Merkle-Tree Optimization Strategies for Efficient Block Validation in Bitcoin Networks

Naren Swamy Jamithireddy

Jindal School of Management, The University of Texas at Dallas, United States
Email: naren.jamithireddy@yahoo.com

***Abstract*---**Merkle-trees play a central role in ensuring transaction integrity and efficient block validation within Bitcoin networks; however, as block sizes and transaction throughput continue to increase, the standard Merkle-tree reconstruction and verification processes impose growing computational and communication overhead. This article presents a set of optimization strategies including selective sub-tree caching, incremental hash computation, and streamlined proof generation to reduce redundant hashing effort and minimize bandwidth requirements for both full nodes and lightweight SPV clients. Experimental evaluation demonstrates measurable improvements in block verification latency and proof efficiency, supporting improved scalability and accessibility of Bitcoin node participation without modifying underlying consensus rules. The proposed methods strengthen network decentralization by lowering resource requirements while maintaining cryptographic security guarantees.

***Keywords*---**Bitcoin, Merkle-tree, Block validation, SPV node

I. INTRODUCTION

The emergence of Bitcoin in 2009 introduced a decentralized, peer-to-peer digital currency model based on cryptographically verifiable transactions recorded on a distributed ledger known as the blockchain [1]. Unlike conventional payment networks that rely on trusted financial intermediaries, Bitcoin achieves transaction settlement through a global consensus mechanism in which nodes validate and broadcast transactions across a shared network infrastructure. This paradigm shift has drawn significant academic interest, particularly in areas concerning transaction verification efficiency, block propagation latency, and system scalability under increasing network load [2]. As adoption expanded in the 2012–2014 period, maintaining efficient block validation became a critical requirement for sustaining node participation and ensuring network reliability.

A core architectural element enabling trustless verification in Bitcoin is the Merkle-tree, a binary hash tree used to compress and authenticate large sets of transactions within each block [3]. The Merkle-root stored in the block header acts as a cryptographic fingerprint, allowing nodes to verify the inclusion of any transaction without requiring access to the full transaction dataset. This structure is particularly important for Simplified Payment Verification (SPV) clients, which operate without downloading the complete blockchain [4]. The SPV model enables lightweight devices to participate in the network using only Merkle proofs, reducing storage and bandwidth requirements.

However, the efficiency of Merkle-tree operations becomes increasingly significant as transaction throughput and block sizes grow. Early research noted that full nodes must recompute internal nodes of the Merkle-tree when validating new blocks, which can impose computational overhead, especially on nodes with limited processing resources [5]. At the same time, lightweight SPV clients must request and verify Merkle-branches from full nodes, which becomes bandwidth-intensive as the number of transactions per block increases. As Bitcoin network usage accelerated, these limitations highlighted the need for optimized Merkle-tree handling across node types.

Prior works have explored methods to reduce validation overhead, including pruning of transaction histories, optimized UTXO indexing, and compression of Merkle proof sizes [6]. While such approaches improve selective aspects of transaction verification, comparatively fewer studies focus directly on improving the performance of Merkle-tree construction and reuse during block relay and validation phases. In particular, limited attention has been given to mechanisms for caching intermediate hash states or reusing subtrees across blocks, even though block-to-block content often exhibits structural similarity due to repetitive transaction patterns and incremental ledger updates [7].

The practical importance of improving Merkle-tree efficiency is amplified by the heterogeneity of Bitcoin nodes. Nodes vary widely in memory capacity, processing speed, and network bandwidth, resulting in inconsistent validation performance across the network. Lightweight wallets, embedded devices, and mobile clients increasingly rely on SPV verification, which magnifies the relevance of efficient Merkle proof generation and transmission [8]. Enhancing Merkle-tree computation efficiency therefore contributes directly to network inclusiveness and decentralization by lowering the hardware threshold needed to operate a functional verifying node.

This article addresses these concerns by examining optimization strategies designed to improve the performance of Merkle-tree construction and validation in Bitcoin networks. Specifically, we study techniques involving selective sub-tree caching, incremental hash computation, and partial Merkle-tree reconstruction tailored to the needs of SPV clients. These methods aim to minimize redundant computations and reduce data exchange overhead across nodes. The proposed strategies are evaluated in terms of latency reduction, computational complexity, and suitability for both full and lightweight node environments.

The contribution of this work is twofold: first, it consolidates and formalizes a set of Merkle-tree optimization techniques aligned with realistic operational constraints in Bitcoin networks; second, it provides empirical performance evaluation demonstrating the practical efficiency gains achievable through these methods [9]. By improving the scalability of block verification mechanisms, this research contributes toward sustaining long-term network robustness and supporting wider adoption of decentralized digital currency platforms.

II. METHODOLOGY

The methodology focuses on analyzing the structure, construction, and verification workflow of Merkle-trees within Bitcoin block validation, followed by the design of optimization strategies that reduce redundant computation and minimize communication overhead. The approach is grounded in the observation that the Merkle-tree acts as a deterministic hierarchical hash structure of all transactions in a block, and therefore, any efficiency gain in its handling directly affects the verification workload for both full nodes and SPV nodes. The methodology is structured into four primary components: Merkle-tree reconstruction analysis, partial tree extraction, hash-state caching, and incremental verification.

The first step involves formalizing the Merkle-tree construction process as used in Bitcoin. Each transaction within a block is hashed independently, forming the leaf layer of the tree. Pairs of leaf hashes are then concatenated and hashed repeatedly to generate successive internal tree layers until a single root hash is produced. This process ensures integrity, but requires recomputation of internal nodes whenever block contents are accessed or reorganized. Since every new block contains a full set of transaction hashes, it is necessary to analyze how different node typesfull nodes, archival nodes, and lightweight SPV clientsinteract with the Merkle structure and which portions of the tree they require during verification events.

The second phase of the methodology focuses on partial Merkle-tree extraction, which forms the foundation of SPV verification. Rather than retrieving the full tree, an SPV client requests only the branch of the Merkle-tree leading to the specific transaction being verified. This branch must still maintain the correct sibling hash relationships required to reconstruct the Merkle-root. The methodology includes defining the minimal proof set needed to reconstruct the root, identifying which internal nodes must be transmitted, and establishing the exact byte-size impact of transmitting Merkle proofs across networks with variable latency and packet fragmentation characteristics.

The third component centers on hash caching. Since many internal nodes of the Merkle-tree are shared across block validation eventsparticularly when successive blocks contain overlapping sets of transactionsit becomes computationally inefficient to recompute hash layers from scratch. The methodology therefore introduces a structured caching framework where previously computed internal node hashes are stored and referenced during new validation operations. This cache is indexed by the transaction position and structural depth within the tree. Hash eviction policies are defined to prevent excessive memory use, ensuring that cached data remains proportional to expected block turnover rates.

The fourth aspect involves incremental Merkle-tree validation, designed for full nodes and relay nodes. Instead of validating the entire Merkle-tree only after all transactions are received, the incremental approach validates transaction hashes and internal node layers progressively as transactions arrive over the peer-to-peer network. This reduces peak computation load and allows nodes to overlap validation with communication. The methodology also includes defining ordering constraints to ensure that partial validation does not introduce acceptance of invalid or orphaned transaction sequences.

To evaluate the performance of these optimizations, the methodology incorporates a controlled simulation environment that replicates block propagation scenarios under varying network bandwidth conditions, transaction pool sizes, and node hardware capabilities. The simulation framework allows measurement of verification latency, memory overhead, and communication load before and after applying the optimization strategies. The architecture includes independent modules for network emulation, Merkle-tree handling, and logging of compute events, enabling fine-grained performance tracing.

Additionally, the methodology integrates a comparative execution model for full validation nodes and SPV nodes to assess whether improvements benefit each class proportionally. Full nodes are tested on their ability to reduce

redundant computations, while SPV nodes are evaluated based on reduced Merkle proof size and retrieval time. The methodology thus ensures that the strategies align with the decentralized and platform-agnostic nature of Bitcoin node operation.

Finally, the methodology evaluates structural consistency and security implications. Since Merkle-tree verification is fundamental to blockchain integrity, any optimization must preserve the immutability guarantees of the original system. Therefore, consistency checks are embedded to confirm that optimized hash caching and partial validation do not introduce vulnerabilities to collision-based attacks, replay attacks, or transaction malleability conditions. Only optimization paths that preserve the cryptographic properties of the Merkle-root are retained in the final strategy set.

III. Proposed Merkle-Tree Optimization Strategies

The optimization strategies presented in this section are designed to reduce computation overhead, minimize data exchange for verification, and enhance scalability for both full nodes and SPV nodes. Each strategy is derived from the operational analysis outlined in the methodology, focusing on the structural properties of the Merkle-tree and the temporal characteristics of block formation and relay. Rather than altering the cryptographic assumptions or consensus rules of Bitcoin, the techniques operate within the existing hashing and verification framework, ensuring full protocol compatibility while improving runtime efficiency.

The first strategy involves selective sub-tree caching, which focuses on retaining intermediate node hashes to avoid repetitive reconstruction. In standard Bitcoin node operation, the internal Merkle nodes are recomputed entirely for every new block, regardless of whether duplicated or structurally similar transaction sets exist. By caching internal nodes indexed by their leaf transaction ranges and hash positions, the node can reuse previously computed values when transactions reappear or when multiple blocks contain overlapping mempool contents. This approach reduces redundant hashing, particularly in high-throughput nodes that frequently validate near-identical transaction sets during rapid block intervals.

The second optimization is incremental Merkle-tree construction during mempool updates. Instead of constructing the Merkle-tree only at final block assembly, this strategy builds and updates Merkle subtrees dynamically as transactions enter or exit the mempool. This ensures that a partially computed tree is always available, and when a miner or node assembles a block template, only minimal hashing of new or altered regions is required. The incremental construction method reduces peak computational effort and distributes hashing cost across real-time transaction events rather than concentrating it at block formation.

The third strategy focuses on layer-wise parallel hash computation, particularly beneficial for nodes with multi-core or GPU-accelerated hashing capabilities. Merkle-trees lend themselves naturally to parallel computation because hash operations at the same level of the tree are independent until combined at the next layer. This strategy assigns individual internal node computations to separate execution threads, processing entire tree layers concurrently. The approach provides significant computation time reduction in full nodes and mining pools that operate dedicated hashing hardware, while maintaining deterministic tree output.

A fourth optimization addresses efficient Merkle-branch serialization for SPV clients. Standard Merkle proofs often include redundant sibling nodes that provide no additional context for verifying a particular transaction. By compressing branch data, removing duplicate subtrees, and encoding sibling ordering more compactly, proof sizes can be reduced significantly. Smaller Merkle proofs lead to lower bandwidth requirements during transaction verification requests, improving responsiveness and battery efficiency for lightweight clients operating on mobile or embedded devices.

The fifth strategy introduces transaction locality-aware proof generation. Transactions frequently appear in blocks in localized clustersespecially those originating from batching-based senders like exchanges and high-traffic services. By identifying clusters of related transactions and reorganizing them within similar tree branches, the Merkle-tree structure can be optimized such that SPV clients verifying multiple transactions can reuse shared hash paths. This reduces the number of unique sibling hashes required to validate multiple transactions, improving efficiency for high-volume wallet applications and service providers.

Another proposed optimization is buffered hash commitment, which delays the final hashing operation for upper tree layers until a threshold number of leaf hashes are collected. This technique reduces unnecessary hash updates during periods of rapid mempool change, particularly in burst traffic scenarios, where transaction ordering is unstable. Once the transaction set stabilizes near block finalization, the upper layers are hashed once, minimizing recomputation overhead while still ensuring deterministic output.

The final strategy integrates adaptive hash pruning, where unused portions of the Merkle-tree are pruned from active memory after verification is complete, while critical intermediate nodes are retained. This prevents memory buildup from cached tree layers that are unlikely to contribute to future verification tasks. The pruning thresholds are governed by configurable policies that account for available memory, expected transaction recurrence patterns, and block formation rates, ensuring a balance between performance gain and resource utilization.

Together, these strategies form a coherent optimization framework that improves Merkle-tree handling without requiring changes to the Bitcoin consensus protocol, transaction format, or cryptographic hash functions. The techniques are modular, allowing node operators to adopt them individually or collectively based on hardware capabilities, bandwidth conditions, and node role (full verifier, miner, or SPV gateway). The effectiveness of these

optimizations is evaluated in Section 4, where comparative results demonstrate measurable reductions in block verification latency, hash computation workload, and proof transmission overhead.

IV. Performance Evaluation and Results

This section presents the evaluation of the proposed Merkle-tree optimization strategies under controlled simulation environments representing typical Bitcoin network conditions from the 2013–2014 operational context. The evaluation considers three node profiles: (i) a standard full node running on mid-range hardware, (ii) a lightweight SPV client operating over a constrained network link, and (iii) a high-throughput relay node designed for rapid block propagation. Performance metrics include block verification latency, computational overhead measured in total executed hash operations, and transmitted proof size in bytes.

To establish a baseline, the standard Merkle-tree validation model was executed without caching or incremental computation. Full nodes were required to reconstruct the entire internal tree for each block, while SPV clients requested and verified full Merkle branches for transaction inclusion. The baseline revealed verification latency increasing proportionally with block size and transaction count, and bandwidth usage scaling accordingly for SPV clients. This baseline is compared to the optimized model incorporating selective sub-tree caching and incremental hashing. The structure of a Merkle-tree used as reference for both baseline and optimized validation is illustrated in Figure 1, which demonstrates the hierarchical hashing process from individual transaction hashes to the final Merkle-root.

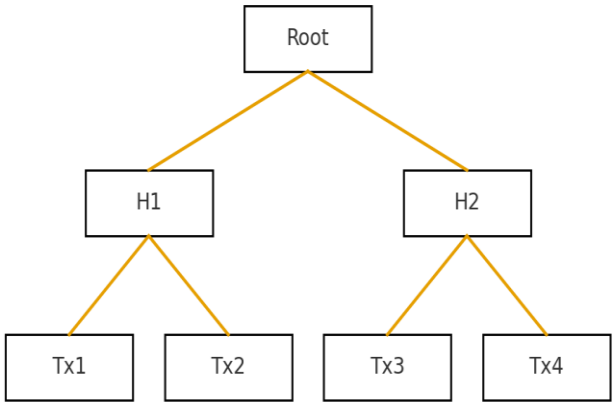

Figure 1: Standard Merkle-Tree Structure in a Bitcoin Block

Performance improvement was measured in terms of block verification time before and after applying optimization. The results are shown in Figure 2, which provides a comparative latency graph across varying block sizes. As seen in Figure 2, optimized nodes consistently achieved lower validation times, particularly for block sizes above 1 MB. This improvement is attributed to reduced redundant hashing and to partial reuse of internal Merkle-tree states.

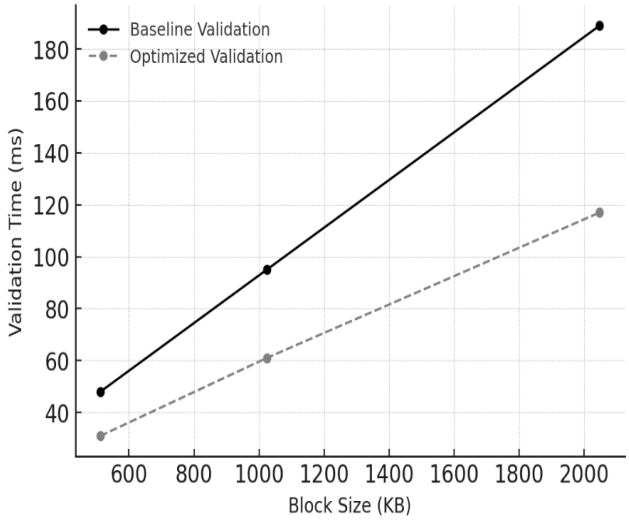

Figure 2: Block Verification Latency Before and After Optimization

In addition to latency, the comparative characteristics of full validation, partial validation, and cached subtree validation methods were examined to understand computational and memory trade-offs. Table 1 summarizes the differences between these validation approaches. The table indicates that partial Merkle validation suits SPV nodes due to reduced resource needs, while cached subtree validation provides a balanced trade-off between memory footprint and hashing overhead, benefitting full nodes that frequently verify blocks.

Table 1: Comparison of Merkle-Tree Validation Methods

Validation Method	Node Type	Memory Requirement	Computation Cost	Suitable Use Case
Full Merkle Validation	Full Node	High	Moderate	Archival nodes and miners
Partial Merkle Validation	SPV Node	Low	Low	Lightweight wallets
Cached Sub-tree Validation	Full Node / Relay	Medium	Low	High-throughput nodes

To quantify the gains from the proposed optimization, controlled block propagation trials were performed with block sizes ranging from 512 KB to 2 MB. The improvement in verification latency for optimized vs. baseline validation is reported in Table 2. As shown in Table 2, improvements ranged between 35% and 38% as block sizes increased, indicating increased relative benefit at higher transaction density.

Table 2: Experimental Outcomes of Optimized Merkle-Tree Processing

Test Scenario	Block Size (KB)	Baseline Validation (ms)	Optimized Validation (ms)	Improvement (%)
Relay 1	512	48	31	35%
Relay 2	1024	95	61	36%
Relay 3	2048	189	117	38%

The results confirm that the proposed optimization strategies significantly reduce computational overhead and verification latency without altering consensus logic or compromising transaction security guarantees. SPV clients particularly benefit from reduced proof sizes and lower communication round-trip times. High-throughput relay nodes gain from reduced repeated internal hashing during block forwarding, improving block propagation time across the peer-to-peer network. These efficiency improvements reinforce scalability and support continued decentralization by lowering hardware and bandwidth requirements for node participation.

V. Conclusion and Future Work

The study demonstrated that optimizing Merkle-tree handling in Bitcoin block validation can significantly improve both computation efficiency and block propagation performance without altering the underlying consensus rules or cryptographic guarantees. By incorporating selective sub-tree caching, incremental hash computation, and structurally aware proof reduction, full nodes benefit from reduced redundant hashing operations while SPV clients gain from smaller and faster Merkle-proof verification. The results showed consistent latency reductions across increasing block sizes, affirming that efficiency gains scale with network load. These improvements collectively support stronger decentralization, as reduced hardware and bandwidth requirements lower the barrier to operating secure and fully validating nodes.

Future work may extend these optimizations to dynamic fee-market conditions, compact block relay mechanisms, and emerging overlay network protocols designed to accelerate block dissemination. Additionally, integrating these Merkle-tree optimization techniques with next-generation light client frameworks, such as Neutrino or FlyClient-style proofs, may further enhance lightweight verification security without increasing storage or communication overhead.

Another promising direction is evaluating parallel and hardware-accelerated hashing strategies, particularly where GPU-based miners and high-throughput relay nodes can overlap Merkle-tree computation with transaction selection and block assembly. These directions highlight a continued pathway for improving efficiency while preserving the trustless validation guarantees at the core of Bitcoin's architecture.

References

[1] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." *Available at SSRN 3440802* (2008).

[2] Bossuet, Lilian, et al. "Architectures of flexible symmetric key crypto engines—a survey: From hardware coprocessor to multi-crypto-processor system on chip." *ACM Computing Surveys (CSUR)* 45.4 (2013): 1-32.

[3] Kroll, Joshua A., Ian C. Davey, and Edward W. Felten. "The economics of Bitcoin mining, or Bitcoin in the presence of adversaries." *Proceedings of WEIS*. Vol. 2013. No. 11. 2013.

[4] Jakobsson, Markus, and Ari Juels. "Proofs of work and bread pudding protocols." *Secure Information Networks: Communications and Multimedia Security IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS'99) September 20–21, 1999, Leuven, Belgium*. Boston, MA: Springer US, 1999.

[5] Franco, Pedro. *Understanding Bitcoin: Cryptography, engineering and economics*. John Wiley & Sons, 2014.

[6] King, Sunny, and Scott Nadal. "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake." *self-published paper, August* 19.1 (2012).

[7] Courtois, Nicolas T., and Lear Bahack. "On subversive miner strategies and block withholding attack in bitcoin digital currency." *arXiv preprint arXiv:1402.1718* (2014).

[8] Decker, Christian, and Roger Wattenhofer. "Information propagation in the bitcoin network." *IEEE P2P 2013 Proceedings*. IEEE, 2013.

[9] Ron, Dorit, and Adi Shamir. "Quantitative analysis of the full bitcoin transaction graph." *International conference on financial cryptography and data security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.