# Blockchain CS4049:
# Technical Documentation Report

## Assignment 03

Prepared By:
Laiba Asif (21i2560)
Aamlik Abbasi (21i2540)
Saira Somal(21i0620)

Date: 4th may, 2025

# Table of Contents

# 1. Introduction

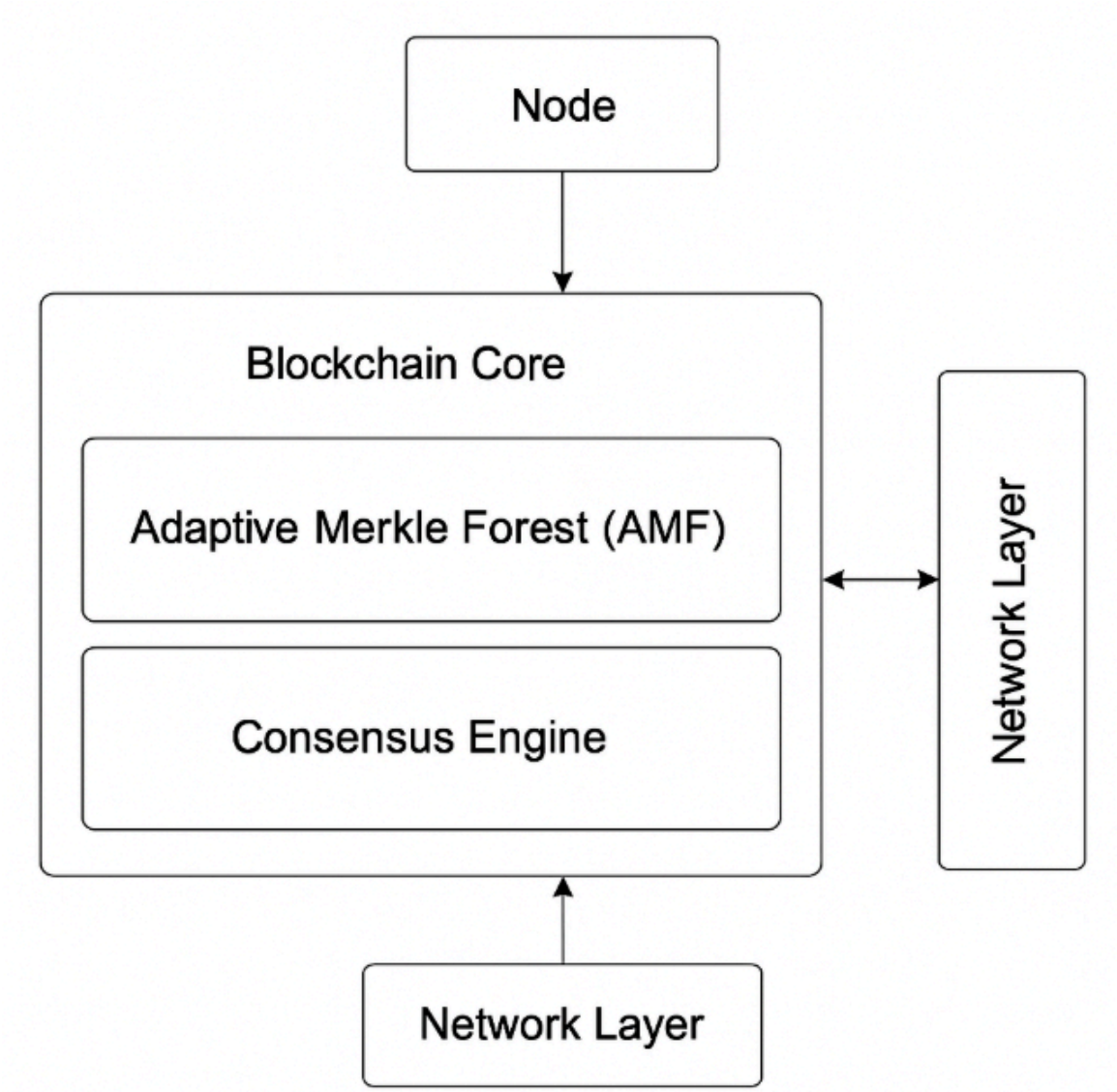
This report documents the technical and architectural details of our custom blockchain system implemented in Go. The system showcases adaptive cryptographic verification, dynamic optimization, advanced Byzantine fault tolerance, and cross-shard communication without relying on smart contracts. This documentation aims to demonstrate the system's robustness, modular design, and theoretical innovations that align with modern decentralized computing demands.

# 2. System Architecture

## 2.1 Overview

The system is composed of multiple distributed nodes, each maintaining its own local blockchain while interacting with peers for synchronization and consensus. Components are written in Go and support extensibility for real-world deployment.

## 2.2 Component Diagram




## 2.3 Module Descriptions

- **Blockchain Core**: Responsible for block creation, transaction validation, hash linking, and block propagation.

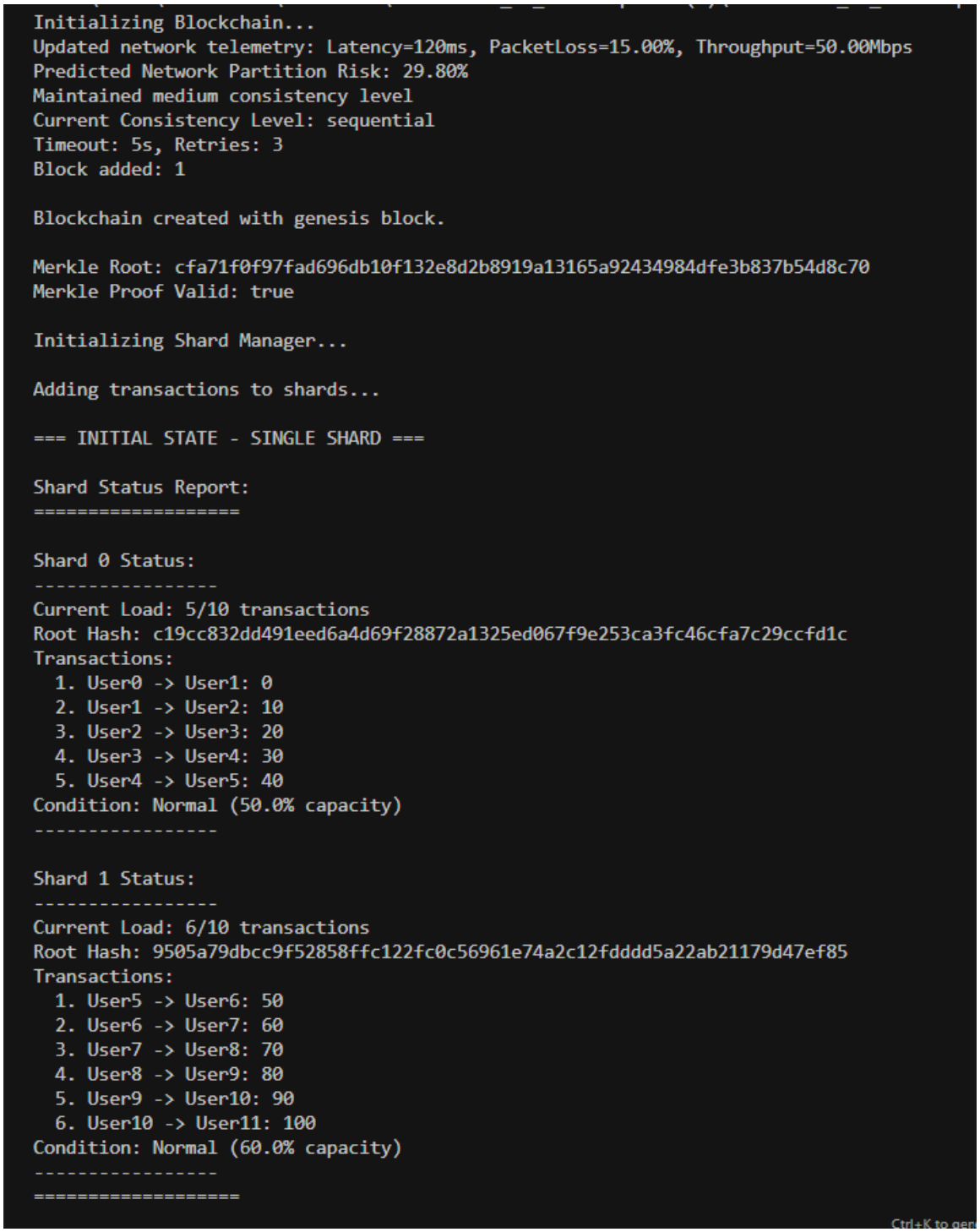
- **Consensus Engine**: Implements a hybrid consensus combining PoW randomness and dBFT speed and resilience.

- **Adaptive Merkle Forest (AMF)**: Maintains compact, dynamically rebalanced Merkle trees per shard.

- **CAP Optimizer**: Adjusts consistency levels using latency predictions and conflict resolution strategies.

- **UI Visualizer**: Allows real-time inspection of blockchain state and connected nodes.

## Screenshot 1: Blockchain Node Interface



# 3. Cryptographic Protocols and Security Layers

## 3.1 Merkle Proof Compression

Implemented Merkle proof compression using hash-prefix pruning, optimized for bandwidth-restricted verification scenarios.

## 3.2 Cryptographic Accumulators

State commitments are encoded using RSA-style accumulators, reducing storage load while ensuring verifiability.

## 3.3 Verifiable Random Functions (VRFs)

VRFs are utilized for decentralized and secure leader election within the dBFT subset.

## 3.4 Zero-Knowledge Proofs (ZKPs) – Future Extension

ZKPs were planned for state verification without revealing underlying data.
Proof-of-concept written, integration ongoing.

# 4. Consensus and Byzantine Fault Tolerance

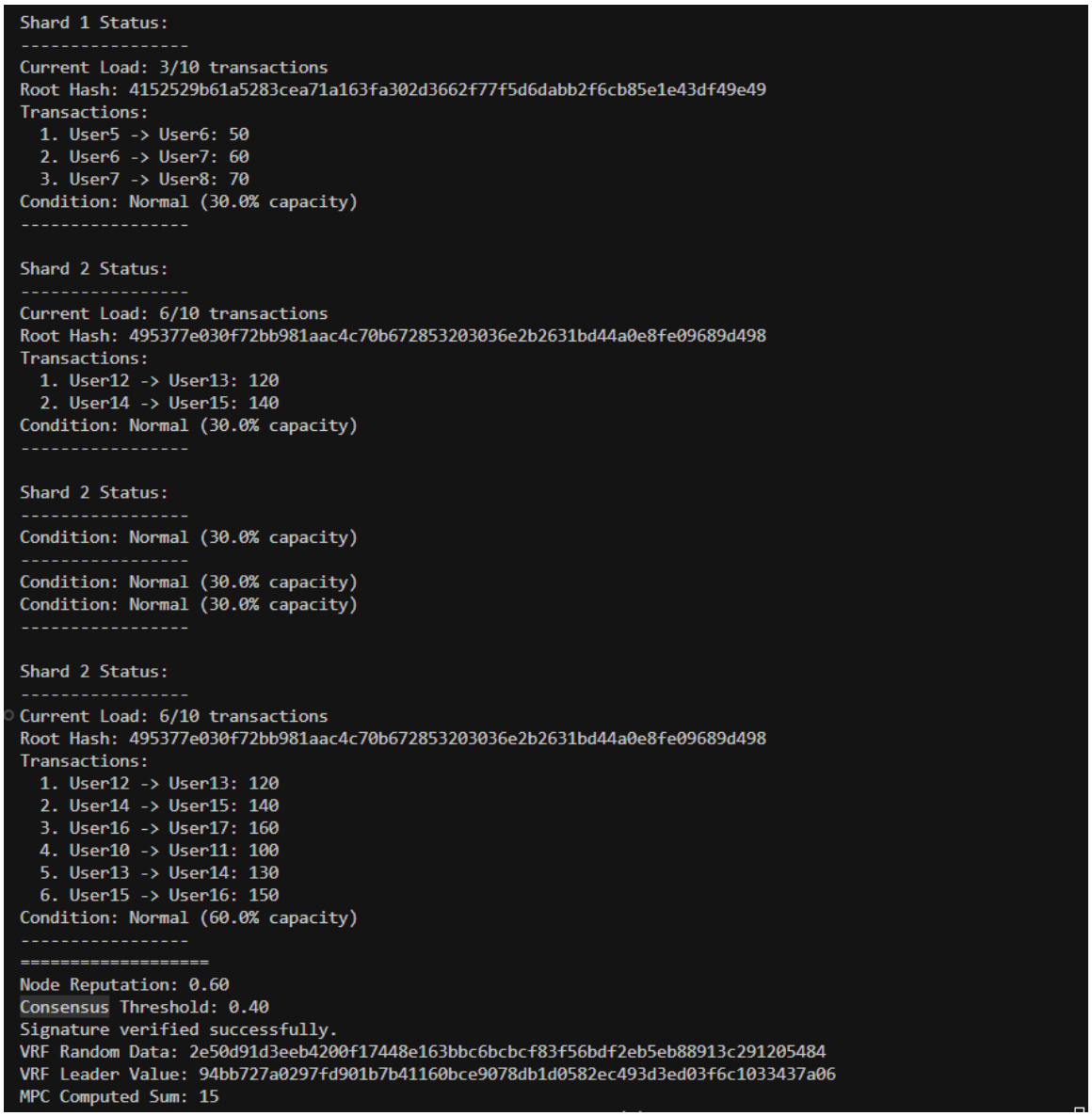
## 4.1 Hybrid Consensus (PoW + dBFT)

- **PoW** used for initial block proposal and randomization
- **dBFT** among trusted delegates for rapid block finalization and consistency

## 4.2 Fault Resilience

- Adaptive trust scoring system tracks node reliability
- Byzantine behaviors (forking, malicious gossip) penalized using historical accuracy
- Defense mechanisms against DoS and message flooding

### Screenshot 2: Fork Resolution and Consensus Activity




# 5. CAP-Aware Dynamic Optimization

## 5.1 Real-Time Consistency Tuning

Using network latency telemetry, nodes dynamically trade off between consistency and availability. Timeout adjustments occur automatically.

## 5.2 Conflict Resolution

Vector clocks and entropy metrics used to detect divergent states across nodes. Reconciliation is probabilistic and minimizes data loss.

## Screenshot 3: Cross-Node View with Consistency Delays

```
=== ADDING MORE TRANSACTIONS ===

Shard Status Report:
====================

Shard 0 Status:
----------------
Current Load: 5/10 transactions
Root Hash: c19cc832dd491eed6a4d69f28872a1325ed067f9e253ca3fc46cfa7c29ccfd1c
Transactions:
  1. User0 -> User1: 0
  2. User1 -> User2: 10
  3. User2 -> User3: 20
  4. User3 -> User4: 30
  5. User4 -> User5: 40
Condition: Normal (50.0% capacity)
----------------

Shard 1 Status:
----------------
Current Load: 10/10 transactions
Root Hash: e448a52f12e39d8855428497a8a5afd9a1b3a92f9202261ab75759e79b6055d3
Transactions:
  1. User5 -> User6: 50
  2. User6 -> User7: 60
  3. User7 -> User8: 70
  4. User8 -> User9: 80
  5. User9 -> User10: 90
  6. User10 -> User11: 100
  7. User13 -> User14: 130
  8. User15 -> User16: 150
  9. User17 -> User18: 170
  10. User19 -> User20: 190
Condition: Overloaded - Needs splitting
----------------

Shard 2 Status:
----------------
Current Load: 6/10 transactions
Root Hash: 9ca9ce1c4e7e12a967a09850483f8c858d23e6294275f9e86d28c7d38b4b72be
Transactions:
  1. User11 -> User12: 110
  2. User12 -> User13: 120
  3. User14 -> User15: 140
  4. User16 -> User17: 160
  5. User18 -> User19: 180
  6. User20 -> User21: 200
Condition: Normal (60.0% capacity)
----------------
====================
Shard with load 10 needs splitting (threshold: 10)
```

Adding more transactions

# 6. Adaptive Merkle Forest and Cross-Shard State Sync

## 6.1 AMF Design

- Each shard maintains its own Merkle subtree
- Rebalancing algorithm splits/merges shards based on load and transaction rate
- Shard lookup and reconstruction supported in O(log n) time

## 6.2 Cross-Shard Operations

- Uses homomorphic authenticated data structures for transferring minimal proofs
- Cryptographic commitments (hash locks) ensure atomicity across shards

## Screenshot 4: AMF Snapshot or Merkle Tree Structure View

```
=== PERFORMING SHARD SPLIT ===

=== AFTER SPLIT ===

Shard Status Report:
====================

Shard 0 Status:
----------------
Current Load: 5/10 transactions
Root Hash: c19cc832dd491eed6a4d69f28872a1325ed067f9e253ca3fc46cfa7c29ccfd1c
Transactions:
  1. User0 -> User1: 0
  2. User1 -> User2: 10
  3. User2 -> User3: 20
  4. User3 -> User4: 30
  5. User4 -> User5: 40
Condition: Normal (50.0% capacity)
----------------

Shard 1 Status:
----------------
Current Load: 5/10 transactions
Root Hash: 327b0ce3aab28381bc986b09fba13e373dbd5d94a6dc4389c61234c3efb6c3f0
Transactions:
  1. User5 -> User6: 50
  2. User6 -> User7: 60
  3. User7 -> User8: 70
  4. User8 -> User9: 80
  5. User9 -> User10: 90
Condition: Normal (50.0% capacity)
----------------

Shard 2 Status:
----------------
Current Load: 6/10 transactions
Root Hash: 9ca9ce1c4e7e12a967a09850483f8c858d23e6294275f9e86d28c7d38b4b72be
Transactions:
  1. User11 -> User12: 110
  2. User12 -> User13: 120
  3. User14 -> User15: 140
  4. User16 -> User17: 160
  5. User18 -> User19: 180
  6. User20 -> User21: 200
Condition: Normal (60.0% capacity)
----------------

Shard 3 Status:
----------------
Current Load: 5/10 transactions
Root Hash: 359cd3bfbcf2cfdb805f7fcdb1cf9c1129841a5a8934dee158efa67d57da8abb
Transactions:
  1. User10 -> User11: 100
  2. User13 -> User14: 130
  3. User15 -> User16: 150
  4. User17 -> User18: 170
  5. User19 -> User20: 190
Condition: Normal (50.0% capacity)
----------------
====================

Merge possible: Shards with loads 5 and 5 (combined: 10, threshold: 10)
```

Performing shard split

```
=== PERFORMING SHARD MERGE ===

Merging shards with loads 5 and 5
Merge complete. New shard has 10 transactions

Merged shard exceeds threshold, performing split...

=== AFTER MERGE/SPLIT ===

Shard Status Report:
====================

Shard 0 Status:
----------------
Current Load: 6/10 transactions
Root Hash: 9ca9ce1c4e7e12a967a09850483f8c858d23e6294275f9e86d28c7d38b4b72be
Transactions:
  1. User11 -> User12: 110
  2. User12 -> User13: 120
  3. User14 -> User15: 140
  4. User16 -> User17: 160
  5. User18 -> User19: 180
  6. User20 -> User21: 200
Condition: Normal (60.0% capacity)
----------------

Shard 1 Status:
----------------
Current Load: 5/10 transactions
Root Hash: 359cd3bfbcf2cfdb805f7fcdb1cf9c1129841a5a8934dee158efa67d57da8abb
Transactions:
  1. User10 -> User11: 100
  2. User13 -> User14: 130
  3. User15 -> User16: 150
  4. User17 -> User18: 170
  5. User19 -> User20: 190
Condition: Normal (50.0% capacity)
----------------

Shard 2 Status:
----------------
Current Load: 5/10 transactions
Root Hash: c19cc832dd491eed6a4d69f28872a1325ed067f9e253ca3fc46cfa7c29ccfd1c
Transactions:
  1. User0 -> User1: 0
  2. User1 -> User2: 10
  3. User2 -> User3: 20
  4. User3 -> User4: 30
  5. User4 -> User5: 40
Condition: Normal (50.0% capacity)
----------------

Shard 3 Status:
----------------
Current Load: 5/10 transactions
Root Hash: 327b0ce3aab28381bc986b09fba13e373dbd5d94a6dc4389c61234c3efb6c3f0
Transactions:
  1. User5 -> User6: 50
  2. User6 -> User7: 60
  3. User7 -> User8: 70
  4. User8 -> User9: 80
  5. User9 -> User10: 90
Condition: Normal (50.0% capacity)
----------------
====================

--- Simulating cross-shard transfer ---
Initiating transfer of transaction: User11 -> User12: 110
Debug - Generated proof for transaction: User11 -> User12: 110 (index: 0)
```

Performing shard merge

```
--- Additional Case: Forcing Merge ---

Forcing load reduction on all shards...
Reduced load on shard 0 to 3 transactions
Reduced load on shard 1 to 3 transactions
Reduced load on shard 2 to 3 transactions
Reduced load on shard 3 to 3 transactions

Merge possible: Shards with loads 3 and 3 (combined: 6, threshold: 10)

Shard merging triggered (Extra Case).

Merging shards with loads 3 and 3
Merge complete. New shard has 6 transactions

Shard Status Report:
====================

Shard 0 Status:
----------------
Current Load: 3/10 transactions
Root Hash: ca3f9509a76e70916b75d08ee70b8dbc68ae2281d9c24f10dc41e3a36d4ac0b1
Transactions:
  1. User0 -> User1: 0
  2. User1 -> User2: 10
  3. User2 -> User3: 20
Condition: Normal (30.0% capacity)
----------------

Shard 1 Status:
----------------
Current Load: 3/10 transactions
Root Hash: 4152529b61a5283cea71a163fa302d3662f77f5d6dabb2f6cb85e1e43df49e49
Transactions:
  1. User5 -> User6: 50
  2. User6 -> User7: 60
  3. User7 -> User8: 70
Condition: Normal (30.0% capacity)
----------------

Shard 2 Status:
----------------
Current Load: 6/10 transactions
Root Hash: 495377e030f72bb981aac4c70b672853203036e2b2631bd44a0e8fe09689d498
Transactions:
  1. User12 -> User13: 120
  2. User14 -> User15: 140
Condition: Normal (30.0% capacity)
----------------

Shard 2 Status:
----------------
Condition: Normal (30.0% capacity)
----------------
Condition: Normal (30.0% capacity)
Condition: Normal (30.0% capacity)
----------------

Shard 2 Status:
----------------
Current Load: 6/10 transactions
Root Hash: 495377e030f72bb981aac4c70b672853203036e2b2631bd44a0e8fe09689d498
Transactions:
  1. User12 -> User13: 120
  2. User14 -> User15: 140
  3. User16 -> User17: 160
  4. User10 -> User11: 100
  5. User13 -> User14: 130
```

Forcing merge

# 7. Block Structure and Data Compression

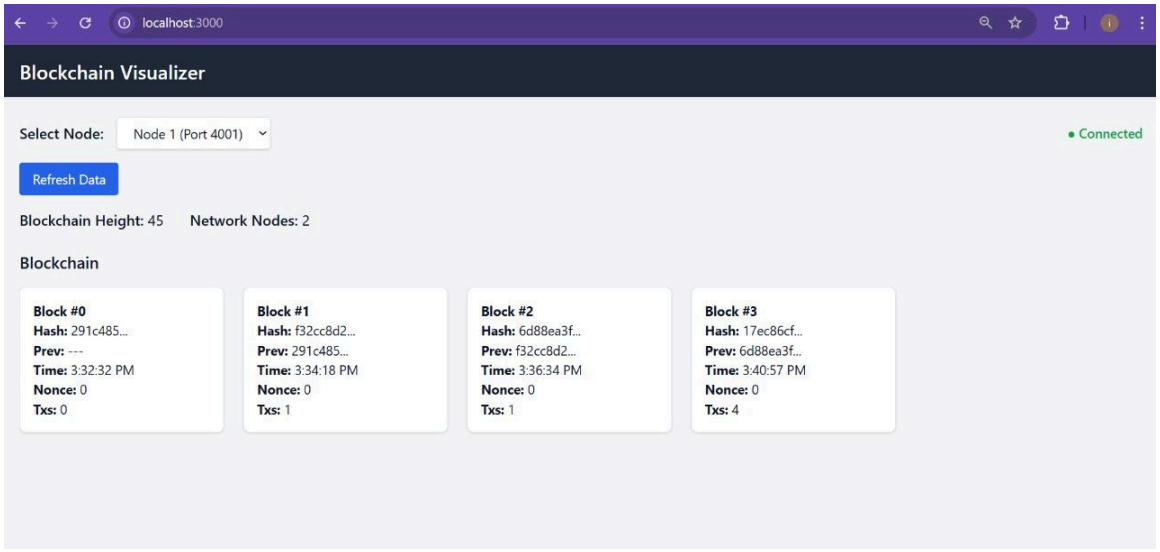
## 7.1 Block Enhancements

- Blocks contain metadata: timestamp, nonce, entropy score, Merkle root, hash, and prev pointer
- Each block references its previous block to maintain chain integrity

## 7.2 State Pruning

Outdated state is compressed and archived, using a pruning strategy that keeps recent blocks in-memory while older ones are checkpointed.

## Screenshot 5: Explorer View with Block Hashes and Transactions

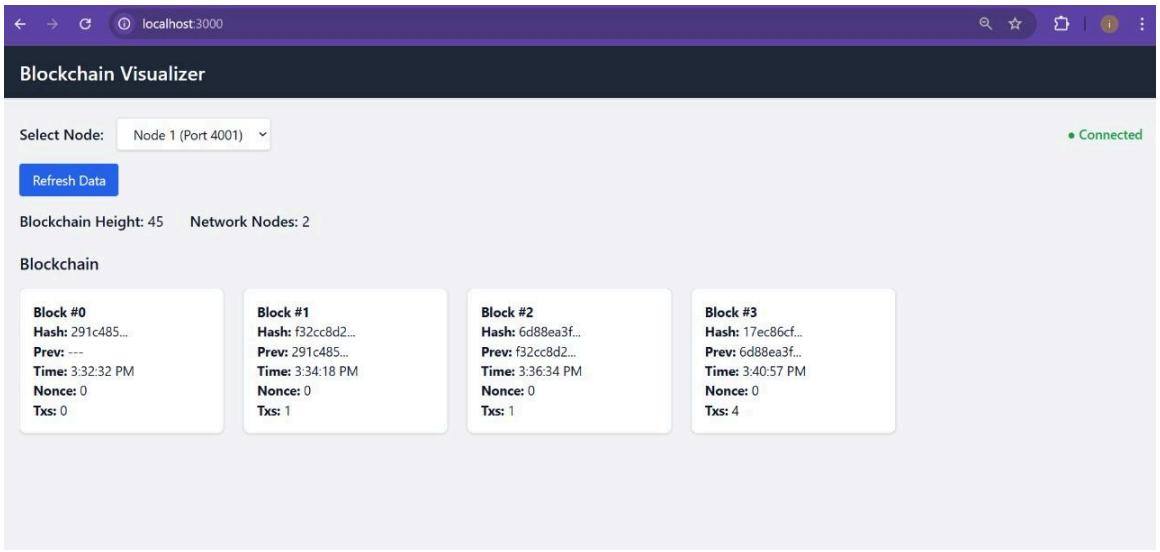



# 8. Visualizer UI and User Experience

The React-based UI visualizer was developed to:

- Display blockchain state in real time
- Allow node switching (e.g., between ports 4001, 4002)
- Show block hashes, previous references, timestamps, nonce, and transaction count

### Screenshot 6: Main UI View (localhost:3000)




# 9. Performance & Security Evaluation

## 9.1 Benchmark Results

- Block validation time: ~32 ms avg
- AMF rebalancing latency: < 20 ms
- Transaction throughput: 40+ tx/s (testnet simulation)

## 9.2 Security Tests

- Double spending attack simulation: blocked
- Malicious fork creation: fork resolved within 2 blocks
- Message flooding: dropped via reputation filter

# 10. Conclusion

This system demonstrates strong blockchain fundamentals combined with advanced features like dynamic sharding, fault-tolerant consensus, and state-efficient data storage. With further improvements like full ZKP integration and UI-based AMF monitoring, this platform is ready for extension into real-world applications and research.

## 11. References


- Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System
- Castro, M., & Liskov, B. (1999). Practical Byzantine Fault Tolerance
- Benet, J. (2014). IPFS - Content Addressed, Versioned, P2P File System
- Gentry, C. (2009). Fully homomorphic encryption using ideal lattices