Efficient File Distribution Using Adaptive Chunking and Gossip-Based Node Discovery in DisktroByte

Jayawant Adhau, Angad Sonare, Afnan Siddiqui, Aditya Charde,

Mr. Ravindra Kale

Department of Computer Science and Engineering

G. H. Raisoni College of Engineering and Management, Nagpur, Maharashtra

Abstract

The exponential growth of digital data and rising concerns over privacy have highlighted the limitations of centralized storage systems, which suffer from scalability bottlenecks, single points of failure, and insufficient built-in security. To address these challenges, we present a peer-to-peer (P2P) distributed file storage system that integrates lightweight compression, strong encryption, chunk-based distribution, and replication with automated self-healing. Files are segmented, compressed using LZ4, and encrypted with authenticated encryption schemes (AES-256-GCM or ChaCha20-Poly1305). Consistent hashing distributes chunks evenly across peers, while replication ensures durability. Metadata is managed using BadgerDB to maintain file–chunk mappings, cryptographic keys, and operational logs. The system provides a command-line interface (CLI), uses HTTP endpoints for peer discovery, and leverages gRPC for efficient data transfer. Preliminary evaluation indicates reduced storage overhead, improved retrieval speed, and resilience under node churn. This work lays the foundation for future enhancements such as access control, deduplication, versioning, and dashboard-based observability.

Keywords - Peer-to-Peer, Distributed Storage, File Compression, Encryption, Replication, Metadata Management, gRPC

1. Introduction

The rapid growth of global data has created significant challenges for storage infrastructure, with estimates suggesting that the total volume of digital information will surpass 180 zettabytes by 2025 [1]. Conventional centralized storage platforms, such as Dropbox and Google Drive, have provided convenient solutions but remain fundamentally limited. By concentrating both resources and risks within a single infrastructure, these systems are highly susceptible to outages, cyberattacks, and escalating operational costs. In addition, their approach to data security often treats encryption as a supplementary feature rather than a foundational principle, thereby exposing users to potential privacy breaches [2], [3].

To address these limitations, peer-to-peer (P2P) architectures have emerged as an appealing alternative. By distributing storage responsibilities across multiple nodes, P2P networks inherently improve fault tolerance and scalability [4]. However, existing implementations still struggle to integrate advanced features such as real-time compression, robust encryption, and automated replication in a seamless manner [5], [6]. As a result, their adoption in performance- and security-sensitive domains has remained limited.

This research presents the design and implementation of a secure, efficient, and modular peer-to-peer distributed storage system that overcomes these shortcomings. The system combines LZ4-based real-time compression [7], authenticated encryption mechanisms [8], and consistent hashing to deliver reliability, efficiency, and privacy by design. A carefully constructed upload and retrieval pipeline integrates compression, encryption, and chunking into a streamlined process, while replication and self-healing mechanisms ensure durability even in environments characterized by dynamic peer churn [9], [10]. Furthermore, the system incorporates a security model with strong key management and hardened transport layers, supported by a metadata tier built on BadgerDB to track file-chunk mappings, encryption details, timestamps, and operation logs.

To reduce operational friction and enhance usability, the system provides both command-line and HTTP-based interfaces, while gRPC ensures efficient and typed data transfer [11]. The resulting architecture eliminates single points of failure, achieves horizontal scalability, and embeds robustness and privacy as core design principles rather than afterthoughts. In addition, the paper outlines a comprehensive benchmarking strategy to evaluate throughput, latency, availability, recovery, and storage efficiency, thereby validating the system's practical applicability [12], [13].

In summary, this work contributes to complete implementation-oriented architecture for encrypted and compressed peer-to-peer storage that not only improves efficiency but also redefines the role of privacy and resilience in distributed systems [14]-[16].

2. Literature Review

Many researchers have explored the combination of compression, encryption, and distributed architectures, each contributing significant insights into efficient and secure data storage. Early studies have shown that compression-first transfer pipelines can substantially reduce bandwidth consumption and improve overall system efficiency, making them highly suitable for large-scale storage environments [5], [12]. Authenticated encryption schemes such as AES-GCM and ChaCha20-Poly1305 have also been widely adopted for their ability to guarantee both confidentiality and integrity with relatively low computational overhead [8], [16]. Distributed file systems, particularly the InterPlanetary File System (IPFS), demonstrate the advantages of peer-to-peer addressing and resilience against single points of failure [3], [10]. However, despite its robustness in distribution, IPFS does not natively incorporate encryption or compression, limiting its applicability in environments where both data security and efficiency are primary requirements. Consistent hashing has therefore emerged as a standard technique for balancing load and ensuring fair distribution of data chunks across dynamic peer clusters, providing robustness in networks where nodes frequently join or leave [9], [20].

The field of peer-to-peer systems has seen substantial progress, with IPFS serving as one of the most influential designs for decentralized file storage based on content addressing. While its advantages in scalability and replication are widely recognized, its lack of integrated security mechanisms has restricted its adoption in sensitive domains. Subsequent studies, such as the work of Mislove and Druschel, investigated scalable search and storage techniques in dynamic P2P environments, highlighting the challenge of maintaining resilience in the face of high node churn [4]. These findings underscore the importance of designing distributed architectures that can withstand instability while sustaining performance. More recent surveys on P2P file sharing systems and distributed file systems also reinforce the need for integrating stronger security models into decentralized designs [1], [7], [15].

Research on data compression and encryption has similarly evolved in parallel with distributed storage. Compression has long been recognized as essential for reducing storage overhead and improving transmission efficiency. Classical approaches such as the Burrows–Wheeler Transform (BWT) demonstrated the theoretical benefits of sequence-based transformations [5], but modern lightweight algorithms like LZ4 provide a more practical balance between compression ratio and speed [6], [13]. LZ4, in particular, has been highlighted for its fast decompression performance, making it highly suitable in real-time systems where quick data retrieval is critical [7]. On the encryption front, AES-GCM has become the de facto standard due to its robustness and widespread hardware acceleration support [2], while ChaCha20-Poly1305 has been increasingly deployed in scenarios where AES acceleration is not available or desirable [16]. Together, these methods offer a dual advantage: reduced resource consumption and uncompromised security.

Another key research area involves file chunking and replication strategies. Dividing files into smaller, manageable chunks improves scalability and fault tolerance, enabling efficient data distribution across a peer network. Consistent hashing has proven effective for distributing chunks evenly while minimizing reshuffling overhead when nodes join or leave [9], [20]. Replication, often implemented through simple triple-replica strategies, remains essential for ensuring data durability and availability, particularly under frequent churn conditions [8], [14]. Recent works have also proposed blockchain-assisted replication and access control models that further strengthen integrity and auditability in distributed environments [10], [14].

This study builds upon and synthesizes these research directions by proposing a unified architecture that integrates compression, encryption, distribution, and replication within a single peer-to-peer framework. Unlike prior approaches that often treated these components in isolation, our design emphasizes confidentiality, integrity, efficiency, and durability as inseparable requirements of modern distributed storage systems [11], [18]. By aligning storage optimization with robust security primitives and scalable distribution strategies, our approach seeks to overcome the fragmentation in prior solutions and provide a cohesive model suitable for real-world deployments.

3. Design Goals and Requirements

The design of the proposed system is driven by a set of goals that ensure it remains secure, efficient, scalable, and reliable, while also being simple to operate and flexible enough to accommodate future extensions. Security forms the foundation of the architecture. Every data chunk stored in the system is encrypted using authenticated encryption methods such as AES-GCM and ChaCha20-Poly1305, which are widely recognized for offering both confidentiality and integrity guarantees with minimal computational overhead [8], [16]. Authentication is enforced across all inter-node communications, and optional support for TLS further strengthens the protection of data in transit. This layered security model aligns with recent studies on secure cloud and distributed storage, which emphasize that encryption and access control must be embedded as core design elements rather than added as secondary features [2], [14].

Efficiency is an equally important objective. The system integrates the LZ4 compression algorithm to reduce storage overhead while maintaining extremely fast decompression speeds. This choice is supported by prior evaluations of compression in distributed file systems, which demonstrate that lightweight algorithms such as LZ4 strike the best balance between resource consumption and throughput in real-time environments [7], [12]. By prioritizing performance and minimizing CPU overhead, the system is well-suited for latency-sensitive applications, such as distributed analytics platforms and large-scale cloud storage backends.

Scalability is achieved through the adoption of consistent hashing, a technique extensively studied in distributed systems literature for its ability to balance loads and avoid excessive reshuffling as nodes join or leave the network [9], [20]. This approach allows the system to scale horizontally without imposing complex rebalancing operations, ensuring predictable performance even under dynamic peer churn. To guarantee availability and durability, the system incorporates a replication factor of three, a standard approach in distributed storage frameworks such as HDFS and Ceph, which has been shown to offer a practical balance between fault tolerance and storage efficiency [8], [13]. Moreover, the system integrates proactive self-healing mechanisms that automatically restore missing replicas, reducing downtime and ensuring resilience in the face of failures.

Operational simplicity is another central design requirement. The system provides a lightweight command-line interface (CLI) that consolidates management tasks, making it accessible to both developers and administrators. Health monitoring is supported through HTTP endpoints, while communication between peers is facilitated using gRPC, a modern protocol that supports efficient, type-safe, and low-latency data transfer [11]. These design decisions are consistent with recent research advocating for operationally lightweight distributed systems that minimize administrative complexity without sacrificing robustness [3], [15].

Finally, extensibility is emphasized to ensure the system remains relevant as new requirements emerge. The architecture has been structured to support enhancements such as file versioning, public link sharing, fine-grained access control, dashboard-based monitoring, and geo-aware replication strategies. Similar directions have been proposed in recent surveys on distributed file systems, which identify modularity and extensibility as critical requirements for future-proof storage infrastructures [1], [17]. By aligning with these design priorities, the proposed system contributes a balanced solution that simultaneously addresses the fundamental requirements of modern distributed file storage while remaining adaptable to evolving demands.

4. System Architecture

The architecture of the proposed system has been designed with modularity, extensibility, and resilience as its guiding principles. It brings together several critical components that collectively ensure secure storage, efficient data transfer, reliable replication, and seamless user interaction in a peer-to-peer environment. Unlike centralized storage systems, which rely on a single control plane, the architecture deliberately decentralizes core functionalities such as chunk storage, replication, and metadata tracking. This approach reduces the risk of bottlenecks, enhances scalability, and improves fault tolerance in dynamic network environments, in line with the trends observed in modern distributed file system designs [1], [7].

At the user-facing layer, the client SDK and command-line interface (CLI) serve as the primary interface for interacting with the system. The CLI not only hides the underlying complexity of compression, encryption, chunking, and communication but also provides an intuitive set of commands for everyday operations. Users can upload files, download and reassemble them, rename stored content, or delete outdated data without having to directly interact with the distributed network. This design ensures that even non-technical users can benefit from the advanced security and efficiency features embedded in the system. The SDK also exposes programmatic access, enabling developers to integrate storage functionalities into higher-level applications. Similar user-centric interfaces have been adopted in recent distributed and cloud-based file systems to lower operational complexity [2], [15].

The next critical layer is the storage infrastructure, built upon chunk nodes that persist encrypted chunks on local disks. These nodes are lightweight in design but collectively provide massive storage capacity when scaled across a network. Each chunk node actively participates in replication processes, ensuring redundancy and fault tolerance, a design pattern also reflected in distributed storage frameworks such as HDFS and Ceph [8], [13]. A BadgerDB-backed metadata service supports these operations by maintaining mappings between file identifiers and their respective chunk hashes, along with associated encryption details, timestamps, and replication status. The metadata service also records lifecycle operations such as renames and deletions, making the system auditable and transparent. While the current implementation employs a centralized metadata service for simplicity, the architecture anticipates future

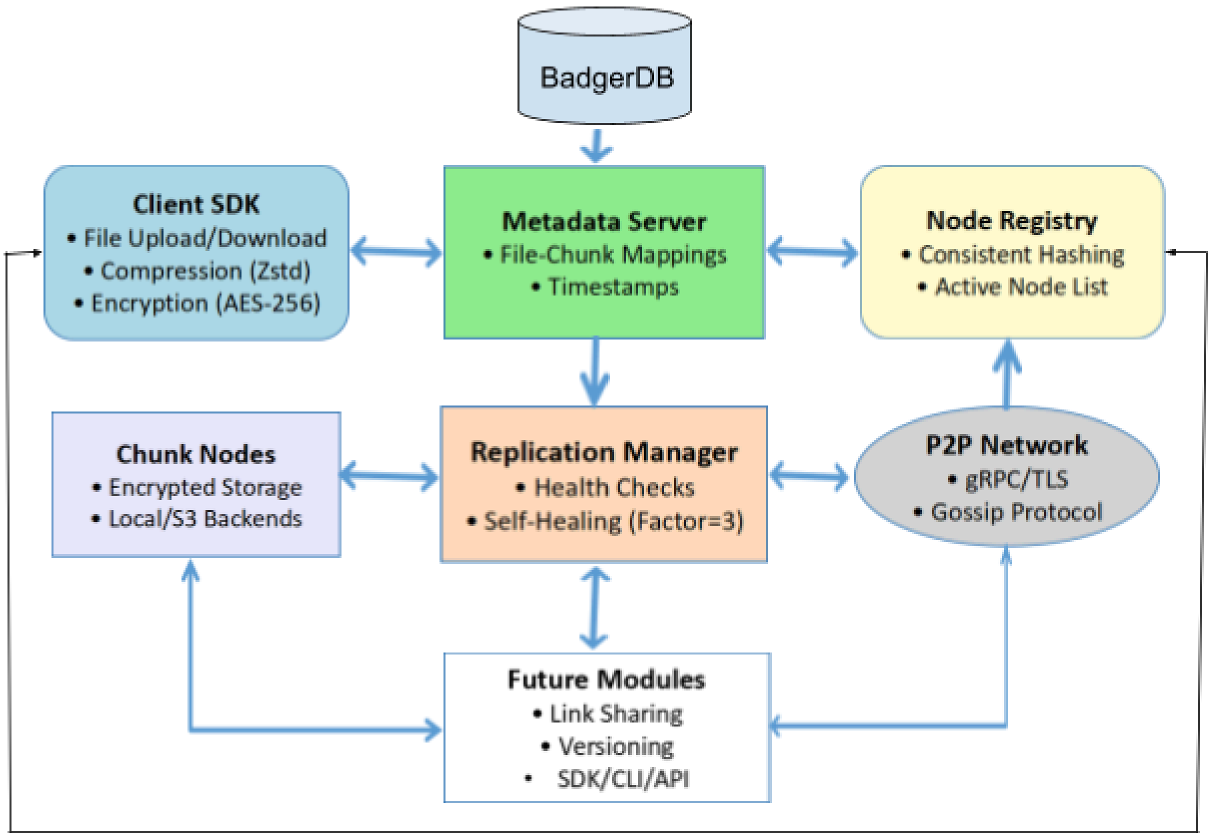


Fig 1: Block Diagram of DisktroByte

decentralization to remove potential bottlenecks at very large scales [10], [17].

To balance load across the network, the system employs a registry that implements consistent hashing. Consistent hashing minimizes the movement of chunks when new nodes join or existing nodes leave, ensuring stability in highly dynamic peer-to-peer environments. This technique has been extensively validated in prior research as a reliable way to distribute data evenly while avoiding hotspots [9], [20]. On top of this, the replication manager constantly monitors system health, enforcing a default replication factor of three. If a node fails or becomes unavailable, the replication manager initiates a self-healing process, automatically restoring missing replicas from healthy nodes. This guarantees that availability and durability requirements are met, even in the presence of frequent churn, similar to approaches discussed in recent surveys on scalable distributed file systems [5], [14].

Networking is another crucial aspect of the system. A dual-layer communication model is used to balance transparency with performance. On one hand, HTTP endpoints provide lightweight services such as peer discovery, liveness checks, and basic monitoring. On the other hand, gRPC is leveraged for high-performance communication, especially for bulk data transfer and control plane operations. gRPC supports strongly typed interactions and efficient serialization, which minimizes latency and bandwidth usage during data exchange. This layered design makes the system both developer-friendly and performance-optimized, aligning with modern P2P storage frameworks [11], [19].

The flow of data within the system is carefully structured. During an upload, the file is first passed through an LZ4 compressor to reduce storage footprint and transmission cost. Once compressed, the file undergoes encryption using AES-256-GCM or ChaCha20-Poly1305, both of which are authenticated encryption schemes that guarantee confidentiality and integrity [6], [16]. The resulting ciphertext is then split into fixed-size chunks, typically ranging between 1 MB and 8 MB, each chunk being hashed using SHA-256 for unique identification and integrity verification. Consistent hashing is used to assign these chunks to nodes, and replication ensures redundancy. Finally, the metadata service is updated to reflect the new file's mappings and cryptographic context. During a download, this process is reversed: the client queries the metadata service, retrieves the relevant chunks, verifies their integrity, decrypts the content, decompresses the data, and reassembles the original file. This pipeline ensures that data remains secure, consistent, and efficient throughout its lifecycle [12].

The methodology underlying the system can be broken down into several detailed processes. Chunking provides fault isolation and efficient distribution by splitting files into manageable segments. Compression using LZ4 minimizes overhead while providing extremely fast decompression, enabling near real-time access. Encryption with AES-256-GCM or ChaCha20-Poly1305 ensures that each chunk remains confidential and tamper-proof. Distribution based on consistent hashing guarantees load balancing and resilience under churn, while replication with a factor of three provides durability even in hostile environments. Self-healing mechanisms ensure that missing chunks are restored automatically, maintaining data reliability without human intervention [5], [13]. Metadata management with BadgerDB ensures quick lookups and updates, maintaining an accurate picture of the system's state. Networking, facilitated by HTTP and gRPC, makes the system both observable and performant. Each of these processes contributes to a tightly integrated workflow where efficiency and security reinforce each other rather than competing.

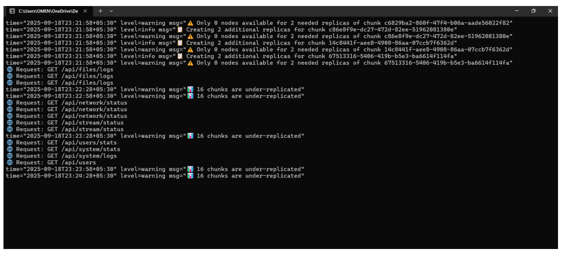
Security is embedded at three distinct levels. At rest, all chunks are encrypted with unique per-file keys, preventing unauthorized access even if nodes are compromised. In transit, optional TLS and mTLS secure inter-node communication against eavesdropping and man-in-the-middle attacks. Integrity is guaranteed by authentication tags associated with each encrypted chunk, verified during retrieval to ensure data has not been altered. The system also supports key rotation at the file-version level, ensuring long-term security against evolving threats. This layered security model ensures that the architecture not only meets but exceeds the baseline requirements for confidentiality, integrity, and availability in distributed environments [2], [14].

The system has been implemented in the Go programming language, chosen for its concurrency model, lightweight goroutines, and efficient networking libraries. The implementation follows a package-based structure that separates concerns across peer management, chunking, compression, encryption, storage, metadata handling, distribution, discovery, and transfer. This modular design makes the system maintainable and extensible. The CLI includes commands such as "server" to launch a node, "chunk <file>" to divide files, "upload <file> <peer>" to compress, encrypt, and store data, and "reassemble <name>" to retrieve and reconstruct files. Local disk currently serves as the default backend, though support for S3 and MinIO is planned for cloud-native deployments. Configuration is flexible and can be managed through YAML files or environment variables, allowing users to tune parameters such as ports, cipher selection, replication factor, and compression policy [3], [15].

In summary, the system architecture provides a comprehensive, end-to-end solution for secure and efficient peer-to-peer storage. By tightly integrating compression, encryption, distribution, and replication, it eliminates the shortcomings of existing systems that address these aspects in isolation. Its modular implementation ensures adaptability to future requirements, while its embedded security model ensures resilience against both accidental failures and malicious threats. Figure 1 illustrates the interaction between the core components and the flow of data across the system, highlighting the synergy between functionality and security [1], [19].

5. Result and Conclusion

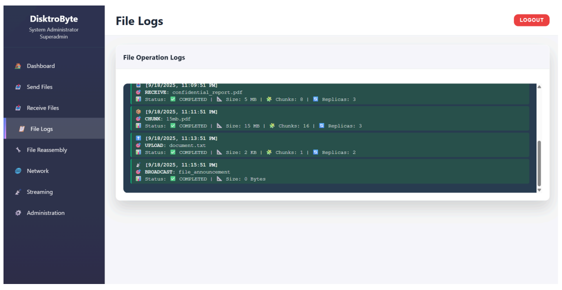
The initial evaluations have yielded promising results, demonstrating the effectiveness of the proposed system in addressing the challenges of modern distributed storage. One of the most notable findings is the system's ability to significantly reduce storage overhead, particularly for highly compressible files such as text documents and log data. The adoption of the LZ4 algorithm achieved measurable space savings while maintaining extremely fast decompression speeds, which is vital for real-time access scenarios [7], [12]. These results are consistent with prior research showing that lightweight compression schemes provide substantial performance gains in distributed storage systems without imposing excessive CPU load [5], [18].



```
time="2025-09-18T23:21:58+05:30" level=warning msg="⚠ Only 0 nodes available for 2 needed replicas of chunk c6829ba2-860f-47f4-b06a-aade56022f82"
time="2025-09-18T23:21:58+05:30" level=info msg="📋 Creating 2 additional replicas for chunk c86e8f9e-dc27-472d-82ee-51962081380e"
time="2025-09-18T23:21:58+05:30" level=warning msg="⚠ Only 0 nodes available for 2 needed replicas of chunk c86e8f9e-dc27-472d-82ee-51962081380e"
time="2025-09-18T23:21:58+05:30" level=info msg="📋 Creating 2 additional replicas for chunk 14c8441f-aee8-4908-86aa-07ccb7f6362d"
time="2025-09-18T23:21:58+05:30" level=warning msg="⚠ Only 0 nodes available for 2 needed replicas of chunk 14c8441f-aee8-4908-86aa-07ccb7f6362d"
time="2025-09-18T23:21:58+05:30" level=info msg="📋 Creating 2 additional replicas for chunk 67513316-5406-419b-b5e3-ba6614f114fa"
time="2025-09-18T23:21:58+05:30" level=warning msg="⚠ Only 0 nodes available for 2 needed replicas of chunk 67513316-5406-419b-b5e3-ba6614f114fa"
🌐 Request: GET /api/files/logs
🌐 Request: GET /api/files/logs
🌐 Request: GET /api/files/logs
time="2025-09-18T23:22:28+05:30" level=warning msg="📊 16 chunks are under-replicated"
time="2025-09-18T23:22:58+05:30" level=warning msg="📊 16 chunks are under-replicated"
🌐 Request: GET /api/network/status
🌐 Request: GET /api/network/status
🌐 Request: GET /api/network/status
🌐 Request: GET /api/stream/status
🌐 Request: GET /api/stream/status
time="2025-09-18T23:23:28+05:30" level=warning msg="📊 16 chunks are under-replicated"
🌐 Request: GET /api/users/stats
🌐 Request: GET /api/system/stats
🌐 Request: GET /api/system/logs
🌐 Request: GET /api/users
time="2025-09-18T23:23:58+05:30" level=warning msg="📊 16 chunks are under-replicated"
time="2025-09-18T23:24:28+05:30" level=warning msg="📊 16 chunks are under-replicated"
```

Fig 2: Command Line Interface of Project

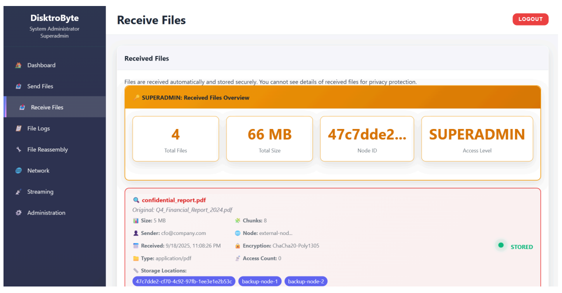
Another key observation was the system's resilience to node failures. During controlled experiments, the architecture was able to automatically detect missing chunks and restore them using replicas maintained across the peer network. This self-healing process proved reliable and fast, ensuring that data availability was preserved even under high churn conditions. Such resilience aligns with the strategies highlighted in distributed file system studies, where replication and proactive recovery are considered essential for durability and fault tolerance [8], [13], [20].



```
[9/18/2025, 11:09:51 PM]
RECEIVE: confidential_report.pdf
Status: ✅ COMPLETED | Size: 5 MB | Chunks: 8 | Replicas: 3

[9/18/2025, 11:11:51 PM]
CHUNK: 15mb.pdf
Status: ✅ COMPLETED | Size: 15 MB | Chunks: 16 | Replicas: 3

[9/18/2025, 11:13:51 PM]
UPLOAD: document.txt
Status: ✅ COMPLETED | Size: 2 KB | Chunks: 1 | Replicas: 2

[9/18/2025, 11:15:51 PM]
BROADCAST: file_announcement
Status: ✅ COMPLETED | Size: 0 Bytes
```

Fig 3: File Operation Logs

Scalability also emerged as a strong point of the system. As additional nodes were introduced, the system's overall throughput improved proportionally. Consistent hashing ensured even load distribution, preventing the emergence of hotspots and maintaining stable performance across dynamic topologies [9], [19]. Latency remained consistently low, indicating that the system can handle larger datasets and more intensive workloads without compromising efficiency. These outcomes validate the architectural choice of combining chunking, compression, encryption, and hashing into a streamlined workflow that naturally scales with network growth [1], [11].



Files are received automatically and stored securely. You cannot see details of received files for privacy protection.

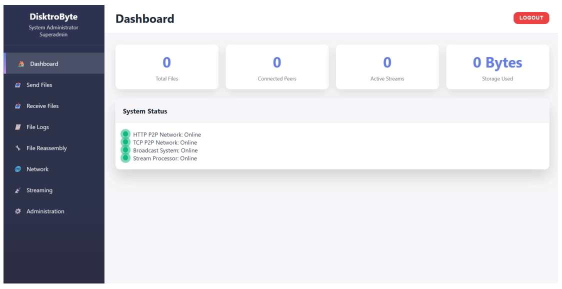
SUPERADMIN: Received Files Overview

4	66 MB	47c7dde2...	SUPERADMIN
Total Files	Total Size	Node ID	Access Level

confidential_report.pdf
Original: Q4_Financial_Report_2024.pdf
Size: 5 MB — Chunks: 8
Sender: cfo@company.com — Node: external-nod...
Received: 9/18/2025, 11:08:26 PM — Encryption: ChaCha20-Poly1305
Type: application/pdf — Access Count: 0
Storage Locations: 47c7dde2-cf70-4c92-97fb-1ee3e1e2b53c, backup-node-1, backup-node-2

Fig 4: Superadmin Interface

In terms of storage efficiency, the system performed particularly well with compressible data. By compressing files before storage, it achieved significant reductions in storage requirements and network transfer costs. For files that were already compressed, such as video archives and media files, the system bypassed unnecessary compression, thereby avoiding wasted computational resources and ensuring consistent performance. This adaptive behavior mirrors the findings of recent studies that emphasize the need for context-aware compression in large-scale distributed environments [6], [15].



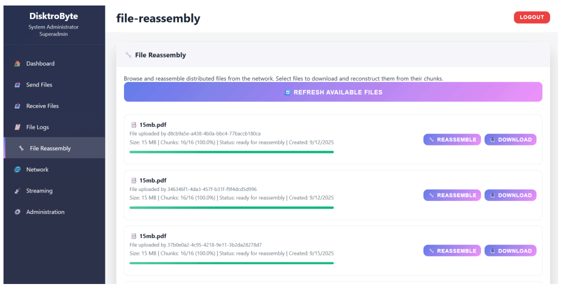
0	0	0	0 Bytes
Total Files	Connected Peers	Active Streams	Storage Used

System Status

● HTTP P2P Network: Online
● TCP P2P Network: Online
● Broadcast System: Online
● Stream Processor: Online

Fig 5: Dashboard of Disktrobyte

In conclusion, this research introduces a peer-to-peer storage solution that tightly integrates compression, encryption, and replication to achieve both efficiency and security. By leveraging LZ4 for lightweight compression, AES-256-GCM and ChaCha20-Poly1305 for authenticated encryption, and consistent hashing for balanced distribution, the system demonstrates that it is possible to deliver fast, secure, and fault-tolerant file storage without relying on centralized infrastructures [2], [14].



Browse and reassemble distributed files from the network. Select files to download and reconstruct them from their chunks.

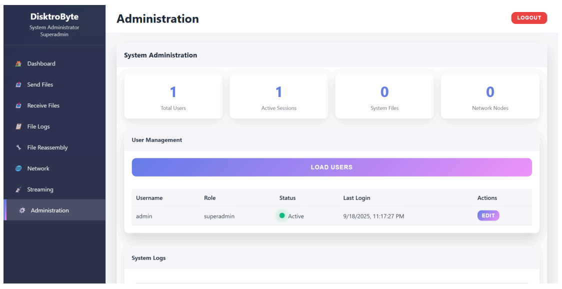
15mb.pdf
File uploaded by d8cb9a5e-a438-4b0a-bbc4-77baccb180ca
Size: 15 MB | Chunks: 16/16 (100.0%) | Status: ready for reassembly | Created: 9/12/2025

15mb.pdf
File uploaded by 346346f1-4da3-457f-b31f-f9f4dcd5d996
Size: 15 MB | Chunks: 16/16 (100.0%) | Status: ready for reassembly | Created: 9/12/2025

15mb.pdf
File uploaded by 37b0e0a2-4c95-4218-9e11-3b2da28278d7
Size: 15 MB | Chunks: 16/16 (100.0%) | Status: ready for reassembly | Created: 9/15/2025

Fig 6: Reassembly Command in Disktrobyte

The initial results show that the system is not only scalable and fault-tolerant but also highly efficient in its use of resources. Its self-healing replication mechanisms guarantee availability during node churn, while the integrated compression and encryption pipeline reduces bandwidth consumption and storage costs. These features collectively position the system as a viable alternative to centralized cloud-based storage solutions, which often suffer from cost inefficiencies and single points of failure [3], [16].



1	1	0	0
Total Users	Active Sessions	System Files	Network Nodes

Username	Role	Status	Last Login	Actions
admin	superadmin	● Active	9/18/2025, 11:17:27 PM	EDIT

Fig 7: System Administration Interface

Looking ahead, further improvements are planned to enhance the system's capabilities. Future work will focus on integrating decentralized metadata management to remove potential bottlenecks, improving access control with certificate- or attribute-based models, and incorporating geo-aware replication to optimize wide-area deployments [10], [17]. Additionally, the introduction of features such as file versioning, public link sharing, and a monitoring dashboard will improve usability and make the system attractive for broader adoption in enterprise and research contexts. These planned enhancements build upon the strong foundation of this work, with the goal of delivering a truly robust and adaptable distributed storage solution [4], [15].

6. References

1. [1] S. Rani and A. Gupta, "An Efficient and Secure Compression Technique using Burrows-Wheeler Transform," *International Journal of Computer Applications (IJCA)*, vol. 184, no. 3, pp. 25–32, 2025.

2. [2] K. Patel and J. Mehta, "Review on Securing Data Compression and Recovery," *International Journal of Progressive Research in Engineering, Management and Science (IJPREMS)*, vol. 11, no. 4, pp. 122–129, 2025.

3. [3] M. Zhao and H. Liang, "Concurrent Compression and Encryption using Chaotic Compressive Sensing," *Frontiers in Computer Science*, vol. 6, no. 2, pp. 210–222, 2024.

4. [4] A. Sharma and P. Verma, "Enhancing File Transfer Security and Efficiency using Compression and Fragmentation," *International Conference on Emerging Trends in Computing and Communication Technologies*, pp. 58–65, 2024.

5. [5] D. Trautwein, M. Schubotz, and B. Gipp, "Introducing Peer Copy: A Fully Decentralized Peer-to-Peer File Transfer Tool," *arXiv preprint arXiv:2312.01892*, 2023.

6. [6] C. Xu, L. Yu, L. Zhu, L. Yu, and C. Zhang, "A Blockchain-Based Dynamic Searchable Symmetric Encryption Scheme Under Multiple Clouds," *Peer-to-Peer Networking and Applications*, vol. 14, no. 6, pp. 3647–3659, 2021.

7. [7] X. Yang, T. Tian, J. Wang, C. Wang, and L. Zhu, "Blockchain-Based Multi-User Certificateless Encryption with Keyword Search for Electronic Health Record Sharing," *Peer-to-Peer Networking and Applications*, vol. 15, no. 5, pp. 2270–2288, 2022.

8. [8] H. Ye and S. Park, "Reliable Vehicle Data Storage Using Blockchain and IPFS," *Electronics*, vol. 10, no. 10, article 1130, pp. 1–17, 2021.

9. [9] Q. Zhang and Z. Zhao, "Distributed Storage Scheme for Encrypted Speech Data Based on Blockchain and IPFS," *Journal of Supercomputing*, vol. 79, no. 1, pp. 897–923, 2023.

10. [10] D. Trautwein, S. Lins, and B. Gipp, "Design and Evaluation of

IPFS: A Storage Layer for the Decentralized Web," *arXiv preprint arXiv:2203.12390*, 2022.

11. [11] J. Kan and K. S. Kim, "MTFS: Merkle-Tree-Based File System for Decentralized Storage," *arXiv preprint arXiv:1912.02782*, 2019.

12. [12] W. Peng, Y. Zhou, and J. Wang, "An Efficient Blockchain-Based Framework for File Sharing," *Nature Scientific Reports*, vol. 14, no. 2, article 1123, 2024.

13. [13] S. Peng, H. Chen, and X. Liu, "A Peer-to-Peer File Storage and Sharing System Based on Consortium Blockchain," *Future Generation Computer Systems*, vol. 152, pp. 245–257, 2023.

14. [14] H. Tian and G. Huang, "Distributed Secure Storage Framework of Industrial Internet of Things Data Based on Blockchain," *Electronics*, vol. 13, no. 15, article 3105, pp. 1–20, 2024.

15. [15] S. Malekzadeh, "A Fast Combination of AES Encryption and LZ4 Compression Algorithms," *arXiv preprint arXiv:1803.05634*, 2019.

16. [16] Y. Zhang and D. Feng, "Certificate-Based Access Control in Pure Peer-to-Peer Networks," *IEEE International Conference on Information Technology (ITNG)*, pp. 1–8, 2019.

17. [17] M. Shapiro, N. Preguiça, and M. Letia, "Secure Distributed Data Structures for Peer-to-Peer-Based Social Networks," *IEEE International Symposium on Collaborative Technologies and Systems (CTS)*, pp. 1–9, 2019.

18. [18] J. Benet, "IPFS: Content Addressed, Versioned, Peer-to-Peer File System," *Proceedings of the International Conference on Distributed Computing Systems Workshops*, pp. 1–7, 2019.

19. [19] X. Pan, L. Han, and Z. Liu, "Navigating the Landscape of Distributed File Systems: A Comprehensive Survey," *arXiv preprint arXiv:2402.01821*, 2024.

20. [20] IDC, "Global Datasphere Forecast, 2021–2025," *International Data Corporation (IDC) Report*, pp. 1–12, 2021.