Open Audio Protocol

Yellow Paper

Technical Specification v1.0

Audius Protocol Team

July 9, 2025

Abstract

The Open Audio Protocol is a decentralized infrastructure for audio content distribution, storage, and monetization. This specification describes a novel architecture combining Byzantine Fault Tolerant consensus (CometBFT), distributed content-addressed storage (Mediorum), and Ethereum integration for governance and economic incentives. The protocol enables permissionless participation, cryptographic content integrity, and efficient audio streaming at scale while maintaining decentralization principles and creator sovereignty.

Keywords: Blockchain, Audio Streaming, Decentralized Storage, Byzantine Fault Tolerance, Content Distribution Network

Contents

1 Introduction

The Open Audio Protocol represents a paradigm shift in digital audio infrastructure, moving from centralized platform control to a decentralized, creator-owned ecosystem. This protocol specification defines the technical architecture for a distributed system that enables:

- **Decentralized Content Storage**: Content-addressed storage with cryptographic integrity guarantees
- **Byzantine Fault Tolerant Consensus**: High-throughput transaction processing with finality guarantees
- **Economic Incentive Alignment**: Token-based rewards for network participation and service provision
- **Interoperability**: Standards-compliant APIs and multi-language implementation support
- **Creator Sovereignty**: Direct creator control over content and monetization

1.1 Design Goals

The protocol is designed with the following core objectives:

1. **Scalability**: Support for millions of tracks and billions of plays
2. **Availability**: 99.9% uptime through redundancy and fault tolerance
3. **Integrity**: Cryptographic proofs preventing content tampering
4. **Efficiency**: Sub-second response times for content access
5. **Decentralization**: No single points of failure or control
6. **Extensibility**: Modular architecture enabling protocol evolution

1.2 Network Participants

The protocol defines several participant types, each with distinct roles and incentives:

- **Creators**: Upload and monetize audio content
- **Listeners**: Discover and stream audio content
- **Core Validators**: Maintain consensus and validate transactions
- **Storage Providers**: Store and serve audio content
- **Discovery Providers**: Index and query protocol state

2 System Architecture

2.1 High-Level Overview

The Open Audio Protocol employs a modular architecture consisting of three primary layers:

```
+-------------------------------------------------------------+
|                    Application Layer                        |
|  +-----------------+  +-----------------+  +--------------+ |
|  |    Web Client   |  |  Mobile Client  |  |  Third-party | |
|  |     (dApp)      |  |     (dApp)      |  |  Integrations| |
|  +-----------------+  +-----------------+  +--------------+ |
+-------------------------------------------------------------+
                                |
+-------------------------------------------------------------+
|                     Protocol Layer                          |
|  +-----------------+  +-----------------+  +--------------+ |
|  | Core Validators |  | Storage Network |  |   Discovery  | |
|  |   (CometBFT)    |  |   (Mediorum)    |  |   Providers  | |
|  +-----------------+  +-----------------+  +--------------+ |
+-------------------------------------------------------------+
                                |
+-------------------------------------------------------------+
|                   Infrastructure Layer                      |
|  +-----------------+  +-----------------+  +--------------+ |
|  |    Ethereum     |  |      IPFS       |  |   CDN/Edge   | |
|  |  (Governance)   |  |   (Metadata)    |  |    Network   | |
|  +-----------------+  +-----------------+  +--------------+ |
+-------------------------------------------------------------+
```

Figure 1: Open Audio Protocol Architecture Layers

2.2 Component Interaction

```
+--------------+    gRPC/HTTP    +--------------+    CRUDR      +--------------+
|              | --------------> |              | ----------> |               |
|   Clients    |                 | Core Network |             |   Mediorum    |
|              | <-------------- |              | <---------- |   Storage     |
+--------------+                 +--------------+             +---------------+
                                        |                             |
                                        |                             |
                                        v                             v
                             +--------------+             +---------------+
                             |   Ethereum   |             |   Audio       |
                             |  Contracts   |             |  Processing   |
                             +--------------+             +---------------+
```

Figure 2: Protocol Component Interactions

3 Core Consensus Layer

3.1 CometBFT Integration

The protocol utilizes CometBFT as its consensus engine, providing Byzantine Fault Tolerant consensus with the following characteristics:

- **Block Time**: 400ms average block production
- **Finality**: Immediate finality after block commitment
- **Fault Tolerance**: Byzantine tolerance up to 1/3 malicious validators
- **State Machine**: ABCI (Application Blockchain Interface) implementation

3.2 ABCI Application State Machine

The core application implements the ABCI interface with the following transaction types:

Algorithm 1 Transaction Processing Pipeline

```
1: $tx \leftarrow$ incoming transaction
2: $result \leftarrow$ CheckTx($tx$)
3: if $result$.IsValid() then
4:     AddToMempool($tx$)
5: end if
6: $block \leftarrow$ ProposeBlock()
7: $validatedBlock \leftarrow$ ValidateBlock($block$)
8: $finalizedBlock \leftarrow$ FinalizeBlock($validatedBlock$)
9: Commit($finalizedBlock$)
```

3.3 Transaction Types

3.3.1 Play Transactions

Records audio stream events with geographic and timestamp data:

```
{
  "type": "Play",
  "track_id": "uuid",
  "user_id": "uuid",
  "timestamp": "rfc3339",
  "location": {
    "country": "ISO-3166",
    "region": "subdivision"
  },
  "source": "web|mobile|api"
}
```

Listing 1: Play Transaction Structure

3.3.2 Entity Management Transactions

Handles creation and updates of protocol entities (tracks, playlists, users):

```
{
  "type": "ManageEntity",
  "entity_type": "track|playlist|user",
  "action": "create|update|delete",
  "entity_id": "uuid",
  "metadata_cid": "bafybeihash",
  "signature": "0xsignature"
}
```

Listing 2: Entity Management Transaction

3.3.3 Validator Registration

Enables dynamic validator set updates:

```
{
  "type": "ValidatorRegistration",
  "comet_address": "validator_address",
  "pub_key": "ed25519_pubkey",
  "power": 25,
  "delegate_wallet": "0xethaddress",
  "endpoint": "https://node.domain.com",
  "node_type": "discovery|storage",
  "eth_block": 12345678,
  "sp_id": "service_provider_id"
}
```

Listing 3: Validator Registration

3.3.4 Storage Proof Transactions

Cryptographic proofs of data availability:

```
{
  "type": "StorageProof",
  "height": 12345,
  "cid": "bafybeihash",
  "address": "validator_address",
  "proof_signature": "signature",
  "prover_addresses": ["addr1", "addr2", "addr3"]
}
```

Listing 4: Storage Proof Transaction

3.4 State Synchronization

The protocol implements efficient state synchronization mechanisms:

- **Snapshots**: Periodic PostgreSQL dumps for fast sync
- **State Sync**: CometBFT state sync with trusted heights
- **Block Pruning**: Configurable retention policies
- **Compaction**: Automatic state compaction for efficiency

4 Mediorum Storage System

4.1 Architecture Overview

Mediorum provides decentralized content-addressed storage with the following properties:

- **Content Addressing**: SHA-256 based content identifiers (CIDs)

- **Replication**: Configurable redundancy across multiple nodes
- **Availability**: Geographic distribution and fault tolerance
- **Integrity**: Cryptographic verification of content

4.2 CRUDR Protocol

The Create, Read, Update, Delete, Replicate (CRUDR) protocol manages data lifecycle:

Algorithm 2 CRUDR Operation Propagation

```
1: op ← NewOperation(action, data, metadata)
2: signature ← Sign(op, privateKey)
3: ApplyLocally(op)
4: if IsOriginNode() then
5:     for peer in PeerSet do
6:         Broadcast(op, peer)
7:     end for
8: end if
9: PersistOperation(op)
```

4.3 Rendezvous Hashing

Content placement utilizes rendezvous hashing for deterministic, decentralized placement:

Algorithm 3 Rendezvous Hash Placement

```
1: nodes ← ActiveStorageNodes()
2: scores ← []
3: for node in nodes do
4:     hash ← SHA256(cid || node.ID)
5:     scores.append(hash, node)
6: end for
7: sorted ← SortByHash(scores)
8: return sorted[0:ReplicationFactor]
```

4.4 Audio Processing Pipeline

4.4.1 Transcoding

Multi-format transcoding for optimal delivery:

```
formats:
  - codec: "mp3"
    bitrate: "320k"
    quality: "high"
  - codec: "mp3"
    bitrate: "128k"
    quality: "medium"
  - codec: "opus"
    bitrate: "96k"
    quality: "efficient"
```

Listing 5: Transcoding Configuration

4.4.2 Audio Analysis

Automated extraction of musical features:

- **BPM Detection**: Using aubio library for tempo analysis
- **Key Detection**: Musical key detection via KeyFinder
- **Spectral Analysis**: Frequency domain feature extraction
- **Preview Generation**: Automatic snippet creation

4.5 Proof of Storage

The protocol implements a cryptographic proof system ensuring data availability:

Algorithm 4 Storage Proof Generation

```
1: $cid \leftarrow \text{SelectCIDFromBlockHash}(blockHash)$
2: $data \leftarrow \text{RetrieveContent}(cid)$
3: $augmented \leftarrow data \mid\mid blockHash$
4: $proof \leftarrow \text{MD5}(augmented)$
5: return $proof$
```

4.6 Repair and Replication

Automated repair mechanisms maintain data availability:

- **Health Monitoring**: Periodic availability checks
- **Repair Triggers**: Automatic repair on node failures
- **Rebalancing**: Dynamic replica redistribution
- **Cleanup**: Garbage collection of orphaned content

5 Ethereum Integration

5.1 Smart Contract Architecture

The protocol integrates with Ethereum mainnet through a suite of smart contracts:

```
+------------------+
|    Registry      | ---+
+------------------+    |
                        |
+------------------+    |   +------------------+
|   Governance     | ---+-- |    Staking       |
+------------------+    |   +------------------+
                        |
+------------------+    |   +------------------+
|ServiceProvider   | ---+   | ClaimsManager    |
|    Factory       |      +------------------+
+------------------+
```

Figure 3: Ethereum Smart Contract Dependencies

5.2 Validator Set Management

5.2.1 Registration Process

Validators register on Ethereum with staking requirements:

1. Stake minimum 200,000 AUDIO tokens
2. Submit service endpoint and public key
3. Provide delegate wallet for L1 operations
4. Undergo community attestation process

5.2.2 Registry Bridge

Maintains synchronization between Ethereum and L1 validator sets:

1. Get validator lists from both Ethereum and L1
2. Identify missing validators in L1
3. Identify deregistered validators no longer on Ethereum
4. Submit registration transactions for missing validators
5. Submit deregistration transactions for removed validators

5.3 Governance Mechanism

5.3.1 Proposal System

On-chain governance for protocol upgrades:

- **Proposal Submission**: Stake-weighted proposal creation
- **Voting Period**: 7-day voting window
- **Execution Delay**: 24-hour timelock for security
- **Quorum Requirements**: Minimum participation thresholds

5.3.2 Treasury Management

Community-controlled protocol treasury:

- **Funding Sources**: Protocol fees and token inflation
- **Allocation**: Development grants and ecosystem incentives
- **Transparency**: On-chain spending proposals

6 RPC Service Layer

6.1 Service Architecture

The protocol exposes functionality through multiple RPC services using ConnectRPC:

Service	Purpose	Port
Core	Consensus operations, blocks, transactions	26657
Storage	File upload, streaming, metadata	1991
ETL	Analytics, queries, aggregations	26659
System	Health, monitoring, diagnostics	26659
Eth	Ethereum integration, contracts	26659

Table 1: RPC Service Endpoints

6.2 Core Service API

6.2.1 Transaction Operations

```
service CoreService {
  rpc SendTransaction(SendTransactionRequest) returns (SendTransactionResponse)
    ;
  rpc GetTransaction(GetTransactionRequest) returns (GetTransactionResponse);
  rpc GetBlock(GetBlockRequest) returns (GetBlockResponse);
  rpc GetStatus(GetStatusRequest) returns (GetStatusResponse);
  rpc GetNodeInfo(GetNodeInfoRequest) returns (GetNodeInfoResponse);
}
```

Listing 6: Core Service Methods

6.3 Storage Service API

6.3.1 Content Operations

```
service StorageService {
  rpc UploadFiles(UploadFilesRequest) returns (UploadFilesResponse);
  rpc StreamTrack(StreamTrackRequest) returns (stream StreamTrackResponse);
  rpc GetUpload(GetUploadRequest) returns (GetUploadResponse);
  rpc GetStreamURL(GetStreamURLRequest) returns (GetStreamURLResponse);
}
```

Listing 7: Storage Service Methods

6.4 Protocol Compatibility

Services support multiple protocols for maximum compatibility:

- **ConnectRPC**: Native protocol with JSON/Protobuf support
- **gRPC**: Standard gRPC with HTTP/2 transport
- **gRPC-Web**: Browser-compatible gRPC variant
- **REST**: HTTP/1.1 JSON API for simple integration

7 Cross-Language Implementation

7.1 Protocol Definition

The protocol specification enables implementations in multiple programming languages through:

- **Protocol Buffers**: Language-agnostic schema definitions
- **OpenAPI Specs**: REST API documentation and code generation
- **Reference Implementation**: Go-based audiusd as canonical implementation

7.2 Rust Implementation Considerations

For high-performance Rust implementations, key considerations include:

7.2.1 Consensus Layer

```
pub trait AbciApplication {
    fn check_tx(&self, tx: &[u8]) -> ResponseCheckTx;
    fn begin_block(&mut self, req: RequestBeginBlock) -> ResponseBeginBlock;
    fn deliver_tx(&mut self, tx: &[u8]) -> ResponseDeliverTx;
    fn end_block(&mut self, req: RequestEndBlock) -> ResponseEndBlock;
    fn commit(&mut self) -> ResponseCommit;
}
```

Listing 8: Rust ABCI Trait Definition

7.2.2 Storage Layer

```
#[async_trait]
pub trait StorageProvider {
    async fn put(&self, cid: &str, data: &[u8]) -> Result<(), StorageError>;
    async fn get(&self, cid: &str) -> Result<Vec<u8>, StorageError>;
    async fn exists(&self, cid: &str) -> Result<bool, StorageError>;
    async fn delete(&self, cid: &str) -> Result<(), StorageError>;
}
```

Listing 9: Rust Storage Interface

7.3 Language-Specific Optimizations

7.3.1 Audio Processing

Different languages excel at different aspects:

- **C++**: High-performance audio analysis (aubio, KeyFinder)
- **Rust**: Memory-safe systems programming with zero-cost abstractions
- **Go**: Excellent concurrency for network services
- **Python**: Machine learning and data analysis workflows
- **JavaScript**: Browser and mobile client applications

8 Security Model

8.1 Cryptographic Primitives

The protocol employs industry-standard cryptographic primitives:

Component	Algorithm	Purpose
Content Addressing	SHA-256	Content identification
Digital Signatures	ECDSA/secp256k1	Transaction authentication
Consensus Keys	Ed25519	Validator consensus
Storage Proofs	MD5	Challenge-response verification

Table 2: Cryptographic Algorithm Usage

8.2 Attack Vectors and Mitigations

8.2.1 Consensus Attacks

- **Nothing-at-Stake**: Prevented by BFT consensus properties
- **Long Range**: Mitigated by checkpointing and weak subjectivity
- **Eclipse**: Protected by peer diversity requirements

8.2.2 Storage Attacks

- **Data Withholding**: Countered by proof-of-storage challenges
- **Sybil**: Mitigated by stake-based registration
- **Grinding**: Prevented by deterministic placement algorithms

8.3 Privacy Considerations

- **Listener Privacy**: Optional pseudonymous play recording
- **Creator Privacy**: Selective metadata disclosure
- **Geographic Privacy**: Configurable location granularity

9 Performance Characteristics

9.1 Throughput Metrics

Operation	Throughput	Latency
Transaction Processing	1,000 tx/sec	400ms finality
Content Upload	100 MB/s	5-10s processing
Stream Initiation	10,000 req/sec	<100ms response
Storage Proof	1 proof/block	1-2s generation

Table 3: Performance Benchmarks

9.2 Scalability Analysis

9.2.1 Horizontal Scaling

- **Validator Set**: Up to 100 validators with current consensus

- **Storage Nodes**: Unlimited horizontal scaling
- **Geographic Distribution**: Global CDN-like performance

9.2.2 Vertical Scaling

- **Database Sharding**: Horizontal partitioning by content type
- **Caching Layers**: Multi-tier caching for hot content
- **Compression**: Adaptive bitrate and codec selection

10 Economic Model

10.1 Token Mechanics

10.1.1 Utility Functions

The AUDIO token serves multiple utility functions:

- **Staking**: Validator registration and slashing collateral
- **Governance**: Voting power for protocol upgrades
- **Incentives**: Rewards for network participation
- **Payment**: Creator monetization and premium features

10.1.2 Inflation Schedule

```
total_inflation: 7% annually
distribution:
  validators: 3.5%
  storage_providers: 2.5%
  creators: 1.0%
```

Listing 10: Annual Inflation Distribution

10.2 Incentive Alignment

10.2.1 Validator Rewards

- **Block Rewards**: Proportional to stake and performance
- **Transaction Fees**: Share of network fee revenue
- **Uptime Bonuses**: Additional rewards for high availability

10.2.2 Storage Provider Rewards

- **Capacity Rewards**: Payment for available storage
- **Bandwidth Rewards**: Compensation for data transfer
- **Availability Bonuses**: Uptime and replication bonuses

11 Governance and Upgrades

11.1 Governance Framework

11.1.1 Proposal Types

1. **Parameter Changes**: Inflation rates, block sizes, timeouts
2. **Contract Upgrades**: Smart contract improvements
3. **Treasury Spending**: Community fund allocation
4. **Emergency Actions**: Security-critical interventions

11.1.2 Voting Mechanics

- **Voting Power**: Proportional to staked tokens
- **Delegation**: Stake-weighted vote delegation
- **Quorum**: Minimum 40% participation required
- **Threshold**: 67% approval for passage

11.2 Upgrade Mechanisms

11.2.1 Soft Forks

Backward-compatible protocol improvements:

- Parameter adjustments
- New transaction types
- Enhanced validation rules

11.2.2 Hard Forks

Breaking changes requiring coordinated upgrade:

- Consensus algorithm changes
- State transition modifications
- Major protocol restructuring

12 Future Directions

12.1 Planned Enhancements

12.1.1 Layer 2 Scaling

- **Payment Channels**: Micropayment streaming for creators
- **State Channels**: Real-time collaboration features
- **Sidechains**: Application-specific execution environments

12.1.2 Advanced Features

- **AI Integration**: Automated content categorization and recommendation
- **Cross-Chain Bridges**: Multi-blockchain asset support
- **Zero-Knowledge Proofs**: Enhanced privacy and scalability

12.2 Ecosystem Development

12.2.1 Developer Tools

- **SDKs**: Multi-language development kits
- **APIs**: RESTful and GraphQL query interfaces
- **Documentation**: Comprehensive integration guides
- **Testnets**: Sandbox environments for development

12.2.2 Community Initiatives

- **Grant Programs**: Funding for ecosystem development
- **Hackathons**: Innovation competitions and bounties
- **Education**: Technical workshops and certification programs

13 Conclusion

The Open Audio Protocol represents a fundamental advancement in decentralized content infrastructure, combining proven technologies like Byzantine Fault Tolerant consensus with novel approaches to content addressing and economic incentive design. By eliminating centralized intermediaries while maintaining performance and usability, the protocol creates new opportunities for creator empowerment and user sovereignty.

The modular architecture enables gradual adoption and iterative improvement, while the comprehensive governance framework ensures community-driven evolution. As the protocol matures, it will serve as a foundation for next-generation audio applications and services built on principles of decentralization, transparency, and user ownership.

The technical specification provided in this document serves as a blueprint for implementation teams and a reference for the broader community. As development continues, this specification will evolve to reflect protocol enhancements and community feedback, maintaining its role as the definitive technical documentation for the Open Audio Protocol.

Acknowledgments

The Open Audio Protocol is the result of collaborative effort from the global Audius community, including core developers, validators, storage providers, creators, and listeners who contribute to the network's growth and success. Special recognition goes to the research and engineering teams who designed and implemented the foundational technologies described in this specification.

References

1. Tendermint Inc. "CometBFT: A Byzantine Fault Tolerant Consensus Engine." https://docs.cometbft.com/

2. Protocol Labs. "IPFS: InterPlanetary File System." https://ipfs.io/

3. Ethereum Foundation. "Ethereum Development Documentation." https://ethereum.org/developers/

4. Buf Technologies. "ConnectRPC: A better gRPC." https://connectrpc.com/

5. Audios Inc. "Audius Protocol Documentation." https://docs.audius.org/