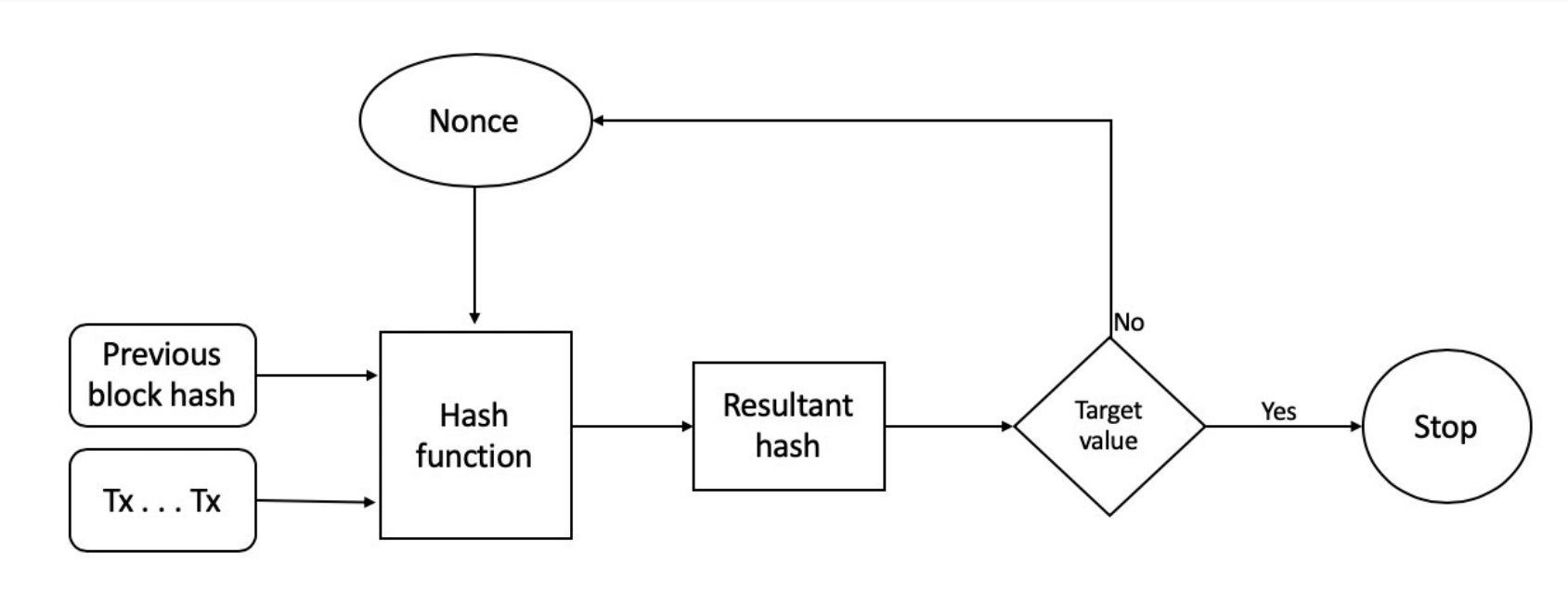
When the leader receives N - F pre-commit votes, it creates a pre-commit quorum certificate and broadcasts it to other nodes as the commit message

When the leader receives N - F commit votes, it creates a commit quorum certificate. This commit quorum certificate is broadcast to other nodes in the decide message

Proof of Work (PoW) i

Um nós que propõe um bloco tem que encontrar um *nonce* (número) tal como:

$$H(nonce||previoushash||Tx||Tx||...||Tx) < Thresholdvalue.$$



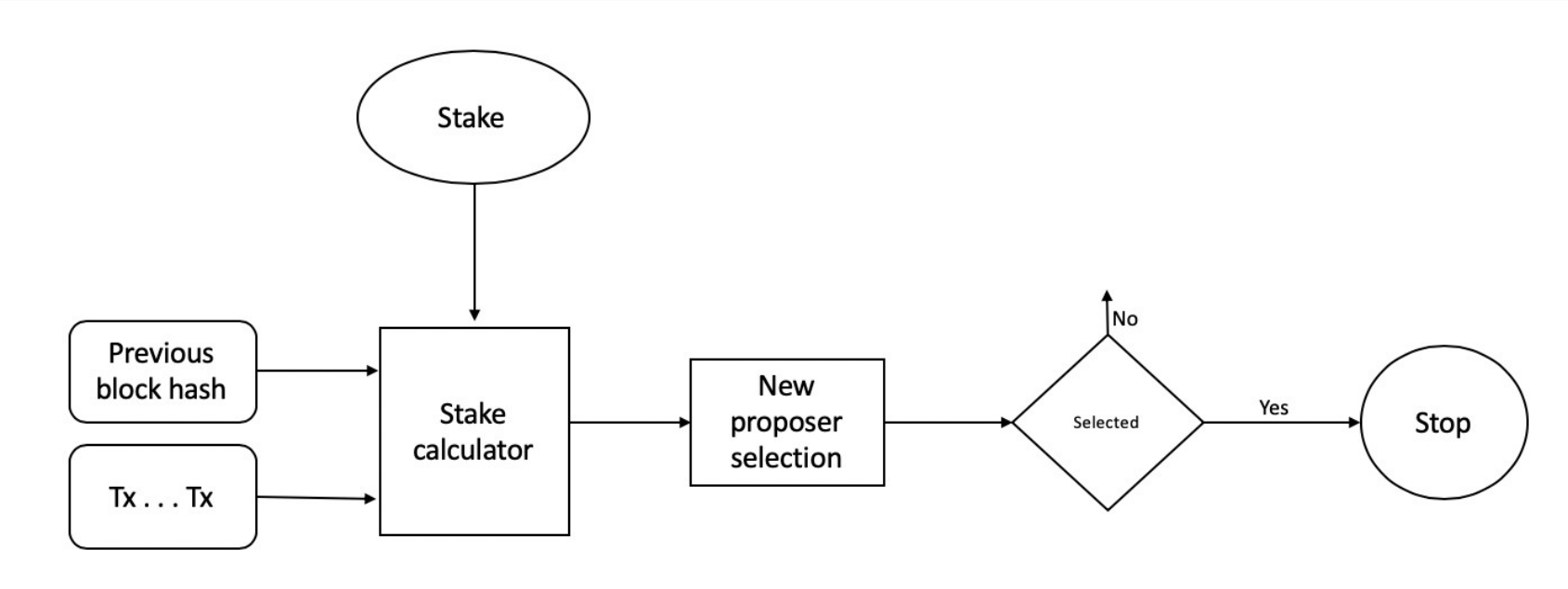
Proof of Stake (PoS) i

Tipos de PoS:

- *Chain-based PoS*
- *Committee-based PoS*
- *Delegated PoS*

Aqui, uma função de cálculo da aposta (stake) é usada para calcular o montante de fundos apostados e com base nisso, seleciona um novo proponente:

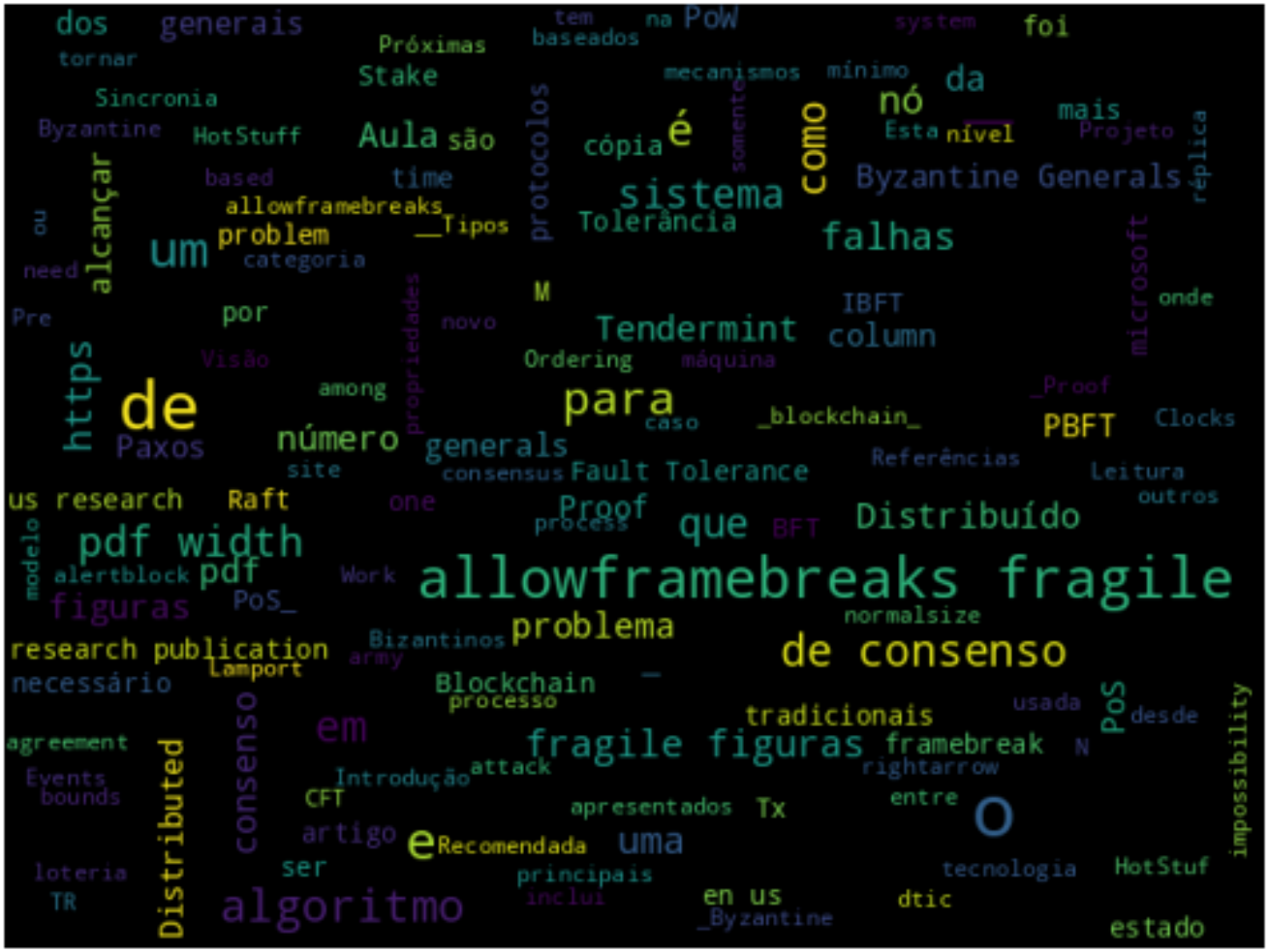
Proof of Stake (PoS) ii



Escolha de algoritmo i

- Finalidade
- Velocidade
- Desempenho
- Escalabilidade

Word Cloud


dos
generais
tornar
Próximas
Stake
Sincronia
Byzantine
HotStuff
Aula
são
based
time
allowframebreaks
problem
__Tipos
categoria
um
alcançar
ou
need
Pre
por
novo
Visão
https
de
Paxos
among
número
site
generals
us research
Raft
one
pdf width
alertblock
pdf
figuras
PoS_
modelo
Work
research publication
Lamport
necessário
Bizantinos
army
problema
Blockchain
processo
consenso
em
Distributed
agreement
Events
bounds
Introdução
attack
CFT
artigo
e
Recomendada
ser
loteria
TR
algoritmo
principais
inclui
apresentados
Tx
uma
en us
_Byzantine
dtic
tecnologia
HotStuf
estado
entre
O
rightarrow
framebreak
N
tradicionais
usada
desde
PoS
impossibility
normalsize
de consenso
fragile figuras
allowframebreaks fragile
Distribuído
que
BFT
Proof
process
Fault Tolerance
consensus
Referências
Leitura
outros
PBFT
_Proof
Clocks
blockchain
caso
para
Ordering
máquina
Tendermint
IBFT
column
M
propriedades
protocolos
sistema
Tolerância
falhas
Byzantine Generals
como
é
cópia
são
Esta
nível
Projeto
mais
nó
da
mecanismos
mínimo
tem
baseados
na
PoW
system
foi
microsoft
onde
réplica
somente

Leitura Recomendada

Leitura Recomendada

Capítulo 5: Consensus Algorithms

Livro: IMRAN BASHIR. Mastering Blockchain : Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition.

Próximas Aulas

Próximas Aulas: Tecnologia Blockchain

- Introdução a `Bitcoin`

Referências

Referências i

Castro, Miguel, and Barbara Liskov. 1999. “Practical Byzantine Fault Tolerance.” In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI ’99, USA: USENIX Association, 173–86.

Fischer, Michael J., Nancy A. Lynch, and Michael S. Paterson. 1985. “Impossibility of Distributed Consensus with One Faulty Process.” *J. ACM* 32(2): 374–82. https://doi.org/10.1145/3149.214121.

Imran, Bashir. 2018. *Mastering Blockchain : Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition*. Packt Publishing. https://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=1789486&lang=pt-br&site=eds-live&scope=site.

Referências ii

Lamport, Leslie. 1978. “Time, Clocks and the Ordering of Events in a Distributed System.” *Communications of the ACM 21, 7 (July 1978), 558-565. Reprinted in several collections, including Distributed Computing: Concepts and Implementations, McEntire et al., ed. IEEE Press, 1984.*: 558–65. https://www.microsoft.com/en-us/research/publication/time-clocks-ordering-events-distributed-system/.

Lamport, Leslie, Robert Shostak, and Marshall Pease. 1982. “The Byzantine Generals Problem.” *ACM Transactions on Programming Languages and Systems*: 382–401. https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/.