# S3 COMPATIBLE DISTRIBUTED STORAGE SYSTEM

Sep 02, 2024

Version - 1.0.0

www.larissa.network | www.silos3.com

# 1 Introduction: Embracing Decentralized Storage

The internet is a vast network of interconnected devices that is largely decentralized in nature. However, the storage of its vast data is often in the hands of a few large tech entities. Myriad challenges, including significant data breaches, periods of outage, high storage costs, and the necessity to rapidly scale infrastructure to meet the growing demand for quicker and larger data access, confront these entities.

In response, decentralized storage solutions are a unique and innovative solution to these challenges. In contrast to centralised data centres, they provide improved security, privacy, cost efficiency, and performance, which are in close alignment with the decentralized architecture of the internet.

The urgency of such solutions is underscored by recent trends: the volume of data is increasing at unprecedented rates, and data intrusions have surged. Global data is anticipated to surpass 181 zettabytes by 2025 (Edge Delta). The decentralized storage market is expected to experience substantial growth, which is being driven by the demand for more efficient data management solutions and the rise in data generation (Edge Delta).

In comparison to conventional storage methods, decentralized systems not only enhance the security and privacy of data but also reduce costs. The design of these systems is centred on the optimization of a variety of parameters, including cost efficiency, trustworthiness, capacity, and speed, to ensure that they can adapt to the changing requirements of users and industries.

A decentralized storage system that is globally scalable and utilises encryption and sharding to improve data security and distribution is proposed in this paper, SILO Network. Our methodology entails the integration of a modular system architecture that incorporates a variety of components to provide secure, efficient, and cost-effective data storage solutions.

In the subsequent chapters, we will explore the SILO Network's design, framework implementation, operational dynamics, future developments, and the computational models that underpin our technology. This will establish the foundation for a comprehensive discussion on the transformative potential of decentralized storage.

# 2 Design Constraints of SILO

A precise comprehension of the specific requirements that orient its development is necessary when designing a decentralized storage system such as SILO. The design space is vast; however, the focus can be considerably narrowed by establishing a few critical constraints, which ensures that the system is in alignment with specific product and market fit objectives.

These constraints are essential for SILO in order to create a system that is as universally applicable as feasible within the constraints of our design parameters. This method guarantees that each component of SILO—including security and scalability—is specifically designed to satisfy these predetermined criteria.

## 2.1 Security and Privacy in SILO

Regardless of whether the object storage platform is centralised or decentralized, it is crucial to prioritise the security and privacy of stored data. Nevertheless, decentralized systems such as SILO encounter further complexities as a result of the inherently untrustworthy nature of the nodes involved. Decentralized storage systems, unlike centralised systems, cannot depend on conventional security mechanisms like firewalls and DMZs. Instead, they must be constructed with security and privacy as fundamental components. This involves the implementation of end-to-end encryption and the improvement of security at all levels of the system.

Moreover, adherence to diverse regulatory systems is essential. For example, the processing of data must adhere to the Health Insurance Portability and Accountability Act (HIPAA) in the United States and the General Data Protection Regulation (GDPR) in Europe. These regulations impose strict criteria for safeguarding data. Furthermore, international clients may want guarantees that their data storage options minimise vulnerability to U.S. jurisdiction as a result of geopolitical anxieties.

In order to cultivate trust, clients must have the ability to authenticate that the SILO software is resilient against possible security vulnerabilities, both anticipated and unforeseen, and satisfies all of their specifications. We provide transparency and guarantee users that the system functions as intended, upholding the highest levels of security and privacy.

## 2.2 Embracing Decentralisation

Decentralisation in application design means that no one entity is responsible for operating or bearing the whole expense of running the service, and no individual entity has the power to disrupt the service for others. This concept is especially attractive for minimising infrastructure expenses, such as maintenance, utilities, and bandwidth, by using unused resources at the periphery of the network.

During the development of the SILO decentralized storage network, we have discovered a significant amount of unused resources among several smaller operators. While these resources, whether they pertain to cost-effective power or cooling solutions, may not be enough on their own to sustain a large-scale data centre, they may make a substantial contribution to a decentralized network. For example, a small company or a household using Network Attached Storage (NAS) may have the capability to accommodate a certain number of drives. Together, these modest companies may provide a strong, economical, and widely spread storage solution.

Our dedication to decentralisation is driven by the goal of providing an alternative to the prevailing centralised storage providers, therefore reducing the dangers associated with relying on a single institution to handle a large amount of global data. These risks include any alterations in service or policy that might jeopardize data privacy or the sustainability of the service.

Decentralisation enables the establishment of a storage network that remains functional and accessible even in the event of SILO's cessation of activities, guaranteeing the continuity and accessibility of data. This strategy not only tackles the constraints and hazards that are naturally present in centralised systems, but it also caters to a wide range of storage requirements, from long-term storage solutions to content delivery networks (CDNs), without the need for different designs that centralised systems need.

## 2.3 Marketplace Dynamics and Economics of Decentralized Storage

The market for public cloud computing, namely cloud storage, has shown significant economic growth, increasing from $186.4 billion in 2018 to an estimated $302.5 billion by 2021. This paradigm provides customers with the option to easily adjust the size and capacity of their operations, while also reducing expenses by removing the substantial fixed expenditures often involved with maintaining physical data centres.

The market for public cloud computing, namely cloud storage, has shown significant economic growth, increasing from $186.4 billion in 2018 to an estimated $302.5 billion by 2021. This paradigm provides customers with the option to easily adjust the size and capacity of their operations, while also reducing expenses by removing the substantial fixed expenditures often involved with maintaining physical data centres.

Public cloud storage, although advantageous, inherently results in market concentration because of the significant initial expenditures and economies of scale that benefit just a few major suppliers. This concentration restricts competition and consolidates control over extensive data stores.

SILO, a kind of decentralized storage, offers an appealing option by dispersing data over a network of autonomous nodes. This approach helps to save expenses and enhance control over data ownership. In order for SILO to successfully compete, it must provide better economic advantages, not just in terms of price for storage and bandwidth, but also in terms of the complete value proposition, which includes security, performance, and dependability.

- End Users: Offering competitive features of public cloud storage such as scalability and no upfront costs, while providing better value and security.

- Storage Node Operators: Making it economically viable to contribute to the network, ensuring they receive fair compensation and potentially profit by utilising or creating new capacity.

- Demand Providers: Attracting developers and businesses to bring data and customers to the network, rewarding them with a share of the revenue, potentially tapping into open-source communities that significantly drive cloud workloads.

- Network Operator: SILO intends to sustain its development and operations by retaining reasonable profits through cost-effective service offerings and revenue sharing with node operators and demand providers.

Furthermore, the SILO network is specifically built to provide streamlined billing and payment procedures, support a wide range of transaction types, including both cryptocurrency and conventional payment methods, and guarantee adherence to international regulatory norms.

This economic model has the dual objective of distributing data storage and ensuring that the economic advantages of the cloud storage business are accessible to everyone, so offering a feasible, expandable, and economically advantageous alternative to centralised cloud providers.

## 2.4 Embracing S3 Compatibility

Amazon Web Services (AWS) continues to dominate the cloud storage industry, mostly because of its early entry into the market and wide-ranging ecosystem. Amazon S3, specifically, is considered the most commonly used cloud storage protocol. In order to successfully compete and encourage the use of decentralized storage, it is essential to provide interoperability with Amazon S3.

SILO's objective is to minimise the obstacles faced by consumers while shifting from centralised to decentralized storage systems. SILO offers a smooth transition for customers who are used to the services provided by Amazon S3 by ensuring compatibility with the Amazon S3 API. Ensuring compatibility is crucial for reducing the expenses associated with moving and seamlessly integrating into established operations without causing significant disruptions.
SILO's approach includes:

- Bucket Operations: Functions like creating, deleting, and listing storage buckets.
- Object Operations: Handling data through commands to get, put, delete, and list objects within these buckets.

These operations ensure that applications developed for Amazon S3 can operate on SILO with little to no modification required, making it an attractive alternative for users seeking enhanced security and privacy without sacrificing performance or durability.

To cater to diverse user needs, SILO supports two primary integration models:

- Single Tenant Nexus: This model allows users complete control over their data, with end-to-end encryption and direct peer-to-peer data transmission to storage nodes.
- Multi-Tenant Nexus: In this setup, a trusted provider manages encryption and data transmission, requiring users to provide encryption keys with each request to ensure security in a hosted environment.

By supporting a broad range of the S3 protocol and offering flexible integration options, SILO positions itself as a robust, secure, and user-friendly decentralized storage solution.

## 2.5 Durability, Device Failure, and Churn Management

In order for a storage system to be efficient, it must consistently store and retrieve data, while ensuring strong durability and limiting the possibility of data loss, even in the event of device failures and network problems. Component failure is an unavoidable occurrence in any hardware setting, as hard drives deteriorate, servers malfunction, and network connections get disrupted. SILO is specifically developed to address these circumstances by using resilient redundancy solutions that guarantee data is not permanently lost due to any one failure point.

Decentralized systems such as SILO are especially susceptible to churn, which refers to the frequent addition and removal of nodes from the network. Research has shown that in peer-to-peer networks, the length of time a node is active may vary greatly, typically lasting just a few hours or minutes. The frequent loss of nodes due to high turnover requires extra redundancy in order to compensate for this, resulting in an increased bandwidth need for the network to operate at its best.

In order to tackle these difficulties, SILO promotes the stability of storage nodes by providing incentives for long-term engagement. This minimises network turnover and eliminates the need for unnecessary duplication and data transmission capacity, which are crucial for sustaining system effectiveness and cost-efficiency. The network's architecture incorporates methods for data preservation, repair, and the seamless substitution of lost redundancy to guarantee the long-term security and accessibility of data.

To get a comprehensive understanding of how repair bandwidth fluctuates in response to node churn, please refer to section 7.3.3 and the accompanying research that investigates the relationship between network stability and operational efficiency.

## 2.6 Optimising for Low Latency

Decentralized storage systems such as SILO provide significant opportunities for using parallelism. This may result in improvements in data transmission speeds, computational capacities, and overall system efficiency, even in cases when individual network connections are sluggish. Nevertheless, parallelism by itself does not automatically decrease latency, despite the advantages it offers.

Latency continues to be a crucial constraint on performance, especially in applications that need high levels of performance. The latency of a single network connection, if it is part of a transaction, establishes the least duration that the operation may need. Hence, it is essential for decentralized systems to not only use parallelism but also prioritise the reduction of latency across the whole system design.

Consistent and assertive optimization endeavours are vital to attain minimal delay at both the process and system levels. This entails optimising the network's architecture to minimise data transmission delay and enhance responsiveness, so assuring that SILO can successfully cater to the requirements of latency-sensitive applications.

## 2.7 Bandwidth

The global availability of bandwidth has been consistently growing, while access to it remains unequal throughout the globe. While some users may benefit from fast and unrestricted data transfer rates, others have notable restrictions. The difference between the two creates distinct difficulties for decentralized networks such as SILO.

Residential internet service providers (ISPs), particularly in nations such as the United States, often provide asymmetric connections in several areas. These plans promote fast download speeds but provide much slower upload rates. In addition, Internet Service Providers (ISPs) often enforce bandwidth limitations, which restrict the maximum amount of data that may be sent during a given month. For example, an Internet Service Provider (ISP) could promote download speeds of 10 megabytes per second (MB/s) but impose a restriction on customers, allowing them to transmit a maximum of 1 terabyte (TB) of data every month. This limitation effectively caps the average data transfer rate at around 385 kilobytes per second (KB/s) to prevent users from incurring additional charges for exceeding their data allowance.

These limitations not only influence the way data is stored and retrieved, but also effect the amount of bandwidth needed for maintaining and repairing the data. These factors are crucial in a decentralized system where device failures and changes are anticipated. A system that fails to effectively regulate bandwidth may inadvertently give preference to operators having access to fast, unrestricted connections, therefore diminishing the decentralisation of the network.

In order to guarantee optimal performance of SILO in various situations, it is essential to actively reduce bandwidth use. This method ensures the preservation of a really decentralized storage network, offering equitable chances for user involvement irrespective of their internet connectivity capabilities.
For a deeper exploration of how bandwidth availability and repair traffic affect usable space, refer to section 7.1.1.

## 2.8 Optimising for Object Size

Storage systems may often be classified into two types depending on the average size of the data items they handle. Within this framework, files that are a few megabytes or greater are categorised as "large". Databases are generally more suitable for keeping several little bits of information, while object storage systems or file systems are more ideal for handling bigger files.

SILO is mainly designed to operate as a decentralized object storage platform that is specifically geared for handling bigger files. The foundation of our protocol is based on the assumption that the bulk of objects would have a size of 4MB or more. Storing smaller files may not be cost-efficient since storage and retrieval operations are geared for bigger data volumes.

This design approach is inclusive of use scenarios that entail accessing a large number of tiny files. Users have the option to use a packing approach, which involves combining several little files into a single, larger entity. SILO enables users to access smaller files without the need to download the full aggregated object. This is achieved via support for searching and streaming, which provides for flexibility while minimising storage volumes.

## 2.9 Handling Byzantine Faults

SILO functions in a decentralized and untrusted environment, unlike centralised storage systems like Amazon S3. In this context, storage nodes are independently operated and not intrinsically trustworthy. The decentralized structure implemented via the public internet allows for the inclusion of any individual as a storage provider, resulting in a range of differences in the dependability and functioning of nodes.

To navigate this, SILO adopts the Byzantine, Altruistic, Rational (BAR) model to categorise participants:

- Byzantine Nodes: These are nodes that may behave arbitrarily or maliciously, deviating from the protocol due to failures, malicious intent, or other reasons. They act independently of the protocol’s utility function and can attempt to disrupt the system.
- Altruistic Nodes: These nodes adhere strictly to the protocol, participating as intended even when it might not be in their direct interest to do so. They are considered "good actors" in the network.
- Rational Nodes: These nodes make decisions based on their self-interest, participating in or deviating from the protocol based on what benefits them the most. They are "neutral actors," neither strictly following nor subverting the protocol unless it aligns with their interests.

In centralised systems, such as typical cloud storage, nodes are presumed to be altruistic due to being managed and maintained by a single party. On the other hand, SILO functions inside a decentralized setting, where each node is autonomously controlled. This configuration requires the network to operate on the premise that most nodes are rational, but a smaller portion may be Byzantine, without making any assumptions about charity.

In order to secure the reliable functioning of the system, SILO includes incentives that motivate rational nodes to behave in a manner that is similar to altruistic nodes, therefore aligning their behaviours with the anticipated protocol. At the same time, it incorporates techniques to reduce or remove the influence of Byzantine nodes, guaranteeing strong performance even in the presence of possible aberrations.

Crucially, while SILO has to deal with Byzantine behaviour, it does not depend on a Byzantine fault-tolerant consensus mechanism. Instead, it chooses other ways to successfully handle fault tolerance.

## 2.10 Minimising Coordination

Eliminating the need for coordination in distributed systems may greatly improve performance and scalability. Coordination refers to the need for processes that are being executed simultaneously to communicate or halt in synchronisation in order to accomplish tasks. This adds delays that may significantly affect the overall throughput of the system.

Studies have shown that systems that limit the need for coordination may achieve much higher throughput compared to those that depend on coordinated operations to ensure accuracy. For instance, techniques that circumvent coordination, such Highly Available Transactions, have shown performance improvements of two to three orders of magnitude compared to standard coordinated systems in wide-area networks. The Anna database demonstrates this notion by attaining speeds up to 10 times faster than systems like Cassandra and Redis via minimising coordination wherever feasible.

Certain activities need coordination to guarantee consistency and accuracy. However, frameworks such as Invariant Confluence and the CALM principle aid architects in determining when cooperation is required. Nevertheless, in order to attain scalability to exabyte magnitudes, it is imperative to minimise coordination to the greatest extent feasible. Systems that optimise coordination get greater scalability, since the addition of resources immediately enhances throughput and performance. On the other hand, systems that rely on coordination, such as those that use global ledgers or consensus protocols, do not see substantial advantages from more resources.

In order to efficiently scale SILO, our objective is to reduce the need for coordination, restricting it to tiny areas that are within the control of the user. This method decreases dependence on global ledgers or blockchain-like solutions, enabling the attainment of greater scalability and performance in a decentralized storage setting.

# 3 Core Framework

This chapter introduces the fundamental foundation of SILO, outlining the necessary components required to satisfy the defined design requirements. The primary purpose of this fundamental framework is to provide adaptability and durability, allowing SILO to adjust to changing requirements without requiring significant restructuring.

The framework establishes the necessary elements to guarantee adherence to our design principles, enabling the modification or substitution of individual components without affecting the whole system. The modular nature of the SILO framework guarantees that it can effortlessly incorporate advancements and adjust to new difficulties in the quickly evolving digital ecosystem, ensuring its continued relevance and efficiency over the next decade.

## 3.1 Framework Overview

The SILO framework is specifically developed to carry out many fundamental duties that are crucial for decentralized storage. Its primary objective is to guarantee the security, dependability, and availability of data across the network. Every element within this structure helps to attaining these objectives by concentrating on distinct tasks:

1. Store Data: Data storage begins with the client encrypting and dividing the data into multiple fragments. These fragments are then distributed across various peers in the network. Metadata is generated during this process to track where each piece of data is stored.
2. Retrieve Data: To retrieve stored data, the client uses the metadata to locate the data fragments across the network. These fragments are then collected and reassembled into the original data on the client's device.
3. Maintain Data: To ensure data reliability, the system monitors redundancy levels. When redundancy falls below a certain threshold, the system regenerates and replaces the missing data fragments to maintain data integrity.
4. Pay for Usage: Users compensate the network for the storage services rendered, typically through a unit of value or cryptocurrency.

To enhance clarity and manageability, the design is divided into eight independent components, which are integrated to form the complete SILO framework:

1. Storage Nodes: These are the individual devices or servers that store data fragments.
2. Peer-to-Peer Communication: This component manages the communication between nodes, facilitating the transfer and retrieval of data.
3. Redundancy: Ensures multiple copies of data fragments are stored across the network to prevent data loss.
4. Metadata: Keeps track of where data fragments are stored, enabling efficient data retrieval.
5. Encryption: Protects data privacy by encrypting data before it is stored on the network.
6. Audits and Reputation: Monitors the reliability of storage nodes and maintains a reputation system to encourage good behaviour.
7. Data Repair: Automatically regenerates and replaces any lost data fragments to maintain the required level of redundancy.
8. Payments: Manages the financial transactions between users and storage providers, ensuring fair compensation for services.

This modular approach allows for flexibility and adaptability, enabling the SILO network to evolve and improve over time while maintaining its core functionalities.

## 3.2 Storage Nodes

The primary function of storage nodes in the SILO network is to store and retrieve data, while ensuring sufficient network capacity and responsiveness. The selection of these nodes is dependent on several parameters, such as ping time, latency, throughput, bandwidth constraints, available disk space, geographic location, uptime, and dependability in appropriately responding to audits. Storage nodes are rewarded for their involvement by receiving remuneration for both the storage of data and the management of data transfers.

The process of selecting storage nodes in SILO is dynamic and non-deterministic, since it is impacted by external factors that are subject to change. This method requires the monitoring of metadata to keep track of the nodes that are chosen for each data upload, like the architecture seen in systems such as GFS, HDFS, or Lustre. Contrary to systems like Dynamo that use a deterministic node selection procedure, SILO's approach prioritises flexibility and adaptability by considering current circumstances and needs to determine the appropriate node selection.

The importance of a strong metadata storage system inside the SILO framework is emphasised by the fact that it is necessary for managing storage nodes effectively. This system ensures that data retrieval and maintenance are done accurately and efficiently throughout the network.

## 3.3 Network Communication Protocols

The SILO network relies on a defined protocol to provide dependable and secure data flow between peers. This protocol is crucial for preserving the integrity and efficiency of the decentralized network and must comply with many fundamental requirements:

- Peer Reachability: The protocol must provide communication between peers, even in the presence of firewalls or network address translators (NATs). To overcome network restrictions and enable communication between peers, several solutions such as Session Traversal Utilities for NAT (STUN), Universal Plug and Play (UPnP), or NAT Port Mapping Protocol (NAT-PMP) may be used.
- Authentication: In order to thwart man-in-the-middle attacks and guarantee secure communications, the protocol utilises cryptographic authentication techniques that are similar to those used in S/Kademlia. Every individual in the network must use cryptographic methods to confirm the identity of other individuals they interact with, hence increasing trust and security inside the network.
- Complete Privacy: The protocol must ensure that all conversations are confidential and impervious to interception, safeguarding against unauthorised surveillance. Confidentiality is of utmost importance in processes such as bandwidth measurement, where the exchange of data between clients and storage nodes must be kept private. By default, all forms of communication are encoded to guarantee confidentiality and deter illegal entry.

Furthermore, the framework necessitates a mechanism to get peer network addresses by using a distinct identity. This enables any peer to establish a connection with another peer using this identification, similar to the functioning of the Domain Name System (DNS) on the internet. However, in contrast to DNS, this system needs to prevent centralised registration and instead depends on a decentralized network overlay constructed on the peer-to-peer communication protocol.
For more detailed implementation specifics, refer to Section 4.6.

## 3.4 Redundancy

Redundancy is an essential aspect of the SILO network to guarantee the continuous accessibility of data, even in the event of some storage nodes being unavailable. This is especially crucial in decentralized systems, since nodes are autonomously controlled and have the potential to exit or malfunction at any given moment. In order to provide consistent access and dependability, our redundancy method is specifically formulated to store data in a manner that maximises the likelihood of accessibility, even in situations when some nodes are not operational.

Limitations of Traditional Replication: Conventional approaches to ensure data durability often use basic replication, where numerous copies of data are kept across distinct nodes. Nevertheless, this method establishes a direct connection between durability and the expansion factor, which is the ratio of stored data to real data. This results in substantial storage overhead and higher expenses. For instance, if data is replicated eight times to achieve high durability, it leads to an expansion factor of 8x, or 800%, which requires a significant amount of storage space and bandwidth.

Erasure coding is used by SILO to address the drawbacks of replication. This sophisticated technique allows for efficient redundancy without compromising durability due to bandwidth consumption. Erasure codes are widely used in distributed storage systems and provide a more efficient method for guaranteeing data integrity and availability.

Erasure coding involves splitting data into multiple pieces and encoding them such that the original data can be reconstructed from a subset of these pieces. This approach is defined by two parameters, k and n:

- $k$: The number of data pieces needed to reconstruct the original data.
- $n$: The total number of pieces after encoding.

For instance, with a ($k$=20, $n$=40) erasure code, there are 40 encoded pieces, and any 20 of these pieces are sufficient to reconstruct the original data. This method provides higher durability than a ($k$=10, $n$=20) code with the same expansion factor of 2, because the risk is spread across more nodes.

To illustrate:

- A (k=4, n=8) erasure code has an expansion factor of 2 and a high probability of durability (99.86%).
- In contrast, a replication scheme with a similar expansion factor offers much lower durability for the same data overhead.

Calculating Durability: Durability in a decentralized network can be estimated using a probabilistic model. Assuming a node churn rate ppp (the probability of a node going offline in a given period), durability P(D) can be modelled using a Poisson distribution. The formula for calculating the probability that data remains accessible is given by:

$$P(D) = e^{-\lambda} \sum_{i=0}^{n-k} \frac{\lambda^{i}}{i!}$$

where λ=pm, representing the expected number of pieces lost due to node churn. This formula calculates the cumulative probability that at most n-k pieces are lost, allowing the original data to be reconstructed. Advantages of Erasure Coding: Erasure coding provides several advantages:

- Lower Expansion Factor: Achieves high durability with less storage overhead.
- Reduced Bandwidth Usage: Minimises the amount of data transferred across the network, essential for scalability.
- Higher Payout for Nodes: Allows storage node operators to receive more direct income per byte stored, as high expansion factors dilute earnings.

By using erasure coding, SILO ensures robust data redundancy and durability, optimising both cost and performance for decentralized storage.

### 3.4.1 Impact of Erasure Codes on Streaming

Erasure codes are widely used in many streaming applications, such as audio CDs and orbital communications, because of their effectiveness in error correction and data recovery. Incorporating erasure coding into the SILO design does not impede our need to provide streaming, which is essential for achieving compatibility with Amazon S3.

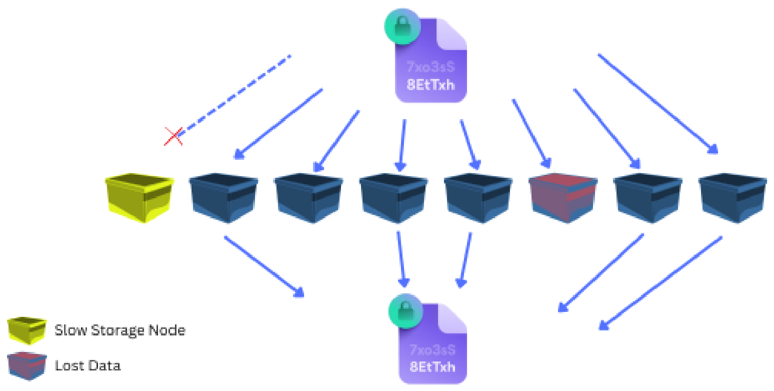
The key element of integrating erasure coding into streaming is the ability to encode and decode data in small chunks, rather than processing the whole file at once. This approach enables seamless data delivery, enabling real-time streaming without significant latency or buffering issues. SILO employs sequential encoding of brief data segments to provide efficient streaming capabilities, while simultaneously using the durability and redundancy provided by erasure codes.

This technique ensures the long-lasting and flexible nature of our system, allowing it to accommodate many situations, including those requiring high speed and lowest latency, such as video streaming. To get more details on the incorporation of erasure codes in our streaming support system, go to Section 4.8.

## 3.4.2 Impact of Erasure Codes on Long-Tail Latency

Erasure codes provide significant performance benefits, especially in reducing the impact of extended reaction times in dispersed systems. A long-tail response refers to the occurrence of abnormally slow reaction times in a server or storage node, resulting from unknown reasons and causing delays in overall operations. Although rare, these delays may vary greatly and cause performance issues, shown as a "long tail" on a probability density graph.

In distributed systems such as MapReduce, delays like this are mitigated by using "backup tasks," which are duplicate requests specifically intended to compensate for underperforming nodes, sometimes known as "stragglers." Despite the redundancy of these backup activities, they effectively prevent sluggish nodes from causing delays in the overall operation. Without this method, the duration of operations in MapReduce might increase by 44%.



The SILO system utilises erasure coding to effectively manage long-tail answers in storage. Optimization of uploads and downloads may be achieved by using a greater (k, n) ratio, which involves encoding data into more pieces than required for ensuring specified durability guarantees. During an upload, after a sufficient number of pieces have been transferred to meet the necessary redundancy, any more uploads may be terminated, enabling the process to finish as swiftly as the most efficient nodes. This strategy circumvents the need to wait for the nodes with the slowest processing speed and reduces the total duration of the upload process.
Similarly, when downloading, the system has the capability to choose the quickest peers to get data from, disregarding slower or momentarily inactive nodes. The capability to choose the swiftest nodes guarantees that operations are not delayed by slower respondents, essentially transforming a potential drawback into a performance advantage.

SILO utilises over-encoding of files to distribute the burden of frequently accessed material evenly throughout the network, resulting in efficient and rapid data retrieval. This technique offers resilient load balancing features while ensuring optimal performance and dependability. To get further information on load balancing and its implementation in active files, go to Section 6.1.

## 3.5 Metadata Management

Within the SILO network, it is essential to retain comprehensive records of the particular nodes that store individual data bits once an object has been separated via erasure coding and distributed over several nodes. This is particularly crucial since users could choose storage nodes based on factors like geographical proximity, performance attributes, or available capacity. SILO utilizes an explicit node selection technique instead of an implicit one, such as consistent hashing in systems like Dynamo, in order to accommodate user preferences and ensure compatibility with Amazon S3.

Metadata Requirements and Structure:

To effectively manage and retrieve data, the metadata system in SILO needs to support several key functionalities:

- Hierarchical Objects: The ability to manage paths with prefixes, allowing for an organised and hierarchical structure of data similar to directory paths.

- Per-Object Key/Value Storage: The ability to store additional metadata for each object, which might include data like creation dates, modification dates, and other custom attributes.
- Arbitrary File Sizes and Quantities: The system must handle both large files and a large number of files efficiently, without size limitations.
- Arbitrary Key-Based Access: Users must be able to store and retrieve data using any arbitrary key, often structured like a file path.
- Deterministic Key Iteration: Support for paginated listing, which is crucial for applications that need to browse or manage large sets of stored objects.

The metadata system must be able to handle a high amount of churn caused by frequent additions, revisions, or removals of objects, according to these criteria. Additionally, it must be able to adapt and expand in accordance with the general expansion of the network. For example, if 1 exabyte of data is stored with an average object size of 50MB and an erasure code is used with a parameter n=40, the management of metadata would be required for 20 billion objects. If each item is around 40×64+192 bytes, the total amount of metadata would be approximately 55 terabytes.

**Partitioning and Flexibility:**
In order to effectively handle this large amount of data, SILO's metadata system is specifically built to be extensively divided per user. Consequently, the metadata of each user is handled independently, thereby minimising the burden on individual users. For instance, if a user stores 100 terabytes of 50MB items, the metadata overhead would amount to around 5.5 gigabytes.

SILO also prioritises the interchangeability of the metadata storage component. Users may enhance flexibility and customization by choosing the most suited choice for their requirements among different implementations of metadata storage. This method is in line with SILO's objective of minimising user cooperation, decreasing possible bottlenecks, and eliminating single points of failure.

**Distributed and Resilient Design:**
In order to guarantee both availability and fault tolerance, the metadata services are dispersed among numerous regions within each geographic area. This distribution provides resilience against a range of failures, including device malfunctions, server outages, network problems, and even interruptions at a regional level.

**API Compatibility:**
For compatibility with Amazon S3, the metadata API in SILO provides straightforward and essential functions:

- Put: Store metadata at a specified path.
- Get: Retrieve metadata from a specified path.
- List: Provide a paginated and deterministic listing of existing paths.
- Delete: Remove metadata from a specified path.

SILO's features enable efficient metadata management, catering to diverse storage requirements while ensuring user-friendliness and seamless integration with established cloud storage protocols.

## 3.6 Encryption

In order to guarantee utmost security and confidentiality, it is essential that all data and metadata inside the SILO network be encrypted. Encryption should be implemented at the earliest stage of the data storage process, preferably prior to the data being sent from the user's device. Therefore, in order to ensure the anonymity of data from its source, it is necessary for any interfaces or client libraries that are compatible with Amazon S3 or comparable protocols to be operated on the same device as the user's application.

Pluggable Encryption Mechanism: SILO has a flexible encryption system that enables users to choose their own encryption strategy. Users have the ability to choose an encryption method that aligns with their particular security requirements, thanks to this flexibility. Furthermore, the system retains information about the selected encryption method, which is essential for data retrieval in the event that encryption settings are modified or enhanced in the future.

Unique Encryption Keys for Files: To provide improved access control and heightened security, every file in SILO is encrypted using a unique key. This strategy guarantees that accessing one file does not result in accessing others, hence preserving a stringent segregation between various files. Users have the ability to share individual files without jeopardizing the encryption keys or information of other files, providing precise control over data sharing and access.

Encryption Metadata Management: To ensure the security and reliability of the encryption techniques, it is crucial to securely and reliably retain information about the keys and algorithms used for encrypting each file. The metadata, along with other file information such as its path, is saved in the aforementioned metadata storage system. In order to guarantee security, this metadata is also encrypted using a deterministic, hierarchical encryption technique.

Hierarchical Deterministic Encryption Scheme: SILO utilises a hierarchical encryption method that is based on BIP32, a widely accepted standard for hierarchical deterministic wallets in the field of cryptocurrencies. This technique facilitates the safe and adaptable exchange of files, allowing the sharing of subtrees while keeping the parent nodes concealed. Additionally, it guarantees the ability to share individual files separately from others, therefore improving privacy and control. To get a comprehensive analysis of our path-based hierarchical deterministic encryption technique, please refer to Section 4.11.

## 3.7 Data Integrity Audits and Node Reputation

Ensuring the reliable storage of data by storage nodes is vital for the success of the SILO network. In order to maintain trust and dependability, it is crucial to provide processes that authenticate and confirm the proper storage of data by storage nodes.

Establishing Node Identity and Initial Reputation: Upon joining the SILO network, storage nodes create a distinct identity by performing a brief proof-of-work task. Establishing this initial identification is crucial for fostering confidence and confirming the authenticity of the node. During the vetting stage, newly added storage nodes are subjected to rigorous audits and uptime tests to verify their proper functioning. Initially, a fraction of the storage node's profits is retained as a precautionary measure to mitigate possible losses in the event of the node's sudden departure from the network. The withheld money may be used to pay the expenses of repairing any data kept on the node in the event of its departure.

Audit Mechanism: The SILO network utilises a resilient audit system to oversee and authenticate the integrity of data stored on every node. This procedure entails conducting probabilistic audits, referred to as proofs of retrievability, on all files stored on a node. These audits are conducted as random tests to verify, with a significant level of confidence and little additional costs, that a storage node is functioning successfully, preserving the data's integrity, and not encountering hardware malfunctions or engaging in harmful activities.

When a storage node fails an audit, it is designated as unreliable, and the data held on that node is redistributed to other nodes to maintain data integrity. This technique contributes to the preservation of the network's health and stability by circumventing faulty nodes and guaranteeing that data is consistently saved on trustworthy and dependable nodes.

Probabilistic Audits and False Positives: The audit technique used by SILO does not include scrutinising each individual byte of every file. Instead, it employs a probabilistic methodology to authenticate the data, which may sometimes lead to false positives—cases where the system erroneously concludes that the data is intact while it has really been altered or partly lost. Nevertheless, the probability of a false positive may be computed and reduced by conducting repeated audits, so guaranteeing that any missing or tampered data is identified with a high level of certainty.

Reputation System: SILO utilises a reputation system to monitor and control the dependability of storage nodes by maintaining a record of the audit results for each node. This solution enables the network to make well-informed judgments about the reliability of nodes, hence improving the overall security and integrity of the data storage network. To get further information on our first strategy for reputation management, please refer to Section 4.15.

## 3.8 Data Repair

Data loss is an inherent risk in every distributed storage network, resulting from a variety of sources. The SILO network faces several factors that may lead to data loss, including corruption, malicious activities, hardware failures, software mistakes, and user-initiated deletions. However, the most prominent risk to data integrity in the SILO network is the frequent replacement of storage nodes, also known as storage node churn. Storage node churn is the recurring process of nodes entering and departing the network, which, if not properly managed, may result in the loss of stored data.

As mentioned in Section 2.5, the typical duration of storage nodes' sessions in several decentralized systems might vary from several hours to a few minutes. The significant pace at which employees leave their positions is a considerable obstacle in ensuring the long-term persistence of data. Although data corruption and hardware breakdowns are worrisome, they are not as substantial a threat to data integrity as the continuous variability of storage nodes.

Data Repair Process:
In order to reduce the possibility of data loss caused by node churn,

SILO implements a resilient data repair mechanism. This procedure encompasses many essential stages:

1. Audit Detection: The audit system consistently checks storage nodes to verify the accurate storage of data. When a node fails to accurately store data or becomes disconnected, the system identifies this divergence from the anticipated behaviour.
2. Data Reconstruction: When the system detects a failure, it uses the remaining data fragments to rebuild the missing data. Erasure coding enables the complete recovery of the original data from a portion of its fragments.
3. Data Regeneration:The system reconstructs the original data and then regenerates the missing portions, which are then stored on new and dependable storage nodes. This phase guarantees the preservation of data redundancy, even when nodes enter and exit the network.

**Incentivizing Stability:**
In order to decrease the occurrence of node churn and encourage sustained involvement, SILO has devised a payment method that incentivizes storage node owners who maintain their nodes in an active state for prolonged durations. By offering rewards to nodes for staying connected over extended periods, maybe spanning months or even years, the network can improve the longevity of data and minimise the need for periodic maintenance.

This method not only contributes to the general well-being of the network but also decreases the operational expenses related to data restoration. SILO's objective is to provide a stable and robust storage environment in decentralized networks by motivating storage node owners to maintain continuous uptime and dependability, hence overcoming the inherent obstacles.

## 3.9 Payment Systems

Payments, value allocation, and invoicing are crucial elements in sustaining a stable and robust decentralized storage environment. Effective and dependable payment systems provide equitable compensation for storage providers and maintain the network's seamless operation by ensuring a consistent availability of storage resources.

Avoiding Blockchain Dependency for Payments:
In order to minimise the delay and maximise the rate of data transfer in the SILO network, it is essential to eliminate any interdependencies between transactions and the blockchain technology.

As mentioned in Section 2.10, it is not feasible to wait for blockchain confirmations in a system that needs actions to be performed within milliseconds. Transactions that are based on blockchain technology often experience substantial delays since they need agreement among several distributed nodes. This may negatively impact performance in a storage environment that requires fast processing.

Contrarily, the SILO framework utilises game-theoretic models to encourage network members to behave rationally and honestly in order to obtain compensation. These models draw inspiration from real-world financial interactions that prioritise trust and collaboration as essential factors. SILO promotes the beneficial contribution of all members to the network's general health and stability by imitating these interactions.

**Background Settlement Process:**
Transactions in the SILO network are handled using a background settlement mechanism, where players who follow the rules work together to resolve financial obligations. This solution eliminates the need for immediate reliance on a single payment method, which results in more seamless transactions and decreases the potential risks connected with untrustworthy payers. Storage nodes are advised to minimise their interaction with unfamiliar or untrusted individuals who want to make payments until a certain degree of trust is built. This precaution ensures that the nodes are more likely to be adequately compensated for the services they provide.

**Tracking and Billing for Service Usage:**
The framework also incorporates systems for monitoring and consolidating the worth of services used by customers who store data on the network. SILO's use of usage fees enables the establishment of a viable economic framework for its decentralized storage marketplace. This strategy guarantees equitable remuneration for storage providers in accordance with their resources, while customers are invoiced in a transparent manner, taking into account their use.

**Flexibility in Payment Methods:**
Although the SILO network is meant to be flexible in terms of payment options and does not enforce a single payment method, it presently defaults to utilise a native token as the main form of payment. While this token is now the default, SILO has the capability to accommodate several different payment methods in the future. These options include cryptocurrencies such as Bitcoin and Ethereum, conventional payment methods like credit or debit cards and ACH transfers, as well as unorthodox payment formats.

SILO intends to provide a smooth experience for all users in the network by implementing a payment system that is both versatile and adaptive, allowing it to meet a broad variety of user preferences.

# 4 Concrete Implementation

Due to the design restrictions of our framework, we possess a clearly established basis on which to go further. Nevertheless, within this structure, there exists adaptability in the manner in which each element is executed. Within this part, we provide our preliminary approach for implementing the project, acknowledging that these specific plans may change as time progresses. We prioritise the implementation of a cloud storage solution that is safe, high-performing, and long-lasting, in accordance with the criteria stated in our framework.

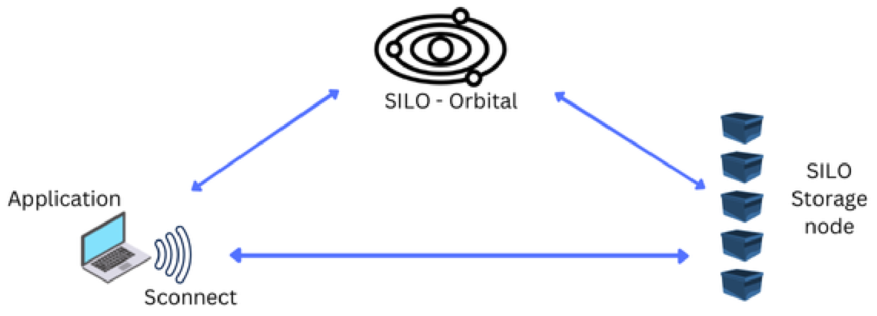
## 4.1 Definitions

This section defines key terms used throughout the description of our concrete implementation:

### 4.1.1 Actors

- Client: An end-user or application responsible for uploading or downloading data from the SILO network.
- Peer Class: A category of network participants that perform specific roles and services. There are three peer classes within the SILO network:
    - Storage Node: These nodes are responsible for storing data for clients and are compensated for their storage space and bandwidth usage. Storage nodes register directly with Orbitals (previously known as orbitals) that they trust.
    - Sconnect: This peer class, represents any application or service using the libsconnect library to store or retrieve data. Sconnect peers perform encryption, erasure encoding, and coordinate with other peer classes on behalf of the client. Unlike other classes, Sconnect peers are not required to be online continuously and are relatively lightweight.
    - Orbital: Previously referred to as orbitals, Orbitals are responsible for various network management tasks. They allow storage nodes to register, store metadata, manage node reputation, perform audits and repairs, aggregate billing data, process payments, manage user accounts, and oversee authorization through AuthShield.

- libsconnect: A library that provides all the necessary functions to interact directly with storage nodes and Orbitals. This library will be available in multiple programming languages to accommodate different user needs.
- Nexus: Nexus provides a compatibility layer between other object storage services (such as Amazon S3) and libsconnect, exposing an S3-compatible API for seamless integration.
- Sconnect CLI: A command-line interface that facilitates uploading and downloading files from the network, managing permissions, sharing, and managing user accounts.
- ShareHub: A service for managing link sharing and permissions, enabling users to share access to specific files or directories within the network.



These definitions establish a foundational understanding of the various components and participants in the SILO network, outlining their roles and responsibilities.

## 4.1.2 Data Definitions

This section provides definitions for key data-related terms used in our implementation:

- Collection (Bucket): A named, unbounded collection of objects or files, each identified by a unique object key within the collection.
- Object Key: A unique identifier for an object within a collection. It is an arbitrary string that functions like a file path, using forward slashes to denote access control boundaries (e.g., videos/carlsagan/gloriousdawn.mp4). Object keys are encrypted before leaving the client's computer for privacy and security.
- Data Object (File): The primary data type within the system, represented by an ordered collection of one or more segments. An object can contain any amount of data, with no minimum or maximum size, and supports user-defined key/value metadata fields. Data and metadata are encrypted client-side.
- Object Metadata: Encrypted key/value fields defined by users, associated with a specific object.

- Segment: A single array of bytes within an object, ranging from 0 to a maximum size defined by the Orbital (previously orbital). Segments can be either remote or inline.
- Remote Segment: A segment that is erasure encoded and distributed across the network, with a size greater than the metadata needed to manage it. This type of segment is stored across multiple nodes.
- Inline Segment: A smaller segment where the data occupies less space than the metadata needed for remote storage. This data is stored directly, or "inline," without being distributed across the network.
- Encryption Block: A fixed-size byte array used as a boundary for encryption. Each block is encrypted individually, often aligned with the stripe size for efficiency.
- Stripe: A fixed-size byte array used as a boundary for erasure encoding within a segment. Stripes are encoded individually to generate erasure shares and are the units on which audits are performed.
- Erasure Share: A piece generated from encoding a stripe. Multiple erasure shares are created, but only a subset is needed to reconstruct the original stripe. Each share has an index to identify its position.
- Piece: A concatenation of all erasure shares with the same index from a remote segment's stripes. For example, if a stripe generates n erasure shares, there will be n pieces, each comprising shares with the same index from different stripes.
- Pointer: A data structure that either contains inline segment data or tracks the locations of remote segment pieces across storage nodes, along with other relevant metadata.

These definitions clarify the various components involved in managing data within the SILO network, ensuring efficient storage, security, and retrieval.

## 4.2 Network Participant Roles

Our design strategy builds upon previous iterations and parallels distributed storage architectures like the Google File System (GFS) and Lustre. In these systems, three primary roles are essential: metadata servers, object storage servers, and clients. Here's how these roles are adapted and enhanced in the SILO network:

- Storage Nodes: These nodes, previously referred to as Storj Share, are responsible for storing the bulk of data in the network. Storage nodes manage the data that clients upload and are selected based on various performance criteria, including uptime and bandwidth capabilities.

- Orbitals: Formerly known as orbitals, Orbitals are metadata servers that manage and store metadata, handle storage node registrations, and coordinate data integrity checks. They also perform audits, manage user accounts and permissions through AuthShield, and facilitate payments. Unlike the previous centralized model, any user can now operate their own Orbital, although many may choose to rely on trusted third-party providers.
- Sconnects: Sconnects represent clients that interface with the network to store and retrieve data. They are lightweight and do not require constant online presence. Sconnects handle tasks like encryption, erasure coding, and coordinating with other network components. The libsconnect library provides all necessary functionalities for Sconnects to interact seamlessly with storage nodes and Orbitals.

This refined structure optimizes network performance and resilience by distributing roles among different participants. By leveraging proven distributed system architectures, SILO ensures robust, scalable, and high-performance decentralized storage, accommodating a variety of user needs while maintaining data integrity and security.

## 4.3 Data Node Operations

**Main Role of Data Nodes:**
Data nodes, formerly known as storage nodes, play a vital role in the SILO network, chiefly responsible for the dependable storage and retrieval of data. Data node operators are usually individuals or companies who have extra hard drive space and want to generate cash by leasing this space to others on the network.

**Node Setup and Configuration:**
Operators are responsible for installing and configuring the SILO software on their own computers. During this process, they choose the amount of disk space and bandwidth they are prepared to commit for each Orbital (formerly known as orbital). During the registration procedure, data nodes submit details on their accessible resources and indicate their token wallet address for payment purposes.

**Data Storage and Management:**
Data nodes contain data fragments together with optional "time-to-live" (TTL) configurations, which determine the duration for which a data fragment should be retained before it is removed. If a Time to Live (TTL) is not given, the data will be held forever. Data nodes are responsible for maintaining a database that manages expiry periods and periodically removes expired data in order to minimize storage.

**Bandwidth and Payments:**
Data nodes also monitor and record the amount of bandwidth they provide for the data they provide, which is then sent to Orbitals for payment processing. Nodes have the ability to choose which Orbitals to collaborate with, and they may earn income from various sources based on the agreements in place. Users are charged for the storage of data when it is not being actively used, as well as for accessing the data. However, there is no charge for uploading the data initially, in order to discourage users from using the system.

**Audit and Disqualification:**
Data nodes are required to consistently and dependably store the data that is allocated to them. In order to guarantee this, Orbitals conduct sporadic checks of stored data. If a node fails these audits, it might be disqualified, leading to the loss of stored data and retained cash, and no more payouts from the network.

**Supported Operations:**
Data nodes support three main operations: get, put, and delete. Each operation requires a piece ID, an Orbital ID, a signature from the corresponding Orbital instance, and a bandwidth allocation.

- Put: Stores data with an optional TTL, ensuring that any subset of the data can be retrieved with a get operation until the data is deleted or its TTL expires.
- Get: Retrieves stored data as long as it remains valid under its TTL or until it is deleted or removed during garbage collection.
- Delete: Removes data when instructed by Orbitals or during scheduled garbage collection based on a probabilistic data structure known as a Bloom filter, which identifies data that is no longer needed.

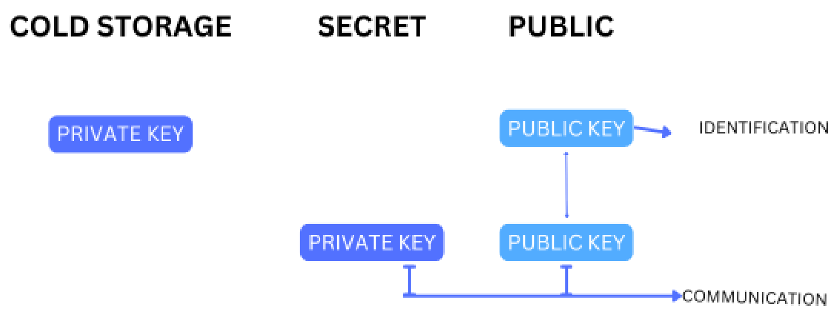
**Open Source Availability:**
The software for data nodes is made available as open source, enabling universal participation in the network and the ability to contribute to its development and safeguarding.

This innovative configuration maximizes the functionality of data nodes, guaranteeing dependable data storage while offering explicit motivations for engagement and adherence.

## 4.4 Establishing Node Identity

**Identity Generation and Certificate Creation:**
During the initial configuration process, every member in the NodeGrid system, whether it is a storage node, Orbital, or Sconnect, creates its own distinct identity and corresponding certificates to ensure safe network activities. This procedure entails generating a certificate authority (CA) that is unique to each node. It necessitates the use of a public/private key pair and a certificate that is signed by the node itself.

- Certificate Authority Management: It is advisable to keep the private key for the certificate authority in a highly secure cold storage to avoid any unwanted access or compromise of the key. Ensuring strong operational security for the CA's private key is essential since any breach requires a full re-issuance of the node's identity, which might impede network operations.



Node Identification and Proof of Work:
The node ID of each node is generated by hashing its public key, using a technique similar to that of S/Kademlia. The node ID functions as a validation mechanism for entering the network. SILO's proof of work algorithm differs from the typical Bitcoin proof of work by emphasizing the discovery of trailing zero bits in the hash result. This approach requires significant computing resources and incurs high costs, thereby mitigating the likelihood of Sybil attacks.

**Leaf Certificates and Key Rotation:**
Nodes also produce a revocable leaf certificate and key pair, which are authenticated by the node's Certificate Authority (CA). The leaf certificate serves as a means of daily communication and has a signed timestamp to monitor its current validity.

- Certificate Renewal and Compromise Handling: If a breach occurs, the node has the ability to generate a new leaf certificate that includes an updated timestamp, which is then monitored by Orbitals. Peers will acknowledge the recently issued leaf certificate and refuse connections that try to utilize older, obsolete certificates. A streamlined exception occurs when the leaf certificate and the CA certificate have the same timestamp, hence

- reducing the need for further timestamp verification in such instances.

This method guarantees that every node in the network maintains safe and verifiable identities, which prevents impersonation and improves the overall security of the network.

## 4.5 Direct Node Communication

The SILO network employs a DRPC protocol, which is an open-source, gRPC-like communication protocol, for inter-node communication. The purpose of this protocol is to provide both safe and efficient communication across the network, establishing a strong basis for interactions between nodes.

**Protocol Security and Structure:**
DRPC utilizes a hierarchical protocol stack including of Transport Layer Security (TLS) and the Noise Protocol Framework in IK mode. These protocols are implemented over either TCP or QUIC, which is a protocol that works over UDP. These layers have specific functions:

- TCP/QUIC: Ensure the dependable and sequential transmission of data over the network, guaranteeing that packets are received in the proper sequence without any loss.
- TLS/Noise: Enhance security by using encryption and authentication measures. TLS is used for providing extensive security measures, such as mutual authentication, while Noise may be utilized in situations that prioritize reduced latency and do not need forward secrecy to be of utmost importance.
- DRPC:Enables the consolidation of several communication channels into a single connection, while offering a user-friendly interface for developers.

**Authentication and Proof of Work:**
By using secure communication protocols such as TLS or Noise, every node may verify the authenticity of the peer it is interacting with. This is done by validating the certificate chain and hashing the public key of the peer's certificate authority. This procedure enables nodes to build confidence and evaluate the computational effort (proof of work) linked to a peer's node ID, which is defined by the quantity of trailing zero bits in the hash.

In order to maintain network security and mitigate Sybil attacks, Orbitals have the ability to establish a minimum proof of work prerequisite for nodes to successfully undergo audits. The threshold may be modified gradually to improve the robustness of the network and guarantee that only nodes with enough processing resources are allowed to participate.

The implementation of this organised communication protocol guarantees that SILO maintains a network environment that is both safe and efficient, while also being capable of scaling to accommodate a larger number of network users. This protocol facilitates dependable interactions between all participants in the network.

## 4.6 Dynamic Node Discovery

Within the SILO network, storage nodes are often linked via diverse consumer internet services, normally situated behind routers that possess dynamically altering IP addresses. This poses a difficulty in regularly identifying and establishing communication with these nodes over a prolonged period. In order to address this issue, SILO utilizes a dynamic node discovery system that is specifically built to monitor and update the position of nodes based on their distinct identities, much to how DNS functions for the public internet.

Every Orbital in the SILO network is furnished with a node discovery cache that retains vital information necessary for communication with storage nodes. This cache contains data such as the node's storage capacity, current IP address, and other important metadata. The decentralized caching mechanism allows the SILO network to effectively identify and communicate with storage nodes, allowing the seamless storage and retrieval of data as required.

When a storage node enters the network, it establishes communication with every Orbital on its trust list in order to officially declare its existence. The Orbital validates the connection, confirming that the node is accessible and capable of retaining data. After verification, the Orbital updates its node discovery cache with the node's current status and metadata, guaranteeing that future data storage activities have access to the most recent information.

In order to ensure the accuracy of the data for active nodes, each storage node transmits a periodic signal, known as a "heartbeat," to the Orbitals it is registered with. This signal is usually sent every hour. The purpose of this heartbeat is to verify the continuous availability of the node and to communicate any changes made to its configuration. If a node fails to transmit its heartbeat within the designated time period, the Orbital system will make an effort to establish communication with the node. If the node becomes inaccessible, the Orbital system will designate it as inactive, so halting any more data assignments and initiating any required data recovery procedures.

The use of this dynamic method for node discovery guarantees that the SILO network maintains its resilience and responsiveness, allowing it to adjust to fluctuations in network circumstances and uphold a superior degree of service dependability. SILO effectively manages its distributed storage network and offers reliable data storage solutions by consistently updating the status of nodes.

## 4.7 Data Redundancy

SILO employs Reed-Solomon erasure coding to guarantee data durability and minimise the possibility of data loss caused by node failures or other disturbances. This method is efficient in spreading data across several nodes, enabling data retrieval even in the event of node disconnection or data corruption. SILO employs four critical parameters, namely u, v, w, and x, in the Reed-Solomon coding scheme for each stored data item. It is important to note that the values of u, v, w, and x follow the condition $u \leq v \leq w \leq x$.
In this setup:

- 'u' denotes the least quantity of fragments necessary to restore the initial data.
- 'x' represents the aggregate quantity of items produced during the encoding procedure.
- The threshold 'v' is the minimum number of available pieces at which SILO will trigger a repair operation to avoid any potential data loss. This guarantees that there will always be a minimum of u pieces accessible. This threshold ensures the maintenance of a constant degree of redundancy and data availability.
- The variable 'w' reflects the ideal quantity of components required to provide the specified level of endurance without causing costly repairs or retaining excessive data. During the process of uploading or fixing data, after the successful storage of certain components on the website, any more uploads up to a certain limit, denoted as x, are stopped in order to improve efficiency and efficiently use resources.

This redundancy method enables SILO to successfully manage sluggish or momentarily down nodes. The system has the ability to handle a maximum of $x - w$ nodes that have a poor response time during uploads. This improves the overall performance of the network by minimising the negative effects of these extended delays. In addition, SILO has the capability to handle the offline status of up to $w - v$ nodes without requiring quick repairs, thereby offering a strong safeguard against data loss. The distinction between the letters "v" and "u" provides an extra level of security, guaranteeing the preservation of data integrity even while undergoing repair procedures.

SILO achieves optimal data durability and network efficiency by meticulously setting these parameters. This strategy decreases the need for frequent repairs and limits the additional space required for storage, all while maintaining a high level of dependability and efficiency. To get further information on the selection of Reed-Solomon parameters and data repair procedures in situations when durability diminishes, refer to sections 7.3 and 4.14.

## 4.8 Organized Data Management

### 4.8.1 Metadata Management for Objects

SILO facilitates improved file management and offers the capability to include information for each item saved on the network, hence providing extra context. This feature has resemblance to the "object metadata" feature offered by Amazon S3 and the "extended attributes" present in most POSIX-compatible systems.

Within the SILO system, users have the ability to provide custom key-value pairs to any item, enabling them to store supplementary data that is relevant to their information. The metadata is kept along with other object-specific information, such as storage location and access rules, which allows for easy access and facilitates more effective data retrieval and administration. SILO enables several use cases by providing support for metadata, allowing for tasks such as basic file descriptions and intricate data classification and indexing.

### 4.8.2 Segmenting Objects

Our revised method incorporates precise terms to prevent any ambiguity that may have arisen in prior editions. In the past, the word "shard" denoted fragments kept on separate nodes, while "sharding" denoted the act of dividing a file into smaller segments for more convenient management. The use of erasure coding in previous versions led to ambiguity in this nomenclature. To provide further clarification, we now designate this procedure as "segmenting," and the main partition of an object's data flow is referred to as a "segment."

A segment denotes a significant fraction of the data included inside a file. If the file is sufficiently compact, it may include just a single segment. An "inline segment" refers to the situation when the size of a segment is lower than the metadata needed to store it on the network, resulting in the data being saved along with the information.

When dealing with bigger files, the data is divided into many "remote segments." Partitioning data in this manner offers a multitude of advantages pertaining to security, privacy, efficiency, and accessibility. Distributed storage systems, such as Google File System (GFS) and Lustre, divide portions of big files (such as films) among several network nodes. This helps evenly disperse the bandwidth burden and prevents any one node from becoming a bottleneck. This strategy not only promotes data privacy and security, but also helps overall network speed by enabling simultaneous data transmission, comparable to the capability seen in peer-to-peer networks such as BitTorrent. Furthermore, by the process of standardising the size of segments, we can achieve a higher level of consistency in storage among nodes. This ensures that storage needs are evenly distributed and enables more effective retrieval of data.

### 4.8.3 Segments Defined as Stripes

Accessing a particular subset of a bigger dataset is often essential in many situations. Some file formats, such those used for films or disk pictures, often provide "seeking," which allows for reading just a specified portion of data. This capacity is crucial in many applications, such as audio CDs, where the decoding of short segments is vital to effectively support operations.

In order to provide selective access, we establish a "stripe" as a smaller portion of a segment. Stripes are often small, usually a few kilobytes in size, which enables efficient management of data and precise access. Encryption in our system takes place at the border of encryption blocks, which are tiny multiples of stripes. Meanwhile, erasure encoding is used to improve the durability and integrity of data, and it is applied to each individual stripe.

Our use of authenticated encryption results in a little additional cost for each encryption block. Consequently, opting for somewhat bigger encryption sizes may be advantageous in minimising this cost. Nevertheless, due to the fact that audits are performed at the stripe level, it is crucial to maintain minimal stripe widths in order to reduce the amount of bandwidth used during audits. This meticulous equilibrium guarantees that data is both safe and kept effectively, while keeping the resources needed for frequent audits and verifications at a minimum.

For those acquainted with the zfec library, our notion of a stripe roughly corresponds to what zfec denotes as a "chunk" in its filefec mode, demonstrating the interchangeable use of terminology across many systems and situations.

### 4.8.4 Converting Stripes into Erasure Shares

As mentioned before in Sections 3.4 and 4.7, the use of erasure codes is crucial for ensuring data persistence, especially in decentralized networks where storage nodes may be unreliable. Erasure coding is a technique that enables the recovery of data even if certain sections of it are missing. This is especially beneficial for preserving data integrity in a network with several remote nodes.

Our approach utilises erasure coding at the stripe level. In an erasure code setup given by parameters (k, n), a stripe is partitioned into n erasure shares. For instance, when using a setup with k = 20 and n = 40, it implies that the initial stripe is divided into 40 distinct erasure shares. A minimum of 20 out of the 40 shares is required to restore the original stripe, ensuring that the system can withstand the loss or corruption of up to 20 shares without any loss of data. As a result, each erase share would be one twentieth of the size of the original stripe.

Encoding data stripe-by-stripe has several advantages. It allows for the extraction of tiny sections from big data segments without requiring the complete download of the full segment. This feature is particularly beneficial for applications that need rapid retrieval of certain data segments, such as video streaming or databases that provide partial reads or searches. Moreover, this approach enables the seamless transmission of data into the network in real-time, eliminating the need for prearrangement and enhancing the efficiency and adaptability of the storage procedure.

To get a comprehensive examination of how modifying the parameters of erasure codes, such as tweaking k and n, affects the availability and redundancy of data, please see Section 7.3.3. This section offers valuable information on how to optimise these settings in order to achieve a balance between storage efficiency and data dependability.

### 4.8.5 Consolidating Erasure Shares into Pieces

Considering that stripes are already quite diminutive data units, the erasure shares generated from these stripes are considerably more little. Handling each of these erasure shares separately would need a substantial quantity of information, resulting in an inefficient system. In order to address this issue, we use a tactic in which all erasure shares that have the same index are merged together to form a unified entity known as a piece.

In a standard erasure coding configuration denoted by (u,x)(u, x)(u,x), where uuu represents the minimum number of pieces needed for reconstruction, and xxx represents the total number of pieces, each erasure share is assigned an index. Each portion of a stripe will consistently generate the corresponding portion when encoded. Instead of individually managing each of these shares, we consolidate all shares with the same index into a single unit. In a (u,x) scheme, there are xxx pieces, and each piece iii consists of all the erasure shares with the same index iii. Therefore, each erasure share corresponds to 1u\frac{1}{u}u1 of a stripe, and each piece corresponds to 1u\frac{1}{u}u1 of a segment. To reconstruct the whole segment, only uuu pieces are needed.

When a new upload occurs, the Orbital generates a unique root piece ID. The root piece ID is kept secret inside the Sconnect system and is used to generate a unique piece ID for every storage node. The piece ID is produced by applying a Hash-based Message Authentication Code (HMAC) to the root piece ID and the ID of the individual node. This technique guarantees that storage nodes are unable to readily discern which components are associated, hence bolstering data security.

In order to enhance data security and facilitate management, the parts are arranged according to their respective Orbital ID. If two orbitals have the same piece ID, they are regarded to be separate from one other. Each Orbital would presume that the parts, albeit having the same ID, had distinct data and have independent lifecycles. This technique efficiently mitigates data conflicts and guarantees smooth data administration across many Orbitals.

### 4.8.6 Managing Data with Pointers

In order to successfully recover a distant segment from the network, the data owner must possess knowledge of the segmentation process and the specific storage locations of each individual piece. The data is stored in a data structure called a pointer.
A pointer includes several key elements:

- Storage Node Information: Information on the specific nodes that are responsible for storing different parts of the segment.
- Encryption Data: Information required to decrypt the data upon retrieval.
- Erasure Coding Parameters: Specifications about how the data has been erasure coded, including the number of pieces generated and how many are needed for reconstruction.

- Repair Thresholds: Defined limits that trigger a repair process when redundancy drops below a certain level. This helps maintain data integrity and availability.
- Repair Success Criteria: The required number of pieces that must be restored for a repair to be considered successful.
- Inline Segment Data: For inline segments, the pointer contains the actual binary data rather than information about piece locations.

In past versions, the system used two distinct data structures, referred to as frames and pointers, to handle these particulars. Nevertheless, in the present iteration, these structures have been merged into a solitary, cohesive entity referred to as a pointer. This simplicity improves efficiency by lowering the intricacy of maintaining data locations and information throughout the network, making both the storage and retrieval operations more streamlined.

## 4.9 Metadata Management

Within the Silo network, the metadata storage system has the main role of storing pointers, which are crucial for the identification and administration of data across the decentralized storage network. These pointers serve as references, providing guidance to the network on the location and management of data items.

An uncomplicated method for integrating metadata storage is to let each user to use their desired and reliable database. Users have the option to choose from a range of database systems, including MongoDB, MariaDB, Couchbase, PostgreSQL, SQLite, Cassandra, Spanner, or CockroachDB. Users, particularly those who handle substantial quantities of data, may take use of dependable backup solutions to save their metadata due to this adaptability. Nevertheless, this approach has both benefits and drawbacks.

Challenges of Using Traditional Database Systems:

- Availability: The accessibility of a user's data is contingent upon the presence and functionality of their selected metadata server. Although Cassandra, Spanner, and CockroachDB are very dependable distributed solutions that provide good availability, effectively administering these systems involves significant work. Furthermore, the absence of a single metadata service does not impact the other parts of the Silo network, guaranteeing that the network continues to function even if certain nodes experience periods of inactivity.
- Durability: Data loss may occur if the metadata server has a major failure and there are no adequate backups in place.

- As metadata contains encryption keys, losing it results in the loss of access to the stored data. In order to reduce this risk, it is possible to regularly create backups of metadata inside the Silo network itself, hence minimising the quantity of crucial data that must be kept elsewhere.
- Trust: Users are required to place their faith in the metadata server to maintain the integrity and security of their metadata. This creates a dependence on other systems or services.

Benefits of Allowing User-Controlled Metadata Storage:

- Control: Users maintain full authority over their data. This decentralisation eradicates the risk of a single point of failure and empowers users to choose their metadata storage, therefore achieving a balance between different trade-offs based on their individual preferences. Similar to the decentralized social network Mastodon, this method enables users to operate their own metadata services, guaranteeing independence and durability in the face of centralised breakdowns.
- Simplicity: By using current database technologies, the Silo network may bring a working product to the market more quickly, without the need to spend years constructing Byzantine-fault tolerant consensus systems for information storage. This approach avoids the performance and complexity trade-offs associated with such systems.
- Coordination Avoidance: Users only need to synchronise with other users on their designated node or server. Users with high demand may create their own nodes, which helps to decrease the need for cooperation and improves performance. Users have the option to choose databases that have lower consistency requirements, such as Highly Available Transactions, which helps to reduce the need for coordination among their nodes.

Silo offers a versatile and effective solution for decentralized storage management by enabling users to handle their information using a database of their own. This technique also facilitates the development of future improvements and novel solutions that might further increase the optimization of metadata management inside the network. To get more understanding of our present methodology and upcoming strategies for metadata management, please refer to Appendix A and Section 6.2.

## 4.10 Orbital

The Orbital Node, sometimes referred to as the Orbital, functions as a centralised hub responsible for overseeing metadata inside the Silo network. Users establish accounts on a designated Orbital instance, which serves the purpose of storing file information, overseeing data access rights, monitoring the dependability of storage nodes, restoring data when redundancy drops below acceptable thresholds, and facilitating payments to storage nodes on behalf of the user. An Orbital instance is not restricted to a solitary server; it may be implemented as a group of servers backed by a scalable, reliable database to guarantee optimal availability.

The Silo network utilises a thin-client approach in which the responsibility of managing file location information is assigned to the Orbital, which efficiently oversees data ownership. The configuration allows the Orbital to accommodate a diverse range of client applications, necessitating a substantial amount of infrastructure resources and ensuring a high level of availability, particularly when handling an active file set. Similar to other elements of the Silo network, the Orbital service is created as open-source software, enabling any person or organisation to operate their own Orbital and connect to the network.

The Orbital is primarily built as a resilient and dependable application server that encompasses a secure database solution, such as PostgreSQL, Cassandra, or any other appropriate system selected for managing metadata. Users authenticate themselves into a designated Orbital using their account credentials. The data available inside one Orbital instance is not immediately accessible within another. However, potential future upgrades may provide the ability to export and import data across different Orbitals.

Crucially, the Orbital does not receive data that is not encoded and does not retain the keys used for encryption. The only information it may provide to other entities is metadata, including the presence of a file, its estimated magnitude, and use trends. This design decision guarantees the preservation of client confidentiality and the client's complete authority over data access, while the Orbital assumes the duty of ensuring file availability on the network.

Users have the option to use Orbitals that are operated by third-party entities. Due to their lack of data storage and absence of encryption key access, these Orbitals provide a security paradigm that surpasses that of conventional data centres. The reputation and selection of storage nodes by Orbitals are enhanced by network effects. As the reputation data expands, it gains more value, hence establishing a compelling economic motivation for sharing infrastructure and information inside an Orbital.

Service providers have the option to run public Orbitals, which enable developers to assign confidence to a particular Orbital for data placement on the network. This is similar to how trust is established in a conventional object store, but with less risk involved. Anticipated future upgrades will bring about diverse arrangements and allocations of duties between client applications and Orbitals, enabling the incorporation of differing degrees of reliance.

An Orbital instance comprises several key components:

- A comprehensive node discovery cache (see Section 4.6)
- A metadata database indexed by encrypted paths for each object (see Section 4.9)
- An account management and authorization system (see Section 4.12)
- A storage node reputation, statistics, and auditing system (see Section 4.13)
- A data repair service (see Section 4.14)
- A storage node payment service (see Section 4.16)

Although the current implementation of several Orbitals represents a notable improvement compared to previous versions, it is but a single stage in the continuous process of decentralisation. There are future intentions to decentralise further components of the Silo network.

## 4.11 Data Encryption

The encryption system used in the Silo network ensures data confidentiality and integrity by using authenticated encryption. It supports both AES-GCM and the Salsa20 and Poly1305 combination, which is known as "Secretbox" by NaCl [62]. Authenticated encryption is selected to ensure that any manipulation of the data may be detected, hence ensuring a dependable guarantee of data integrity.

Data encryption is performed in discrete units called blocks, which are usually made up of tiny groups of stripes. It is suggested that each block should have a size of 4KB or less [63]. While each encryption batch inside a segment uses the same encryption key, separate segments may use distinct encryption keys. The nonce for each encryption batch must increase consistently from the preceding batch inside the segment, and if the nonce counter reaches its maximum value, it resets to zero. In order to mitigate the risk of reordering attacks, the starting nonce for each segment is systematically established using the segment number. In situations when many segments are being uploaded simultaneously, such with Amazon S3's multipart upload capability, the initial nonce for each segment is determined by combining the file's initial nonce with its segment number. This encryption method guarantees that the storage nodes are unable to access the content, hence preserving data secrecy. The data owner, who has the encryption key, maintains total control over the data.

Furthermore, the data channels are encrypted to enhance security. Like the BIP32 hierarchical deterministic wallet standard [44], the encryption used here is both hierarchical and deterministic. Each component of the route is encrypted individually. This procedure entails the identification of a confidential value for each segment of the route. The value is created in a recursive manner using an HMAC function that relies on the secret of the preceding segment.

For example, if we have an unencrypted route consisting of components p1, p2,..., pn, and the user selects an initial root secret s0, the next secrets are defined recursively as si=HMAC(si-1,pi). A deterministic derivation of a key K(si) may be obtained from each sis_isi. Each encrypted route component, ei, is generated by encrypting the path component, pi, using the key acquired from the previous path component, si-1. This encryption method enables users to provide access to specified sub-pathways while keeping parent paths and irrelevant paths hidden, hence boosting privacy.

Path encryption is an optional feature that is automatically activated. Nevertheless, when used, it may create complexity in sorted path listings since items are arranged based on their encrypted path names. This sorting algorithm is predictable, meaning that it will always provide the same output for a given input. However, it may not be practical or beneficial when client applications need unencrypted pathways. Users may choose to deactivate route encryption. When path encryption is off, only the user's chosen Orbital can see unencrypted paths. The storage nodes are kept oblivious of the data's path and information.

## 4.12 Access Control and Authorization

Encryption protects the confidentiality and integrity of data by preventing unwanted access and detecting tampering. However, authorization methods are necessary to prevent illegal alterations and regulate access. Only those with the appropriate authority should have the ability to upload, remove, or modify files. On the other hand, it is important to limit the ability of unauthorised users to carry out these actions. Authorization controls both data alterations and access to metadata operations. Users verify their identity via their corresponding Orbital, which determines their privileges for different activities according to their authorization settings.

The first method we use for metadata permission makes use of macaroons [64], which are bearer tokens that provide versatile and detailed access control. Macaroons are very advantageous since they provide decentralized delegation, enabling numerous entities to impose limitations on a token without the need of a single authority to verify such limits. This approach allows us to enforce precise restrictions for giving or denying access, such as restricting activities to specified encrypted routes or providing read-only or append-only access. Every user account is linked to a primary macaroon, and all actions must adhere to the limitations specified in the macaroon's disclaimers.

Our solution offers the flexibility to add expiry dates and revocation tokens to macaroons, allowing users to programmatically revoke access as necessary. This feature is essential for preserving security and guaranteeing that obsolete or hacked tokens do not result in unlawful entry.

Our authorization approach is based on encrypted pathways, since Orbitals only handle encrypted paths and limit Orbital actions. In order to facilitate delegation for certain route prefixes, it is crucial that the distinctions between path components remain discernible even after the process of encryption. This requirement may restrict the functionality or performance for path delimiters that are not the conventional forward slash.

After a Sconnect client is successfully authenticated with an Orbital, the Orbital grants permission and generates signatures for any future activities involving storage nodes. This includes tasks such as allocating bandwidth, as explained in section 4.17. The Sconnect client is required to get legitimate signatures from the Orbital for every transaction involving storage nodes. Each action performed on a storage node requires a distinct Orbital ID and its matching signature. Storage nodes will refuse any actions that do not have the appropriate Orbital ID signature, guaranteeing that one Orbital cannot unintentionally or maliciously impact data controlled by another, unless specifically authorised by the data owner's Orbital.

Our early approach lacks protections to prevent unexpected file deletions or rollbacks initiated by a possibly malicious Orbital. The foundation of our trust model is based on the idea that the user's Orbital operates with integrity, ensuring the proper management and repair of data. If an Orbital is seen untrustworthy, it is improbable that it would carry out dependable data repairs for the customer. Subsequent upgrades may provide stronger methods for identifying and addressing such situations, similar to the mechanisms used in systems like as SUNDR, SiRiUS, or Plutus [65-67], in order to enhance protection against unwanted Orbital activity.

## 4.13 Data Auditing for Integrity

Ensuring the accuracy of data storage and provision by nodes in a decentralized storage network is crucial for sustaining the network's dependability, especially considering that these nodes operate autonomously and may not always be trustworthy. Audits are conducted to verify that nodes are retaining the data they claim to possess and are operating as intended. During the audit procedure, an auditor, usually an Orbital, initiates a challenge to a storage node, which is then required to provide evidence that it has the requested data in its original state.

Merkle tree proofs were used for audits in some distributed storage systems, such as older versions of SILO. This approach entails creating barriers and anticipated reactions throughout the storing process, which serve as evidence of retrievability. The use of a Merkle tree reduces the amount of information needed to hold these challenges and replies because of its efficient hierarchical hashing structure. Nevertheless, Merkle tree-based proofs are classified as "limited schemes," requiring pre-generation of audit difficulties. Storage nodes may take advantage of this restriction by optimising storage capacity by the selective tracking of predicted responses instead of recording all of the actual data.

In order to address the constraints of pre-determined tasks, we have included Reed-Solomon erasure coding to enable more flexible and resilient audits. Reed-Solomon coding eliminates the need for pre-generated obstacles in audits. Instead, they use erasure coding to rebuild data in real-time, allowing for verification without requiring all of the original data bits. This method enables the possibility of conducting random audits, in which a sample of data is chosen at random and examined for correctness.

The audit procedure starts by randomly picking a segment of data. The Orbital subsequently asks the deletion of certain data segments from all relevant storage nodes. The system employs algorithms such as Berlekamp-Welch error correction to examine these shares. If the data that is returned is identical to the anticipated output, then the audit is said to be successful. If any inconsistencies are detected, the node may be marked as non-compliant for its failure to maintain data integrity.

Nodes that fail to reply, offer inaccurate data, or become unresponsive will be subjected to further examination. When a node is momentarily unable to answer, maybe because it is being overloaded with requests, it enters "containment mode." During this state, the Orbital system keeps trying to do the audit on the node until it either succeeds, fails, or is tagged as offline because it has been unavailable for an extended period of time.

The reputation system logs audit findings, including failures, latency measures, throughput, and overall responsiveness. This continuous assessment aids in sustaining a superior level of service and guarantees that only dependable nodes stay operational inside the network. Systematic, indiscriminate audits conducted on all data provide thorough coverage and instil trust in the accuracy of the data, eliminating the need to inspect each byte or file separately.

To get further information on the frequency of audits and the statistical guarantee of data accuracy, please see Section 7.2.

## 4.14 Data Repair Mechanism

In a decentralized storage network, the data's availability is of utmost importance, particularly when storage nodes go down, resulting in the temporary or permanent inaccessibility of the stored bits. In order to ensure the integrity and availability of data, it is essential to rebuild and replace any missing parts if the number of accessible pieces for a segment drops below a preset threshold, represented as m.

When the Orbital detects that a storage node is offline, it designates the data pieces stored on that node as missing. The system then consults the node discovery cache to ascertain the most current state of storage nodes. When a storage node that was recently connected to the network is discovered to be disconnected, the Orbital system does a search in a reverse index located in the metadata database. This search is done to determine all the references linked to the data segments that were stored on that particular node.

If a segment falls below the necessary redundancy level, meaning it has less than m pieces, the Orbital will begin a repair operation. This process entails retrieving the already available fragments of the segment from the remaining nodes and recreating the missing data by using Reed-Solomon erasure coding. After the rebuilding process is finished, fresh components are created and transferred to new storage nodes in order to restore the redundancy of the segment. Once the new parts are uploaded successfully, the metadata pointer is changed to accurately represent the changes. This update guarantees that the data remains easily available and appropriately safeguarded.

Network users have the flexibility to choose their preferred degree of durability by selecting an Orbital. This choice may affect the price and other operational aspects of the network. The chosen degree of durability, together with the findings from continuous audits, guides the selection of Reed-Solomon erasure code parameters for both new and repaired files. It also establishes the thresholds for successful uploads and the need for repairs. To get further information on these computations and user inputs, please see Sections 3.4 and 7.3.

It is necessary to emphasise that the uninterrupted functioning of the Orbital is vital for this healing process. In the case of an Orbital system failure, repair operations will be suspended, potentially leading to data loss as a result of node turnover within the network. This is comparable to the value storage and republishing procedure in Kademlia, where the owner must stay connected to guarantee the availability of data.

Although the audit and repair procedures need a substantial amount of incoming bandwidth because of the extensive data needed for audits and reconstruction, the outgoing bandwidth is very little as only the newly rebuilt components are sent. The uneven distribution of bandwidth utilisation makes hosting providers that give free incoming bandwidth particularly attractive to customers that run Orbitals, as it helps to decrease overall operating expenses related to data upkeep.

### 4.14.1 Minimising Repair Overhead with Piece Hashes

Restoring data in a decentralized storage network is a crucial but demanding task that requires significant resources. It necessitates a significant amount of bandwidth, memory, and computing power, frequently burdening a single storage operator with a substantial workload. In order to enhance network efficiency and save expenses, it is essential to limit the use of resources during the repair procedure.

When it comes to segment repair, the objective is to minimise the bandwidth use required for data recovery. This refers to the act of downloading just the essential amount of components needed to reassemble a section. However, using merely Reed-Solomon erasure coding for mistake correction is inadequate when only a portion of redundant parts is accessible, since it does not intrinsically verify the integrity of individual pieces.

In order to overcome this constraint, we have developed a technique that utilises piece hashes. Every item kept on a storage node is accompanied by a cryptographic hash, which serves as a unique fingerprint for its contents. Furthermore, a validation hash, which serves to authenticate the collection of all piece hashes, is kept in the pointer metadata.

During a repair operation, the piece hashes are first fetched from the storage nodes and then compared to the validation hash in the pointer. This stage guarantees that all components used for reconstruction are accurate and unaltered. By validating the integrity of each individual piece prior to commencing a complete download, the repair process may securely use the minimum number of pieces required to reassemble the data, without the need for further redundancy.

This strategy not only guarantees the correctness of the data but also preserves vital network resources by minimising needless data transmissions. The utilisation of piece hashes greatly improves the effectiveness and dependability of the data repair procedure in the network.

## 4.15 Node Reputation Management

Robust reputation systems are essential in decentralized storage networks to ensure dependability and data integrity of storage nodes. This method facilitates the identification and elimination of untrustworthy or malevolent individuals from the network, hence boosting the overall security and resilience. The reputation system is structured based on four fundamental elements: evidence of work, first evaluation, filtration, and preference.

Every storage node is required to do a proof of work in order to generate its own identity. This first proof serves as a preventive measure against Sybil attacks, in which an attacker may generate several counterfeit identities to disrupt the network. The complexity of this proof is determined by the Orbital operator and may be modified gradually. Nodes that do not fulfil the necessary proof of work are not qualified to store fresh data, guaranteeing that only dedicated participants may join the network.

Storage nodes that are newly introduced are subjected to a thorough evaluation procedure to determine their level of dependability. During this timeframe, nodes undergo testing by being included into a reduced amount of data storage activities, hence avoiding potential risks while collecting performance data. This procedure guarantees that only nodes with demonstrated stability and performance are thoroughly included into the network. After successfully completing the vetting phase, a node becomes qualified for standard data storage

and is issued a certification by the Orbital to certify its status.
The reputation system consistently monitors the behaviour of nodes in order to identify and exclude nodes that are not dependable. Nodes may be excluded due to excessive audit failures, failure to provide data promptly, or inadequate uptime maintenance. Nodes that have been disqualified are removed from the active storage pool, and whatever data they were holding is transferred to nodes that are more dependable. Disqualified nodes are required to undergo the vetting process again in order to restore confidence, hence discouraging any irresponsible or malicious actions.

In order to enhance data storage efficiency, the system incorporates a preference mechanism. Nodes are evaluated according to characteristics such as velocity, dependability, geographical position, and the amount of time they are operational. The purpose of this data is to determine the priority of nodes for future data storage activities, giving preference to nodes with superior performance while still ensuring that all eligible nodes have some level of involvement. This strategy facilitates the equitable distribution of storage duties and safeguards against concentration, so guaranteeing a decentralized and robust network.

At first, each Orbital manages node reputations separately, which means that a node that has been rejected by one Orbital may continue to take part in others. Over time, the reputation system strives to grow increasingly interconnected, providing a thorough assessment of each node's performance across the whole network. The implementation of this comprehensive strategy for reputation management ensures the maintenance of stringent requirements for the dependability of data storage and the security of the network.

## 4.16 Payment Systems

The payment mechanism in the SILO network is specifically intended to streamline transactions among clients, storage nodes, and Orbitals, guaranteeing seamless operations and equitable remuneration for used resources. Customers that store data on the network remunerate the Orbital for handling their data, which thereafter allocates funds to storage nodes according to the storage capacity and bandwidth they provide.

Customers have the option to choose from a range of payment methods, including bitcoin, credit cards, or invoicing, providing them with freedom in how they make payments. Nevertheless, the storage nodes are remunerated with the SILO currency, which is derived from the Larissa blockchain. SILO serves as the storage layer of the Larissa blockchain, enabling a decentralized and safe payment system. This decision utilises the advantages of blockchain technology, such as transparency and security, while being customised to meet the unique requirements of the network.

Conventional decentralized storage systems often depend on fixed contracts for payments, where data is stored for a certain duration, such as 15 or 30 days, until extended. The SILO network deviates from this concept by abstaining from using set contracts for storage periods. Conversely, it presupposes that data will be retained forever unless explicitly stated differently. This strategy is more suitable for situations when indefinite storage is preferred, hence minimising the possibility of unforeseen data loss caused by contract termination.

Storage nodes get compensation for both storing data and providing download bandwidth. Payments are determined by the quantity of data saved and the amount of bandwidth used, usually on a monthly basis. Nevertheless, the storage nodes do not get compensation for the first data upload, hence promoting effective data management and discouraging superfluous data uploads. Nodes that have a continuous record of being up and reliable are chosen, and there is a motivation for storage nodes to be active in the network for long durations. In order to substantiate this claim, Orbitals have the ability to retain a percentage of a node's profits until it has shown its ability to consistently provide dependable service for a certain duration, often six months or more. If a node departs from the network prematurely or fails to reach the required level of dependability, the Orbital has the opportunity to retrieve the monies that were being kept.

Storage nodes are anticipated to rapidly acknowledge delete requests from Orbitals in order to prevent the accumulation of unnecessary data. If a node fails to execute a delete command, it will not be compensated for the storage it is unnecessarily using and may finally remove the data via a trash collection process. This approach guarantees that only operational and trustworthy nodes get compensation, hence enhancing the efficiency and cost-effectiveness of the network.

Orbitals continuously monitor the storage and bandwidth utilisation of each node throughout the month, keeping track of daily records to ensure precise accounting. After each billing cycle, money are collected and allocated to the storage nodes appropriately. In addition, Orbitals generate income by levying fees on customers for a range of services, including auditing, data restoration, and metadata storage. This ensures the long-term viability of the network and covers operating expenses.

## 4.17 Bandwidth Management

A crucial component of our approach is precisely monitoring the use of network capacity between two interconnected entities. In our prior design, we depended on exchange reports to document the amount of bandwidth used during interactions between peers. Both parties would provide reports to a central agency for the purpose of reconciliation at the conclusion of an operation. This strategy functioned seamlessly when both peers reached a consensus on the reported bandwidth use. Nevertheless, the inconsistencies necessitated more scrutiny to uncover any fraudulent conduct, so complicating the procedure and allowing for disagreements.

Our objective with the existing system is to eliminate cheating by directly tackling the problem at the protocol level. In order to do this, we get inspiration from Neuman's Proxy-based authorization and accounting for distributed systems. This accounting system provides a decentralized and delegated approach to precisely quantify resource use.

According to Neuman's paradigm, an account server creates a digitally signed check, also known as a proxy, that grants permission for a particular amount of resource consumption. Within our specific circumstances, this examination is often known as bandwidth allotment. The information provided consists of the account server (Orbital), the payer (Sconnect), the payee (storage node), the maximum permissible use of resources (bandwidth), a distinct identification to avoid duplicate transactions, and a specified date of expiry. The Orbital allocates bandwidth for the Sconnect only if the Sconnect is permitted for the desired activity. The Sconnect subsequently transmits the allocated bandwidth to the storage node at the start of a storage activity. After confirming the signature of the Orbital, the storage node carries out the operation within the allowed bandwidth limit and sends the bandwidth allotment to the Orbital for payment.

Additionally, we include a system influenced by Filecoin's off-chain retrieval market, which involves the transfer of data in tiny increments. This architecture mitigates the possibility of one party engaging in fraudulent behaviour by dividing activities into smaller requests. This technique operates in a manner akin to an optimistic, incremental-release, equitable exchange protocol, where neither participant is subjected to substantial loss in the event of early termination of the protocol.

To manage this efficiently without burdening the Orbital with excessive overhead, we use restricted bandwidth allocations. These limited allocations, similar to macaroons, may be further constrained without eliminating current limits, guaranteeing that only the assigned bandwidth is used. The Sconnect controls these allocations, gradually increasing the limit as the storage node successfully transfers data, ensuring that only the largest allocation received is retained.

For example, if the first bandwidth allocation allows for a maximum of x bytes, the Sconnect begins with a lesser allocation of y bytes, enabling the storage node to authenticate the permission. After verification, the storage node sends the data corresponding to a certain number of bytes, denoted as 'y', and then waits for the next allocation. This method continues iteratively until all x bytes are sent. In the event of an unexpected termination of the operation, the storage node will save the highest allocation it has received and forward it to the Orbital for payment.

This bandwidth management system mainly focuses on monitoring the amount of bandwidth used during storage operations, namely the storage and retrieval of data fragments. However, it does not take into consideration the bandwidth consumed by maintenance traffic or node discovery activities. By including these strong measures, we guarantee precise monitoring of bandwidth and secure transactions, therefore improving the overall dependability and confidence in the network.

## 4.18 Orbital Reputation

Within the SILO, the dependability and credibility of Orbitals are essential for sustaining a resilient and distributed storage system. If an Orbital shows below-average performance, problems with payment, or poor production of demand, storage nodes are motivated to refuse to take data from that Orbital.

Storage nodes have the independence to choose the Orbitals with whom they want to cooperate and have the option to abstain from collaborating with those they deem untrustworthy. If an Orbital fails to fulfil expectations or acts in a manner that undermines trust, storage nodes have the ability to express their lack of faith by ceasing their engagement with that particular Orbital.

In order to optimise this procedure and guarantee a uniform standard of service, storage node operators have the option to automatically place their faith in a carefully selected list of authorised Orbitals that are supplied by the Larissa Network. The selection of these Orbitals is done via a rigorous evaluation process that includes stringent quality controls, operating requirements, and payment service level agreements (SLAs). This technique provides storage node operators with a level of confidence that they are collaborating with trustworthy Orbitals that comply with predetermined operational and financial criteria.

In order for an Orbital operator to be included in the Larissa Network's recommended list, they must adhere to certain operating rules, payment processes, and price structures. Furthermore, it is necessary for them to establish a legal agreement with Larissa Network, guaranteeing that they maintain a superior level of service that is in line with the network's and its members' standards.

SILO promotes transparency and accountability among Orbital operators by introducing a reputation-based system. This creates a favourable environment where storage nodes and Orbitals may both get advantages from a reliable and effective storage network.

## 4.19 Data Cleanup and Garbage Collection

Efficient storage system maintenance in the SILO network relies on the appropriate management of unwanted or old data, also referred to as trash collection. When clients want to relocate, substitute, or erase data, Orbitals (formerly known as Satellites) or client applications operating on behalf of Orbitals will inform storage nodes that the data in issue is no longer needed.

In some setups, the client directly initiates the deletion of messages, and the metadata system guarantees that evidence of deletion is sent to a limited number of storage nodes. This guarantees that as soon as data is removed, all accessible and active storage nodes are instantly informed, hence maintaining the organisation and efficiency of the storage network.

Nevertheless, there are occasions when storage nodes may experience temporary unavailability or offline status, resulting in their failure to receive these delete messages. In such instances, data that is no longer required is referred to as "garbage." Furthermore, in scenarios where the client does not send delete messages, unnecessary data builds up as trash. Crucially, Orbitals only incur costs for data that they anticipate will be kept. This implies that storage nodes containing huge quantities of useless data would get lower earnings compared to those that effectively manage their storage, unless they implement a garbage collection mechanism.

Garbage collection entails a systematic procedure for finding and eliminating superfluous resources. A precise garbage collection system in computing ensures optimum efficiency by collecting all superfluous data without any residual remnants. Conversely, a conservative garbage collection system may intentionally retain certain rubbish in order to optimise efficiency and resource allocation. SILO's first implementation adopts a cautious strategy, guaranteeing that storage nodes get sufficient compensation to cover the expenses related to retaining a minimal quantity of residual waste.

SILO's first release will use a conservative garbage collection mechanism to handle nodes that do not get the initial delete messages. This strategy will be further improved in subsequent releases. At regular intervals, storage nodes will ask their respective Orbital for a customised data format that assists in detecting differences between stored data and the anticipated condition. At its most basic level, this data structure might be a hash of stored keys, enabling efficient identification of data that is no longer necessary. After identifying data that is not synchronised, it is possible to use a different data structure, such as a Bloom filter, to precisely determine which data has to be eliminated.

Orbitals facilitate the removal of unnecessary data by providing customised data structures at regular intervals. This allows storage nodes to clean up trash data based on a pre-set tolerance level, resulting in a more efficient and organised garbage collection procedure. This system guarantees the efficiency of storage nodes, optimises data storage, and ensures that the whole network maintains a high level of speed and dependability.

## 4.20 Sconnect Interface

Within the SILO network, the word "Sconnect" encompasses any program or service that utilises the LibSconnect library to support interactions between Orbitals and storage nodes. Sconnect functions as the user interface for the SILO network and is available in several versions to cater to different use scenarios:

**LibSconnect**: This library serves as the foundation for storing and retrieving data inside the SILO network, including essential features. Developers have the ability to incorporate LibSconnect into their programs, allowing for easy use of the distributed storage features provided by SILO.

**Nexus Gateways**: These gateways serve as intermediary layers, connecting services or applications with the SILO network. Nexus Gateways operate as co-located services that directly interface with storage nodes, hence reducing central bandwidth expenses. In essence, a Nexus Gateway functions as a streamlined service interface built on the foundation of LibSconnect. The first version of this gateway provides an interface that is compatible with Amazon S3. This allows users and programs to store data in the SILO network without the need to manually handle the intricacies of a distributed storage system.

**Sconnect CLI**: The Sconnect Command Line Interface (CLI) is a command-line tool that utilizes LibSconnect to do a range of activities, including file uploading and downloading, bucket creation and removal, and file permission management. The Sconnect CLI is specifically intended to provide a user experience that closely resembles that of well-known Linux/UNIX utilities such as scp or rsync. This design ensures that those who are already acquainted with these environments can easily access and use the Sconnect CLI.

SILO's dedication to openness and community-driven development is shown by the fact that all Sconnect software components, such as storage nodes and Orbitals, are created and made available as open-source software. These components facilitate many use cases and transactions, ranging from basic file storage and retrieval to more intricate processes. They provide users a resilient and adaptable framework for managing their data in a decentralized setting.

# 5 Typical Data Operations

Below are many typical use case examples illustrating various forms of data transfers inside the system.

## 5.1 Upload Process

When a user starts a file upload in the SILO network, the process begins by passing data to an instance of Sconnect, which is the client-side interface for dealing with Orbitals and storage nodes.
The steps involved in the upload process are as follows:

1. Encryption and Data Segmentation:
    - Sconnect chooses an encryption key and an initial nonce for the first data segment and promptly begins encrypting the incoming data using authorised encryption techniques. This guarantees that data is protected from the beginning.
    - Sconnect receives incoming data and temporarily stores it in a buffer. It then analyses the data to decide whether it is a tiny enough segment to be saved directly on the Orbital, or if it is a larger segment that has to be distributed over other storage nodes in the network.
2. Preparing for Storage:
    - If the data is sufficiently substantial to need a distant segment, Sconnect initiates a request to the chosen Orbital to make arrangements for storing the segment. This request contains essential credentials, such as macaroons and identification certificates.
    - Upon receiving this request, the Orbital validates the legitimacy of Sconnect's permission and ensures that they have the cash to cover the upload activity. Prior registration with this particular Orbital is a must for the user.
    - The Orbital system chooses storage nodes by considering factors such as resource availability, durability, performance, geographical proximity, and reputation needs. Sconnect receives a list of specific nodes, including their contact information, bandwidth allotment, and a root piece ID.
3. Data Upload and Erasure Encoding:
    - Sconnect establishes many simultaneous connections to the designated storage nodes, while monitoring the amount of bandwidth used throughout the operation.

- The segment is partitioned into smaller entities known as stripes, which are then encoded using an erasure coding algorithm. The erasure shares are consolidated into segments that are simultaneously communicated to each storage node.
- In order to minimise the impact of long-tail effects and enhance performance, Sconnect employs a technique called over-encoding, which involves producing an excess number of data bits above what is absolutely required for reconstruction. This feature enables the system to terminate slower uploads and give priority to quicker nodes, hence improving the overall speed and efficiency of uploads.
- The process of transferring data continues until either the maximum segment size is reached or the data stream concludes. The hashes of all items are added at the end of each data stream.

4.Data Storage and Payment:

- Every storage node saves data using a distinct piece ID that is linked to the Orbital's ID. Additionally, each node keeps track of the highest limited bandwidth allotment it got throughout the upload process. In the event of an upload being terminated prematurely, the node will destroy all remaining data except for the greatest allocation, which will be retained for the purpose of payment.
- After a successful upload, Sconnect employs a deterministic hierarchical key scheme to encrypt the randomly selected encryption key for the file. It then proceeds to upload a pointer object back to the Orbital. This pointer contains critical metadata, such as:
  - The storage nodes that successfully received the data
  - The encrypted path for the segment
  - The erasure code algorithm used
  - The piece ID and encrypted encryption key
  - The hash of the piece hashes and a signature

5. Finalising the Upload:

- Sconnect iterates through each consecutive segment until the full data stream is uploaded. Each subsequent segment use a distinct encryption key, with the initial nonce increased from the preceding section.
- Once all segments have been uploaded, Sconnect transmits the final information to the Orbital. This metadata includes the total number of segments, their sizes, the beginning nonce, and any extra item metadata.
- Storage nodes regularly report their highest bandwidth allocations to the Orbital in order to get payment, guaranteeing

- that they are remunerated for the storage and bandwidth they have supplied.

The organised upload procedure guarantees data confidentiality, optimal resource use, and fair remuneration for storage nodes, in line with the decentralized principles of the SILO network.

## 5.2 Download Process

When a user starts a download request for an item from the SILO network, the procedure is optimised to be efficient and reduce delay. This is the sequence of events that occurs during the download process:

1. Initial Request and Metadata Handling:
    - The user initiates a download request to Sconnect, indicating the specific item they want to get.
    - Sconnect's objective is to reduce the amount of contacts with the Orbital by proactively seeking information about the item, which includes references to the initial segment. If Sconnect had prior knowledge of the desired byte range, it may immediately notify the Orbital about these particular byte ranges, therefore motivating the Orbital to provide the required segment pointers.

2. Orbital Actions and Response:
    - Upon receiving a segment pointer request, the Orbital takes the following actions:
        - Validation: The Orbital verifies whether Sconnect has the requisite authorizations to access the designated segment and ensures that there are enough money to facilitate the download.
        - 
        - Bandwidth Allocation: Each item that makes up the segment is allocated an unlimited amount of bandwidth, as explained in section 4.17.
        - Node Information Lookup: The Orbital obtains the contact information for the storage nodes specified in the segment pointer.The Orbital obtains the contact details for the storage nodes specified in the segment pointer.

Response to Sconnect: The function provides the desired segment pointer, as well as the bandwidth allocations and contact details of the storage node for each piece.

3. Segment Retrieval:
   - Sconnect evaluates if further segments are required to satisfy the user's data inquiry. When more segments are needed, it retrieves the extra segment pointers from the Orbital.
   - Once Sconnect has obtained all the required segment pointers, it establishes parallel connections to the relevant storage nodes. The system requests the specific ranges of data to be erased from each stored piece, while simultaneously monitoring the amount of bandwidth being used, as described in section 4.17.

4.Optimising Download Performance:
   - Given that not all erasure shares are necessary for data reconstruction, Sconnect optimises this by minimising "long tails", which refers to slower downloads. Sconnect significantly enhances speed and reduces total download time by eliminating slower transfers.
   - Sconnect merges the obtained erasure shares into stripes and decrypts the data, reconstructing the original object for the user.

5. Handling Aborted Downloads:
   - In the event of a download being stopped or aborted, each storage node will save the highest limited bandwidth allotment it has received so far, but will delete all other request-related data.
   - Irrespective of the result, storage nodes then provide their highest limited bandwidth allotment to the relevant Orbital in order to permit payment for the bandwidth used during the download.

This procedure guarantees the effective retrieval of data, optimises the allocation of network resources, and assures appropriate compensation mechanisms for storage nodes, hence boosting the overall functionality and dependability of the SILO network.

## 5.3 Deletion Process

When a user wishes to delete a file on the SILO network, the request is first sent to Sconnect. Sconnect retrieves all segment pointers associated with the file that is scheduled for deletion. If the configuration allows for instant removal, Sconnect works along with Orbital to supervise the process. Orbital confirms that Sconnect has the requisite permissions and sufficient authorization to delete the file. Afterward, it generates signed contracts for each segment, instructing

Sconnect triggers the transmission of deletion commands to the relevant storage nodes. Each node confirms the receipt of the command and indicates whether the file has been deleted or if it has already been wiped. Sconnect collects these verifications and sends them to Orbital. Orbital requires a certain number of confirmations to verify the effective completion of the deletion. After confirming, Orbital removes the information associated with the file, stops charging the user for storage, and ceases payments to storage nodes for the deleted data. Sconnect then generates a report for the user, confirming the successful completion of the operation.

When it is not feasible to delete data directly or when storage nodes are temporarily unavailable, the network relies on a garbage collection process. Storage nodes often request garbage collection updates from Orbital, which helps them find and remove redundant data. This system ensures efficient storage space management in the network, guaranteeing its availability for future use and maintaining optimum performance.

## 5.4 File Relocation

When a user intends to transfer a file to a different place on the SILO network, the procedure starts with Sconnect receiving the request for relocation. Sconnect then retrieves all segment pointers connected with the file from Orbital. Orbital performs checks to verify that Sconnect has the requisite rights to read the file. If the validation is successful, Orbital retrieves and provides the segment information for each pointer.

Sconnect decrypts the segment information by using an encryption key that is obtained from the current file location. Subsequently, it computes the updated destination route and applies a fresh encryption key that is tailored to the new location to encrypt the metadata once again. Sconnect initiates a request to Orbital to update the metadata by include the revised segment pointers for the new route and eliminating the previous pointers. This request is performed as an indivisible compare-and-swap operation, guaranteeing uniformity and avoiding conflicts while carrying out the relocation procedure.

Orbital does a conclusive validation to verify that Sconnect has the necessary authorization to carry out these modifications and that the content of the file has remained unaltered since the commencement of the transfer operation. Once all tests are successfully completed, Orbital proceeds to perform the relocation and updates the file

information appropriately. It is crucial to emphasise that this procedure just entails updating the metadata, without altering the actual data on the storage nodes or sending any extra requests to the storage nodes.
Efficient file relocation may not be provided in the first release of the network due to the intricate nature of guaranteeing atomicity and consistency during pointer modifications. Subsequent versions will strive to enhance this feature in order to facilitate smooth file management operations.

## 5.5 File Listing

To begin the process of listing files on the SILO network, the user must first send a request to Sconnect to get a page containing the desired objects. Sconnect receives this request and converts the unencrypted file paths into encrypted paths in order to provide security and privacy. After encrypting the pathways, Sconnect initiates a request to Orbital for the encrypted paths' associated page.

Orbital verifies the request by confirming that Sconnect has the requisite rights to view the files. After the validation is successfully completed, Orbital sends the specified list of encrypted pathways back to Sconnect. Ultimately, Sconnect employs decryption to reveal the initial file paths to the user, thus concluding the process of listing files. This solution guarantees both the security and efficiency of navigating through the user's data on the network.

## 5.6 Audit

For the purpose of preserving data integrity, every Orbital is responsible for maintaining a roster of segment stripes that need auditing on the network. The process of populating this list involves choosing stripes from storage nodes that have had less recent audits, thereby guaranteeing an equitable distribution of checks. The Orbital algorithm randomly chooses a stripe of data from a storage node, which requires obtaining data from other nodes in the erasure-coded group as well. This randomization ensures that data checks are equally spread across the network.

The Orbital then systematically processes the audit list and records any discrepancies:

- The Orbital does a comprehensive data retrieval operation for each stripe audit, excluding any nodes that are presently marked for containment. Unlike typical download operations, these audits are not limited by duration, enabling the Orbital to wait for node answers for an extended period.

- After collecting the data, the Orbital examines the erasure shares to identify any discrepancies. If a storage node consistently provides incorrect data, it will be disqualified, resulting in its exclusion from future data storage selection and cessation of payment reception.
- When a storage node becomes unresponsive, the Orbital system generates a hash of the anticipated audit outcome and saves it. This action puts the node in a condition where it is required to react to many audit efforts until it either successfully passes or is disqualified.
- The Orbital requests the whole data piece and compares it with the anticipated hash to detect any inconsistencies in nodes' answers. This guarantees the preservation of the network's integrity and the presence of only trustworthy nodes inside the network.

This auditing process ensures that the network continuously verifies data accuracy and identifies unreliable storage nodes, maintaining a high standard of trust and reliability.

## 5.7 Data Repair

The network data recovery procedure consists of two essential stages: identifying files that are susceptible to data loss and then restoring those files. The detection phase guarantees the continued accessibility of all stored data by examining the state of storage nodes and the files they contain.

- Detection Phase: The Orbital conducts periodical pings to all storage nodes in order to assess their state, either as part of normal audits or during node discovery activities. When a storage node does not reply, it is designated as offline. In addition, nodes that do not pass audits are labelled as untrustworthy. The Orbital then analyses the data points linked to these malfunctioning or unreliable nodes. When the count of reliable, connected nodes storing a certain data segment drops below a predetermined threshold, that segment is marked for restoration.
- Repair Phase: A worker process manages repair jobs by retrieving flagged segment pointers from a repair queue. The worker downloads sufficient components for each damaged section to fully rebuild it, guaranteeing the presence of all essential data for precise restoration. Subsequently, the worker verifies the authenticity of these pieces by comparing them to the corresponding hashes stored alongside them. Any inaccurate pieces are discarded, and the source nodes from which these pieces originated are identified as having passed audits. After

gathering a enough quantity of valid parts, the worker restores the missing pieces. Additional storage nodes are chosen to store these regenerated fragments, and the new fragments are uploaded as part of a standard data upload process. Ultimately, the metadata pointer for the segment is modified to accurately represent these changes.

This repair system guarantees the consistency and accessibility of data across the network by constantly monitoring and resolving any data loss, ensuring strong and dependable storage.

## 5.8 Compensation Process

The compensation procedure in the network guarantees precise remuneration for the contributions made by storage nodes. This is the operational process:

At the beginning, the Orbital establishes a predetermined time period, usually lasting one day, to calculate payments for data that is held on the network. This period is only used for accounting reasons, while the actual disbursement of funds occurs according to a separate timetable. During every roll-up period, the Orbital examines all data it thinks is held on each storage node and computes the payment due depending on the time of storage and the rates established for data at rest.

Aside from storage fees, the Orbital system also handles bandwidth use statistics that are regularly sent by storage nodes. These reports provide a comprehensive breakdown of the amount of bandwidth used for data transfers, which is included into the computation of the payment. After gathering all pertinent data about storage and bandwidth utilisation, the Orbital calculates the exact amount owing to each storage node.

During the payment issuance process, the Orbital computes the combined amount owed for both storage and bandwidth. Afterwards, the monies obtained are sent to the wallet addresses given by the storage nodes, guaranteeing that they are fairly and effectively paid for their services in a clear and efficient way.

# 6 Upcoming Enhancements

We are always working on improving our network, with several changes and new features scheduled for next releases. This section delineates certain crucial domains in which we strive to enhance and broaden our existing execution to more effectively cater to the ever-changing requirements of our customers and the distributed storage ecosystem.

## 6.1 Scaling for High Demand Files

Occasionally, certain files on our network may see an unforeseen increase in popularity, leading to a substantial rise in access requests. Although storage node operators may initially see advantages from the higher bandwidth utilisation, the demand might rapidly surpass the existing bandwidth capacity, requiring the implementation of a dynamic scaling strategy.

Since orbitals have the authority to supervise and permit all access to file pieces, they are able to efficiently monitor and regulate access to files that are in high demand. When the demand surpasses the current capacity, the orbital may temporarily limit access, enhance the file's redundancy by spreading it over more storage nodes, and then restore access to provide a smoother delivery.

Reed-Solomon erasure coding has a remarkable property that allows for scalability. With a (k,n) encoding technique, the data may be reconstructed using any k components. This feature enables easy scalability by producing supplementary components without modifying the existing ones. For instance, if a file is first encoded using a (20, 40) scheme and there is a need to increase the redundancy by two-fold, the orbital may generate additional fragments to transition to a (20, 80) scheme. By distributing the file among several storage nodes, the load can be balanced and the additional demand may be accommodated.

Although the increase in redundancy results in greater storage expenses owing to the inclusion of more data fragments, it offers substantial advantages. This technique effectively handles enormous volumes of traffic and improves the global distribution of data, resulting in superior performance and dependability for delivering information in different places.

## 6.2 Enhancing Metadata Management and User Autonomy

During the initial phase of our implementation, the orbital operator is primarily responsible for maintaining the quality of service. This encompasses the provision of regular backups, timely payments, durability, and high availability. Users are likely to favor a more autonomous approach to managing their data and metadata, and they will likely seek to reduce their dependence on orbital operators over time. This may entail the elimination of the burdensome process of manual downloads and uploads, thereby enabling more seamless data transfer between various orbitals.

We intend to implement a metadata import/export system in the near future. This system will enable users to autonomously back up their metadata and simplify the process of transferring data between orbitals. The objective of this preliminary phase is to grant users more autonomy over their data and to improve their capacity to transfer data between various platforms as required.
In the medium term, our objective is to further optimise this process by automating the backup process and reducing the size of metadata exports. We intend to create a system that directly backs up metadata to the network on a regular basis, thereby ensuring that the process is as seamless as feasible. This would reduce the need for user intervention and establish a more robust, automatic safeguard against data loss.

Our long-term objective is to eliminate the necessity for centralised orbital control over metadata. This would ideally entail the implementation of a Byzantine-fault tolerant consensus algorithm to manage metadata in a decentralized manner. The difficulty lies in the delicate balance between the necessity to prevent excessive coordination and the desire to ensure that the platform maintains high performance standards that are similar to those of traditional cloud storage services. We are dedicated to the continuous pursuit of research and development in this field in order to investigate viable solutions that are consistent with these objectives.

This direction is consistent with our long-term strategy to promote a genuinely decentralized storage environment by reducing central points of control and enhancing user independence. To learn more about our coordination avoidance strategy and our decision not to immediately prioritize Byzantine fault tolerance, please refer to Section 2.10 and Appendix A.

# 7 Detailed Calculations and Analysis

## 7.1 Estimating Object Repair Costs

One of the main challenges in maintaining a decentralized storage network is selecting the right parameters to balance the expansion factor and repair bandwidth while ensuring a high level of data durability. Our goal is to optimise these factors, keeping repair costs and bandwidth usage minimal without compromising data integrity.

We can draw from existing research, such as the study titled "Peer-to-Peer Storage Systems: A Practical Guideline to be Lazy" [34], which provides valuable insights and formulas for understanding the trade-offs involved. The framework developed from this research helps in determining network durability and repair bandwidth based on Reed-Solomon encoding parameters, average node lifetime, and reconstruction rates.

The key variables and their descriptions are as follows:

- MTTF: Mean time to failure.
- α: Failure rate, which is the inverse of the mean time to failure (1/MTTF).
- MRT: Mean reconstruction time.
- γ: Reconstruction rate, which is the inverse of the mean reconstruction time (1/MRT).
- D: Total data stored in bytes across the network.
- n: The total number of pieces each segment is divided into (Reed-Solomon encoding).
- k: The number of pieces required to rebuild a segment (Reed-Solomon encoding).
- m: Repair threshold indicating the minimum number of pieces that must be available to avoid repair.
- LR: Loss rate, indicating the likelihood of data loss.
- 1-LR: Durability, representing the probability of data remaining intact.
- ED: Expansion factor, showing the storage overhead required.
- BR: Repair bandwidth ratio, indicating the fraction of data that must be repaired over time.
- BWR: Total repair bandwidth, reflecting the total bandwidth needed for repair operations.

The formulas governing these parameters are:

$$LR = \frac{1}{(m+1)\,ln(n/m)}\;\frac{m!}{m!\,(k-1)!}\left(\frac{\alpha}{\gamma}\right)^{m-k+2}$$

$$ED = \frac{n}{k}$$

$$BR = \frac{\alpha(n-m+k)}{k ln(n/m)}$$

$$BWR = D\cdot BR$$

These equations highlight that while repair bandwidth is linearly affected by node churn, data durability is exponentially sensitive to increased churn. This necessitates maintaining highly stable nodes with longer lifespans to achieve desired durability levels. By understanding these relationships, we can make informed decisions about the storage network's configuration to balance performance, cost, and reliability effectively.

This section provides an overview of the calculations needed to understand the costs associated with repairing data in a decentralized storage network, emphasising the importance of stability and efficient management of resources.

### 7.1.1 Impact of Bandwidth on Storage Capacity

The repair procedure has a dual impact: it affects both the bandwidth use of storage nodes and restricts the amount of store space they can provide. Let us examine a storage node that has a storage capacity of 1 terabyte and a maximum monthly bandwidth limit of 500 gigabytes. According to our approach, if a node is projected to repair 50% of its stored data on a monthly basis and each stored file is anticipated to be viewed at least once, then the node may effectively store around 333 GB of data. This is due to the fact that the total amount of data sent and the repair tasks performed must not surpass the bandwidth restriction established for the node.

As the frequency of repairs increases, the amount of accessible storage space decreases, particularly for nodes that often handle stored data. The precise rate of paid bandwidth consumption will be contingent upon the specific nature of the data stored and the level of demand for access. In order to achieve the best possible storage efficiency while staying within the limits of available bandwidth, it is crucial to consistently monitor these ratios and make necessary adjustments to the amount of useable storage space as the network conditions change.

This section examines the impact of bandwidth limitations on the storage node's ability to optimize its storage capacity while simultaneously managing data serving and repair tasks.

## 7.2 Evaluating the Risk of False Positives in Audits

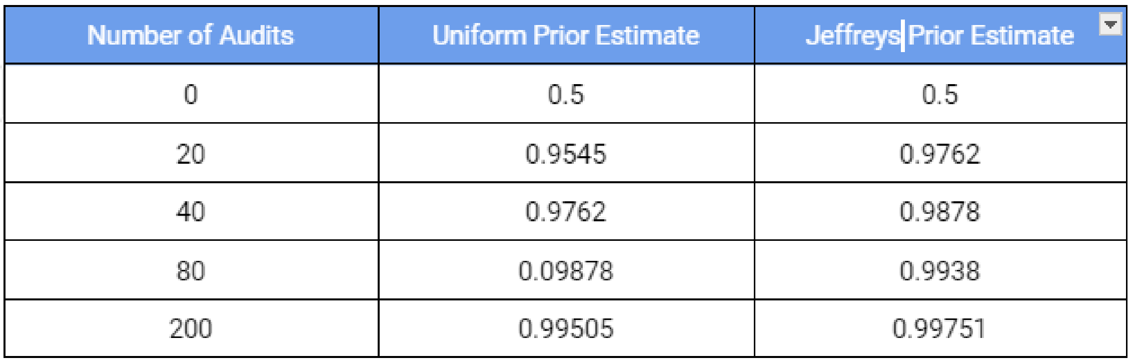
To assess the likelihood that a storage node consistently maintains its stored data correctly, we employ a Bayesian method. This approach helps us estimate the probability that a node will continue to pass audits based on its past performance. The key question we address is: How does the outcome of consecutive audits influence our confidence in a node's reliability?
We model the audit process as a binomial random variable, where the success rate ppp is an unknown parameter within the range [0, 1]. Each audit can be considered an independent Bernoulli trial. The beta distribution, βa,b serves as the conjugate prior for the binomial distribution in this context, meaning that the posterior distribution also follows a beta distribution after observing some data. The Bayesian estimate of the probability of audit success, given as

$$P = \frac{a+x}{a+b+n},$$

depends on prior parameters a and b, and the number of successes x in n audits.

We focus on two widely used priors: the Uniform prior, β1, 1 , which assumes all outcomes are equally likely, and Jeffrey's prior, β(0.5,0.5), which presumes that the success probability is likely to be near 0 or 1. These priors help us to initialise our confidence levels differently based on how we perceive the initial uncertainty.

Here's how the audit success estimates change based on these priors:



| Number of Audits | Uniform Prior Estimate | Jeffreys Prior Estimate |
|---|---|---|
| 0 | 0.5 | 0.5 |
| 20 | 0.9545 | 0.9762 |
| 40 | 0.9762 | 0.9878 |
| 80 | 0.09878 | 0.9938 |
| 200 | 0.99505 | 0.99751 |

As shown in the table, starting with no information (zero audits), both priors yield an initial probability of success at 0.5. As audits accumulate and are passed successfully, the estimated probability rapidly approaches certainty, surpassing 99% with as few as 80 successful audits under Jeffrey's prior. This demonstrates how the Bayesian framework can quickly enhance our confidence in a node's reliability, assuming it consistently passes audits.

## 7.3 Configuring Erasure Coding Parameters

When storing erasure-coded segments on a decentralized network, it is crucial to evaluate piece loss from multiple perspectives to optimise data durability and efficiency.

### 7.3.1 Evaluating Direct Piece Loss

Direct piece loss considers how erasure-coded pieces decrease over time at a loss rate of $0 < p < 1$. Starting with n pieces, the decay follows an exponential pattern, $n(1-p)^t$, where *t* represents time. To manage the repair process, a rebuild threshold *m* is set, indicating when a segment should be rebuilt. The time *t* for pieces to decay from *n* to less than *m* can be calculated using $n(1-p\div a)^{at}=m$ . Solving this for *t*, we find $t > \frac{lm(\frac{m}{n})}{aln(1-\frac{p}{a})}$ .

This formula helps determine the expected lifespan of a segment between repairs, given parameters n, m, a, and p.

### 7.3.2 Assessing Indirect Piece Loss

Indirect piece loss is modelled by considering a fixed rate of nodes that leave the network each month, regardless of whether they hold pieces of a particular segment. To calculate the probability that a certain number of these departing nodes were storing pieces of the segment, the hypergeometric probability distribution is used. If c nodes leave out of a total of C nodes each month, and n nodes were storing pieces of the segment, the probability that d of the departing nodes held these pieces is given by the distribution:

$$p(x=d)=\frac{\binom{n}{d}\binom{C-n}{c-d}}{\binom{C}{c}}$$

To find out how long it takes for the number of pieces to drop below a certain threshold mmm, the model iteratively reduces the number of pieces based on the mean reduction per iteration. If multiple checks per month are assumed, the model adjusts for a reduced churn rate per check, which influences the number of expected rebuilds per month for any segment.

### 7.3.3 Simulations for Modelling Indirect Piece Loss

In order to get a deeper comprehension of the consequences of losing pieces in a decentralized storage network that utilises Reed-Solomon encoding, we conduct numerical simulations using different setups. The objective is to provide decision tables that forecast the average rates at which segments are reconstructed in the most unfavourable situations. The simulations analyse several Reed-Solomon parameters, including the total number of pieces (n), the minimum number of pieces necessary for reconstruction (k), and the repair threshold (m).

We model the occurrence of piece loss at a fixed monthly rate, assuming that pieces are lost as a result of variables such as node turnover or data corruption. In order to determine the average number of rebuilds each month, we divide a segment into n fragments, distribute them randomly around the network, and then simulate node failures. During each simulation, a certain number of nodes are chosen to fail based on a predetermined loss rate. This procedure is repeated until the number of surviving nodes drops below the repair threshold, denoted as m.

The simulation iterates this procedure many times every month, adjusting the rate of component loss based on the monthly number of inspections conducted. We document the frequency of rebuilds that take place over each simulated two-year period, with particular emphasis on the 99th percentile of the distribution to assure reliability and stability. By calculating the average of these values across 1,000 repetitions, we can determine the monthly mean rebuild rate.

Showcasing different combinations of parameters and their impact on repair bandwidth and durability. This helps in identifying the optimal configuration for maintaining durability while minimising repair costs.

| MTTF (months) | k | n | m | Repair Bandwidth Ratio | Durability (# nines) |
|---|---|---|---|---|---|
| 1 | 20 | 40 | 35 | 9.36 | 0.99999999 |
| 6 | 20 | 40 | 30 | 0.87 | 0.99999999999999999 |
| 12 | 20 | 40 | 25 | 0.31 | 0.9999999999999 |
| 1 | 30 | 60 | 35 | 3.40 | 0.9999 |
| 6 | 30 | 70 | 40 | 0.60 | 0.99999999999999999 |
| 12 | 30 | 80 | 45 | 0.31 | 0.999999999999999999999999 |
| 1 | 40 | 80 | 60 | 5.21 | 0.9999 |
| 6 | 40 | 120 | 50 | 0.52 | 0.99999999999999 |
| 12 | 40 | 120 | 45 | 0.24 | 0.99999999999 |

This table demonstrates the relationship between Mean Time To Failure (MTTF), Reed-Solomon parameters, repair bandwidth, and durability, offering insights into optimal configurations for network resilience and efficiency.

### 7.3.4 Summary

To summarise, the selection of erasure coding settings should be in accordance with the particular network characteristics and performance objectives. Various networks may need distinct Reed-Solomon designs in order to achieve an optimal balance between durability and maintenance costs. Typically, when the m/n ratio approaches 1, the frequency of repairs rises, leading to increased bandwidth use. Although a higher m/n ratio may improve the longevity of files, it also results in more regular maintenance and greater use of bandwidth.

To summarise, the selection of erasure coding settings should be in accordance with the particular network characteristics and performance objectives. Various networks may need distinct Reed-Solomon designs in order to achieve an optimal balance between durability and maintenance costs. Typically, when the m/n ratio approaches 1, the frequency of repairs rises, leading to increased bandwidth use. Although a higher m/n ratio may improve the longevity of files, it also results in more regular maintenance and greater use of bandwidth.

In order to minimise the frequency of repairs and enhance longevity, it is advantageous to maximise the total number of components, denoted as n. This technique enables a decrease in the value of m (repair threshold), resulting in a reduction in the frequency of repairs and, subsequently, a decrease in bandwidth utilisation. In essence, increasing the value of n enhances the network's ability to withstand node failures since it prolongs the time required to accumulate enough losses that need a repair.

For example, let's suppose a network that has an average duration before it fails of six months. Comparing a (20, 40) Reed-Solomon encoding scheme to a (30, 80) strategy, if we set the repair threshold to be k+10 in both instances, the (30, 80) scheme has a lower repair bandwidth ratio (0.60) compared to the (20, 40) scheme (0.87). The reason for raising n is to enhance data storage reliability and minimise the need for repairs, despite the fact that both designs initiate repairs when the number of components decreases to k+10. Therefore, the process of choosing the most suitable parameters involves taking into account the requirements for durability as well as the limitations on repair capacity, all within the context of the specific network circumstances.

## A Distributed Consensus Mechanisms

To clarify why Byzantine distributed consensus isn't the focus of our current efforts, it’s useful to explore the history of distributed consensus systems.

## A1 Non-Byzantine Distributed Consensus

Initially, data storage systems in computers were limited to a single machine, which carried a substantial risk of losing data or experiencing downtime if that unit failed. As a result, researchers devised techniques that enable clusters of computers to collaboratively handle data, improving system availability, boosting data processing speed, and dispersing computing tasks. This expedition has been arduous yet has yielded pioneering innovations.

An essential hurdle in reaching agreement across several computers is the possibility of message loss, which is well shown in the "Two Generals' Problem." In this problem, two sides must come to a consensus despite the presence of unreliable communication. The issue highlights the intrinsic challenge of attaining complete consensus with a limited amount of messages, prompting developers to create systems that handle uncertainty by balancing consistency and availability.

The CAP theorem concisely states that a distributed system can provide just two of the following properties: consistency, availability, and partition tolerance. Due to the inevitability of network failures, systems are required to prioritise partition tolerance, which results in a trade-off between consistency and availability. Depending on the architectural design of a system, some prioritise consistency, ensuring that every read action mirrors the most recent write operation. On the other hand, some prioritise availability, allowing the system to remain operational even if data consistency cannot be ensured.

Linearizability, the most robust consistency model, has been a primary objective for distributed systems since it allows for the development of dependable distributed locks and other coordination mechanisms. Initial efforts to establish linearizable consensus resulted in the development of two-phase and three-phase commit protocols. However, these protocols were later shown to be inadequate in ensuring consistency when faced with message loss. As a result, more resilient algorithms such as Viewstamped Replication and Paxos were developed.

Paxos, although being widely used in distributed consensus algorithms, is renowned for its intricate nature, which has led to the development of more straightforward alternatives such as Raft. Currently, Paxos, Raft, and other consensus protocols like as Viewstamped Replication and Chain Replication are extensively used in large-scale distributed systems.

These protocols serve as the foundation for many of the highly resilient data storage solutions utilised today.
These advancements demonstrate the intricate progression of non-Byzantine distributed consensus, a crucial field of study and innovation that has influenced the field of distributed computing.

## A2 Byzantine Fault-Tolerant Consensus

The architecture of our system assumes that the majority of nodes will operate in a rational manner, some nodes may act maliciously (Byzantine), and very few, if any, would act totally altruistically. The previously stated consensus algorithms operate under the assumption that all participating nodes behave altruistically, which is not appropriate in scenarios where nodes may exhibit hostile or illogical behaviour. These conventional consensus methods have had a significant impact on many applications that need dependable fault-tolerant storage. Nevertheless, attaining distributed agreement in the presence of Byzantine defects has proved much more difficult.

The Byzantine fault-tolerant (BFT) consensus issue has received significant attention in research, particularly since the emergence of Bitcoin and its blockchain system. This region is undergoing continuous development and has yielded numerous algorithms, such as PBFT (Practical Byzantine Fault Tolerance), Q/U (Query/Update), FaB (Fast Byzantine), Zyzzyva, RBFT (Redundant Byzantine Fault Tolerance), Tangaroa, Tendermint, Aliph, Hashgraph, HoneybadgerBFT, Algorand, Casper, Tangle, Avalanche, PARSEC, and others.

Byzantine Fault Tolerant (BFT) algorithms bring about further intricacies and compromises that are not necessary in non-Byzantine algorithms. These complexities are aimed at handling possibly malevolent or uncooperative nodes. For example, PBFT incurs substantial network overhead since each client has to interact with a majority of nodes, and each node must answer individually. Bitcoin restricts the number of transactions it can process by modifying the complexity of its proof-of-work method in order to ensure security. Several protocols that emerged after Bitcoin similarly mandate that all nodes retain a comprehensive record of all modifications to the system's state in order to guarantee its integrity and deter fraudulent activities.

The continued development of Byzantine fault-tolerant consensus algorithms aims to address the basic difficulty of obtaining agreement in contexts where members may be untrustworthy. Each new method offers distinct answers to this challenge. Research in this field is ongoing as systems adapt and explore novel approaches to address faults and ensure uniformity in decentralized networks.

## A3 Rationale for Avoiding Byzantine Fault Tolerance

Due to the present limits of available solutions, we have decided not to include Byzantine fault-tolerant consensus methods into our system. Although algorithms such as Flexible Paxos provide enhancements compared to classic Paxos by minimising the need for coordination in stable situations, they are not appropriate for contexts where nodes may exhibit malicious behaviour. In addition, distributed ledger technologies and tangle-like techniques, which aim to resolve Byzantine faults, often experience significant global coordination overhead and struggle to quickly remove unnecessary past data.

We are quite excited about the creation of a resilient and expandable Byzantine fault-tolerant system that can efficiently manage both performance and security. Nevertheless, until a definitive and effective answer arises, we have opted to minimise our vulnerability to these problems by completely ignoring them in our present design. By adopting this strategy, we may prioritise other elements of system efficiency and security without dealing with the additional burden and intricacies involved with Byzantine fault tolerance.

## B Potential Attack Vectors

Within each distributed system, there are several ways in which attacks might occur, presenting substantial dangers. While many aspects are shared throughout distributed systems, others are specifically applicable to decentralized storage networks.

Identity Hijacking Attacks: Spartacus assaults, sometimes referred to as identity hijacking, happen when a node replicates the node ID of another node in order to impersonate it. This enables the assailant to intercept or modify data that is meant for the originating node. By using node IDs as public key hashes and enforcing message signatures, this attack may be successfully mitigated. This is because the attacker would lack the requisite private key to sign messages and establish authentication inside the network.

Multiple Identity Attacks: Sybil attacks are characterised by the creation of several counterfeit nodes with the intention of disrupting network operations, often by the hijacking or discarding of communications. Our method mitigates the danger of such assaults by using proof-of-work identity creation. In addition, our reputation system has a screening process for new nodes, thereby preventing a sudden influx of new nodes that may have harmful intentions from immediately obtaining access to important data.

Network Isolation Attacks: Eclipse attacks are designed to isolate a certain node or set of nodes, thereby limiting their connectivity to just malevolent nodes. Our network employs public key hashes and signatures to safeguard against man-in-the-middle attacks, which may be difficult to identify. As the network size increases, it becomes more difficult for an attacker to gain complete control over the connections of any one node.

Coordinated Node Attacks: Honest Geppetto assaults include an assailant who controls many nodes that seem to be separate and independent. These nodes gather data and build up a reputation gradually. Ultimately, the assailant utilises these nodes to deliberately disrupt the network, either by withholding data or abruptly disconnecting. Our protection technique includes examining the behavioural patterns of nodes in order to detect and mitigate potential hazards. Additionally, we distribute data among a variety of unconnected nodes.

Extortion-Based Attacks: Hostage byte assaults are a kind of cyber attack where some nodes intentionally withhold data in order to extract more money. Reed-Solomon encoding is used in our network to alleviate this issue by enabling data reconstruction from alternative nodes. To enhance the security against such assaults, we may minimise their impact by distributing the components over several nodes and minimising centralization.

Cheating and Malicious Behaviour: There is a possibility that dishonest storage nodes or orbitals may try to exploit bandwidth allotment or provide false information about their activity. Our solution mitigates these dangers and preserves confidence among participants by mandating cryptographic signatures for bandwidth transactions and implementing a rigorous screening procedure for new nodes.
Compromised Nodes: Untrustworthy storage nodes or orbitals may provide data to unauthorised requestors, hence compromising data security. Nevertheless, the use of strong encryption on the client-side guarantees that data privacy remains uncompromised, even in such circumstances.

Audit Manipulation Attacks: Conventional techniques of verifying Merkle proofs are susceptible to manipulation of pre-generated responses. In order to address this issue, our network use random stripe requests and the Berlekamp-Welch method to verify the integrity of data, hence increasing the difficulty for rogue nodes to successfully pass audits without really storing the data.

Our objective is to establish a robust and reliable distributed storage network by adopting these security measures. This network will be capable of withstanding a range of possible assaults, while also guaranteeing the integrity and availability of data.

## C Key Advantages for Users

The SILO network is specifically engineered to provide customers with superior levels of security, availability, performance, and cost-effectiveness as compared to conventional on-premise or centralised cloud storage systems. This appendix examines the reasons why our decentralized method offers substantial advantages in many scenarios.

**Enhanced Security:**
Our technology guarantees robust security by using client-side data encryption prior to its transmission over the network. The files are fragmented and then spread across several autonomously run storage nodes. Using a standard 20/40 Reed-Solomon setup, a file is distributed among 40 distinct disks in a worldwide network, making it very difficult for any malicious individual to identify and compromise all the essential components required to reconstruct the file. Even if a malicious individual manages to identify and get access to all 40 disks, they would still need the ability to decipher the 256-bit AES encryption. However, only the end user has the capability to do this decryption. The decentralized and encrypted technique effectively establishes significant obstacles against illegal access.

**Superior Availability:**
Decentralized storage provides a clear benefit in terms of availability. Contrary to centralised cloud providers, which might be susceptible to widespread interruptions caused by natural catastrophes, power failures, or assaults, every node in a decentralized network operates autonomously. This autonomy greatly diminishes the probability that a malfunction in one section of the network may impact other nodes. The decentralized architecture guarantees the availability of data even if many nodes fail, providing resilience against correlated failures that may occur in a single data centre setting.

**Improved Performance:**
The SILO network utilises parallelism to improve performance in applications that involve a high volume of reading. The system minimises latency by deploying storage nodes in proximity to users at the network edge, especially for users located distant from centralised data centres. By using erasure coding, our system ensures that download rates are not hindered by slower nodes. Instead, the network adapts in real-time to maintain maximum performance. The SILO network's versatility enables it to achieve accelerated download and

streaming rates, while avoiding the normally expensive nature of content delivery networks (CDNs).

**Cost-Effective Storage:**
The cost of cloud storage from conventional providers has not matched the exponential growth in global data output. This is partially attributed to the substantial financial and operational costs necessary to sustain centralised data centres. On the other hand, the SILO network has an advantage since the storage node operators may profit from the very cheap additional expenses. These operators usually use their current gear that has idle capacity. These operators incur little extra expenditures, enabling them to pass on cost reductions to end consumers. Furthermore, the network's decentralized structure avoids the concentration of market power, guaranteeing fair competition and enabling the provision of storage solutions at a much lower price compared to conventional cloud services.

The decentralized method not only creates economic incentives for all players but also guarantees that storage services are efficient and scalable, providing customers with an appealing alternative to standard storage solutions.