Redundant Byzantine Fault Tolerance - RBFT

PIERRE-LOUIS AUBLIN, SONIA BEN MOKHTAR, VIVIEN QUEMA

Presented By:
Arefe Emrani(40259157)
Mohamadali Sadeghi(40258780)

MARCH 26TH, 2025

Agenda

1. Background and Motivation
2. System Model
3. Analysis of existing robust BFT protocols
   (Prime, Aardvark, Spinning)
4. RBFT overview
5. RBFT detailed protocol steps
6. RBFT monitoring mechanism and Protocol instance change
7. Implementation
8. Performance Evaluation
9. Critical Analysis
10. Conclusion

Background and Motivation

Challenges in Existing BFT protocols:

- degrade significantly (at least 78%) when faults occur.
- Smartly malicious primaries can degrade performance before being detected.

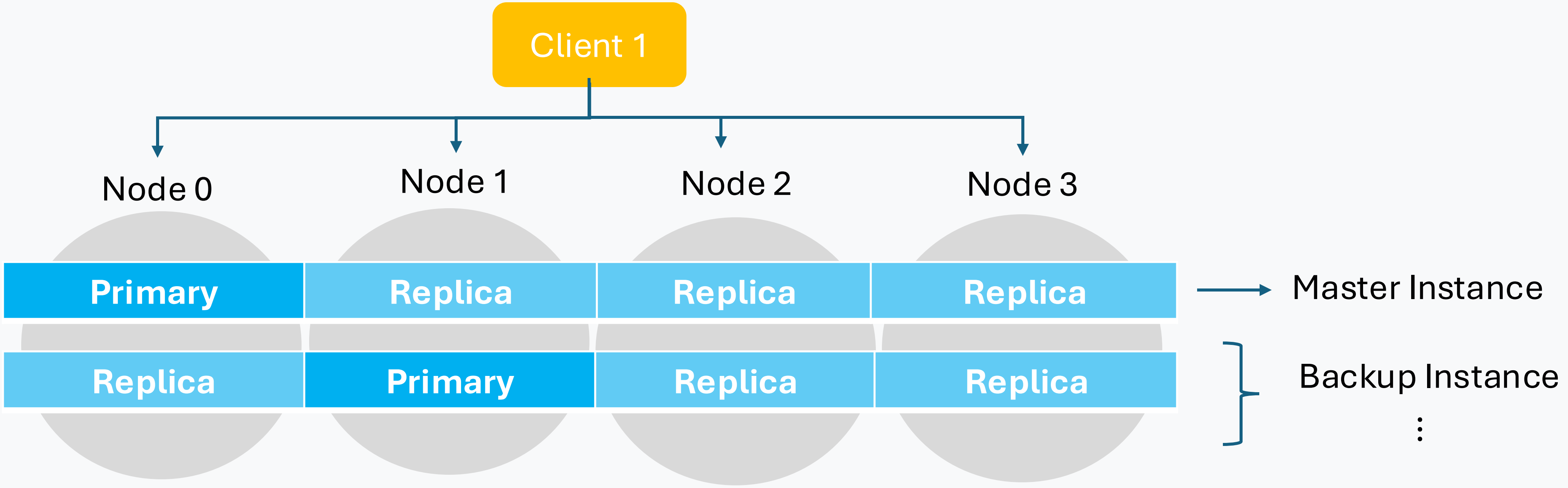
Maximum Throughput Degradation	Prime	Aardvark	Spinning
	%78	%87	%99

- ➢ How can we build a BFT protocol that remains robust under faults?
- ➢ Can we avoid reliance on a single primary to prevent performance bottlenecks?

Background and Motivation

Proposed Solution - RBFT

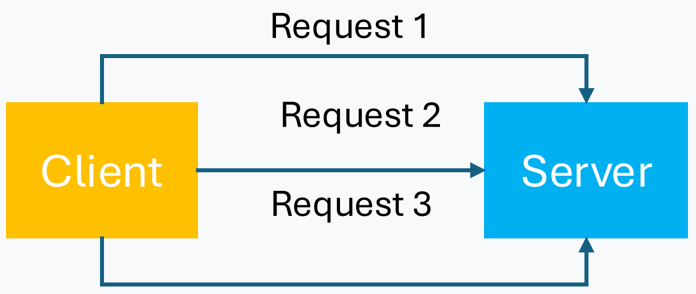
- ✓ **Minimizes Performance Degradation:** Only 3% degradation under fault and similar performance in fault-free scenarios.
- ✓ **Ensures Fairness:** Monitors request latency to fairly process client requests.
- ✓ **Improves Fault Tolerance:** Does not rely on a single primary.



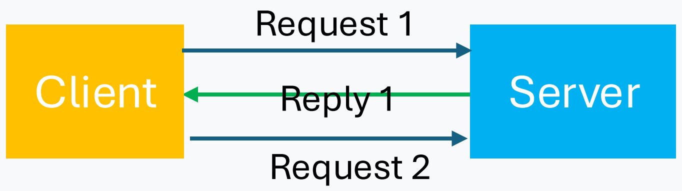
System Model

Assumptions:

- ❖ Faulty node is f = (N−1)/3 -> lower bound.
- ❖ A compromised process means whole machine is compromised.
- ❖ Faulty nodes and clients can collude to attack but cryptographic techniques (signatures, MACs, and hashing) remain secure.
- ❖ The network is Semi-Synchronous
- ❖ It addresses open-loop systems (no need to wait for a reply)



Open-loop System

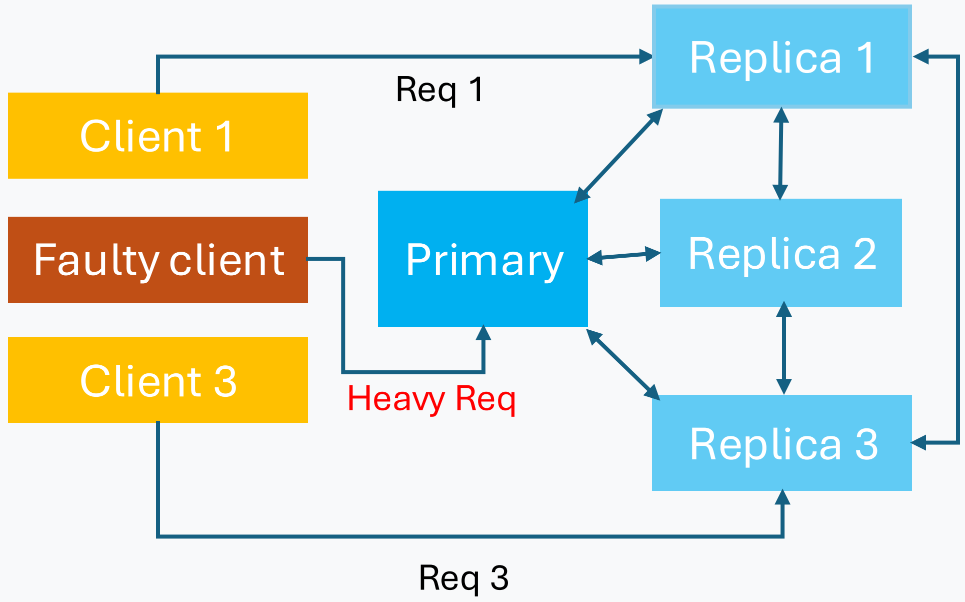


Close-loop System

ANALYSIS OF EXISTING ROBUST BFT PROTOCOLS

1.Prime

- ❖ Requests be sent to any replica.
- ❖ Replicas exchange requests and monitor primary.
- ❖ Replica be changed when observe faulty behavior.
- ❖ Primary send ordering messages at a defined frequency calculated based on:
  - ❖ Round-trip time (RTT) between replicas.
  - ❖ Request execution time.
  - ❖ Network variability factor.



Weakness of Prime

- ➢ The protocol relies on accurate network monitoring.
- ➢ Malicious primary colludes with a client to artificially increase RTT.
  - ➢ This increases the allowed delay for sending messages.
  - ➢ Primary can now delay ordering messages without detection.
- ➢ A faulty client sends heavier requests
  Increases monitored RTT, allowing the primary to delay orders.

Maximum degradation Throughput : 78%.

ANALYSIS OF EXISTING ROBUST BFT PROTOCOLS

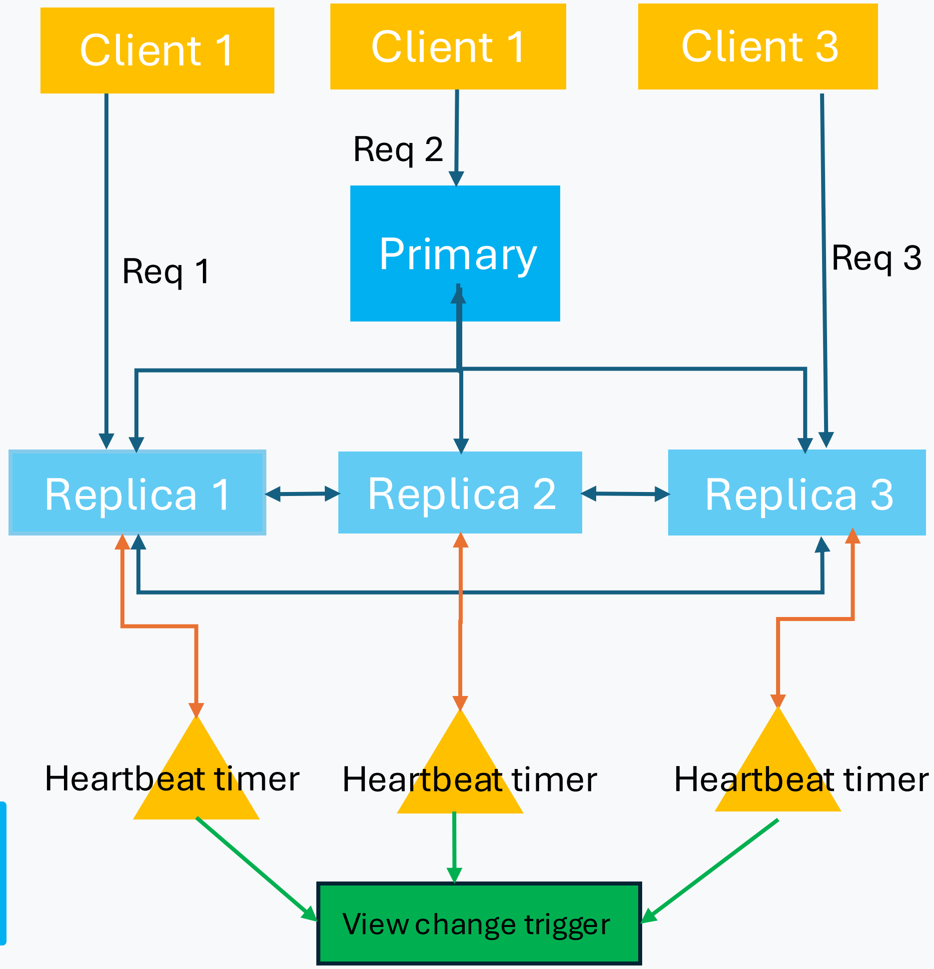
2.Aardvark

- Based on PBFT but adds frequent primary changes.
- Primary change occurs when:
  - Throughput falls below 90% of the average from the last N views.
  - Heartbeat timer expires before next ordering message arrives.
- Uses separate NICs for clients and replicas to prevent slowdowns.

Weakness of Aardvark

- Malicious primary delays requests strategically during low-traffic periods.
- When load increases suddenly, system fails to detect delays quickly.

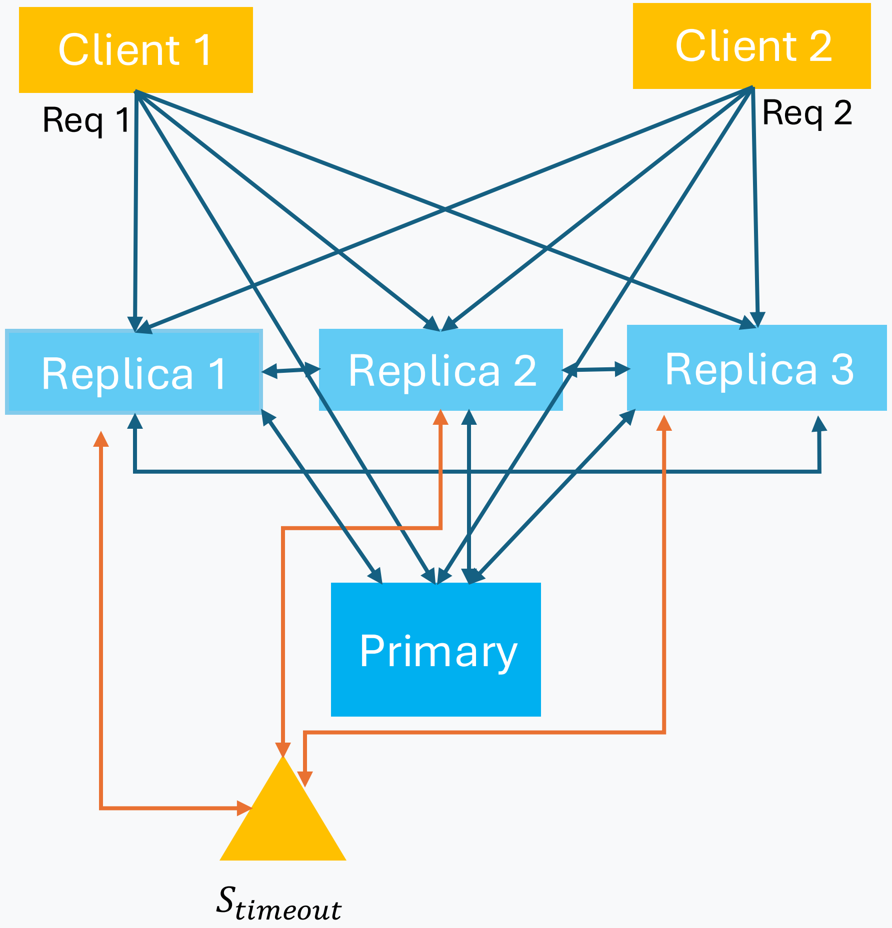
Under static load, throughput remains 76% of normal
Under dynamic load, throughput can degrade to 87%



ANALYSIS OF EXISTING ROBUST BFT PROTOCOLS

3. Spinning

- Regular primary rotation after each batch of requests.
- Clients send requests to all replicas.
- If a non-primary replica does not receive an ordering message within a timeout ($S_{timeout}$):
  - Primary is blacklisted.
  - A new primary is automatically selected.
  - Timeout doubles on each failure.



Weakness of Spinning

- Malicious primary delays ordering messages just under $S_{timeout}$
- This prevents immediate detection while drastically reducing throughput.

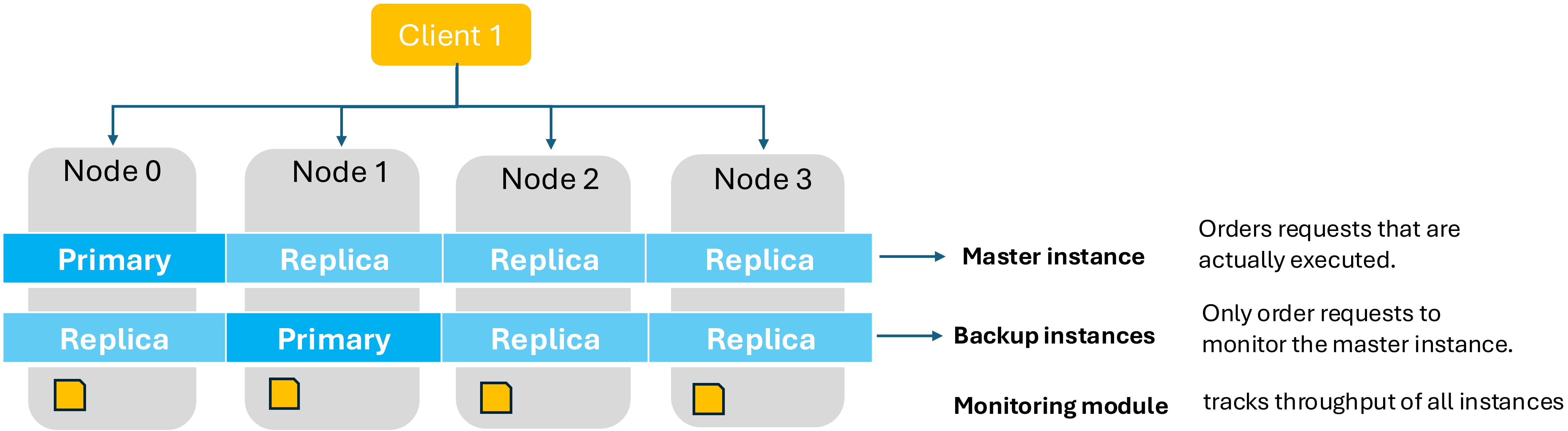
Under static load, throughput can degrade to 99%
Under dynamic load, throughput can degrade to 95.5%

Analysis Summary

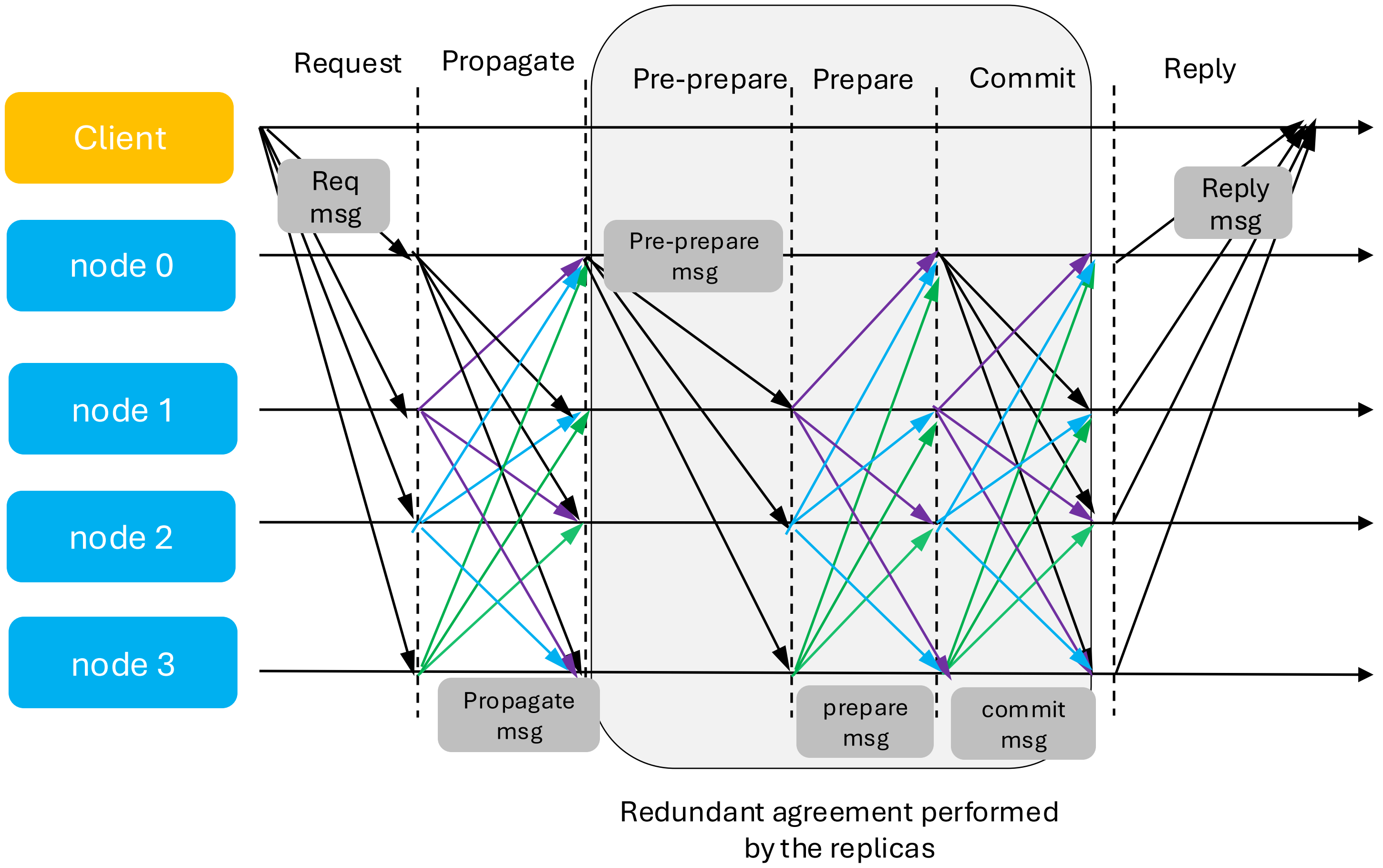
Protocol	Primary Rotation	Attack Strategy	Max Performance Drop
Prime	Replaces slow primary	Increases RTT to allow delays	**78%**
Aardvark	Periodic primary change	Delays requests under low load	**87%**
Spinning	Changes primary every batch	Delays just under timeout	**99%**

RBFT – Overview

- ✓ Requires 3f + 1 nodes.
- ✓ Each node runs f + 1 protocol instances in parallel and must receive the same client requests.
- ✓ The protocol follows a 3-phase commit protocol, similar to PBFT.
- ✓ If **2f + 1 nodes** detect the master instance is underperforming, a new primary is elected.
- ✓ A node **forwards requests to all other nodes** instead of processing them directly.
- ✓ When a node receives **2f + 1 copies** of a request, it forwards it to its local instances.



RBFT - Detailed Protocol Step



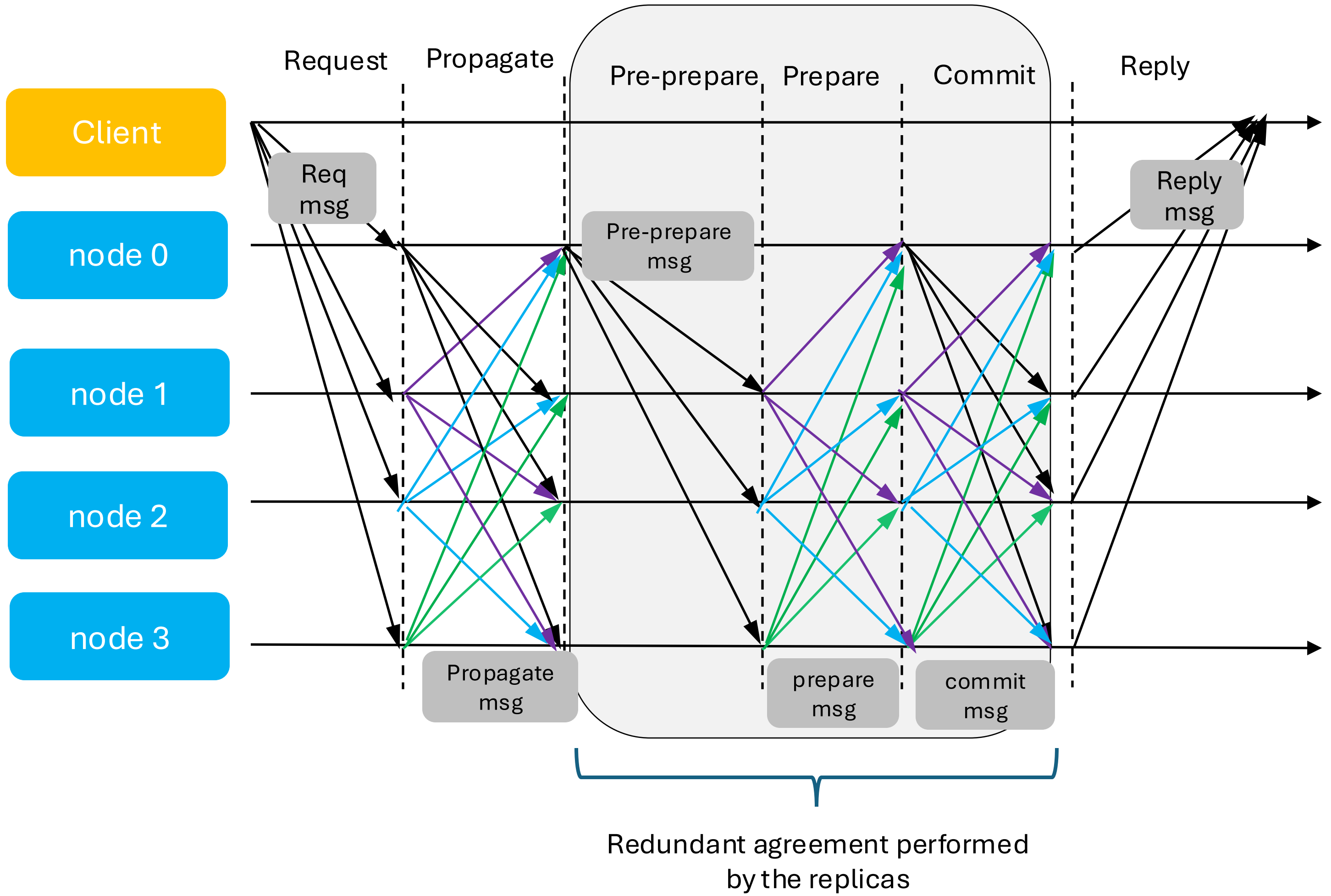
- REQUEST message containing: Operation (o), Req ID (rid), Client ID (c)
- Signed & authenticated using: Digital signature and MAC
- Nodes: check the MAC & signature.

- If a node receives f+1 propagate messages, it considers the request ready for ordering.
- Prevents a malicious primary from manipulating request flow.

- Primary sends a PRE-PREPARE message contains: View number (v), Sequence number (n), Client request ID (rid), Request digest (d)

- Replicas verify the message and send a PREPARE message to all replicas

RBFT - Detailed Protocol Step



- When a replica receives 2f matching PREPARE messages, it sends a COMMIT message.
- When 2f+1 COMMIT messages are received → Request is finalized and ordered.

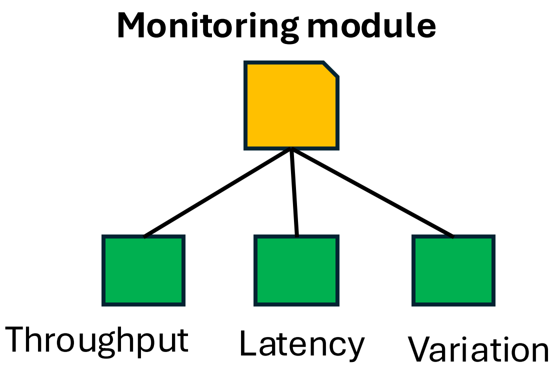
- Once the request is ordered, it is executed by the master instance.
- Each node sends a REPLY message to the client.
- The client accepts the result only if it receives f+1 matching REPLY messages from different nodes.

RBFT - Monitoring mechanism & instance change mechanism

Trigger primary change :

- **Throughput** $t_{master} / t_{backup} < \Delta t$
- **Latency Check** (Λ max latency)
- **Variation Check** (Ω threshold)

Fairness



Instance change mechanism:

- Primary suspect as malicious: send change message
- New primary selected
- Instance change triggered

Implementation

- Implemented in C++, based on the Aardvark BFT protocol.

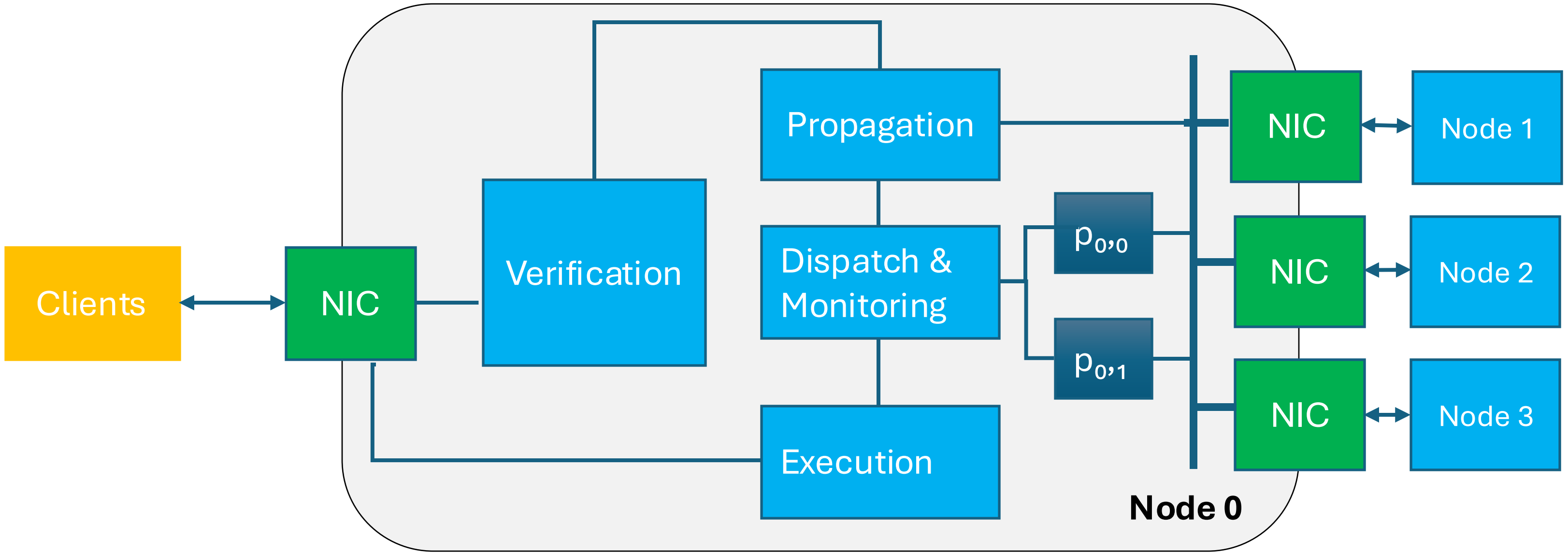
- Uses separate Network Interface Controllers (NICs) for:
  - Isolating client traffic from replica communication.
  - Mitigating flooding attacks by closing a faulty node's NIC temporarily.

- Communication between replicas is via TCP.

- Also implemented a UDP version of RBFT for comparison.

Implementation

1. Client sends request via client NIC.
2. Verification module validates the request.
3. Propagation module sends the request to other nodes and waits for f+1 copies.
4. Once f+1 are received, request is sent to Dispatch & Monitoring.
5. Dispatch & Monitoring forwards to local replicas (e.g., $p_{0,0}$ and $p_{0,1}$).
6. Replicas coordinate with their instance peers on other nodes to order the request.
7. Ordered requests return to Dispatch & Monitoring.
8. Requests from master instance are passed to Execution.
9. Execution runs the request and sends the reply to the client.



Experimental Settings

- Experiments run with up to 2 Byzantine faults ($f \leq 2$)

- Unless specified, default configuration is $f = 1$

- Two workload modes tested:
  - **Static Load:** clients send requests at a constant rate (saturated system)
  - **Dynamic Load:** varying client count to simulate spikes

- Clients operate in open-loop mode

Performance Evaluation

Spinning vs others

- Spinning outperforms other protocols for both requests of 8B and 4KB since it only uses MAC, while others use signatures in addition to MAC
- Spinning has low latency since it uses UDP for communication between replicas and between replicas and clients

RBFT vs Aardvark

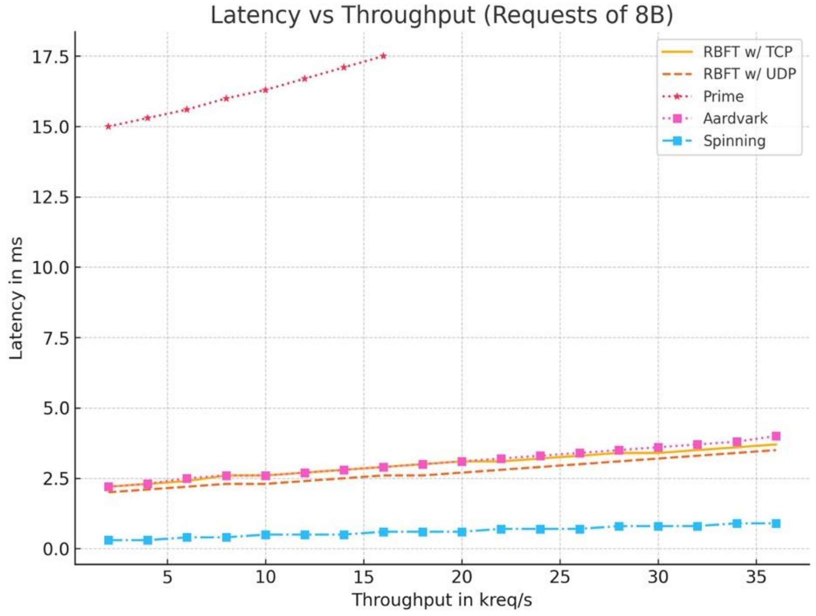
RBFT outperforms Aardvark and this may seem surprising as they both use the same code base the reason for that is that RBFT doesn't perform view changes

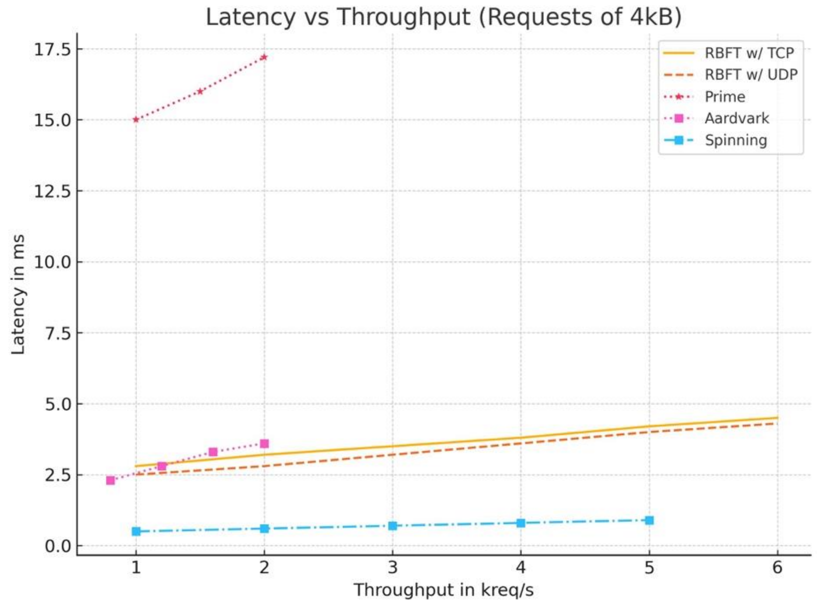
Prime vs others

Its high latency is due to the fact that it solely relies on signatures

TCP vs UDP RBFT

The throughput is the same, but TCP has more latency due to mechanisms it use (ack, flw ctrl, ..)



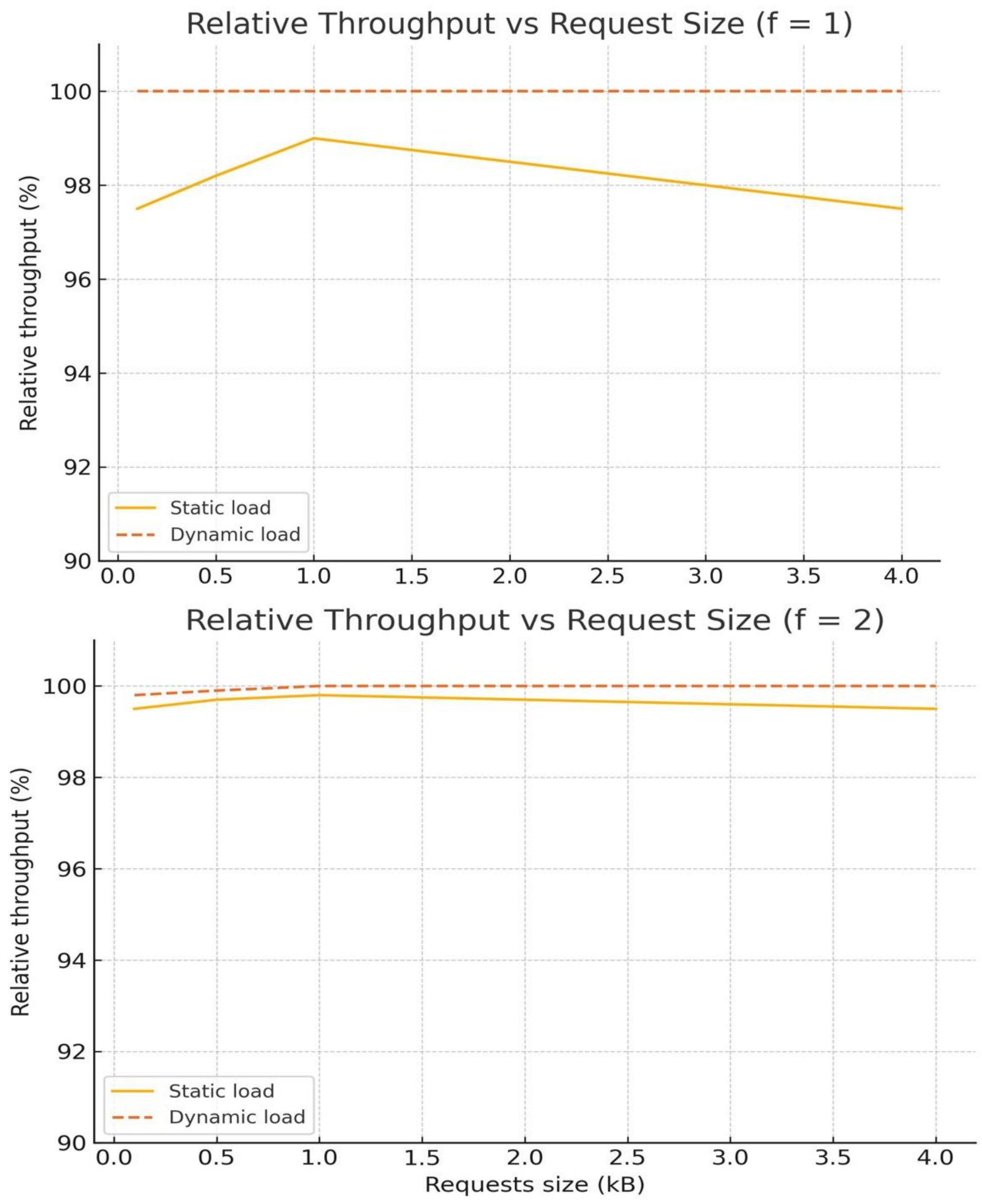


Performance Evaluation - Worst-Attack-1

- f faulty nodes are present.
- All clients are faulty.
- The primary of the master protocol instance is correct and runs on a correct node p.

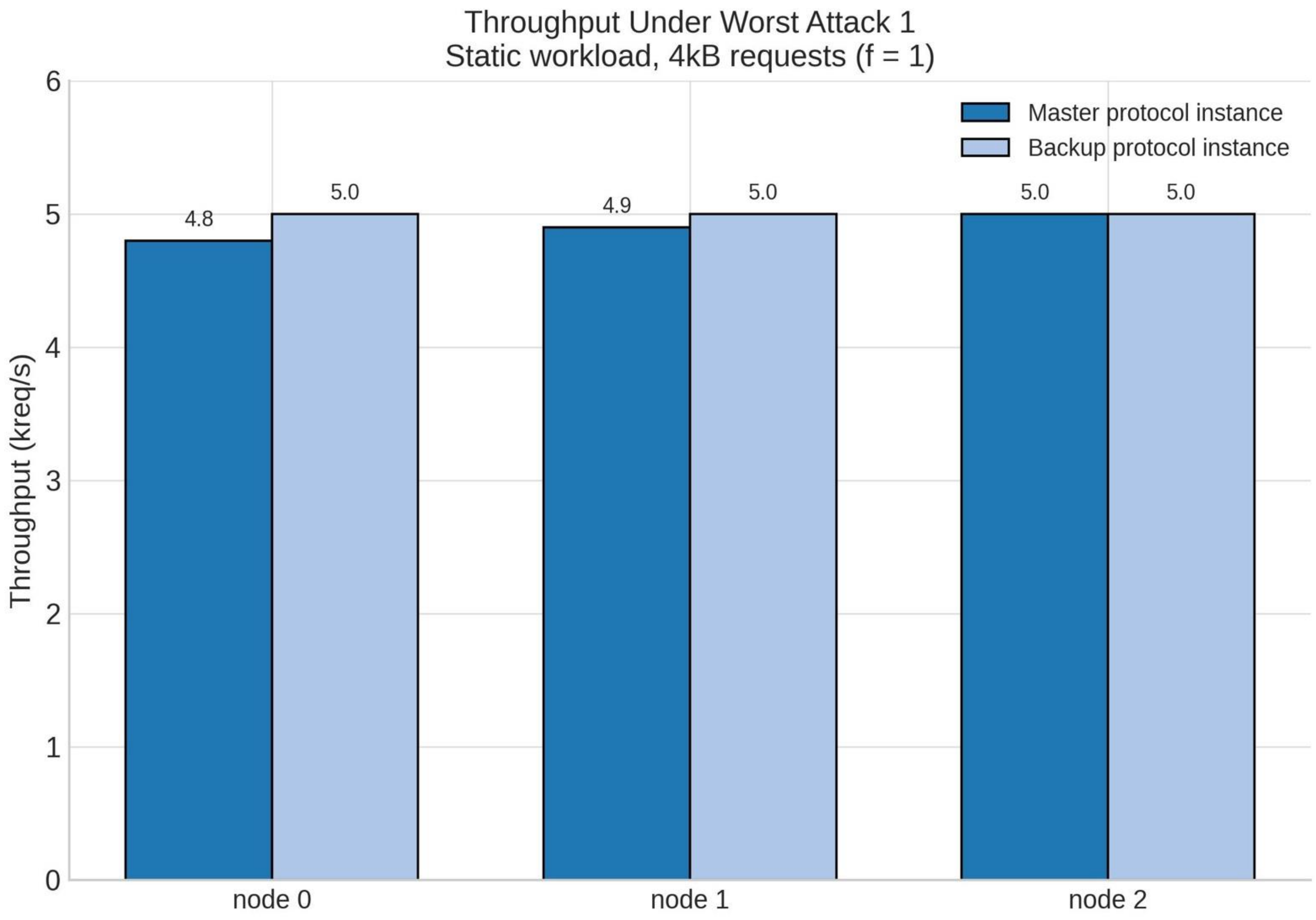
Attack Strategy:

- Targeted Client Traffic
- Flooding with Invalid PROPAGATE Messages
- Replica-Level Flooding
- Protocol Sabotage



Performance Evaluation - Worst-Attack-1

- ❖ Consistent Throughput Across Nodes
- ❖ Master vs Backup Comparison
- ❖ RBFT Monitoring Mechanism

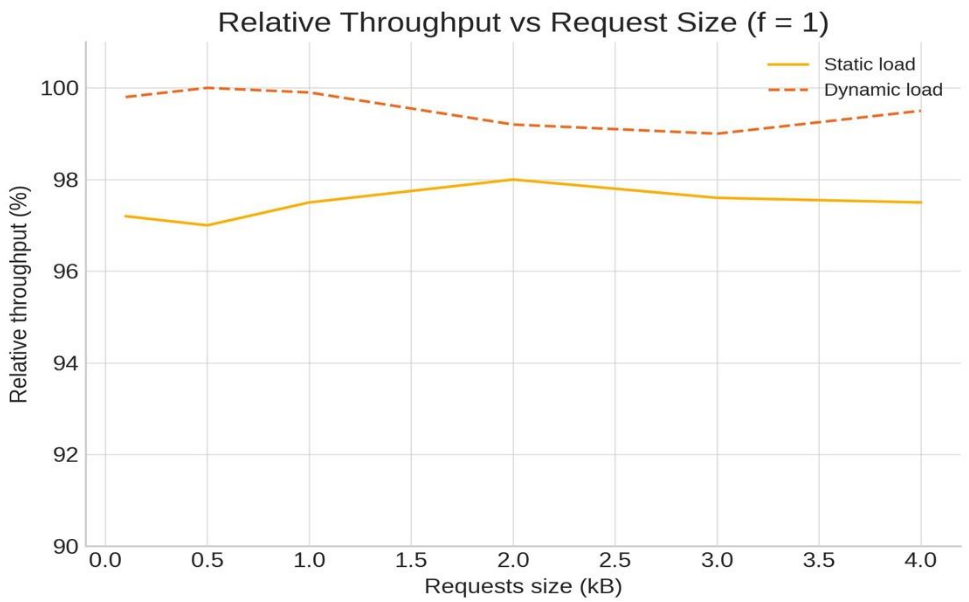


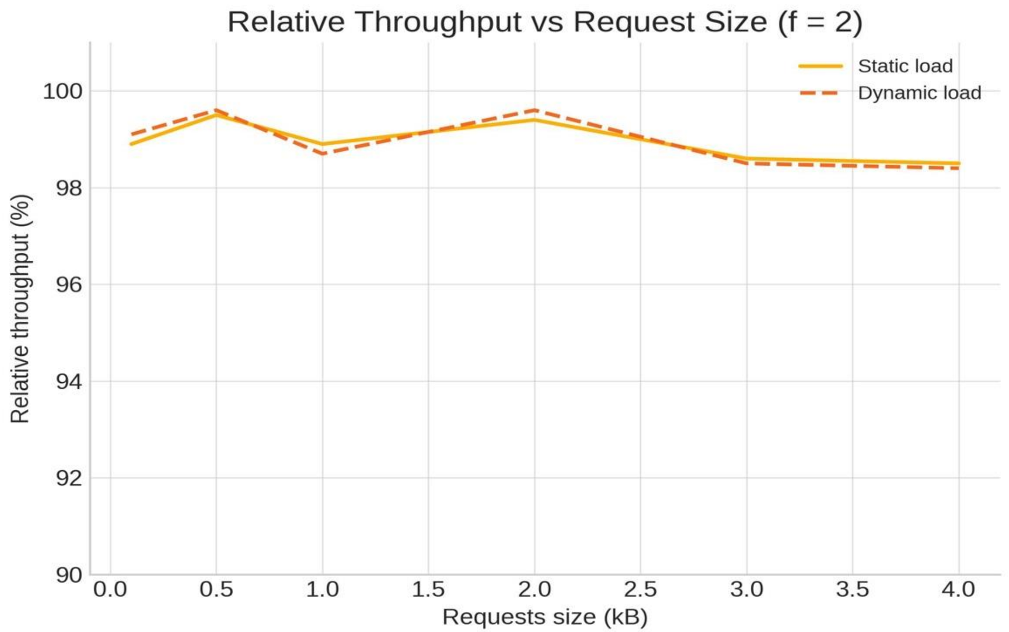
Performance Evaluation - Worst-Attack-2

- f faulty nodes and all clients are faulty
- The primary of the master protocol instance is faulty and runs on a faulty node
- The goal is to make the master appear normal by disrupting backup instances

Attack Strategy:

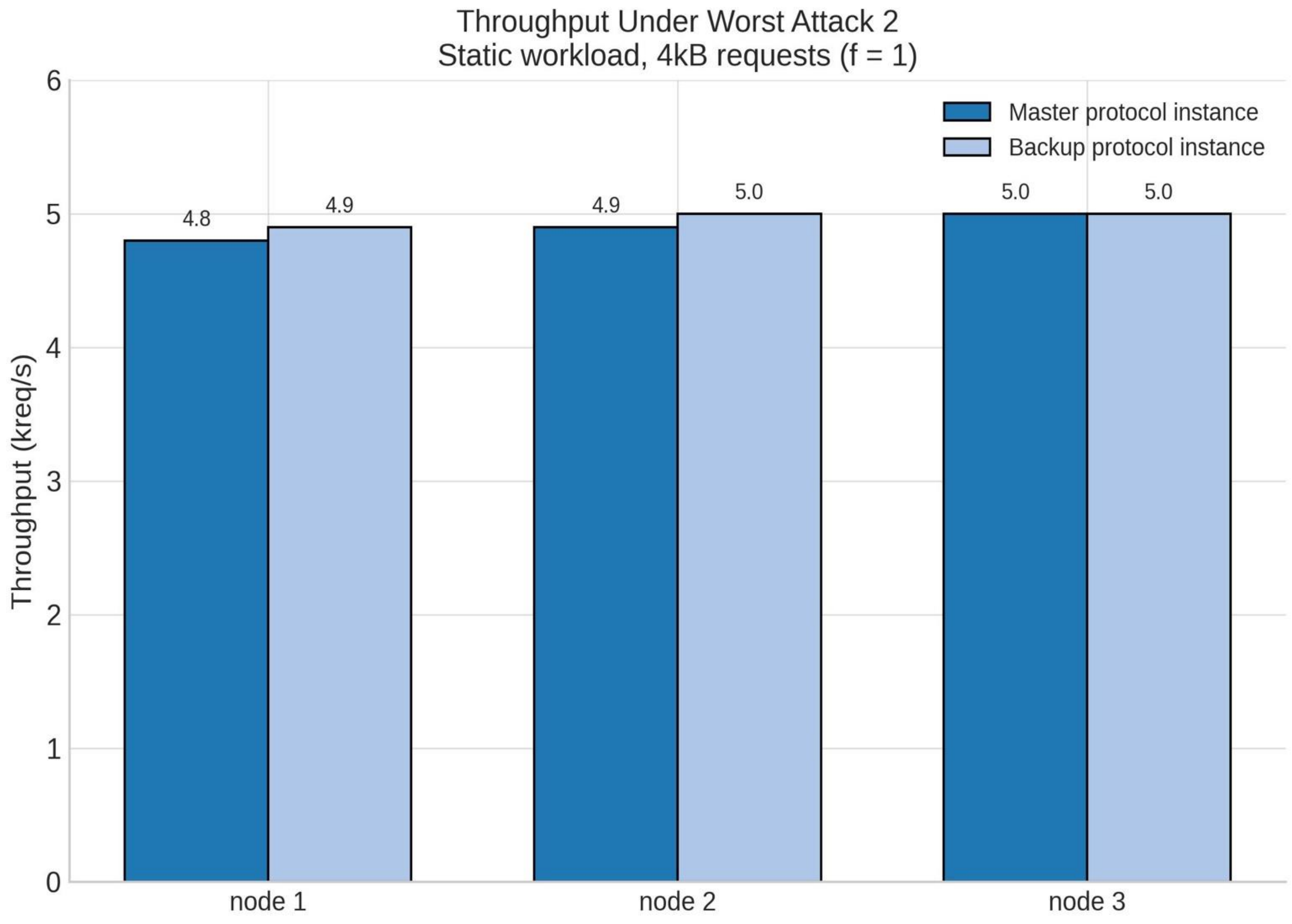
- Targeted Client Traffic
- Flooding with Invalid PROPAGATE Messages
- Replica-Level Flooding





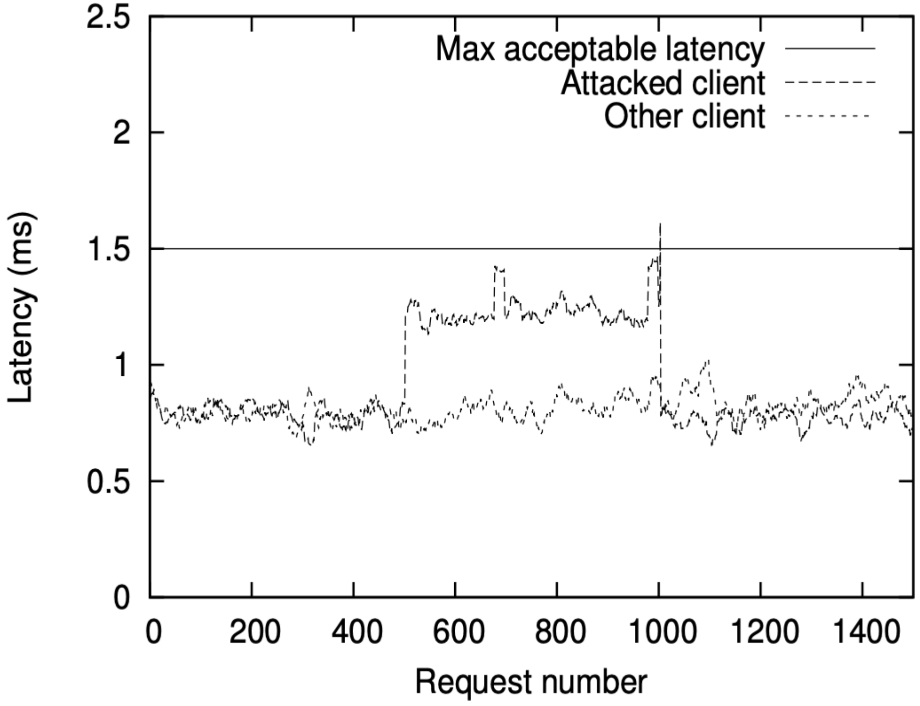
Performance Evaluation - Worst-Attack-2

- ❖ Consistent Throughput Across Nodes
- ❖ Master vs Backup Protocol Instances
- ❖ RBFT's Robustness



Performance Evaluation - Unfair Primary Attack

- The primary of the master protocol instance is malicious, attempting to delay one client's requests
- Λ (Lambda): Maximum acceptable latency per request = 1.5 m
- Ω (Omega): Maximum acceptable difference between the average latency of a client on different protocol instances



RBFT Response:

- Monitoring detects latency violation (Λ exceeded)
- Nodes initiate a Protocol Instance Change
- The malicious primary is evicted
- A correct replica takes over as the new primary
  - Restores fairness → both clients receive consistent,
  - low-latency responses

Critical Analysis

Pros

✅ Throughput monitoring ensures that performance remains stable, even under attack.
Only 3% degradation under fault !

✅ Latency tracking prevents unfair request ordering, ensuring fairness for all clients.

✅ Multiple protocol instances prevent a single primary from controlling the system.

Cons

Limitation – Open-Loop System Focus

❌ No proposed solution for adapting to closed-loop systems.

✓ State synchronization to avoid excessive delays.

Over-Reliance on Performance Monitoring

❌ Fixed detection thresholds (Δ, Λ, Ω) may not adapt well to network variations.

✓ Reputation-based primary selection based on historical performance.

High Overhead – Running Multiple Instances

❌ Running f+1 instances per node increases CPU & memory usage. No cost analysis provided in the paper.

✓ Resource utilization analysis (energy, memory, CPU impact).

✓ Adaptive instance management to scale instances dynamically

Conclusion

Existing BFT Protocols Lack Robustness:
State-of-the-art BFT protocols can suffer severe performance degradation under malicious primaries.

RBFT: A New Approach to Robustness:
Introduces Redundant Byzantine Fault Tolerance by running multiple BFT instances in parallel.
Uses monitoring mechanisms to detect underperforming or malicious primaries.

Resilience Without Compromising Performance:
Fault-free performance of RBFT is on par with leading robust BFT protocols.
In the worst-case scenario, RBFT limits throughput degradation to ≤3%, even with colluding malicious clients and nodes.

Scales Better with Fault Tolerance:
Performance impact is smaller with $f = 2$ than with $f = 1$

Thank you