Fleek Network: A Decentralized Content and Application Delivery Network

Version: 0.0.3

Mahmoud A. Shehata
Fleek
mahmoud@fleek.network

***Abstract*—Content Delivery Networks (CDNs) have seen a high rise in popularity due to the low latency and high transfer speeds at which they can serve content. Traditional approaches typically involve infrastructure-based delivery, wherein users request their content from third-party servers. These CDNs are traditionally permissioned, trust-based, and subject to the control of dominating parties who can censor or limit content. This is incompatible with making Web 3 applications decentralized and end-to-end, which rely on Web 2 infrastructure to provide their services to Web 3.0 users. Since decentralized storage is a reality, a decentralized content delivery network is the missing infrastructure link and the layer that will fully tie all necessary components together to decentralize Web 3.0 applications. This paper gives a technical outlook of the Fleek Network, a model for P2P content delivery for Web 3.0. They are built using proven blockchain infrastructure and P2P protocols with specialized enhancements focused on content delivery.**

***Index Terms*—trustless, CDN, IPFS, decentralized CDN, content routing, content addressing, web3**

1. Background

Fleek Labs has worked for years to operationalize decentralized file storage and delivery through its *Fleek.co* platform and related projects. The Fleek Network results from many years of work and research, with extensive expertise in peer-to-peer protocols, IPFS, security, cryptography, and creating products within various blockchain ecosystems (Ethereum, IPFS, Filecoin, Arweave, and StarkNet). Fleek already powers 40,000+ sites/apps and serves users with over 600 TB of monthly data.

2. Introduction

The existing content delivery model is centralized, shown in Figure 1; third parties can and have compromised services. Fleek Network was designed to change this by using a decentralized network to ensure that the network is trustless, censorship-resistant, and resilient while maintaining scalability, high availability, and fault tolerance. Fleek Network's objective is to create a decentralized content delivery network so that users can serve their content and applications in a censorship-resistant, trustless manner, with no central authority controlling it. It is designed to be scalable, highly available, and fault-tolerant, replicating content across nodes to ensure censorship resistance and enable trustless content retrieval. It is peer-to-peer (P2P) and agnostic, offering a low-cost service with blockchain and consensus algorithms. Fleek Network will add a caching and performant delivery layer on top of independent decentralized storage protocols, allowing users to interact with the network directly rather than only through HTTPS. With no intermediary, the platform offers users an alternate route for accessing online content without censorship.

Fleek Network is a permissionless edge network with no central authority controlling it, and anyone is free to join. Given the reasonable requirements to run nodes, it scales with demand and accelerates any data presented as content-addressable. The network is agnostic to underlying storage providers, utilizing a two-pronged approach from the data (content-addressable) to the underlying decentralized storage protocols. An economy of scale allows for a low service cost, while blockchain and consensus algorithms offer a robust, fault-tolerant system. Importantly, it is a peer-to-peer platform.

Fleek Network will build on existing decentralized storage protocols, such as long-term storage (Filecoin, Arweave) [2] and P2P storage (IPFS) [1], by adding a caching and performant delivery layer. Additionally, the use of Fleek Network will extend these protocols so that users can interact with the network directly instead of only being able to go through HTTPS. For the rest of the stack, such as the gateway and domain name system (DNS), an iterative approach to decentralization is being taken. This provides users an alternate route for accessing online content without censorship, as discussed in section 4.4.

3. Overview

A peer-to-peer protocol was developed, allowing users to request content from peers directly, as shown in Figure 2. A Gossiping Protocol [7] is also used by the network for nodes to gossip over the same topics, message dissemination, content, and block requests. Custom content routing is utilized for fast lookups, and a directed acyclic graph (DAG) based consensus is employed to verify cache accounting claims. In a fully decentralized network, there are no third-party

Figure 1. Traditional CDN setup.

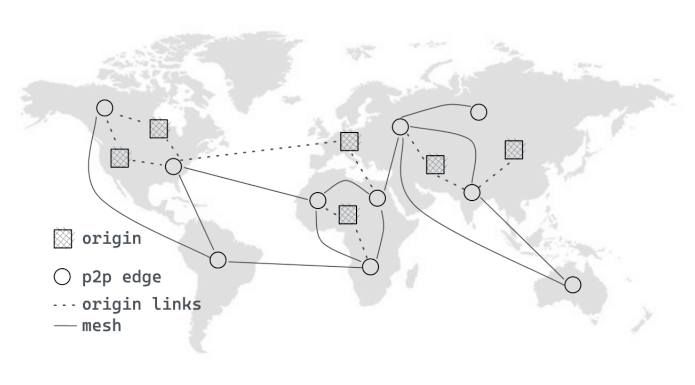

Figure 2. Fleek Network CDN.

servers. It makes the network more resilient, as there is little to no single point of failure, and it is easily maintainable (as end users can provision the network). With an economy of scale, the network can offer a low service cost. In addition, using a consensus algorithm, the network is robust and fault-tolerant. Furthermore, third parties gain access to user data and usage in the traditional model. No matter how centralized systems are, they are still vulnerable to an attack by malicious actors.

Users should have granular control over what information they send out over the internet and be protected from anyone trying to compromise their data or privacy. Fleek Network was designed to create an alternative content delivery network that will maintain high performance and availability for publishers while providing end-users with the experience they are accustomed to. Fleek Network uses an incentivized consensus mechanism to further this goal to ensure trustlessness based on content verification and high uptime. Although it is only in the very early stages, Fleek Network is engineered to be able to grow into a network of nodes and distributed computation tools capable of reliably delivering information that, in the future, could be verified through various means and software development kits (SDKs).

4. Fleek Network Architecture

This section will focus on the protocol architecture, the reasons it can be more robust than current solutions, the potential risks exposed, and how they are sought to be mitigated.

4.1. Overview

The network uses custom low-level peer-to-peer protocols and DAG-based consensus, adding caching, load balancing, and performant delivery to decentralized data-sharing protocols. The core innovation of the network is a combination of high throughput consensus, availability sampling, and content-addressable data to host and serve content. This enables Fleek Network to host and deliver content in a decentralized manner while maintaining data authenticity and network security with minimal overhead. Fleek Network does not store content; it only caches the content. The biggest issue for websites is high bandwidth consumption and data availability; the Fleek Network can reduce bandwidth consumption by reducing round trip time (RTT). Popular content will be cached in memory, allowing quicker trip times and low time to first byte (TTFB); content is evicted based on access popularity. To ensure a trustworthy service, Fleek Network will provides proper censorship resistance through various methods, including consensus, reward mechanisms, and content addressability. While it is essential for users to be able to determine who owns which content, it is also crucial that no central authorities control the data.

4.2. Composition

Fleek Network accelerates, caches, indexes, and replicates content. Requests are partitioned across multiple nodes to distribute the load efficiently. As the number of nodes in the network increases with time, the network's overall bandwidth increases while latency decreases. Content providers (end users, developers, or applications) incentivize content acceleration and retrieval. Nodes are incentivized proportionally for their participation based on provable metrics of servicing the network. The solution comprises the following components: 1) a high throughput DAG-based consensus, 2) content addressability, and 3) an efficient network layer. They are governed by a native protocol token called *FLK*. The network is agnostic to the accelerated data as long it can be represented as addressable content [4]. Figure 7 showcases an overview of interactions between the different actors in the network.

4.2.1. Actors. *Clients*, *Cache Nodes*, *Origin Nodes*, and *Gateway Nodes*.

- *Clients* operate independently from Fleek Network but interact with the network through the Gateway Nodes.
- *Cache nodes* responsible for caching, replicating, delivering content, and participating in the network.

They expose *Put* and *Get* functionality for the Clients.

- *Gateway nodes* bridge users to the closest Cache Node, handling data extraction, transformation, loading, and node failures with rerouting. These nodes act as a reverse proxy to the entire network, handling all client data transmissions through Get requests.
- *Origin nodes* persist Client data and respond to retrieval requests from Cache nodes.

4.2.2. Network. The Fleek Network consists of multiple sets of actors, namely Cache nodes *C*, Gateway Nodes *G*, Origin *O*, and Clients *P*. The network graph $F = (V, E)$ has a vertex set $V = C \cup P \cup G$ and an edge set $E = M \cup N$, where *M* represents the connections from *P* to *G*, and *N* represents the connections from *G* to *C*.x

4.3. Cache Nodes

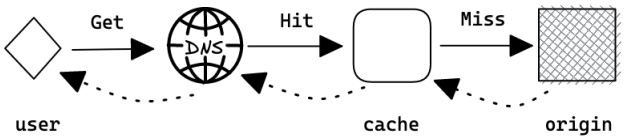

Figure 3. Cache life-cycle.

The Fleek Network problem, presented in Figure 7, can be modeled as a simple mapping, $M = (E, D)$, where u_i is the entity requiring data d_i. With a set of Cache nodes C_i and Gateway Nodes G_i mapped (g_i, C_i). Each gateway node is mapped to a set of connected cache nodes. Cache node c_i is responsible for replicating its data d_i across a set of other C_i nodes based on p_i, defined based on a Cuckoo filter or a weighted LFU cache, more in section 4.4.3. Partitioning the data across the list of all nodes in the network is handled by consistent hashing—more on this in section 4.4.1.

This paper presents an algorithm, content delivery, and routing system following a multi-mesh hierarchical strategy. Unlike previous systems, the approach allows each meshes independently provisioned with multiple network connections without coordination or cooperation. This means that different meshes may have different cache nodes serving different purposes but are all part of the same network. Furthermore, this paper shows how a hierarchical caching strategy can be distributed using multiple independent cache nodes through cooperation at only two levels. A hierarchical node environment improves the reliability of content delivery by splitting an information provider's network into several fault domains.

The Network's cache nodes maintain a cache of objects. O is represented as (K, V) pairs. When a cache node receives an HTTP request, such as in Figure 3, it checks to see if the requested object is already in its cache. If not, the server begins to query other cache nodes in the following order: First, it asks other nodes in its mesh; if none of these nodes have the object, then it may request content routing if needed; and finally, if all else fails, it issues a request for the object to an origin node.

4.4. Content Routing

Content routing algorithms are used, such as a distributed hash table, a gossiping protocol, consistent hashing, global content indexing, and Cuckoo filters. Composed and extended to speed up content routing, these algorithms aim to address the high delay traditionally experienced with hashing-based content routing and lookup schemes. The content routing information is stored using a distributed hash table, enabling scalability and handling large amounts of data space-efficiently. The system employs gossiping, consistent hashing, and Cuckoo filters to improve the data dissemination speed. Later stages of the system involve primitive caching, where "gossiping content routing" sends messages between nodes to distribute content routing information. The routing efficiency is improved by using consistent hashing. With this algorithm, the hash of the content's name is used as a key into a table that stores metadata about the content, allowing the content to be found quickly. To improve the speed of content routing, the cuckoo filter, a space-efficient data structure for approximate membership testing and probabilistic counting, is used. By extending these algorithms, the system is more responsive, but it can be challenging to maintain the consistency of the distributed cache.

4.4.1. Consistent Hashing. Consistent hashing is one of the algorithms Fleek Network uses to divide data objects among all the clusters of nodes. The Consistent Hashing algorithm runs independently of the number of nodes in the system and allows for scaling without affecting overall performance. It is used to distribute requests among its nodes by maintaining a hash ring, a data structure that maps all the nodes in the system to an abstract circle, depicted in Figure 4.

Nodes in the network are part of clusters, and each position on the circle represents a slot allocated to one of those nodes. The hash ring distributes incoming content requests among that subset of nodes. The algorithm starts by generating a hash value for each object that needs to be served. The network then calculates the difference between two consecutive slots on the circle, "modulo." The algorithm uses this difference as an index in a table that maps each node to a slot on the circle. The node with that position on the circle can now serve requests for any object whose hash is less than or equal to its slot number. The hash values for objects nearby on the circle are likely to be close together. The network can therefore serve requests by looking at a relatively small range of slots, which reduces latency and improves performance. Besides that, the algorithm can also use the modulo function to identify a "neighborhood" of objects likely to be requested together. If there are enough requests for nearby objects, the network will serve them from the same nodes, which improves performance by reducing latency and increasing throughput. The key to the modulo function is that it only needs to store a slot number,

not the entire hash space. With the modulo function, shown here in Equation 1, hash space is divided into slots and then used as buckets for storing objects. Then, when a request comes in for an object whose hash is greater than its slot number, the network will use a modulo to determine which slot it should look.

The network consists of *n* nodes C_1, C_2,..., C_n, take n nodes C_i with *i = 1, . . . , n* and define a cryptographic hash function *H*, a one-way function of generating a deterministic, collision-resistant key-pair, *Pair(s_{pub}, s_{priv})*.

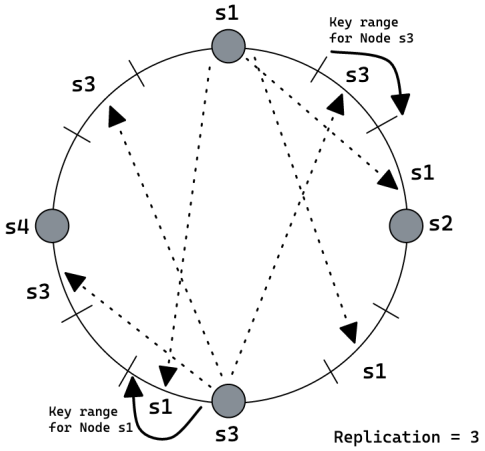

Figure 4. Consistent hashing with replication, r = 3.

$$\begin{aligned} h_1(s_{pub1}) &= p_1 \\ h_2(s_{pub2}) &= p_2 \\ &\dots \\ h_k(s_{pubn}) &= p_{n-1} \end{aligned} \tag{1}$$

$Where:$

s_{pub} : Peer Id.
k : A unique hash function.
p : Position on the hash ring.
n : Hash ring domain.

4.4.2. Data replication and partitioning. Smaller hash ranges are assigned to physical nodes through proxy nodes to ensure a more evenly distributed load. Several smaller ranges are assigned to each physical node by the proxy node, which divides the hash range. Proxy nodes are mapped multiple times across the hash ring to minimize changes to a node's assigned range, as shown in Figure 4. Proxy nodes reduce the number of changes to the hash range. When adding new physical nodes, the hash range for each node must be updated. A proxy node is mapped across all ranges, so changes are only made once instead of multiple times.

Every node in the network will have a unique hash range assigned to them. This hash range is used to identify that node in the network and its position within the ring. This hash range is also used for creating unique identifiers for transactions and blocks so that no two nodes use the same one. If a proxy node fails, the ring assigns another physical node to take over its responsibilities. The goal is to keep the hash range evenly distributed across all nodes while avoiding any single points of failure. The ring topology also allows for high throughput. This is because if one node fails, the system will continue to operate normally as long as other nodes are available. It also means the network can scale horizontally (adding more nodes), improving performance and availability. This allows for eventual consistency and high availability even in network partitions or partial failures. In this model, nodes do not know the other nodes on the ring. Instead, each node keeps in memory a routing table that holds the information required to contact the next node in the ring (according to their predefined order). This means that nodes can be added or removed from communication, and new ones can be used for queries. The second model is called "shared state." Each node knows the other nodes in the system and makes this information available through a shared API. This means that one can query any node for data, but it also has the added complexity of managing which nodes are queried for what pieces of data (e.g., avoid hot spots).

Each node c_i is responsible for caching and replicating a set of data D amongst connected peers C_i. Data availability is ensured by streaming data to multiple nodes in their mesh. In an eventual consistent network, data replication is done asynchronously. In Consistent Hashing, the data d_i is distributed among N nodes in the system such that each node is responsible for several data items. How the data is mapped to nodes ensures that no two items are ever placed on the same node. This ensures the high availability and durability of applications. The algorithm is given a set of data items $D = [0, 1, 2...9]$, which computes each item's hash value. Nodes are assigned to each item from this set of hash values. These nodes are responsible for replicating and caching these items among connected peers. Consider an example of $n = 2$, where each node receives half of all requests so that any of these two nodes can serve each request. This can be achieved in many ways, including picking a random node, rotating through available nodes in sequential order for a given target or hash code, or picking a least loaded node based on current usage or historical metrics. Different implementations can coexist if each node has an equal chance of serving a given request. Reducing the load on the system will be allowed by distributing requests among all available nodes.

4.4.3. Hot content and Set Membership. Hashing algorithms are used ubiquitously in systems that manage large data sets. They have been explored and analyzed extensively, yet only recently have a few data structures been reported that address the problems posed by real-world systems, such as large numbers of elements and high query loads. One such approach is Cuckoo filters [15], which represent their underlying sets using bitwise string operations. This paper describes a new representation that dramatically reduces the memory requirement of Bloom filters by using Cuckoo

Filters while maintaining excellent false positive rates under most conditions. $\mathcal{O}(1)$ time support for storage and retrieval is provided by Cuckoo filters, with a space overhead of $\mathcal{O}(n)$, making them an excellent choice for data structures with many lookups but where exact membership does not matter.

$$h(x) = H^k(x) \bmod n \tag{2}$$

The hash of the element x, used to set the corresponding bit in the Cuckoo filter, is given by Equation 2. The efficiency of the Cuckoo filter is increased by avoiding collisions, which is likely if a large enough range of values for the hash function H is present. In summary, the new representation of a Cuckoo filter relies on a bitwise operation applied to the input element and a modulo operation to obtain the corresponding hash, enabling elements to be stored in the filter without worrying about collisions. The new representation is more efficient than the traditional approach as it reduces the chances of collisions and enables the storage of more elements. This makes it a valuable tool for network security and data mining applications. Furthermore, it is faster than the traditional approach, which is especially important in time-sensitive applications. The critical insight behind the new representation recognizes that a Cuckoo filter is represented as a set of bitwise operations on its input. For example, if element $x = 10101$ of size 5 bits need to be added to a filter of size n bits, the hash $h(x)$ is computed by applying some bitwise operation H, k times. This can be done by performing binary addition (i.e., XOR), which results in a value of 0010, or by using binary multiplication with the following equation:

$$h_n(x) = \left\lfloor \frac{1}{2} \right\rfloor^n \cdot x \tag{3}$$

An extremely compact representation is obtained by representing Cuckoo filters in this way, which can be used in various applications where memory space is at a premium—given a Cuckoo filter bit vector B where each bit is set to 1 if and only if H_k was computed on the corresponding bit of x. This enables the computation of the membership test for x in $\mathcal{O}(1)$ time by simply applying H, k times to B. The key can access the elements used while computing this hash with the computed hash of an element x. In other words, consider a set S and a hash function H, elements in S can be computed whose hash value is equal to $h(x)$ by checking whether they satisfy $H_k = h(x)$, where k is less than n (e.g., 1).

Regarding content delivery, Cuckoo filters are useful to determine whether or not a piece of data is available on the network. Cuckoo filters have some drawbacks, though:

- They use large amounts of memory for each entry, which can negatively impact disk usage.
- They are slow to update and add entries.
- They must be configured to be granular enough not to filter out addresses due to false positives.

In the Fleek Network, two significant forms of replication are employed: uniform and non-uniform. Uniform replication entails each peer holding the same content, potentially a replica of every other peer. On the other hand, non-uniform (or edge) replication involves only a small fraction of the entire set of content being assigned to some peers. This is discussed in the next section. With uniform replication, the peers must synchronize each other's data to maintain consistency, which can be problematic. This can be time-consuming if there are many peers or the content is substantial. Non-uniform replication allows for much more efficient load balancing across peers, but it also has drawbacks:

- If a peer goes down, access is lost to the content it was hosting.
- If this happens at a critical moment in retrieving content, users must wait for it to be re-shared by another peer. There is no "master" copy of the data.
- The network topology must be known in advance. This may require manual configuration or peer coordination (e.g., via gossip protocols). Suppose the system is highly dynamic and peers are constantly joining or leaving. In that case, it can become challenging to determine which peer should hold which data based on the current network topology.

Edge replication is less efficient, but it is easier to manage. Uniform replication requires a lot of bandwidth and communication between peers; edge replication uses much less bandwidth because only the content for which a peer is responsible needs to be sent. The trade-off is that edge replication makes tracking what peers have and what content more difficult. Uniformly replicated systems must have a way to ensure that each peer has the same content as every other peer. This is usually done by having each peer share with others whenever they make a change or "commit" their data. The more often a node commits, the faster the system can recover from crashes and failures; however, it also increases overhead because it takes time to replicate all these changes across the network.

Decisions about handling data replication and reconciliation have been made to keep the network as resilient as possible. Uniform replication is preferred because every peer has access to the same content, which can improve responsiveness when searching for something specific; however, problems with disk usage can arise from uniform replication. Non-uniform (or edge) replication, which allows each peer to hold only a fraction of the entire content set, is more efficient disk usage. The details of how these algorithms work will be skipped over since multiple implementations can exist, although it is essential to understand how they are implemented.

Uniform replication is of interest in this case because a duplicate copy of the data must be available to all peers for every peer who has a piece of content to serve it to any other peer who requests it. Furthermore, scaling with the number of peers is desired in the system. Otherwise, there will be more overhead as the network becomes more prominent.

4.4.4. Reconciliation. Set reconciliation is a fundamental problem in distributed computing that involves computing the union of two sets located at different network nodes. The goal is to reconcile the sets to minimize the amount of redundant data transmission. The communication complexity of the reconciliation process is ideally bounded by the size of the symmetric difference between the two sets. Range-based set reconciliation is an approach that allows nodes to perform the necessary computations with space proportional to the size of the symmetric difference and time proportional to the size of the symmetric difference or the logarithm of the size of the local set, whichever is greater. This approach involves a divide-and-conquer strategy that sorts the sets according to some total order. It initiates reconciliation by sending the fingerprint of all local items within a specific range. If the fingerprints match, the range is successfully reconciled. Otherwise, the range is split into subranges, and reconciliation is performed on the subranges until all ranges are reconciled. Maintaining the fingerprints can be efficiently done by storing the set as a balanced search tree, where each vertex also stores the fingerprint of its subtree.

The complexity of Range-Based Set Reconciliation depends on the size of the sets being reconciled and the range size. In the worst-case scenario, where both sets have no elements in common, the algorithm would need to compare every element in both sets, resulting in a time complexity of $\mathcal{O}(n+m)$. However, in practice, most sets have some overlap, and the algorithm is optimized to take advantage of this fact. When the range size is small, the algorithm can divide the sets into smaller ranges, reducing the number of elements that need to be compared. The algorithm performs much better than traditional set reconciliation algorithms, especially for large sets with a small overlap. The average time complexity can be as low as $\mathcal{O}(n*log(m))$, making it a practical and efficient solution for set reconciliation.

4.5. Gateway Nodes

Although the primary nodes for caching, indexing, and replication are present in the network, nodes are needed in the network that can be pointed to by end-users for their domains, perform the essential functions of web servers, and act as a communication point between external clients and the network. This section describes how external clients, such as browsers and mobile applications, communicate with the Fleek Network. Besides the network nodes, Gateways act as the frontend (in server terminology as opposed to the frontend in web browsers) nodes. The Fleek Network can be used as a complete P2P replacement to DNS and TLS - but they are still required to access most web applications today.

There will be various elements of load balancing done at the boundary layer. Firstly, given a particular domain, it will resolve to an IP address near the requestor user's existing GeoDNS. Following that, there should be multiple gateway nodes in any given region so that this can further resolve into one of the IP addresses in that region using a solution like AnyCast. Optionally, multiple nodes can sit behind a

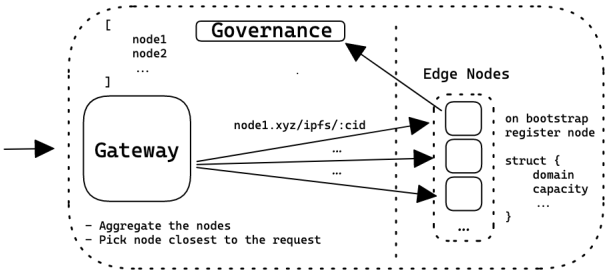

Figure 5. Gateway linkage to Cache nodes.

single IP address (think Kubernetes service endpoint) that other load balance "pods" or containers.

4.5.1. Decentralized Gateways. In a peer-to-peer (P2P) network, decentralized gateways are essential to building a censorship-resistant, resilient, and scalable network. The fair exchange protocol 4.7.1, which guarantees that both parties involved in an exchange are satisfied with the outcome and that neither party gains an unfair advantage, can be leveraged to facilitate the decentralization of gateways. Through cryptographic protocols such as Shamir's Secret Sharing algorithm and symmetric encryption keys, gateways can be decentralized and distributed among multiple nodes in a P2P network. These nodes can use the fair exchange protocol to ensure that they are equitably compensated for their services and that clients receive the requested data securely and reliably. Additionally, by employing streaming hash functions like Blake3, gateways can provide a streaming functionality that enables clients to receive data in real-time, enhancing the user experience. This approach provides a robust and scalable solution that improves the security and reliability of data exchanges while reducing reliance on centralized authorities. The decentralization of gateways using the fair exchange protocol represents a promising direction for developing resilient and scalable P2P networks.

4.6. Index Provider

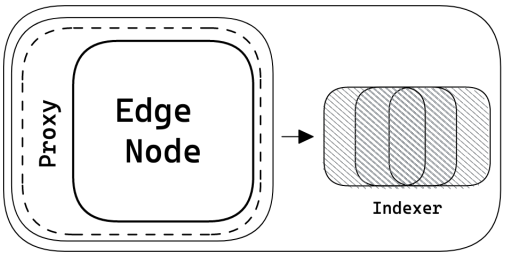

Figure 6. Cache nodes act as an Index provider for the content they cache.

The challenge of content routing is faced by all peer-to-peer networks and becomes more significant as the network scales. This challenge is tackled by integrating a global content indexer. Each cache node will run an index provider,

collecting the Cids (content ids) of the content addressable data and transforming it into an object. Nodes are responsible for sending the objects as an advertisement of locally cached content to the global indexer. Each node maintains a chain of this structure pulled in by the global indexers. The nodes' local storage relies on different replication and caching strategies discussed earlier to keep providing the content. If the content is unavailable locally, the node queries the Indexer, demonstrated here Figure 6, to get a list of providers in the network storing that content. The content is then transferred as blocks of data using data transfer protocols. As these nodes are short-term storage providers, keeping the routing updated when the cache is evicted is vital. Each advertisement has a *contextId* that determines a group of cids that go together. When a block is evicted, the cache node notifies the index provider about the eviction. When the Indexer pulls new data, the chain determines the data history, and evicted data is removed.

4.7. Availability Sampling

Several nodes are being run in a cluster at the core of the network infrastructure. The responsibility of running a single peer instance lies with the node, and scaling the Fleek Network becomes much more manageable since multiple nodes exist in the same cluster. Fault tolerance is also gained from the current layout as the network is scalable, enabling new nodes to replace failed ones if they stop running or misbehave. The nodes and clusters come to a Byzantine Fault Tolerant (BFT) consensus, which will provide the security needed for network transactions. A peer can publish any transaction signed with the correct private keys.

The difference lies in gaining throughput with availability sampling, separate from Data Availability Sampling. Usually, when a new block is produced, all the data behind that block is published to the network, resulting in nodes accepting big blocks having much more work to maintain consensus than nodes accepting small blocks or no blocks. To solve this problem, Fleek Network is agnostic to the underlying storage layer, provided that the data can be represented in a content-addressable structure. The data would translate to a Merkle Tree. The tree's root would be a single hash representing the entire state of the current data. The network must also replicate this information across all nodes in the cluster.

The data cached in Swarms is considered, which provides everyone access to the same data while ensuring they cannot tamper with it. The data is content-addressable, which makes it easy to check consistency between the state at the end of a block and on a given subsequent block's state root. Since the consensus mechanism does not care about the data, only the state root gets committed. Therefore, only certain parts of the state tree are required for sampling. Fetching the whole root data would be expensive. Instead, leaf nodes can be fetched as they are only 256 bytes of data. Validators can randomly pick these leaf nodes to verify the entire content addressable structure of the data in a block. A good outcome of availability sampling is that nodes only need to provide data for a given period (6 months, for example). The node can then delete the data based on content popularity or other data-driven metrics. So if a node decides not to store data and publish a block missing a transaction, the network can detect and slash them without downloading the state of the entire blockchain.

4.7.1. Proof-of-Delivery. Fair exchange is a challenging problem in peer-to-peer (P2P) networks where two parties exchange information and want to be protected. The fair exchange problem involves ensuring that both parties are satisfied with the exchange and that neither has an advantage. The fair exchange problem is a crucial issue in P2P networks, and various approaches exist to address it. One approach is to use a trusted third party to mediate the exchange. However, this approach has limitations, such as increased costs, reliance on a centralized authority, and potential privacy concerns. Another approach to addressing the fair exchange problem in P2P networks is to use cryptographic protocols to ensure the exchange is fair. These protocols involve encryption, digital signatures, and other cryptographic techniques to ensure neither party can cheat or gain an unfair advantage in the exchange.

To address this, Fleek Network uses secret symmetric encryption keys to create a commitment to the request. The client returns the "receipt of payment" after verifying the commitment. A streaming hash function like BLAKE3 [17] enables streaming functionality. In addition to these measures, Fleek Network also uses Shamir's Secret Sharing algorithm further to share every node's secret key with the consensus committee members so that the committee can be used to resolve disputes.

Shamir's Secret Sharing is a well-known cryptographic algorithm that allows a secret to be split into multiple parts, or shares, and distributed among multiple parties. The secret can only be reconstructed by combining a certain minimum number of shares, while any subset of shares fewer than this minimum number reveals no information about the secret. In Fleek Network's fair exchange mechanism, the symmetric encryption key used to encrypt the requested data can be split into multiple shares using Shamir's Secret Sharing algorithm.

In case of a dispute or failure to share the decryption key, committee members can share the shares of the decryption key with the client. The client can reconstruct the decryption key independently if a certain minimum number of shares are available. This provides an additional layer of security and redundancy to Fleek Network's fair exchange mechanism, ensuring the data exchange is fair and secure even during unexpected challenges.

Overall, Fleek Network's implementation of the fair exchange problem is a sophisticated and well-designed solution incorporating state-of-the-art cryptographic techniques to ensure the security and reliability of peer-to-peer data exchanges. The use of secret symmetric encryption keys, streaming hash functions like BLAKE3, and Shamir's Secret Sharing algorithm work together to create a robust and

resilient fair exchange mechanism that can be relied upon by users and nodes.

4.7.2. Validators. The Fleek Protocol also allows each Validator to vote if they deem the data valid and then sends this vote data along with the block to other nodes to validate transactions. If a validator detects something suspicious, they can notify peers with availability proof. These proofs will make it impossible for any party to tamper with data or rewrite the transaction history. The basic idea behind these proofs is to prove the invalidity of a node storing a piece of data without requiring a participant to keep the actual data. The basic idea is that every Validator will keep track of peers; if they think a transaction is invalid, they can easily prove it. They can do this by storing a proof record for all transactions that modify the state of the network. This combination of availability sampling and content-addressable data provides exceptional security while avoiding unnecessary overhead and minimizing resource requirements.

5. Robustness

A Byzantine fault-tolerant algorithm ensures that the system is fault-tolerant. Distributed denial-of-service (DDoS) protection will be used at the gateway nodes so that sites and files hosted on the network have a substantial uptime. Addiontaly, nodes rate limit on the networking layer as a form of DDoS and employ blocklists.

The BFT algorithm allows each participant in the system to broadcast their proposed value to all other participants. If most participants agree on the same value, it is accepted as the consensus value. If there is no majority agreement, the algorithm is repeated until a consensus is reached. The advantage of this algorithm is that it can tolerate up to f (the minority of corrupted participants) faulty nodes with n identical nodes distributed over space and operating in a mode where f bounds the size of corrupted subgroups $f < (n/3)$. This algorithm can guarantee the system's safety, as no single subgroup can control the system. To summarize the network's robustness, the system is resilient to:

- Transient faults.
- Malicious faults.
- Byzantine faults.
- Byzantine Generals attacks.
- All honest nodes in the system will agree on the same value (consensus).
- The system can tolerate up to f faulty nodes and still reach consensus.

5.1. Node Robustness

The cache nodes' robustness will come from the governance and network redundancy. The network leverages availability sampling to ensure nodes are caching data. As such, nodes can distribute the application and related content closer to the user, significantly reducing overhead and round trip times. Consequently, reducing hosting bandwidth and the amount of data an origin server must provide. Fleek Network is built with high availability in mind. This allows them to continue to operate even if an individual node fails. High availability is paramount when building such a critical and scalable system to ensure consistent content delivery and the best possible experience for their clients.

5.2. Gateway Node Robustness

Gateway nodes will be highly available using a multi-region setup with container management through AWS ECS or Kubernetes. Domains will be mapped to multiple IP addresses using a combination of GeoDNS and AnyCast, which directs users to a gateway node in their region and, within that region, load balanced across multiple gateway nodes. Gateway node robustness relies on proven application uptime techniques such as high availability and multi-region setups.

5.3. Security

Fleek Network will not be immune to attacks expected in the blockchain ecosystem. Even though any implementation has its trade-offs, the fact that vulnerabilities are technically feasible means something other than that they are practical or efficient. This section will identify different types of potential attacks and their mitigation strategies.

- **Sybil Attacks.** Representing the data as content addressable, data integrity is upheld. Combining the Distributed hash table (DHT) and content addressable data with a BFT consensus can significantly reduce the DHT query's range of peers returned. Each node has a stake property that creates an ordering between them and ensures that more prominent nodes are higher. There are better defenses against Sybil attacks, but it does mitigate them by creating a couple of orders of magnitude worth of work that Sybils must perform to gain control over many nodes and disrupt the network. In addition, peers can implement multiple anti-Sybil algorithms and possible identity delegation schemes in which reputation can be passed down to entities below you to prevent Sybil attacks. The problem of decentralization and possible data tampering can arise if data integrity is to be ensured. A node can be controlled by any user with malicious intent to modify the stored data. The DHT prevents content spoofing using a cryptographic hash, as it becomes more difficult for an attacker to generate multiple content items that share the same key. The solution to the Sybil problem is called probabilistic node selection. Following rules, each network member can detect when an attacker creates too many nodes and remove them from their routing table.
- **Eclipse Attacks.** The eclipse attacks in a decentralized system are network partitions of a specific

peer. A malicious actor generates many nodes with fake peers to target a specific peer, making them unavailable for any other communication in the network. This is done by advertising fake addresses, which will cause the target peer to connect to it and be unable to reach any other node. Several designs for defending against eclipse attacks exist, such as requiring decisions to be made on a consensus resulting from multiple queries, reducing the system's efficiency. Other design options include a protocol for detecting and rejecting peers that exhibit malicious behavior at the cost of efficiency. Fleek Network improves upon current eclipse mitigation by implementing IP-based rate limits, protecting against node table overflow attacks by raising the cost of control. This makes communication between peers more reliable and secure by leveraging the power of open channels that encourage cooperation and consensus.

- **Spam Attacks.** To prevent such attacks, reliance is placed on the incentives given to users who provide services. The incentivization protocol will motivate participants to maintain the network. The bandwidth requirement is non-trivial, making it economically challenging to carry out such an attack. Honest activity will always be more prevalent than malicious activity, and users who attempt to DDoS the network will get slashed and have their "stake" of participating or reputation gained by helping the network taken away.

6. Consensus

Fleek Network uses an underlying consensus engine for ordering transactions. This is a critical component when building a network with high throughput requirements, as it must consistently and reliably order transactions to allow transactions to be processed correctly. The consensus layer is designed to handle stateful transactions and provide traditional blockchains' functionality. Fleek Network's DAG consensus checkpoints its state periodically. The goal of the consensus layer is to provide a robust and reliable ordering mechanism for stateful transactions, handle high throughput requirements, and process transactions as quickly as possible while maintaining consistency. The partial asynchronous Byzantine fault-tolerant (BFT) algorithm allows Fleek Network to process transactions quickly while maintaining consistency. The BFT algorithm ensures the system converges on a single result, even during failures or malicious actors. The consensus engine uses the following components:

- A network of nodes to maintain a replicated state machine.

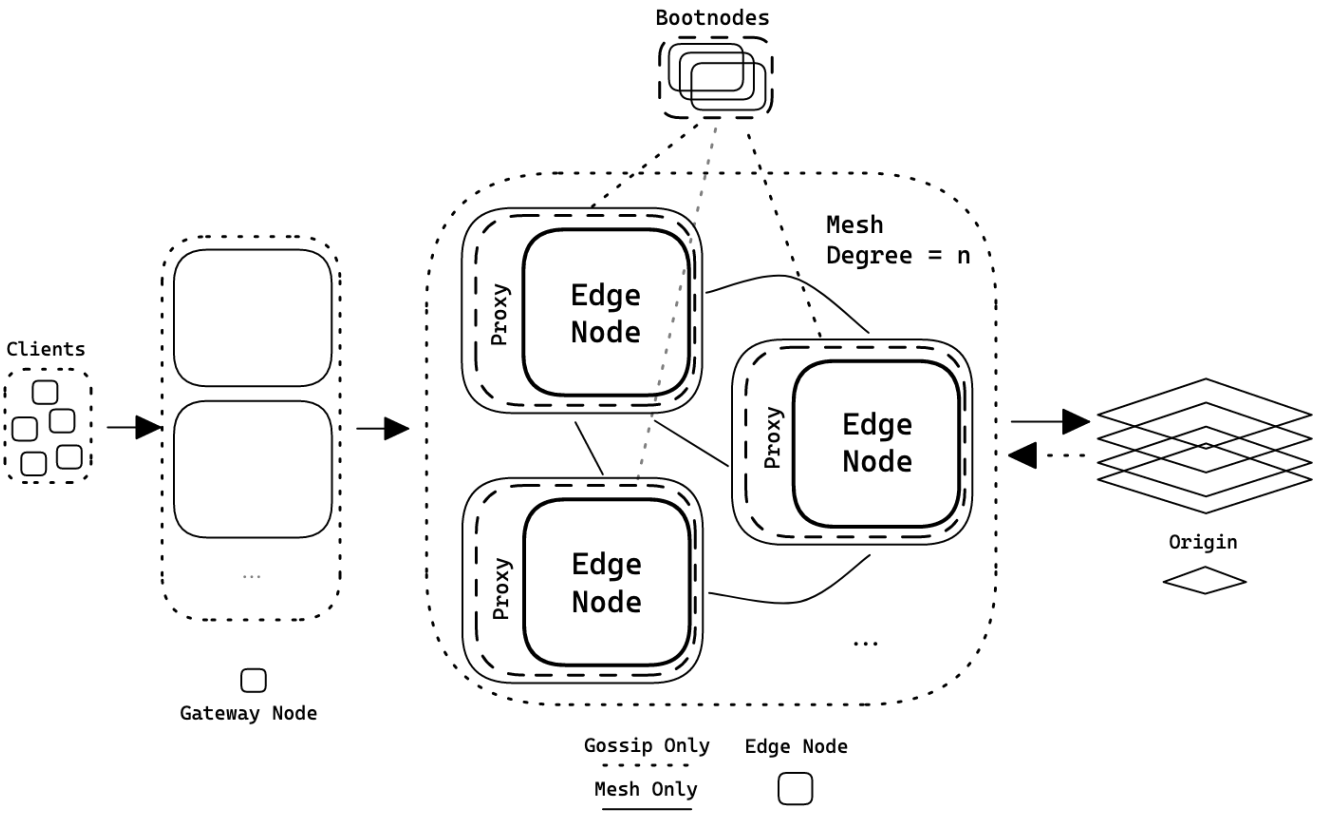

Figure 7. Network Overview.

- A set of rules that govern how nodes interact with each other.
- An algorithm for ordering transactions.

The network of nodes maintains a replicated state machine that stores all the data for Fleek Network. The state machine is a key-value store where each key represents an object. Every node has its copy of this state, and when any node makes changes, they are propagated throughout the network. The set of rules that govern how nodes interact with each other is called consensus. In this system, nodes communicate and agree on a state change before it is committed to the database. The algorithm for ordering transactions used is Bullshark [6] coupled with Narwhal [5] to achieve a high throughput distributed system.

Decoupling data dissemination from metadata ordering eliminates the need for a centralized message broker or coordinator to support the scale of a CDN. Instead, each node can independently broadcast data to the network and order messages using a distributed ledger. This approach allows for scalability and high throughput, as the system can be scaled out by adding more nodes to the network. Furthermore, using a DAG to abstract the network communication layer from the consensus logic can reduce the overhead associated with consensus protocols.

A DAG-based BFT allows for more efficient implementations as messages can be ordered without a global consensus. This makes for a more resilient distributed system that is much harder to disrupt. In addition to these benefits, DAGs are more scalable than traditional blockchain implementations. This is because nodes no longer need to validate each transaction to be included in the next block. Instead, transactions are grouped into batches by timestamp and ordered within those batches based on their transaction IDs. The DAG solution has its own set of problems, however. As the number of transactions increases, so does the data structure's size, which each node must maintain. This can cause issues when scaling out as more nodes are added to increase throughput. To deal with DAG scalability, the DAG needs to be partitioned. This allows nodes only to maintain a portion of the overall graph, making it easier to handle large transactions. The concept of "trusted checkpoints" is introduced to solve this problem. The idea is that every node will maintain a copy of the DAG and store it in memory. As transactions are added, each node will compare them against its local copy of the graph and determine whether they should be included in their DAG structure. If all nodes accept the transaction, it will be added to their DAG, and the network will continue to grow. If two or more nodes disagree on whether a transaction should be included in their DAG, they will create separate DAGs instead of one large one. In this case, both networks would act independently and create different versions of history, eventually converging on one source of truth.

To avoid indefinite growth of the DAG, the nodes will have to periodically prune their DAG down to just those most relevant transactions to guarantee that vertices are garbage collected during every synchronous period at every step. It is impossible to guarantee fairness during asynchronous periods [5], [6]. This is done using the transaction's weight to determine which transactions should be included in the final graph structure. When this happens, it is called "reaching consensus." All nodes agree on the same version of history. The goal of reaching a consensus is to ensure that all nodes are in sync with each other so they can communicate effectively without any security issues that might arise if they are inconsistent. Synchronization is a necessary evil. Blockchains can operate with synchronous periods of every round, which would make topologies idempotent and more vulnerable to attack. To keep things fair, the DAG must grow for a while; otherwise, the larger chain would be favored in consensus decisions and cause the smaller chains to die out.

6.1. Boostrapping

The process in which governance allows a node to become part of the network and expand the number of nodes. For this purpose, the node has to first lock a reasonable amount of the governance tokens for a set number of days, which can be modified by the governance if needed. Then a node will send a join request to the governance layer. Their stake is checked at this point, and they will be listed as Validators if it is enough. In this round, the Validators will run a validation process to estimate this node's bandwidth and compute its availability score. This process runs for a week until a node's capability to perform on the network's standards is verified. From this point forward, if the node meets the criteria, it will be listed as one of the CDN nodes and start handling actual users' requests.

6.2. Punishment

The validator operates in two modes: 1. Discover, 2. Verify. A single validator acts in the same two modes regularly. In the discovery phase, a validator tries to perform a checklist against a node to "find" whether a particular node misbehaves. If such a node is found, the validator puts a bounty on its claim. The governance will list these claims and other validators in their Verifier mode, then retrieve this list of misbehaving nodes and verify that the other validator's claim stands. Furthermore, they also vote on the claims; once enough evidence is collected, the node is punished by deducting tokens from its stake value and distributing them as a reward to the validators.

To have a fair system, the misbehaving of nodes is separated into two groups of intentional and unintentional problems. Generally speaking, any problem that might happen to everyone and is not necessarily proof of a malicious actor is considered an unintentional problem. On the other hand, anything intentional proves for sure that a node runner was acting maliciously. Unintentional problems have lower costs.

6.3. Slashing

Once a node's stake value is decreased to less than a specific value (set by the governance), a node will not be able to operate anymore and will be delisted from the list of CDN nodes. The node owner must increase the nodes-locked tokens and perform the onboarding again. The government layer will be able to evaluate changes to the slashing rules, like permanent slashing and onboarding retries.

6.4. Rewards

Once a node's stake value is decreased to less than a specific value (set by the governance), a node will not be able to operate anymore and will be delisted from the list of CDN nodes. The node owner must increase the nodes-locked tokens and perform the onboarding again. The governance layer will be able to evaluate changes to the slashing rules, like permanent slashing and onboarding retries.

6.5. Economics

The following section describes the token model used for the network and how it helps keep it operational and secure. The decentralizing content delivery market becomes possible with the right incentives. This model could change because it is very early in the development of this network. Much of this model was inspired by the Pokt Network [3], as there are parallels in the market they are creating for API requests, whereas Fleek Network would focus more on bandwidth.

Application consumers of the network stake *FLK* tokens at the beginning to host their content. At first, they can skip paying their tokens to cache providers to bootstrap the network. Once the network gains traction, they must pay based on usage. The amount they stake will be based on bandwidth required in many requests and sizes and additional features like time-to-live (TTL)/image resizing/custom headers.

Node operators will also stake an amount to be determined by governance that would correspond to a level of service in terms of the number of requests and total bandwidth available. This amount will be the minimum amount required to stake by node operators and will be a minor unit corresponding to a single node and a fixed number of requests and bandwidth offerings. i.e., they would add more nodes by staking additional units corresponding to the number of nodes they wanted to offer. Tokens will be minted and released by the protocol initially to reward node operators. Later, the decentralized autonomous organization (DAO) can determine the inflation/reward schedule and other token specifics (usage cost). Staking by node operators will have un-staking periods to incentivize long-term participation.

6.6. Incentivisation

Staking is required for application owners to use the network. Incentivizing network growth further, there will be no payments for the usage initially. Requiring node operators to stake incentivizes proper availability, performance, and integrity. Long-term participation is promoted by requiring or incentivizing stake lock-ups or un-staking periods and allocating rewards accordingly. Site owners are incentivized to ensure a properly functioning network to serve their applications consistently and performantly.

7. Future Work

- **Content optimization.** Tooling to optimize a root CID (hash) and walks the dag optimizing all the assets needed to provide a new root hash that the network could serve. For example, a website with a lot of media files.
- **Distributed Denial of Service.** Will be at the core of a successful delivery network, as seen with services like Cloudflare and Fastly. Implementing machine learning models to analyze the usage across the network to help detect and route spikes in traffic and determine whether or not the traffic is legitimate or originated from an attack. To be clear, DDoS protection protects node operators from such attacks. Still, it will be incremental, and such implementation details are out of scope.
- **Integrity.** Anti-censorship is a priority, and the network will be modified as necessary to ensure it. However, fairness to the authors and artists of original work is also necessary to handle such cases democratically. Once the gateway nodes are decentralized, the community and the DAO can determine how to handle these cases by voting. Different gateway nodes can run their governance to decide what rules to follow when protecting original work, and site owners can choose which gateways to serve their content through.
- **Decentralized Edge.** Given the inherited geo-distribution, Fleek Network offers a promising approach to address the challenges of serverless computing models. By leveraging a modular stateless execution environment based on [16], Fleek Network Economics aims to provide a distributed infrastructure for addressable functions. The proposed SDK would enable large-scale deployments of these functions, and the system's modular design would allow easy integration with popular storage solutions such as *Postgres*, *Gundb*, or any other conflict-free replicated data type (CRDT) database. Furthermore, the Fleek Network proposal emphasizes cost vs. scale, as adding more nodes will decrease cost. Building trustless computing systems in a network where bad actors can participate requires a careful balance between privacy, verifiability, and performance. While recognizing that verifying computation is not a requirement but can be added. Overall, the proposed system could provide a novel approach to decentralized computing, enabling developers to deploy and run stateless functions distributed while providing a

scalable, cost-effective, and secure infrastructure. As such, it could drive the next generation of serverless computing and usher in a new era of decentralized, modular, and distributed computing paradigms.

8. Conclusion

This paper proposes a solution to decentralize CDNs, where content can be routed using a peer-to-peer approach to be reliable, fast, and censorship-resistant by relying on blockchain technology as the core of the technological stack. The network's governance layer and the base protocol allow the network to grow sustainably over the long term. An economic model for content delivery incentivizes people to share their excess capacity with the network and provides them with fair compensation is also suggested. This protocol aims to create a decentralized CDN where people who share their bandwidth can do so equitably. A completely censorship-resistant, fast, and reliable network, with no single point of failure, will be achieved. The idea is that the Internet should be a place where all content is treated equally. However, the interests of big corporations are prioritized over those of the people, and the Internet is controlled by a few powerful companies prioritizing their profits over everything else. That is why a decentralized internet needs to be built, giving people back control over their data. Decentralizing CDNs is the first step to achieving this goal.

Acknowledgment

Janison Sivarajah, Parsa Ghadimi, Muhammed Arslan, and Matthias Wright, thanks for peer reviewing the paper and giving insights on the overall structure and organization of the paper, clarity of the language used, the accuracy of the information and data, overall formatting of the paper, and suggesting possible improvements and directions for further research.

References

[1] IPFS. https://ipfs.io.

[2] Filecoin. https://filecoin.io.

[3] Pokt Network. https://www.pokt.network.

[4] IPLD. The data model of the content-addressable web. https://ipld.io/

[5] Narwhal and Tusk. https://arxiv.org/pdf/2105.11827.

[6] Bullshark. https://arxiv.org/pdf/2201.05677.

[7] GossipSub. https://arxiv.org/pdf/2007.02754.

[8] GossipSub Ethereum 2.0. https://arxiv.org/pdf/2007.02754.

[9] MPC TLS. https://eprint.iacr.org/2021/318.

[10] Informed content delivery across adaptive overlay networks. Available at: https://www.eecs.harvard.edu/ michaelm/postscripts/sigcomm2002.pdf.

[11] Fleek-Network, "Fleek-Network/Ursa: Ursa - A decentralized content delivery network," GitHub. [Online]. Available: https://github.com/fleek-network/ursa.

[12] "Choosing a routing policy - amazon route 53." [Online]. Available: https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-policy.html.

[13] "Algorithmic nuggets in content delivery - akamai." [Online]. Available: https://www.akamai.com/site/en/documents/research-paper/algorithmic-nuggets-in-content-delivery-technical-publication.pdf.

[14] Filecoin-Project, "Filecoin-project/storetheindex: A storethehash based directory of cids," GitHub. [Online]. Available: https://github.com/filecoin-project/storetheindex.

[15] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo Filter: Practically Better Than Bloom. In Proceedings of the 10th ACM International Conference on emerging Networking Experiments and Technologies (CoNEXT 14). Association for Computing Machinery, New York, NY, USA, pp. 75-88, 2014.

[16] Protocol Labs Research. "IPFs-Fan: A Function-Addressable Computation Network." Protocol Labs Research, https://research.protocol.ai/publications/ipfs-fan-a-function-addressable-computation-network/.

[17] BLAKE3-Specs/blake3.Pdf at Master · Blake3-Team/Blake3-Specs - GitHub. https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf.

[18] Meyer, Aljoscha. "Range-Based Set Reconciliation." ArXiv.org, 7 Feb. 2023, https://arxiv.org/abs/2212.13567.