Blockchain Content Fabric for Multi-Chain Content Ownership and Distribution

{michelle.munson, serban.simu, lukas.anliker, michael.parker +team}@eluv.io

August 8, 2022

Abstract

2021 brought the first mainstream monetization of content on the blockchain with the release of high value NFTs created from digital art, the proving of digital collectible communities for fans, and the first direct-to-consumer Web3 distribution models backed by NFTs for blockchain ticketing, content sales, re-trading, and content investing. We at Eluvio were fortunate to have an on-court view of this, helping to power the first just-in-time "on-chain" content supply chains and creator-to-audience blockchain distribution experiences using the Content Fabric. The Fabric is a blockchain controlled, real time and programmable content storage and distribution network in which nodes execute a novel content-native application-specific blockchain protocol, the "Content Fabric Protocol" (CFP).

The CFP helps to make possible a fully decentralized Creator Economy with high quality, highly efficient and verifiable digital content ownership, distribution, and monetization using blockchain. It aims to provide the essential utility functions for managing and distributing digital content over the Internet in a decentralized network for Web3-native content experiences at scale. Its design goals are low latency, high resolution publishing, streaming, download, and interactive access to digital content globally with maximum resource efficiency and in turn lowest cost, combined with on chain ownership, access control, and transactions. These attributes are made possible by a novel decentralized ("nothing shared") design built on the following principles: 1. Self-scaling via a trust-less security model, an open fabric-extend protocol, and strong incentives allowing content owners, viewers, sponsors and infrastructure suppliers to benefit. 2. Extreme bandwidth and storage efficiency compared to traditional cloud, CDN and digital asset management architectures through a new content-centric storage and distribution design. 3. Predictable high resolution delivery and low latency for on demand and live content. 4. Programmable output, allowing content to carry its "code" and just-in-time application when serving output versions. 5. Certifiable, tamper-proof content where all version history including derivatives are transparent and provable from on-chain data. 6. On-chain attestation and verification of ownership and authorization to content.

With the maturation of blockchain token technologies, including NFTs and fungible tokens, this last point has opened a huge range of possible new applications where content can be directly monetized and re-monetized using digital tokens. The purposes of this paper are first, to describe the key design principles of the CFP from both the content and blockchain perspectives, and second, to introduce a new on-chain and cross-chain paradigm for attestation of ownership and authorization in decentralized content distribution using digital tokens. This new paradigm applies to tokenized ownership of media of all variations including fungible and non-fungible tokens (NFTs) and utilizes the Content Fabric's core capabilities to programatically support the secure and verifiable reading of or writing to any representation of digital media content under the control and verification of a policy stored on chain, at scale.

The current production network (main.net955305.contentfabric.io) has been built using a standard open source fork of the Ethereum network and has utilized Ethereum standard addressing, cryptographic signatures and keys, and smart contracts. However the CFP can be independently implemented using any full stack layer 1 blockchain, and thus provides an open and potentially universal design for decentralized content in a federated and heterogeneous blockchain ecosystem consisting of many interoperable blockchains. Accordingly a work-in-progress implementation of the protocol utilizes the Polkadot substrate blockchain and the W3F inter-blockchain validation and messaging features. Furthermore, the CFP applies to content and is independent of where (on which blockchain) proof of ownership exists, opening the possibility for cross chain content ownership and authorization. The paper will conclude with several example use cases on the platform from major media brands and emergent creators.

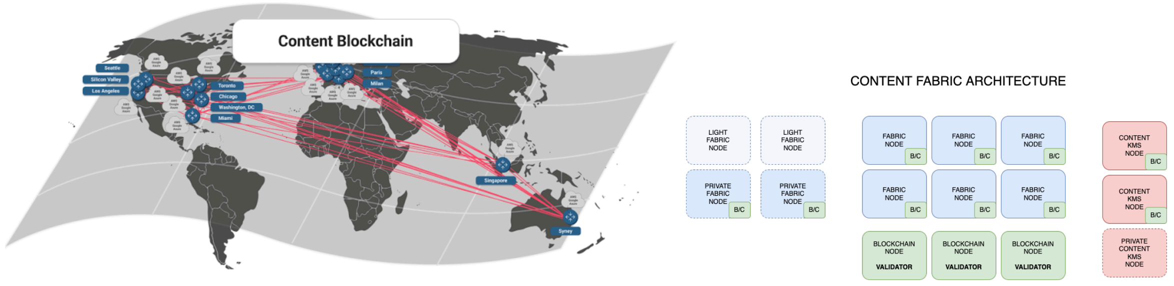
1 Introduction and Functional Overview

The Content Fabric is an incentive driven, open and decentralized platform for just-in-time content distribution, monetization, and asset servicing. It achieves high performance (low latency and high bandwidth) delivery and self-scales because of its technological design and user incentives.

A global network of nodes run the Content Fabric Protocol (CFP). CFP is an application-specific blockchain and a just-in-time componentized content storage and distribution protocol, purpose-built to serve content at scale with direct owner to consumer provenance.

The CFP executes a decentralized data protocol across the global network of hosting nodes through which video, imagery, apps and other active content is dynamically served directly from source objects as live and on-demand streaming and dynamic combinations. Blockchain contracts and on-chain policy authorization secure the ownership, access and versioning

Figure 1: Network of Nodes Executing the Content Fabric Protocol Software



of content. The nodes are peered directly with all of the major public clouds allowing for local ingest from cloud storage buckets, and have high capacity (multi-terabit / second) IP transit capacity for direct ingest of content including live, and egress distribution of streaming and static content to end users.

Functionally, CFP eliminates the need for individual content transcoding, aggregation, management, and distribution services by consolidating many conventional functions such as live ingest, cloud origin, live transcoding, content management, encryption/DRM, program sequencing, rights and avails control, CDN streaming, and static content distribution.

The Content Fabric creates and serves output such as adaptive bit rate streaming manifests and segments, and static content through a just-in-time process that executes within the nodes in the network. This process relies on a novel decentralized data storage and distribution protocol and a component object representation of media essence, metadata and code called a "content object". The content object is a data structure comprised of references to the cryptographic hash signatures of the binary "parts" that comprise the object.

On ingest -- master file media, assets, data, or streams -- are decomposed into such parts, distributed throughout the network, and an object is created. Any re-use of the parts within an object is by reference - rather than by copy - and common bytes are copied only on update. This is carried through rendering, such that all re-rendered output is built from parts, avoiding all file copy representations through the network and in storage for efficiency. A fast part routing algorithm allows for parts to be found in the network in real-time and supports just-in-time transcoding, packaging and extensible A/V processing within the protocol.

2 Protocol Design Principles

- Content Internet Overlay - A single soft stack runs on all nodes for maximum scalability. Nodes communicate with one another to securely store and serve end-user content using a decentralized design, meaning no state need be shared via any centralized entities (databases, tables, ledgers, etc.).

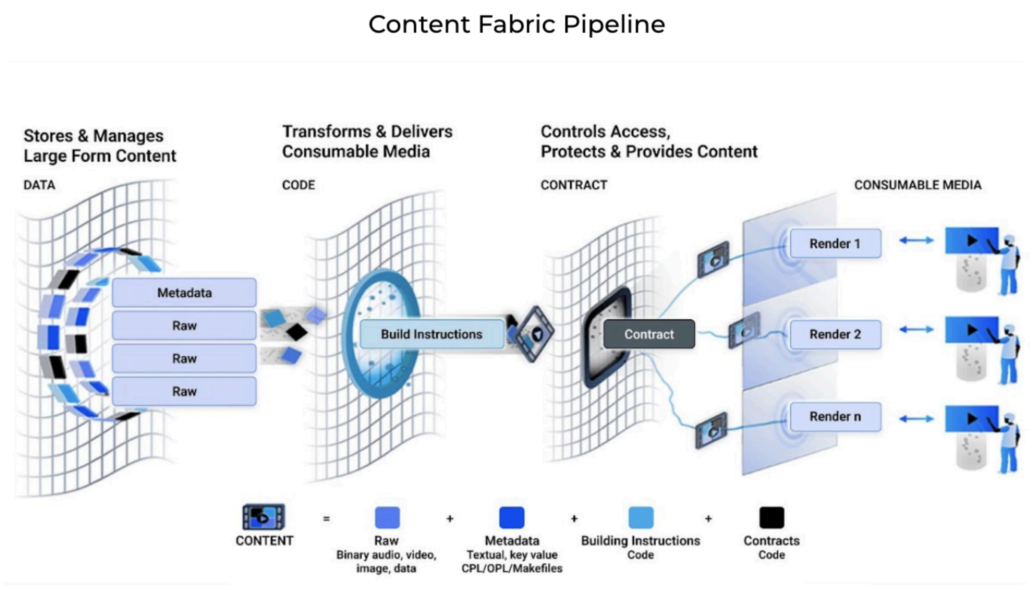
- Low Latency and High Performance with Scalability - Content is served using a maximally resource efficient content routing protocol that ensures low latency (<500 milliseconds time to first segment) and high bandwidth delivery (at the client's bottleneck bandwidth) with minimum server and network bandwidth resources. The performance properties hold with the expansion of the number of nodes and number of content objects stored.

- Content and Metadata are Stored "Once" and End Versions are Rendered on Demand - All content is stored in an object structure consisting of content, metadata, and "code" that operates on the media at serving time ("just in time") to yield the consumable versions. All data ingested into the fabric (comprising the content, metadata, and code) is divided into chunks of bytes ("Parts") that are stored on the disks of nodes running the fabric software. Only the hash of a part is stored into each content object structure and parts re-used in new versions or in derivative objects are never copied: New versions of the same content object and new content objects that derive from existing content objects refer to the hashes of its parts. The "code" that operates on the content and metadata is compiled and executed just-in-time, allowing for flexible re-use of the media, updates of the code without updating the CFP software (for scalability), and sandboxing for security and metering.

- Nodes are Not Trusted and Content is Protected (encrypted) - The CFP is "trustless" in that content is encrypted for owners and re-encrypted for authorized receivers without the software or nodes on which it runs accessing content in plain-text or having access to the content's encryption keys. Nodes that perform trusted operations that require the content to be plain-text (unencrypted) are rated publicly, take a stake in their correct behavior, and are incentivized to behave well with higher compensation.

- Access is Mediated through Blockchain Transactions - All access to content objects – create, update (write), and view (read) – are mediated by transactions on an embedded blockchain ledger. The application interface supports programmable blockchain transactions, e.g. the current mainnet.955305 implements the Ethereum Virtual Machine

Figure 2: Content Fabric Just-in-Time Pipeline


Content Fabric Pipeline
Stores & Manages Large Form Content
DATA
Transforms & Delivers Consumable Media
CODE
Controls Access, Protects & Provides Content
CONTRACT
CONSUMABLE MEDIA
Metadata
Raw
Raw
Raw
Build Instructions
Contract
Render 1
Render 2
Render n
CONTENT
Raw
Binary audio, video, image, data
Metadata
Textual, key value CPL/OPL/Makefiles
Building Instructions
Code
Contracts
Code

and executes EVM smart contracts. Each life cycle operation on a content object is implemented as a transaction against a smart contract for that object, in turn recording the address of the entity/user that made the operation, the identifier of the content object and any details of the transaction.

- Content Operations are Programmable - The base smart contract interface for content objects references the hash of an extensible policy data structure, and offers a generalizable interface for authorization of content operations via on-or off chain data such as identity, token ownership, credit or debit of an account, time, geography or roles, etc. for intrinsic commerce, rights management, and workflow capabilities. The "in object" execution of bitcode allows for extensible content operations such as interactive and personal variants also under blockchain access control.
- Content Versions are Provable and Tamper-resistant - Every content object has a version proof consisting of a Merkle tree root hash of the object's part hashes for fast verification of the integrity of the object. The root value of the Merkle proof is recorded in the blockchain transactions for that content object allowing for a tamper proof record of the version history of the object ('who changed what when') .

3 Key Technology Innovations

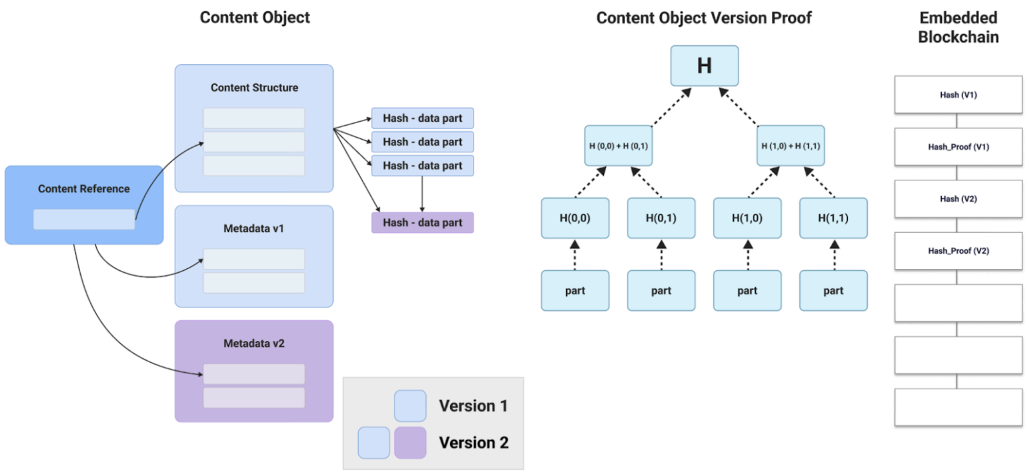
The realization on these design principles relies on four major areas of new technology invention in the Content Fabric Protocol, which we will detail in the following sections. We begin with the foundations of the network, a decentralized, low latency and efficient content native storage model and content routing system. Then we turn to the supporting capabilities including just-in-time composition of metadata, content, and code. Finally, we cap off with the blockchain protocol which combines a trustless security model, blockchain controlled content access using proxy re-encryption of content, and authorization and access control via policy stored within contracts on the blockchain, and content integrity verification and origin traceability implemented with a fast, provable content version history backed by blockchain transactions. This lays the foundation for the new on-chain and cross-chain paradigm for attestation of ownership and authorization in decentralized content distribution using digital tokens described in section 5.

3.1 Content Native Storage Model and Content Routing System

3.1.1 Content Objects

All content in the Fabric is stored as **Content Objects** consisting of a novel copy-on-write structure, aiming to provide a universal (type and size agnostic) content-native distributed storage solution that is fundamentally different than today's distributed file systems and cloud storage in a few key ways: 1) avoids duplication of storage or network bandwidth transmission as the content is re-purposed for various output versions; 2) provides flexible personalization of the media delivered

Figure 3: Content Object Structure and Versioning



("programmability"); and 3) includes intrinsic versioning and an ability to prove the validity of a piece of content and its version history.

A Content Object's data is stored in data containers called **Content Parts**. A Part is the fundamental unit of storage, network transfer and caching in the Content Fabric; it is immutable once finalized and identified by a hash value that is calculated across all of its content. Thanks to the use of a cryptographic hash function, the authenticity of a Part's data can be easily verified by recalculating the hash, and similarly an efficient Merkle proof can be calculated across a set of Parts serving as a unique and verifiable identifier for the entire object and stored on chain, described further in section 5. The hash also serves as criterion for data deduplication.

When data is ingested into the Content Fabric, it is automatically divided into Parts as needed in order to achieve a manageable and approximately constant Part size. Large file data is split up and stored in multiple Parts. Multiple small files are aggregated into a single Part, and in some cases Parts are created by transforming raw data on ingest.

User-provided Content Metadata is also stored in Parts. Even Fabric-internal data structures such as the list of data parts or content verification proofs are stored in Parts. Finally, the Content Object itself is a small data structure that simply references the other Parts by hash and is stored itself . . . as a Part.

Thanks to the hierarchical reference structure, Content Objects scale easily from small to very large. None of the employed structures impose a limit on size, neither for binary data nor for metadata. The reference structure also provides efficient and fast versioning of content. Creating a new version of a content consists in copying only the Content Reference structure, pointing back at the previous version's parts, and then creating new structures and parts for the pieces that change in the new version. For example adding a new file to an existing Content Object results in a new (set of) data parts, a modified subset of metadata, and updated internal structures. All existing file data (represented as existing Parts) and the unchanged metadata subset are not duplicated.

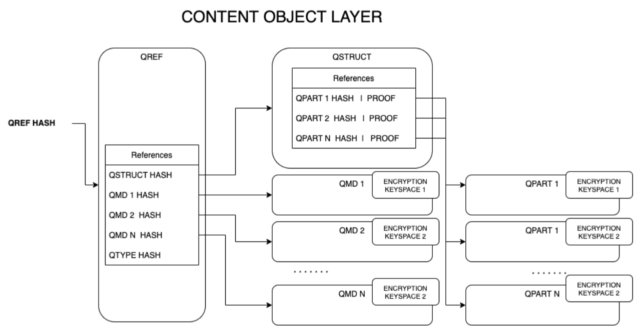
3.1.2 Content Object Detail

The Content Object consists of two data structures, the QREF and QSTRUCT. The QREF stores a hash of the elements that define the object, including a hash of the QSTRUCT, along with hashes of one or more metadata structures (QMD, QMD2), and hashes of the content object's 'types' QTYPEHASH. 'Types' are parts that store 'code' that can execute on binary raw data parts and metadata. The QSTRUCT itself is a table with references to all opaque data parts (storing opaque data such as raw data and/or code). The QMD stores structured content metadata associated with the object and used in operations on it.

3.1.3 Storage

All parts are stored on the file system of the Nodes in the network. Both QREF and the QSTRUCT are encoded in the Concise Binary Object Representation (CBOR) format, which is JSON-like and schema-less, but more efficient in size and faster in processing. The use of this standard encoding format facilitates efficient validation of the binary content, while the open format allows extraction of the metadata from it. Both QMD metadata structures as well as the opaque data parts can

Figure 4: Content Object Detail



be stored in designated encryption key spaces to support isolated unique encryption contexts. Finally, along with the binary parts, the QSTRUCT stores associated proofs that can be used in version verification.

3.1.4 Functions

This Protocol uses this unique Content Object structure for many fundamental features:

1. Locating the content object - looking up content objects by hash (see next section on content routing);

2. Programmable output - executing content code against content object binary data parts and metadata (see next section on just in time content);

3. Efficient "no copy" versioning, on chain - efficiently creating new versions of an object without duplicating data and storing in a blockchain ledger the QREF identifying that version. A new version is as simple as copying a the QSTRUCT for the previous version, updating the references to the new Parts, and recording the new QREF in the contract for the object on the blockchain.

4. Verification of the Content Object Metadata and Binary Parts - a checksum of the CBOR encoding of the QSTRUCT, and the checksum of each of its constituents (QMD, QTYPEHASH, and individual QPARTHASHes) recorded in a Merkle proof structure can efficiently verify if the data matches the expected checksum and provide tamperproof verification for readers.

3.2 Content Routing

The backbone of the Content Fabric is a novel content routing system that locates Parts throughout the fabric network using an original decentralized hashing algorithm. Unlike previous decentralized peer-to-peer networks built on DHT's the approach achieves low latency high bandwidth serving even as the system grows in number of nodes and content objects (to giant scale). Distributed Hash Tables "DHTs" were popularized almost twenty years ago in response to the growth of the original peer-to-peer file sharing networks, to provide a more scalable way to store and search content in a decentralized network of peers or "nodes" (more formally called "content addressable networks"). The essential idea of a DHT is that the nodes in the network are treated as the buckets in a distributed hash table that spans the entire network. The hashtable is keyed by each node's id and the values are any resource associated with or stored by that node, such as file hashes or key words. In this way, the node ID not only serves as identification of the node, but also as a direct map to a set of values to be located in the network, and the crux of the DHT's characteristics are the particular algorithm it uses to search its network to locate the node ID that can return the desired value in response to an application or user request.

3.2.1 Brief History of Content Addressable Networks and Decentralized Hashing

Over the years, and particularly in the heyday of the peer to peer network popularity of the late 1990s and early 2000s, many different distributed hash table algorithms were proposed and compared in the literature and implemented in practice, with varying strengths and weaknesses. Several of the better performing designs reduced search times, and one in particular, Kademlia [4] introduced an elegant way to partition the network by bits in the node ID and to fix local routing knowledge around a minimum distance function calculated on this ID using XOR.

Specifically, in Kademlia content is divided over nodes using each bit in a fixed length (eg 128-bit) node ID, and a routing distance is defined as the XOR distance between two node IDs. Nodes would specifically maintain routes to content on their "nearest" neighbor nodes and part search would explore to ever greater XOR distances to locate an unknown node id. The approach like several others ensures the maximum time to find a content is limited to O(log n) where n is the number of nodes in the network, is relatively easy to calculate, and has resilience to denial of service attacks.

However, while the content routing limits the maximum time to find a part as function of the size of the network (n), there is a severe practical limit to the number of nodes (or amount of content in the network) that can be stored and still serve content with low latency (under 500 milliseconds per request). Considering the routing protocol runs at the application layer (using UDP/TCP/HTTP) using multiple round trip times per routing step, and practical network round trip times in 10s to 100s of milliseconds, finding a part in a network of 10,000 nodes could easily take over 1 second to multiple seconds -- not acceptable for low latency “Internet TV”. Additionally as the network grows with new content and new nodes, content will either have to be frequently reshuffled to new nodes to keep even balance for storage, capacity, resilience etc. (costly in time and complexity) or grow unbalanced with new content only on new nodes. Finally, in Kademlia and similar previous DHT / CANs, there is no direct tie between the selected node (based on addressable/routing distance) and *performance* of the serving experience for the client. In other words whether the client is getting a high bandwidth stream with low time to first byte is not considered in the search selection at all, or in a very limited way, such as considering response latency of the node.

Consequently, overlay routing using DHTs in peer-to-peer networks has not yet demonstrated the ability to scale and ensure the latency and quality demands of live and professional video (“Internet TV”) and the practical results are that to date, no peer-to-peer/decentralized distribution system has yet achieved mainstream adoption for broadcast quality content over the Internet and instead most high quality OTT distribution systems today use a more traditional hierarchical CDN with edge caches, with at most a bit of “peer to peer” at the edge.

3.2.2 Node Identification, Part Location, and Network Partitioning for Scale in the Content Fabric

In the Content Fabric, nodes are each identified with a 32-byte ID (node ID), which is also their address on the embedded blockchain ledger (to be described in the security section). The content part hashes are sharded over the nodes using a novel, scalable partitioning algorithm that, with the routing algorithm, is designed to a) locate parts on their node with low latency (under 500 milliseconds) consistently, even as the number of nodes and content objects in the fabric grows, and b) not require moving around content as new nodes and content are added.

The partitioning algorithm has two global configuration parameters, a “level” (l) which defines the number of partitions and a “number of partitions per node” (np) which defines the number of partitions each node stores, and a static configuration of the number of copies of a partition (cp). Each partition is itself identified by an ID (4 bits for every partition level), and content part hashes are also represented as 32-byte IDs.

Content parts are assigned to be stored in a partition by matching a prefix in their part hash to the partition ID, where the length of this prefix is controlled by the current partition level of the network.

Similarly, nodes are assigned to store and serve a set of partitions by matching a prefix in their node ID to all partition IDs that are within (less than or equal to) an XOR distance calculation that is equal to the configured ‘np’.

For convenient calculation of the XOR distance, the 32-bytes that make up the node ID, the content part hash, and the partition ID are expressed as 32 pairs of hexadecimal characters (each hex character representing 4 bits). When parts are retrieved from the network, the fabric routing protocol locates the nodes that can serve the partition to which the part belongs (see next section), and uses the most favorable node to serve the part.

The key feature of the partitioning method is that as the number of nodes in the network increases to accommodate more content, the network dynamically repartitions with minimum “reshuffling” of existing parts such as no node needs to acquire new content (i.e. nodes only discard existing content that no longer belongs to the smaller partitions they are responsible for), and the routing protocol (see next section) does not become slower in finding parts.

Specifically:

The Fabric protocol starts with a global network configuration level L that controls the specificity of the partitioning of the content parts to node assignments. Each subsequent level includes the next hexadecimal character in the prefix of the part hash when making a partition assignment, and thus each subsequent level L raises the parts capacity by a factor of 16.

Given a number of partitions in a network = p, a number of partitions per node = np, and a number of copies = c, the network needs to have (p / np) * c nodes. For example a network that has 16 partitions, and maintains 8 partitions per node, and 7 copies of each partition, needs (16 / 8) * 7 = 14 nodes.

As those partitions start to fill with new content and the network needs to increase capacity, we can divide each partition space into 16 smaller partitions, increasing the number of partitions from 16 to 256, by increasing the level. Each existing node will shed a part of the now more specific partition space and new nodes will take on the new partition space. Assuming our part hashes are generally evenly dispersed over the partition space, we can introduce the new partitions and scale up the network with minimal moving of content and no renumbering of nodes or content, and still maintain the same redundancy of each partition.

3.2.3 Example Partition Assignment

For example, considering the partition ID:

0f 1a 66 aa 4d 5e 6f 7a ab bc cd de ef 76 e3 a8 44 98 b4 c5 11 00 34 dd 3d 47 a8 91 32 fa 01 12

Level 1 - uses “0” Level 2 - uses “0f” Level 3 - uses “0f1”, etc.

Consider a content part with the hash starting with

0f 1a 66 aa ...

That is assigned to ‘level 3’ partition ID:

0f1

And a node ID with a hash starting with

0f 1a 00 00 ..

The part will stay on this node as the network grows from Level 3 through Level 4 because the XOR distance is 0:

Level 3 : XOR distance (0f1, 0f1) = 0 Level 4 : XOR distance (0f 1a, 0f 1a) = 0

At Level 5 the XOR distance calculation between the nodeID and partition ID yields a value of “6” :

XOR distance (0f 1a 6, 0f 1a 0) = 6

Assuming the number of parts configuration of the network is 6 or greater, the partition will stay on this node.

Finally at Level 6 the partition ID and node ID diverge beyond the XOR distance constraint and the partition is shed from this node and assumed by a new node :

XOR distance (0f 1a 66, 0f 1a 00) » 6

Note that the network grew by 3 orders of magnitude (16*16*16) before we had to make any changes to where this content is located.

3.2.4 Low Latency Content Part Routing

The dynamic partitioning of the nodes and content combines together with a novel content routing that allows for parts to be looked up with low latency, even as the network grows. The Content Routing algorithm takes advantage of a fast direct location of eligible nodes containing the desired content parts. The Protocol aims to provide to applications (and in turn, end users) two basic primitives for content publishing and retrieval: GET and PUT. Each primitive takes a content part hash as its primary argument. The Protocol is then responsible to either locate and return the part (GET) or to publish the part (PUT) to the appropriate nodes based on the partitioning algorithm. This process is seamless to the client/user, is fully symmetric in design (for both PUT and GET processes for scale), and the network can optionally be price-incentivized to provide different classes of service. [1] [5]

3.2.5 Joining the Network

When a node first joins the fabric, it generates its node ID, the current level and the number of partitions per node for the network, and uses these to compute the partition IDs it will cover, and starts a background process for acquiring the content parts it is responsible for. Since the new node is now ‘online’ and can receive requests for content parts it hasn’t yet acquired, the node will also acquire parts on demand through the same mechanism (GET), described below.

3.2.6 Reading a Part "GET"

When a node receives a request from a client to GET a new content part, it checks if it has the part (computing whether the part is in one of its partitions), and if not, transmits the request for the part to the group of nodes corresponding to the part’s partition ID. From the responses, the requesting node selects one of the responding nodes (and can rank the selection based on a quality score) and streams the part from the selected node. One of the great benefits of this design is that over time nodes accumulate popular parts, and when a part is not available on the node locally, the pipelined streaming retrieval of parts and just-in-time processing (next section) ensures low latency serving to the end user nevertheless. Finally, token economics can be used to incentivize participation of nodes in underserved areas.

3.2.7 Writing a Part "PUT"

PUT-ing a new part into the network uses the same process as GET-ing a part. When a client makes a request to a node to PUT the part, the node first transmits the part to the group of nodes associated with its partition ID, and when enough nodes respond to meet the replication requirement for the network, it transmits the new part to these node(s). This publishing and update to content results in a new Content Object version hash, which is committed to blockchain via a transaction on the Content Object’s base smart contract, and couples with the content encryption system described in Section 5.

[1] We have implemented a novel Machine Learning algorithm that allows for continuous relative scoring of nodes and node-to-node paths to achieve target qualities of service in part serving and Fabric output. The algorithm selects nodes and paths in the overlay network that are predicted to serve the requesting client with low latency ($<$ 500 ms time-to-first-segment) and high bandwidth, at the client’s bottleneck bandwidth capacity.

3.3 Just-in-Time Composition of Media, Metadata, and Code

The fast Content Routing algorithm allows for parts to be found in the network in real-time and supports just-in-time transformation, transcoding and packaging, and extensible A/V, dynamic data, and "active" processing within the CFP. End clients request "representations" of content objects which are effectively built on demand, from Parts. This mechanism is a novel break from traditional active web, video, and gaming pipelines, which create output variations a priori and transmit finished or near-finished versions as files (or chunks thereof) through networks, caches and end systems, usually between multiple applications. Instead, in the CFP, parts are fetched and composited to create unique output variants just-in-time in response to read requests from end clients.

This removes byte duplication over the network and in storage, eliminates significant I/O otherwise consumed in reading and writing pre-made file variants, eliminates latencies through the pipeline for any content serving, and allows individual and personalized variants to be built not just at the edge but with minimal data. A key example is low latency live 4K video streaming to broadcast client audiences with 3-5 seconds latency sustained globally (see next section).

The efficiency gain is significant (a >10-50X reduction in the host and network requirements to generate the same net media output to clients as compared to traditional architectures. This process uses code type Parts described in Section 4.1 and a just-in-time (JIT) compilation subsystem that allows built in and pluggable "bit code" deployed as source code (stored in code Parts) to be compiled to an intermediate representation language, e.g., abstract syntax tree (AST) that is further compiled to machine code at runtime.

The CFP implements a bitcode execution environment that can compile and execute this AST code in combination with metadata and binary parts JIT. The AST is managed using LLVM[3] and WASM[10] technologies. Applications of this mechanism are wide.

3.3.1 Low Latency, Scalable Transcoding and Streaming

The interface between the Fabric and the bitcode modules loaded JIT allows for a Fabric method to call into the module, and the module to call back into the Fabric method. This calling context can facilitate reading/writing content and metadata to/from the Fabric, and can facilitate a security sandbox for both authorizing code operations and metering their use of system resources (e.g., for compensation and charging).

The figure below shows this applied for just-in-time (JIT) transcoding, packaging and transport within the content fabric to generate HTTP Adaptive Bit Rate streaming (e.g. DASH/HLS CMAF fragmented mp4 segments). Entities participating include a client, a fabric node, another fabric node having a desired content object part, and an origin fabric node in the case of live streaming. The client access a Fabric URL requesting a playout representation.

The response returns to the client the URL to request from a local egress node IP the playlist and then a segment of the video. If the egress node finds the requested segment in its cache, the segment is returned from the node to the client. If the node is not able to find the requested segment in its cache, the node determines if a corresponding mezzanine-type part, a higher bit rate version of the segment, is stored locally in its cache. If the mezzanine version is found, the egress node transcodes the part and returns the requested segment to the client (and stores the generated segment in its cache). If the requested segment and the mezzanine version of the segment are not in the egress node cache the node can send a GET for the mezzanine version to a node having the part, using the content routing described previously.

This source node then transmits the mezzanine part to the egress node. The egress node can begin transcoding the received mezzanine version into individual segments 1, 2, 3, 4 ... N as it arrives. As the segments are transcoded, the egress node sends the transcoded segments to client, where the segments can be desirably buffered ahead of playback.

Note that when this process is perfectly pipelined and the transcoding is faster than real time, the net latency end-to-end is only the propagation time for the mezzanine part between the nodes (on the order of 500 ms or less for global internet).

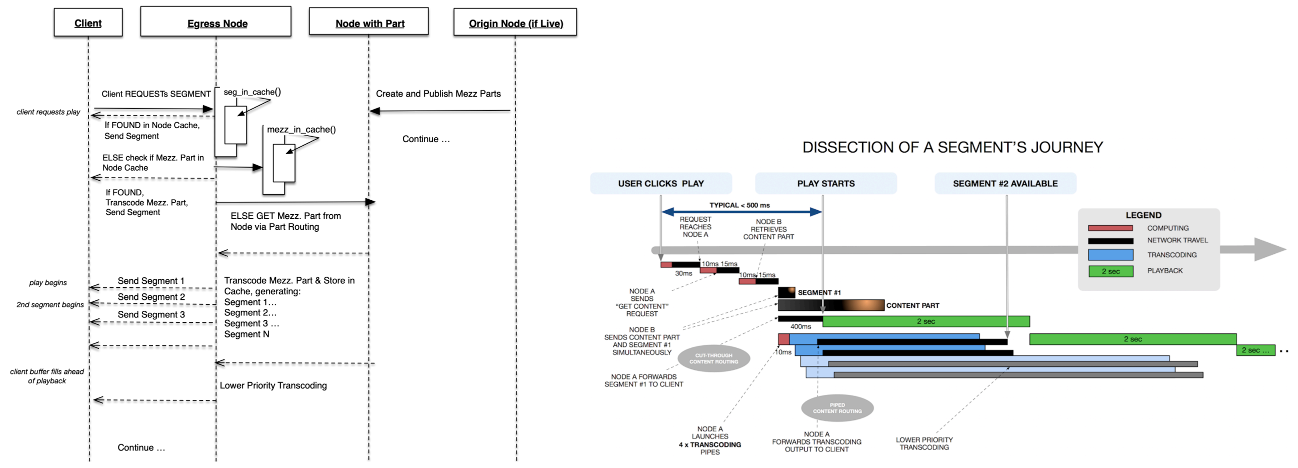
3.3.2 Live Streaming

In the case of live transmission, the mezzanine parts are created and published from the ingress fabric nodes while the request process is occurring, and in a similar pipelined manner, and with the content object structure pre-established at set up time. In this case the bitcode transcodes the incoming live stream to mezzanine parts and PUTs them to the network in a pre-existing content object structure. The latency of this process is deterministic and so the Fabric synthesizes a playlist that has the live edge segments advertised pulled back in time only the duration of the end-to-end latency of this pipeline. Thus high resolution low latency live delivery (3-5 seconds from source) low complexity, low cost, and globally achievable in the decentralized network.[6]

3.3.3 Localized and Multi-Versioned Content from Single Source

The same JIT bitcode mechanism is used to generate new output variants without having to create additional copies of a mezzanine source, such as unique language versions, territory versions, repairs, new playout device formats, etc. The core logic to create an output variant is written as a metadata value in a content object using a JSON object; this object, termed

Figure 5: Just-in-Time Transcoding and Streaming Pipeline


Client
Egress Node
Node with Part
Origin Node (if Live)
Client REQUESTs SEGMENT
seg_in_cache()
Create and Publish Mezz Parts
client requests play
If FOUND in Node Cache, Send Segment
mezz_in_cache()
Continue …
ELSE check if Mezz. Part in Node Cache
If FOUND, Transcode Mezz. Part, Send Segment
ELSE GET Mezz. Part from Node via Part Routing
play begins
Send Segment 1
Send Segment 2
2nd segment begins
Send Segment 3
Transcode Mezz. Part & Store in Cache, generating: Segment 1… Segment 2… Segment 3 … Segment N
client buffer fills ahead of playback
Lower Priority Transcoding
Continue …
DISSECTION OF A SEGMENT'S JOURNEY
USER CLICKS PLAY
PLAY STARTS
SEGMENT #2 AVAILABLE
TYPICAL < 500 ms
REQUEST REACHES NODE A
NODE B RETRIEVES CONTENT PART
LEGEND
COMPUTING
NETWORK TRAVEL
TRANSCODING
PLAYBACK
30ms
10ms 15ms
10ms 15ms
NODE A SENDS "GET CONTENT" REQUEST
SEGMENT #1
CONTENT PART
2 sec
400ms
NODE B SENDS CONTENT PART AND SEGMENT #1 SIMULTANEOUSLY
2 sec
2 sec …
10ms
CUT-THROUGH CONTENT ROUTING
NODE A FORWARDS SEGMENT #1 TO CLIENT
PIPED CONTENT ROUTING
NODE A LAUNCHES 4 x TRANSCODING PIPES
NODE A FORWARDS TRANSCODING OUTPUT TO CLIENT
LOWER PRIORITY TRANSCODING

an "offering," is a grouping of the key names for the audio and video tracks, starting time code, duration and parameters, for example, to create an output version.

For example, consider delivering a language localized consumer stream where the package specifies multiple language versions. When a user requests to stream a DASH or HLS version of a content object in its target language version, the bitcode module reads the appropriate metadata and the metadata pointing at the constituent video and audio parts, reads the content of these parts, and generates a manifest file that advertises the segments to be streamed (also synthesized). The manifest and the segments can be built on-the-fly by the bitcode. In this case, the bitcode drives audio/video processing modules to perform the scaling and bitrate transcoding to generate only the segments the client requests as the requests are made.

3.3.4 Permissioned "Branching" Interactive Content

The bitcode mechanism can be used to synthesize segments in response to view switching requests in a virtual multi camera stream, or to serve embedded HTML static and dynamic files to personalize an interactive AR experience, micro-grames and digital twins/avatars. As we will see in Section 6 this can be combined with the Fabric's policy based authorization to verify ownership of digital tokens as a criteria for serving the representation. Web 3 projects can use these capabilities to create token permission-ed and interactive "metaverses".

3.3.5 Trait-Specific Publishing

The bitcode mechanism allows function updates to be updated or extended without changing or taking down the Fabric, and bitcode stored in content object parts can be versioned and updated without having to change other parts of the pipeline.

3.3.6 Automatic Marking, Clipping and Tagging

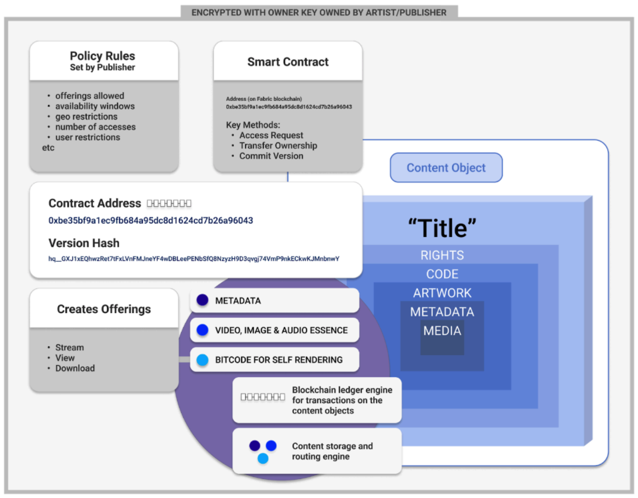
Bitcode can also operate on other content parts such as raw data and metadata for personalizing or customizing output such as creating 1:1 content variations for NFTs (image, video, or metadata traits), applying custom watermarks, "clipping" full length content with no offline processing, and automatic media tagging and classification, e.g. via ML. In this case the ML model that provides the video classification code runs inside a container loaded by the bitcode sandbox, illustrating a breadth of possibilities for creating intelligent content coding pipelines.

3.4 Content Fabric Blockchain Protocol

The publishing, update, and access to content within the Fabric protocol is an application-specific blockchain protocol and is built upon a full-stack chain model. Key concepts are as follows:

- The life cycle of a Content Object is controlled by a decentralized publish and commit process that results in the entire content version history being stored in the blockchain, and all access to write or read content being authorized through a blockchain transaction (either off or on-chain), and recorded as events on the blockchain.
- All content parts are encrypted for the owner and under the control of a blockchain contract that is part of each object. All actors in the system - content owners, content accessors, and nodes - identify and authenticate themselves in the network using blockchain authentication, and all content encryption uses a trustless proxy re-encryption. Taken

Figure 6: Active Content Object Components


ENCRYPTED WITH OWNER KEY OWNED BY ARTIST/PUBLISHER
Policy Rules
Set by Publisher
offerings allowed
availability windows
geo restrictions
number of accesses
user restrictions
etc
Smart Contract
Address (on Fabric blockchain)
0xbe35bf9a1ec9fb684a95dc8d1624cd7b26a96043
Key Methods:
Access Request
Transfer Ownership
Commit Version
Content Object
Contract Address
0xbe35bf9a1ec9fb684a95dc8d1624cd7b26a96043
Version Hash
hq__GXJ1xEQhwzRet7tFxLVnFMJneYF4wDBLeePENbSfQ8NzyzH9D3qvgj74VmP9nkECkwKJMnbnwY
"Title"
RIGHTS
CODE
ARTWORK
METADATA
MEDIA
Creates Offerings
Stream
View
Download
METADATA
VIDEO, IMAGE & AUDIO ESSENCE
BITCODE FOR SELF RENDERING
Blockchain ledger engine for transactions on the content objects
Content storage and routing engine

together multiple owners can safely store their content in the same network and without trusting the nodes in the network.

- Access control to Content Objects is governed by a verification against an extensible policy that is both hashed and referenced by the object's contract, which provides on-chain verification of any attestation by the client and authorization to any of the object's data or dynamic offerings (Figure 6).
- Content is also self-verifiable in this protocol: the cryptographic hash of each object is a Merkle hash of the parts of the object, and is committed in a contract transaction whenever a version change occurs, ensuring that the "parts" that make up the object are verifiable throughout the network. Any reading client can verify content by calculating a fast version proof.

In the following sections we will details the content security model and the application of these principles in new multi-chain / cross-chain protocol for authorization and distribution of content via digital tokens.

4 Content Security Model

4.1 Motivation

The present digital content supply chain depends on multiple trusted technological systems -- systems that are relied upon to safeguard content from being stolen or tampered with by unauthorized parties -- as content passes from owner to consumer, and an equally numerous and complex set of bespoke rights and royalty management systems that collect and distribute revenues. At the same time, content proliferation via secondary ownership, re-trading and derivative work is growing in popularity, and yet the prevailing Internet economic model provides only limited attribution and remuneration to content creators and publishers, and accrues most of the revenues to the trusted platform. Both free ad supported social content and paid subscription streaming have shored up the centralization of control in the supply chain around a few large companies that have been able to scale the delivery channel to the consumer and are the "trusted" point in the content transaction [9]. This has entrenched an economic paradigm in which only the most successful content creators and publishers are significantly profitable.

Blockchain technology offers a new paradigm for protecting and controlling access to content distributed by systems and networks that are not trusted, and not centrally managed by a single organization, and for scaling content transactions person-to-person with intrinsic trust. Blockchains systematically distribute trust over a network of actors that are incentivized economically to behave in a "trustworthy" manner by carrying out a common protocol that is safe and scalable by virtue of its cryptography and mathematical design. What's more, non-fungible and fungible digital asset standards provide for scalable attestation of ownership, provable access statistics, and decentralized payments and transactions, and are now becoming an accepted and practical tool for the mainstream content economy. This creates a unique opportunity for an application-specific content blockchain.

4.2 Blockchain Content Access Control in the CFP

4.2.1 Overview and Comparisons

The CFP builds upon these features of decentralized blockchain ledgers by backing all content access control -- operations to create, update, or access content -- with blockchain transactions and policy attested on chain executed through a native ledger embedded in the CFP stack.

The system ensures that all parties are authentic, and its consensus ensures that only valid (authorized) transactions on the content can be carried out.

The current main.net955305 implementation runs a fork of Ethereum using a proof-of-authority (PoA) consensus executed by a subset of nodes in the network as designated block validators, provided by various stakeholders (media companies, content publishers, infrastructure providers and eluv.io). A public network implementation using the Polkadot substrate framework [2], W3F inter-blockchain messaging and delegated proof-of-stake validation is currently being built.

The CFP is is open and extensible -- new nodes may be added at any time and without impacting the running of the network. At present time the majority of nodes are owned by Eluvio, Inc. although there is an active program of third party node contributors in the PoA network and this participation is expected to grow with the public network.

The CFP couples control over the content's modification and access to the blockchain, while maintaining scalable storage and distribution outside of the blockchain. Most Web3 content infrastructure in practice at least partially defers access control and distribution to external/3rd party systems running separately from the blockchain ledger. The difference is especially obvious for content sold as or permissioned by NFTs. The CFP automatically mints NFTs (ERC 721 and 1155) from any content object as a native function and as a uniquely versioned content object (with its own unique blockchain contract address). Unlike other blockchains for NFTs, the CFP has a strong coupling of the media asset and the token contract, and verifies token ownership via the on chain policy associated with the content object.

Specifically, the content access process evaluates the terms of a policy that is tied to a smart contract for that content, including both access to data as well as access to decryption keys. Thus, there is an inseparable on chain linkage between the the content's identity, the integrity of its source and representations, and access to these by users.

4.3 Functions of the Ledger

The 'ledger' is charged with three essential functions:

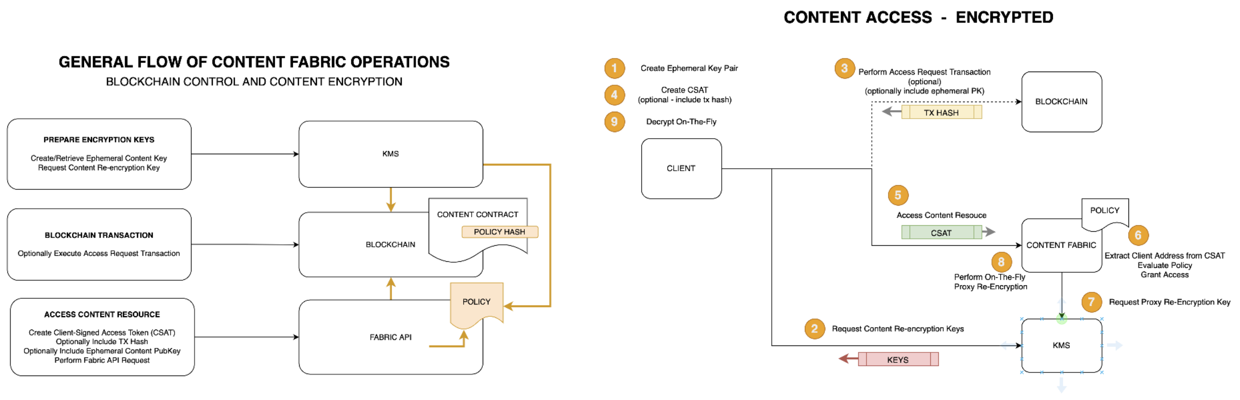
- **Providing the authoritative 'directory' of all content including the only trusted reference to the list of versions of each content object and the final verification 'hash' (the ground truth) for each of these versions;**
- **Optional execution of on-chain 'access control' logic allowing users to read and write content** - on-chain contract terms enforcement and Commercial terms reconciliation (payments and credits);
- **Attestation of all content access operations** - Content access is recorded by content fabric nodes and periodically packaged into a special content object. The hash of these usage parts is committed on blockchain. Optionally when content access requires on-chain enforcement, the corresponding transaction ID is also recorded in the content parts and can be correlated to the contract terms that it enforces (either on-chain or off-chain).

4.3.1 System Actors

All participants in the blockchain fall into these two categories:

- **Account owners** - Account owners are primarily users in control of their actions against the blockchain (for example creation or update of content, accessing or viewing content, etc). Applications operated by people or automated processes are also account owners. These applications are trusted by the people who run them to do what they were constructed to do and they are trusted to operate the blockchain accounts they have been given access to. An "account owner" is identified by its 'address' on the blockchain and owns a public / private key pair that it uses to sign its transactions against the blockchain.
- **Contracts** - On the other hand Contracts are entirely autonomous participants - they operate exclusively based on their 'code', and thus always behave deterministically unless their code is changed and redeployed to the chain. (Since the present implementation of the mainnet955305 network is an EVM based blockchain we refer to programmatic autonomous execution agents in the architecture as "Contracts" but the same functional role can be achieved with other smart contract style technologies.) A contract written to pay credits to each user who supplies a particular record signed by a signature accepted by this Contract will forever behave the same way and pay the credits when the signature is matched, and decline to pay otherwise. A contract is identified by its 'address'; it will only have an address if it has been successfully deployed by its creator. The creator is known because the creation of the contract is done in a 'transaction' where the 'from' address is the contract creator. The life cycle of any content object and the base utility functions of the Fabric are implemented through a set of base Contracts.

Figure 7: Generalized Flow of Content Fabric Operations and Detailed Access Control Protocol


GENERAL FLOW OF CONTENT FABRIC OPERATIONS
BLOCKCHAIN CONTROL AND CONTENT ENCRYPTION
PREPARE ENCRYPTION KEYS
Create/Retrieve Ephemeral Content Key
Request Content Re-encryption Key
BLOCKCHAIN TRANSACTION
Optionally Execute Access Request Transaction
ACCESS CONTENT RESOURCE
Create Client-Signed Access Token (CSAT)
Optionally Include TX Hash
Optionally Include Ephemeral Content PubKey
Perform Fabric API Request
KMS
BLOCKCHAIN
CONTENT CONTRACT
POLICY HASH
POLICY
FABRIC API
CONTENT ACCESS - ENCRYPTED
1 Create Ephemeral Key Pair
4 Create CSAT (optional - include tx hash)
9 Decrypt On-The-Fly
3 Perform Access Request Transaction (optional) (optionally include ephemeral PK)
BLOCKCHAIN
TX HASH
CLIENT
5 Access Content Resouce
CSAT
POLICY
CONTENT FABRIC
6 Extract Client Address from CSAT Evaluate Policy Grant Access
8 Perform On-The-Fly Proxy Re-Encryption
7 Request Proxy Re-Encryption Key
2 Request Content Re-encryption Keys
KEYS
KMS

4.3.2 Base Fabric Entities and their Contracts

The fabric implementation uses a few key concepts to scope the mechanisms by which content is managed:

- **Node Providers** - Each node provider is represented by a dedicated contract. A node provider registers one or more Content Fabric Nodes to participate in the Fabric. Each node has a blockchain account, represented by its public/private key pair and is responsible for running the Fabric software. A Node Provider submits periodic usage records and is paid for the services provided by its nodes.
- **Tenant** - A Tenant is the contractual entity representing an organization or an individual using Content Fabric resources and is represented by a dedicated Tenant contract which periodically pays the charges for the Content Fabric services used by its resources. It is also the containing entity for all content belonging to this tenant.
- **Content Space** - The Content Space in the entity that sets the base rules binding Node Providers and Tenants, including the rules for accepting new Content Fabric Nodes, creating new Tenants, and the rate of the service charges.

  A Content Fabric network may operate as a single, global Content Space or support multiple Content Spaces.
- **Fabric Users** - Each user has a blockchain account represented by its public/private key pair.
- **Library** - A repository of Content created inside a Tenant, setting the policies for how all of its containing Content Objects behave.
- **Content** - The direct representation of a set of digital assets (for example a movie, all its media tracks, associated artwork, metadata, etc.). Content is created inside a Tenant Library and is represented by either a dedicated contract or an entity contained in the Tenant contract.

4.3.3 Logical flow of a Content Object Lifecycle through the Ledger

As shown in Figure 7, operations on content objects in the Fabric follow a two part flow as follows:

First **creating an access token of appropriate scope, signed by the authoritative entity** : Either (1) creating a Client-signed access token attesting to the identity and context of the request signed by the client's private key using a direct or custodial signing operation, optionally including a transaction ID when on-chain authorization is required OR (2) creating an Editor-signed access token or a delegated Authority-signed access token where the signer is an owner or editor of the content or an authority specifically listed and trusted by the tenant and content object contract.

Second, **making an authorized API call on the Content Fabric using this access token**.

4.3.4 Content Access Scopes

The base Content Object contract implements three role scopes under which operations on the objects are executable, and for which access tokens are signed:

1. Client Signed - The client signed token is a JSON payload that includes a user identity (address) and an optional extensible context section that can include any application-specific data needed for the evaluation of the payload or authorization, such as details from the application context or user agent. This token is then signed by the client's private key or 3rd party wallet or custodial signer. The current implementation supports packing of the payload using CBOR, JSON, compressed CBOR, or compressed JSON for flexibility.

2. Editor Signed - Used by back-end operations that operate content on behalf of the content owner, signed by a key with Editor role, typically the owner address or an address specifically listed as an editor in the tenant or content contract.

3. Authority Signed - Signed by an Authority service, a third party service that has been empowered by the owner to act on the owner's behalf, and specifically listed as an authority in the tenant or content contract.

The blockchain-based access control is combined with the content encryption system described in the following section.

4.4 Detailed Content Security Flows

4.4.1 Prerequisites

Publishing a Content Object into the Content Fabric assumes a Tenant entity has been created within a Content Space. A tenancy is organized into Content Libraries, and Content Objects are created inside a Content Library.

4.4.2 Content Creation

The creation of a new content object and updates to existing objects involves the Content Owner ("Alice") (or her application) carrying out the following steps:

1. Executing a blockchain view call on the Library Contract to look up the possible content types and their required security groups offered by the Library.

2. Executing the blockchain call CreateContent on the corresponding Library Contract passing the content type and the chosen security groups. This creates a Content Object entity and returns a transaction ID and the Content Object ID.

3. Calling Content Fabric API 'EditContent', passing in an access token signed by the creator or any address with "Edit" rights on the content, containing the Content Object ID and optionally the transaction ID obtained above. The Content Fabric API returns a content 'write token'.

4. Generating the Content Encryption Key Set: an AES symmetric key and a proxy re-encryption public, private key pair (AFGH) for each security group.

5. Encrypting the Content Encryption Key Set for three parties, the Owner, the fabric's Key Management Service and any Owner Delegates, using their respective public keys.

6. Calling the Content Fabric API SetSecurityMetadata to store this data mapped to these three entities.

7. Uploading content parts to the Fabric (both binary blobs or structured metadata), encrypting each Part on the fly with the AES content key first, and then using the AFGH key.

8. Calling the Content Fabric API 'Finalize' passing in the content write token. The Content Fabric API closes the write token and returns a new 'content hash', which effectively represents a version of the content, uniquely identified by this hash. [Note that each invocation of the 'Finalize' API returns the potential 'content hash' if a version of the content were to be finalized without further modifications.]

9. Executing the blockchain call 'commit' passing the new 'content hash' obtained above.

10. Finally, the Node supporting the operation publishes the new content parts to the rest of the network, obtains an official confirmation from the nodes that received the new content, and executes the blockchain call 'Confirm' on the content object, passing in the final content hash, and evidence of the confirmations received in the previous step, and signing the transaction with the Node's key to prove that this Node supported the operation.

Updating content in the fabric follows an analogous process, except we skip steps 1 and 2 and instead executing the blockchain call 'WriteAccessRequest' on the Content Object to authorize the update. Steps 3-9 are the same.

4.4.3 Content Consumption

The consumption of content from the Fabric by the Content Consumer ("Bob") involves the Consumer (or his application) carrying out the following steps:

1. Creating an Ephemeral Key Set consisting of an AFGH key.

2. Calling the Content Fabric KMS API to retrieve a proxy re-encryption key, passing in the ephemeral key obtained above. The Content Fabric KMS API returns a set of keys: an AFGH proxy re-encryption key and an AES symmetric key.

3. Optionally executing the blockchain call ‘AccessRequest’ on the Content Object’s Blockchain Contract, and optionally passing in the ephemeral public AFGH key and retrieving the blockchain transaction hash.

4. Creating a client-signed access token (CSAT) including the consumer’s (“Bob”) full public key and optionally the transaction hash obtained above, signed by the consumer’s private key (“Bob”).

5. Calling the Content Fabric API ‘ContentOpen’, passing in an access token signed by the Consumer.

6. The receiving Node calls the Content Fabric KMS API passing in the access token above. The KMS verifies the request by verifying the access token (Consumer’s signature and authorization against any policy for the Content) and then generates (a) the proxy re-encryption key using the AFGH key in the Ephemeral Key Set, and returns the re-encryption key to the Node, and (b) an encrypted version of the content AES key, encrypted with the Consumer’s public key (“encrypted key blob”).

7. The Node uses the re-encryption key to re-encrypt on-the-fly the content from the original AFGH key space into the consumer’s ephemeral AFGH key space as it returns the data to the caller.

8. The Consumer reads the re-encrypted content delivered by the Node and the encrypted key blob. The consumer is now able to decrypt the content as follows:

9. Extract the AES content decryption key from the encrypted key blob using its private key. Decrypt using its ephemeral AFGH secret key. Then decrypt the result using the AES content decryption key obtained above.

4.4.4 Proxy Re-encryption in the Fabric’s Content Security

Unlike traditional content management systems, the CFP is designed to run in a distributed, trustless environment. Security assumptions are different: in particular, it is not valid to entirely delegate content security to the content nodes themselves. The CFP achieves strong content security through a two-layer encryption strategy that merges traditional encryption with proxy re-encryption. Secure data is published encrypted with a set of keys generated and stored by the content publisher. Two distinct sets of keys are generated for each piece of content by default: a symmetric key and a public / private key pair for proxy re-encryption. The symmetric key and algorithm is currently AES-128. The proxy re-encryption is implemented with a pairing-friendly elliptic curve (currently BLS12-381), which provides the desired security level with good performance. While each form of cryptography independently provides strong security guarantees, the two distinct key sets are used together to implement the trustless model. The symmetric content keys are encrypted for each content owner address and stored with the content object. Additionally, the keys can also be encrypted for an independent online system (the Content Fabric Key Management Service), and also stored with the content object.

When a user is authorized to read/view a content object, the symmetric key is transmitted directly to the authorized user. This key by itself is insufficient to decrypt the data and no useful information can be recovered with just this key. To perform the proxy re-encryption the authorized user generates their own ephemeral set of BLS12-381 keys. The Content Fabric KMS then creates a re-encryption key based on the original content owner’s key and the content-specific key of the end user. This key is then transmitted to one or more nodes in the Content Fabric. These nodes then proxy the encrypted data, using the provided re-encryption key to transform the data in realtime into the target key of the end user. The end user then first decrypts the symmetric AES key with their private key, then decrypts the first encryption layer using the ephemeral BLS12-381 key and finally the second encryption layer with the symmetric AES key. This form of two-tier encryption guarantees that the keys the end-user controls cannot be used to directly decrypt the original source data that is stored in the fabric.

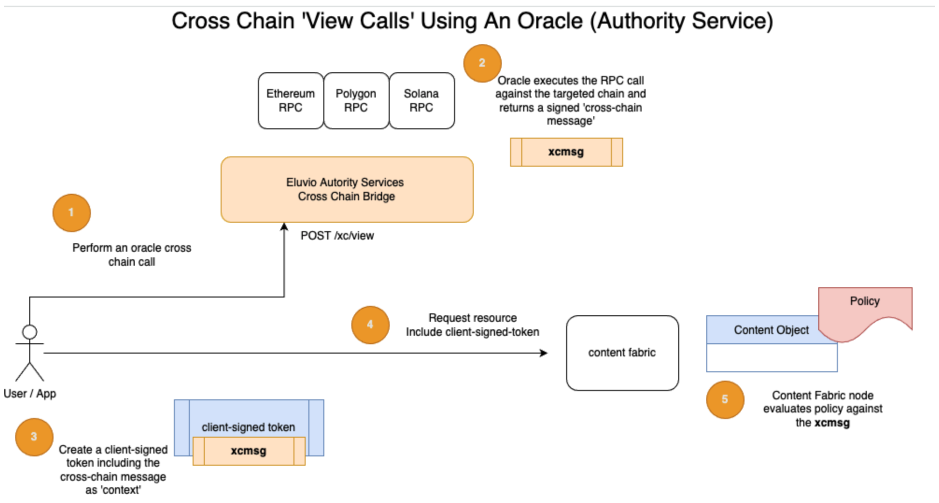
Key generation, storage and management are all performed automatically and transparently on behalf of both the content owner and content consumers by two content fabric components: the Content Fabric nodes and the Content Fabric KMS. All encryption and decryption is performed in real-time while data is stored and retrieved from the fabric. While the Fabric’s two-tier approach adds some additional overhead, performance is assured with optimal implementation of the underlying algorithms. A highly efficient, scalable library is used for all server-side processing and the same library is cross-compiled into Web Assembly (WASM) to execute in client software, including all modern browsers. Production use has shown that the WASM-based client decryption is fast and supports secure, high-resolution (4K) low latency streaming video (typically <500ms latency per 2 sec video segment, faster than real time).

5 Cross/On Chain Attestation of Content Ownership and Authorization

The access token and authorization protocol described in section 4.4 can be applied and extended to verify digital token ownership as a generalized criteria for authorizing any content object access or representation. This has widely applicable use including all of the following:

- “token gating” exclusive content experiences based on utility NFT ownership;
- non fungible tokens for “rental”, “purchase”;

Figure 8: Cross Chain Content Authorization Protocol using Digital Tokens


Cross Chain 'View Calls' Using An Oracle (Authority Service)
Ethereum RPC
Polygon RPC
Solana RPC
2
Oracle executes the RPC call against the targeted chain and returns a signed 'cross-chain message'
xcmsg
Eluvio Autority Services Cross Chain Bridge
1
Perform an oracle cross chain call
POST /xc/view
Request resource Include client-signed-token
4
content fabric
Policy
Content Object
User / App
5
Content Fabric node evaluates policy against the xcmsg
3
Create a client-signed token including the cross-chain message as 'context'
client-signed token
xcmsg

- fractionalized ownership of content and distributed disbursement via fractionalized fungible tokens;
- on chain micro payments models for media consumption enforcement of new re-trading and derivative works rights where the token owner has the permission to create and publish new work from the same content;
- fractionalized payments from content sales, e.g. to promotors, sponsors, and multiple creators.

We believe this system of inter-releated technology capabilities can be a foundation for an on-chain creator economy. Below we propose a generalized protocol that builds on the CFP for on chain and cross chain verification of ownership of digital tokens in content distribution and re-use. The protocol can be carried out entirely on chain, as described in our specification, or supported through API Authority Services. It utilizes inter-blockchain messaging principles from the Web3 Foundation (W3F) and is independent of the blockchains executed upon, although the first implementation has been created for Ethereum mainnet.

The protocol builds on the client-singed access token principles to extend the payload in the token with a cross chain message signed by an Oracle service. The Oracle service acts as an intermediary delegate between an application and a foreign blockchain and implements a signed cross chain message (xcmsg) against a target. That message is the result of executing a contract method on the target chain that ascertains token ownership for an owner address, such as the balanceOf method on ERC-721 NFT contracts.

As shown in Figure 8, the end user application makes a request to the Oracle service with the specific content operation scope of interest, such as “xco/view” . The Oracle service makes the corresponding transaction on the target blockchain via its RPC interface, and returns to the user the signed cross-chain message; the application embeds this into the access token payload, and then signs the token presented to the CFP API accessing the object. The token payload is evaluated against the object policy and if passed, the requested operation such as reading an object or its representation is authorized. This protocol can be applied within the CFP in all manner of cases that may verify token ownership on any chain to authorize Content Object offerings (see Figure 9).

6 Tokenomics

A tokenized content economy consists of a content layer and a business layer. Fundamentally these two layers have to be strongly correlated on chain. However all current blockchain-based content businesses address only the business layer and leave the content layer entirely decoupled, often using conventional cloud platform services.

The Content Fabric Protocol addresses this exact topic by providing the full function of the content utility on-chain, effected through the circulation of the ELV digital token. This allows the business layer to use common fungible and non-fungible tokens to implement content commerce and ownership.

Content access based on token ownership can result in token payments against project contracts and back to the project stakeholders. While there are many variations in dividing royalties and revenue this model allows for flexibility across all kinds of creative projects and stake holder terms.

The next section explains the ELV digital token, and its role in the Content Fabric utility operations.

Figure 9: Cross Chain Content Authorization System View

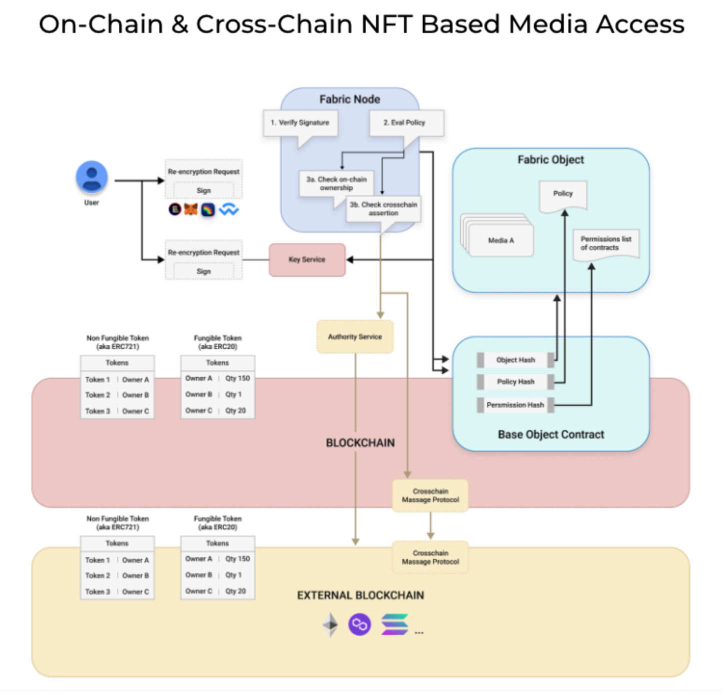
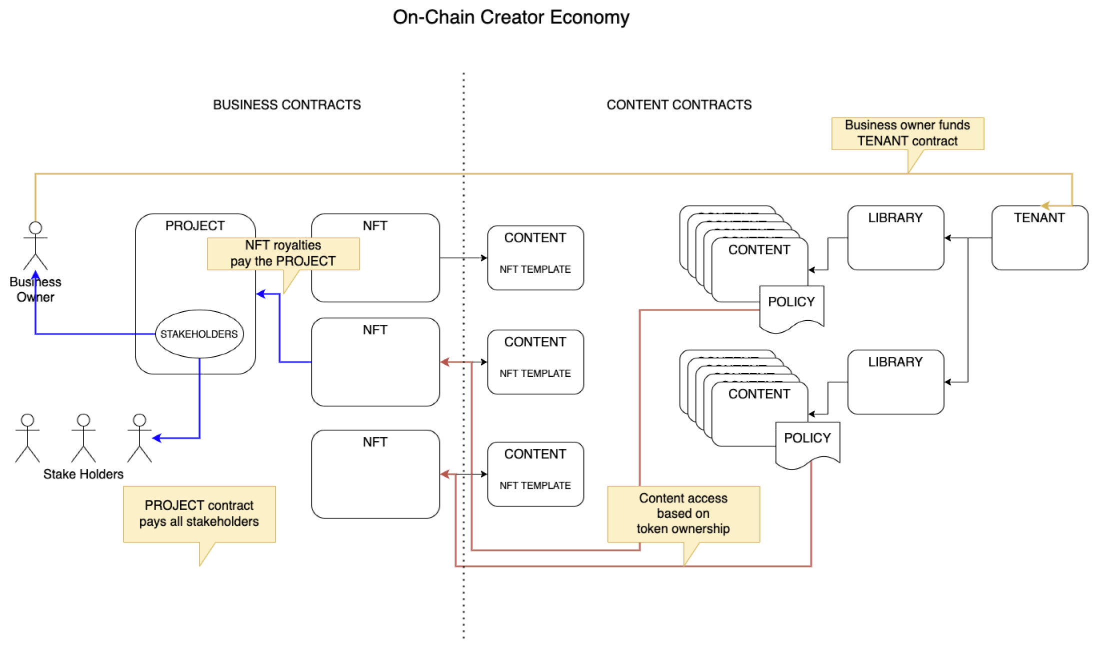


Figure 10: ELV Transaction Flow



Utility Functions and Definition of Terms

As an **application-specific blockchain** providing just-in-time decentralized content creation, storage, and distribution platform, the Content Fabric provides several essential utility functions for digital media. These include, for example, content creatioin and publishing, encrypted content storage, transcoding (to generate consumable output media), egress bandwidth distribution for streaming, interactive application, and file content; internal transmission, minting of tokens (NFTs and fungible tokens), and issuing/redemption of one-time and persistent passcodes ("tickets").

These essential utility functions are complemented by additional application-specific utility functions such as forensic watermarking, application of digital rights management, automated machine learning tagging of content, content search, dynamic clipping, content download/archiving, and others (upon which can be built higher level full Creator Stack API Functions such as payments gateways, marketplaces, live events, market making for recommendations and sponsorhip, etc.)

Stakeholders

Recapping from previous sections, all Stakeholders in the Content Fabric network have an address in the network and access the network via blockchain identity and blockchain signatures using private keys and blockchain tranactions. The Content Fabric is a network of **Nodes** that all run the same Content Fabric protocol, and thus carry out these utility functions on behalf of Content Owners. Each Node has a contract address allowing it to be compensated for processing transactions for these utility functions.Some nodes are **Validator Nodes** that specifically validate the Fabric blockchain transactions on behalf of the network. These Nodes currently run a proof-of-authority based consensus protocol (adapted Ethereum Clique), but also participate in the network with their own governance contracts, which can be staked for a proof-of-stake based consensus and governance in an inter-blockchain network. (A PoS based implementation is currently underway). **Content Owners** or Media Businesses acting on their behalf (**Tenants**) use these utility services to reach their **Users**, in order to gain and hold their attention. Users gain access to view/experience/retrade/remix the Content directly via the Content Fabric: the Fabric serves media to authorized Viewers directly by default. It can also be used to serve it to proxies (content delivery networks/headends, etc. that in turn serve the content to the Users). **Users** are authorized to view/experience/retrade/create derivative content (through the Fabric's trustless owner-to-reader encryption model) via blockchain attestation. This can be self-attestation only where appropriate for low risk, modest value transactions, or fully on-chain for high value (e.g. publishing) operations. The Content Owner thus either directly authorizes the Users to access the Content via the Fabric's capabilities or indirectly, through an authorized authority. The User is explicitly in control of any exchange of value for his/her attention to the Content.Content Owners may sell to third parties (**Sponsors**) the option to present content to the Users alongside the primary content and the ability to offer Users value in exchange for their data.The above relationships between Nodes, Validator Nodes, Content Owners, and Sponsors are currently implemented as a **Tenant** within a Space in the Content Fabric. A Space is a collection of content objects owned by one or more Content Owners. The group of content objects owned by a specific Content Owner and the access groups and policies for accessing these is a Tenant, and the Space is a collection of Tenants that all follow the same general security groups and quality of service guarantees.

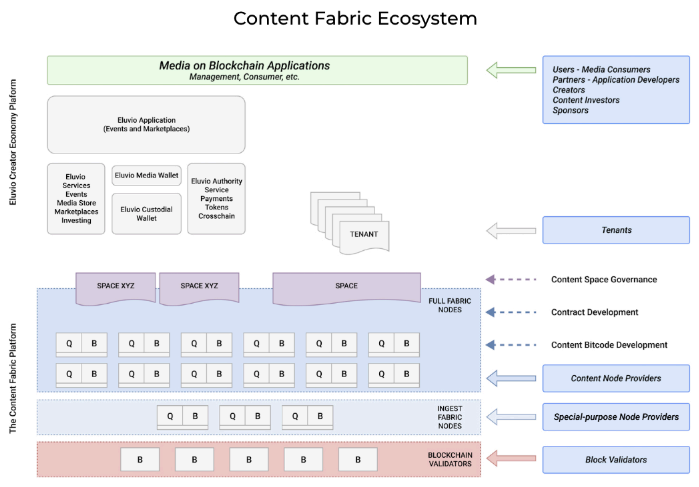
Thus, the identifiable Stakeholders include:

1. Creators, who create content to be stored and distributed on the Platform and who seek to reach an audience for that content;
2. Tenants, who operate a content business (but who may or may not be Creators themselves);
3. Users, primarily individuals who consume content stored on the Platform;
4. Partners, who provide content support to Tenants and may introduce potential Tenants to the Platform;
5. Node Operators, who validate blocks in the network, transcode, transform, and render content media for storage on the network, and serve content to Users;
6. Sponsors, who seek engagement with Users for commercial purposes ("advertising"); and
7. Content Investors, who invest in content in anticipation of revenue that content may generate.

Organization

Today's production Content Fabric has only one active space (Active Space 1), but the implementation allows for the addition of more Spaces that would allow for different classes of security and levels of distribution service as the platform grows. The current mainnet955305 is built on an Ethereum-standards based blockchain architecture and standard EVM contracts (ERC 20, 721, 1155) , and uses a proof-of-authority Clique based protocol (no proof of work), and a network of Validators run by media companies and stakeholders in the network. A substrate-compatible testnet ready for interblockchain communication is being built to allow for decentralizing stake and open scaling. These architectures togeter are not only fast, ecological, and standards based but also flexible for the rapidly evolving future for application-specific chains.

Figure 11: Blockchain Stakeholder Ecosystem


Content Fabric Ecosystem
Eluvio Creator Economy Plaform
Media on Blockchain Applications
Management, Consumer, etc.
Users - Media Consumers
Partners - Application Developers
Creators
Content Investors
Sponsors
Eluvio Application
(Events and Marketplaces)
Eluvio Services Events Media Store Marketplaces Investing
Eluvio Media Wallet
Eluvio Custodial Wallet
Eluvio Authority Service Payments Tokens Crosschain
TENANT
Tenants
SPACE XYZ
SPACE XYZ
SPACE
FULL FABRIC NODES
Content Space Governance
Contract Development
Content Bitcode Development
Q
B
Content Node Providers
The Content Fabric Platform
INGEST FABRIC NODES
Special-purpose Node Providers
BLOCKCHAIN VALIDATORS
Block Validators

Tokens - the ELV

Transactions in the Eluvio Network are effected using digital token - the ELV. Utility work done by Nodes for Content Owners is metered and is used as a normalized measure of the utility's value. A Node that is primarily serving streaming content to end user audiences may for example perform 0.01 units of transcoding, 0.1 units of egress bandwidth, and 0.005 units of storage, scaled to reflect the actual cost of providing these services. Current values are grounded in real world costs based on production use over the past three years. The rate of exchange of utility units to ELV tokens is currently set by the permissioned network's governance (largely Eluvio, Inc.) but as the public network opens with the public ELV, the rate will be independently set by the open market ELV price.

Each Content Owner as a Tenant on the Fabric has the option to offer content services in ELVs or a **tenant specific token** to assign value to User interaction with their Content via these utility services. This Tenant specific token type is implemented as an ERC-20 token and can be tied to the ELV (or not). Each Tenant may determine the number of such tokens in circulation and the token's name. This may function as an Attention Token for that Tenant's Content and can be used to measure, incentivize and monetize User engagment and promotion. The specific use cases and the exchange value to ELVs is under the Tenant's control, and allows Content Owners to create media applications on the Content Fabric with both non-fungible and fungible tokenization of their Content.

ELV Circulation in the Creator Stack

There are three basic functions for the ELV in the token economics : The ELV compensates Node Providers for the utility "work" done by Nodes on behalf of Content Owners; Sets a charging value on the variable cost of the utility work in order to charge Content Owner Tenants; and Provides a utility rate for Content Owners to consider in valuing their tenant specific services. This results in a value flow in the Content Fabric as follows:

- A core flow of ELVs is carried out by the Content Fabric Space continuously (via daily settlement) on behalf of Content Owners (Tenants) and the Nodes:
- The Nodes carry out utility work that is authorized and/or metered as blockchain transactions in work units to record their Transcoding, Egress Transmission, Publishing Transmission and new content Storage, as well as Application-specific utility work.
- Meanwhile, the Space contract (at least once daily) charges a Tenant (Content Owner) contract in ELV according to the work to ELV current exchange rate, and credits the Node contracts.
- The Validator Nodes continuously validate the transactions allowing the network cycle progress, and accumulate transaction fees in work-to-ELV rates on each validation.
- Meanwhile, the Space contract charges the Tenant contract according to the current work-to-ELV exchange rate.
- As Nodes earn more ELV their stake in the network increases.

Figure 12: On Chain Token Economics

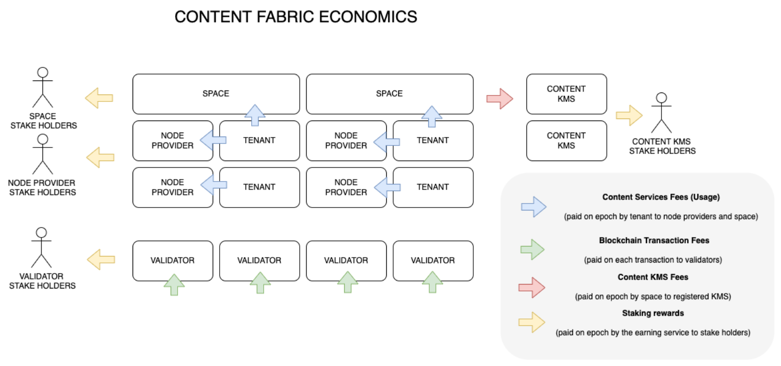
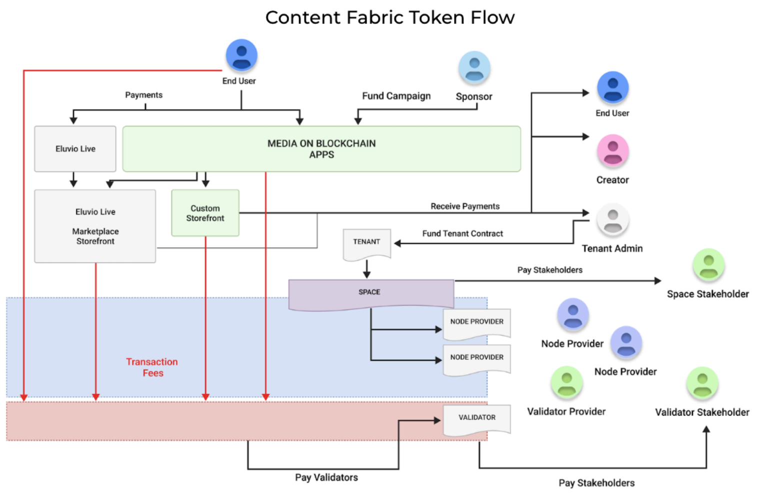

CONTENT FABRIC ECONOMICS
SPACE STAKE HOLDERS
NODE PROVIDER STAKE HOLDERS
VALIDATOR STAKE HOLDERS
SPACE
SPACE
NODE PROVIDER
TENANT
TENANT
VALIDATOR
CONTENT KMS
CONTENT KMS
CONTENT KMS STAKE HOLDERS
Content Services Fees (Usage)
(paid on epoch by tenant to node providers and space)
Blockchain Transaction Fees
(paid on each transaction to validators)
Content KMS Fees
(paid on epoch by space to registered KMS)
Staking rewards
(paid on epoch by the earning service to stake holders)

Figure 13: On Chain Creator Economy



Optionally, Each Tenant's operations can have a liquidity pool or payments backed by ELVs and optionally tied to fixed price token (USD-tether or similar). As the use grows (rewards for fan engagement, for advertising, trading of NFTs in Tenant-Specific-tokens), so grows ELV stake by the Tenants and optionally Users.

The accumulation of ELV token stake by Validators and the increasing popularity of Tenant functions backed by ELVs both will drive the ELV toward a convertible status, and a natural tipping point for a sustainable public value. Eluvio's opportunities to earn revenue in this framework include as Node operator, as a Validator operator, and as a provider of application-specific Creator Stack services transacted in ELV directly or indirectly through transaction fees. It is worth spelling out that Eluvio's commercial interest is directly aligned to the growth of the use of the Content Fabric as a utility platform for the Stakeholers that use it. And, as long as the Content Fabric governance becomes sufficiently decentralized to represent the best interests of the users of the platform (Content Owners, Users, Sponsors, Investors) Eluvio's commercial profit interests will remain objectively, directly aligned with the successful delivery of utility value to the network stakeholders.

The Content Fabric economics incentive an economy in which Content Owners keep their revenue and Users co-own and profit from their engagement, and we believe that there is not only a commercial benefit but also an important positive social impact that the Content Fabric can provide in a era where the Content Internet economy is held by 'web 2.0' limitations (stagnate distribution and advertising technology, user data exploitation, commoditization/monopolization of creative IP, and environmental damage).

7 Unique Benefits for Media Access via NFTs and Use Cases

The on-chain and cross-chain authorization of media via digital tokens described brings several major advantages to Web3 experiences via non-fungible tokens ("NFTs") and has been piloted to significant success over the past year in several projects. We will conclude the paper by describing these advantages and use cases.

Full length and higher value video experiences as NFTs have revealed problems around authenticity, uniqueness, rarity, chain of ownership, user experience, and cost/minting efficiency that stem from, or at least relate to, the inefficiencies of many blockchain platforms for media operations and the weak link between NFTs as contracts and the media assets to which they refer.

Media NFTs are usually stored, managed, and rendered in separate storage systems such as decentralized storage (e.g. IPFS[1]) or centralized cloud storage.

These limitations may not matter as much for early adoption, static/JPEG based NFTs and a limited market, but do restrict mainstream tokenized ownership. The CFP addresses these shortcomings and brings several unique benefits to content NFTs:

7.1 Benefits

- Authenticity: The NFT contract has a direct, inseparable link to the original content object that was used to create it. The CFP hash is a cryptographic fingerprint that proves that the content referred to by the NFT is actually the same as the original content, and can be verified through an open source proof. The original content stored in the Fabric is also encrypted and all authorization to it uses on chain data or a valid transaction, making the original content tamper free and truly unique. Many NFTs refer to storage of the digital content that is unprotected and, sometimes, not verifiable.
- Uniqueness and Rarity: Each NFT minted by the CFP is guaranteed unique, as it refers to a unique version of the original content object, minted with a unique issue number among a finite set. Access to each unique NFT and its viewing experience is enforced through a crypto wallet or a digital ticket code that is cryptographically verified against a blockchain contract that backs the NFT batch and that particular issue number. A unique issue number can be dynamically watermarked into the NFT when accessed with a ticket code or crypto wallet to display proof of ownership to others.
- Any Content: Unlike traditional NFT platforms that require a third party media storage or distribution system to serve the digital content (static images, small videos), the Fabric can render any digital content—a full length movie, concert, or even a continuous performance—as an NFT. This opens up virtually unlimited possibilities for collectible experiences.
- Chain of Ownership and Secondary Payments: Because the Fabric couples the transfer of ownership of NFTs to the content contract, it is possible to have any royalty or other payments to creators part of every NFT transaction and to enforce this in the availability of the content itself.
- User Experience: All NFTs are content objects that can be directly authorized and displayed in user wallets as a native media experience (video/audio streaming, resizing, live, continuous, with metadata, etc.) avoiding all separate content storage and streaming systems. Key management can be delegated to the Fabric avoiding the challenges of tedious crypto wallets.
- Dynamic Gamification and Generative Pipelines: Functions such as spawning, burning, trait mixing, branching stories, and permissioned offerings are natively supported by the Fabric's dynamic content serving and blockchain policy permissions, and generative NFTs can be created and served from single content objects with ease and low cost;
- Enables Low Cost Minting and Extreme Efficiency: The Fabric's content and blockchain efficiencies enable applications to mint on demand, at high speed, with low cost. Applications can offer their users low cost, gas-free minting at scale, and for any content experience the Fabric renders. The CFP's "zero file copy" and dynamic, componentized distribution protocol is dramatically more resource efficient (>50X) than traditional content storage and distribution networks and along with its PoA/PoS blockchain, is a very efficient and low cost solution for the Creator Economy.

7.2 Ongoing Use Cases

The unique advantages of the CFP have been exploited in several major use cases to date in Web3 properties built on the Content Fabric for brands such as WWE, Dolly Parton, Dan Harmon's 'Krapopolis', and FOX's 'The Masked Singer', in partnership with Blockchain Creative Labs; Warner Media and Warner Bros. Studios; Globo; AMC's 'The Walking Dead' in partnership with Orange Comet; and the Black Eyed Peas and Rita Ora in partnership with Paramax Studios; Universal Music in partnership with Cirkay UK; Indieflix, and others. These experiences have brought a consumer-grade digital to several hundred thousand mainstream consumers with on chain content streaming, NFT-authorized media access and media NFTs containing exclusive content ranging from interactive AR worlds to digital song tracks and albums. The media content has been "added to" and "transformed" within the token by dynamically updating the content object (e.g. WWE "Flips')' and exclusive content in multiple projects; unique "1of1s" are created from a single content object; and full length media has been authorized via individual utility tokens for various ownership terms including one-time access, time-bound access, and unlimited access. A next step will be authorizing derivative works created from token-authorized media.

8 Brief Comparison to Existing Technologies

Web3 media experiences that have used decentralized and open technologies such as IPFS [1] or Airweave [11] (decentralized storage networks), or LivePeer [7] (decentralized transcoding) have struggled to provide the quality of service and a practically integrated platform for scalable use by creators and consumer grade media experiences. Consequently many Web3 media experiences are currently provided with media stored and distributed using traditional Web2 systems, and token metadata referring the user to traditional Web2 URLs for video streaming or other content access such as on cloud providers and web2 streaming platforms [8]

The Fabric is a **Web3-native** content storage, distribution, and authorization platform. Compared to traditional content delivery networks and storage platforms, it provides decentralized low latency, high bandwidth streaming and download of digital content and highly efficient storage and bandwidth use, content integrity, and transparency of user data. We have described how these attributes are realized in implementation through a novel decentralized ("nothing shared") design that self-scales via an open extensible routing protocol that scales with network size and number of content objects stored and provides predictable delivery quality for users, a programmable media and commerce pipeline that generates output variants on-demand from source content, and a trustless security model and provable content version history.

9 Conclusion

Building upon the CFP, we have introduced a new on-chain and cross-chain protocol and implementation for ownership and authorization in decentralized distribution using digital tokens. This protocol applies to tokenized ownership of media of all variations including fungible and non-fungible tokens (NFTs), and decentralized storage/distribution/playout of the content. It features secure (trustless) and verifiable access to any representation of the content under the control and verification of a policy stored "on chain". The approach brings many advantages over Web2 paid and social/ad sponsored creative pipelines, such as:

1. A scalable way to attest to / prove individual ownership of media;
2. Low cost transactions between the media owner and the consumer allowing for direct selling and all revenues to the seller;
3. Tamper proof unambiguous measurement of consumption (and any other engagement) as a blockchain transaction;
4. A practical way to distribute royalties and encode digital rights and pay many parties;
5. Escape from the economics that sell audience data and keep a large percentage of creator revenue.

Multiple new applications are possible that scale revenue to content owners, provide transparent control for users over their data and maximize user choice, ensure media integrity, and help sponsors to achieve direct, more personalized experiences.

References

[1] J. Benet. IPFS - content addressed, versioned, P2P file system. *CoRR*, abs/1407.3561, 2014.

[2] J. Burdges, A. Cevallos, P. Czaban, R. Habermeier, S. Hosseini, F. Lama, H. K. Alper, X. Luo, F. Shirazi, A. Stewart, et al. Overview of polkadot and its design considerations. *arXiv preprint arXiv:2005.13456*, 2020.

[3] B. S. D. Fandrey. Clang/llvm.

[4] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. 2002.

[5] M. Munson. Decentralized content fabric for video over the internet, demuxed 2018, 2018.

[6] M. Munson. Low latency live from a different vantage point, demuxed 2020, 2020.

[7] D. Petkanics and E. Tang. Livepeer whitepaper. Technical report, Technical report, Livepeer, 2018.

[8] T. R. S. Samantha Hissong. Kings of leon will be the first band to release an album as an nft, 2021.

[9] Z. Tufekci. How social media took us from tahrir square to donald trump. *MIT Technology Review*, 14:18, 2018.

[10] webassembly.org. Web assembly (wasm), 2022.

[11] S. Williams, V. Diordiiev, L. Berman, and I. Uemlianin. Arweave: A protocol for economically sustainable information permanence. *Arweave Yellow Paper, www. arweave. org/yellow-paper. pdf*, 2019.