

# Allaudio Network

*A decentralized live audio delivery and streaming network with a focus on privacy,  
powered by a sustainable blockchain.*

[vishnu@rainverse.com](mailto:vishnu@rainverse.com)

Version 1.0 | Nov 25 2021

## Abstract

This whitepaper introduces the Allaudio Network, a blockchain and token as an incentive mechanism for decentralized live audio streaming and delivery network without tracking user data and activity.

The Allaudio Network and protocol ensures privacy of its users and solves various challenges the live audio streaming industry faces today. First, with privacy at its focus, a key-based authentication mechanism instead of traditional authentication methods so that the user may take full control and responsibility of their private information. Second, tokens on the Allaudio blockchain are used as an incentive to encourage individual users to share their redundant computing and bandwidth resources as caching or relay nodes for live audio streams. This improves the quality of stream delivery and solves the "last-mile" delivery problem, the main bottleneck for traditional content delivery pipelines, especially for high bitrate lossless audio and next generation audio formats including Dolby Atmos. Third, with sufficient network density, the majority of listeners shall pull streams from peering caching nodes, allowing audio platforms to significantly reduce content delivery network (CDN) costs. More importantly, by introducing tokens as an end-user incentive mechanism the Allaudio Network allows audio platforms to deepen listener engagement, drive incremental revenues, and differentiate their content and listening experience from their competitors.

The Allaudio blockchain adopts three main novel concepts:

- **Multi-Level BFT:** A modified BFT consensus mechanism which allows thousands of nodes to participate in the consensus process, while still supporting very high transaction throughput (1000+ TPS). The core idea is to have a small set of nodes, which form the validator committee, produce a chain of blocks as fast as possible using a PBFT-like process. Then, the thousands of consensus participants, called guardians, finalize the chain generated by the validator committee at regular checkpoint blocks. The name multi-level BFT consensus mechanism reflects the fact that the validator/guardian division provides multiple levels of security guarantee. The validator committee provides the first level of consensus - with 10 to 20 validators, the committee can come to consensus quickly. The guardian pool forms the second line of defense. With thousands of nodes, it is substantially difficult for attackers to compromise the integrity of the network, and thus

provides a much higher level of security. This mechanism achieves a good balance among transaction throughput, consistency, and level of decentralization, the three pillars of the so-called "impossible triangle".

- **Aggregated Signature Gossip Scheme:** A basic all-to-all broadcasting of the checkpoint block hash could work between guardian nodes, but it yields quadratic communication overhead, and therefore cannot scale to 1000+ nodes. Instead, by adopting an Aggregated Signature Gossip Scheme reduces messaging complexity. Each guardian node keeps combining the partially aggregated signatures from all its neighbors, and then gossips out the aggregated signature. This way the signature share of each node can reach other nodes at an exponential rate, leveraging the gossip protocol. In addition, the signature aggregation keeps the size of the node-to-node messages small, and thus further reduces the communication overhead.
- **Resource Oriented Micropayment Pool:** An off-chain "Resource Oriented Micropayment Pool" that is purpose built for live audio streaming. It allows a user to create an off-chain micropayment pool that any other user can withdraw from using off-chain transactions, and is double-spend resistant. It is much more flexible compared to off-chain payment channels.

The Allaudio blockchain also introduces a new concept:

- **Key-Based Authentication and Profile Management:** As an alternative to password authentication, instead of requiring a user's password, it is possible to confirm the identity of each node by using asymmetric cryptography algorithm, RSA, with public and private keys. This empowers the user to own their private information and contradict any forms of targeted advertisement.

table-of-contents

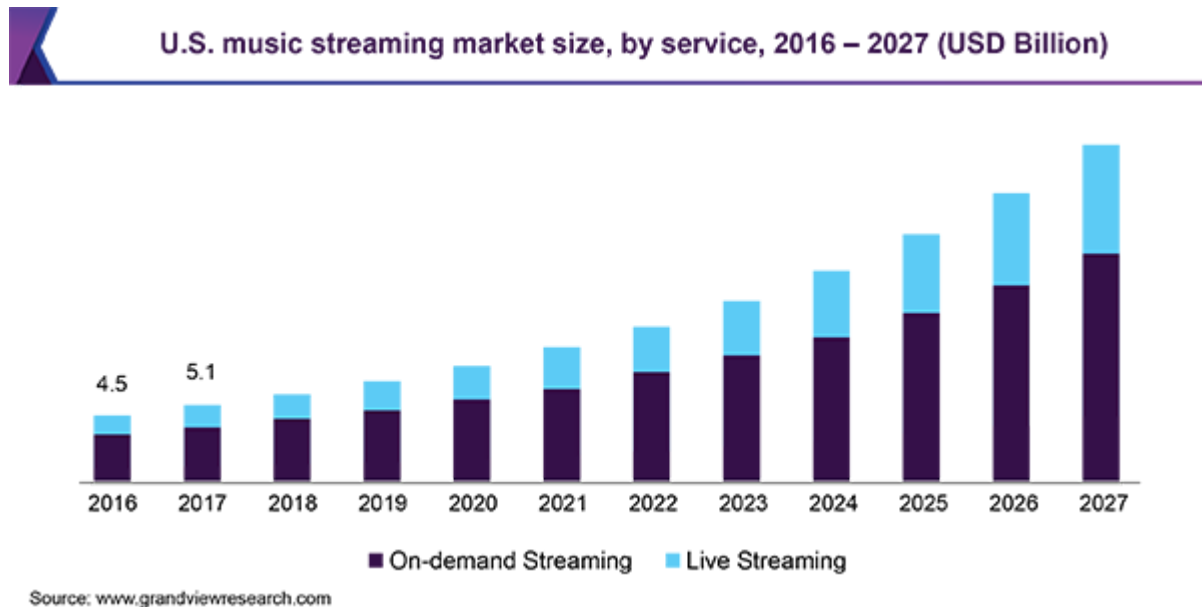
# Vision

## Introduction

### Audio Streaming Market

The global music streaming market size was valued at USD 20.9 billion in 2019 and is expected to expand at a compound annual growth rate (CAGR) of 17.8% from 2020 to 2027. The growing popularity of digital platforms and the increasing adoption of smart devices is expected to positively impact industry growth during the forecast period. According to the China based Tencent Music Entertainment (TME) group, the year-over-year revenue from online music subscriptions increased by 70.0% in the first quarter of 2020. Additionally, the number of online music paying users touched 42.7 million, a year-over-year rise of nearly 50%. Spotify reported a 29% year-over-year growth

in paid subscribers in 2019. In Q4 2019, the company registered 124 million paid users. According to a report from the Recording Industry Association of America (RIAA), paid streaming services in the U.S. added an average of over one million new subscriptions each month, and the overall number of paid subscribers in the U.S. surpassed 60 million in 2019.



## Audio Streaming Challenges

Content Delivery Networks (CDN) play an important role in the live streaming ecosystem. It provides the backbone infrastructure to deliver the audio streams to end listeners. One major limitation of today's CDN networks is the so-called "last-mile" delivery problem. Typically, CDN providers build data centers called Point-of-Presence (PoPs) in many locations around the globe, with the expectation that these PoPs are geographically close to the listeners. However, the number of PoPs are limited, hence are not near enough to the majority of listeners, especially in less developed regions. The "last-mile" link is usually the bottleneck of today's streaming delivery pipeline and often leads to less optimal user experience including choppy streams, bad audio quality and frequent buffering.

To streaming sites and content platforms, another major concern is the CDN bandwidth cost. For popular sites, the CDN bandwidth cost can easily reach tens of millions of dollars per year. Even if platforms own proprietary CDNs, maintenance costs are often high.

These issues are becoming more prominent while streaming lossless audio and Dolby Atmos audio format.

- Low quality - Comes with a bit rate of 96 Kbps, using 0.72 MB a minute or 43.2 MB every hour.
- Medium Quality - Usually has a 160 Kbps bit rate, which translates to 1.20 MB bandwidth usage every minute or 72 Mb per hour.

- High quality - 320 Kbps or slightly higher rate, costing 2.40 MB per minute and 144 MB per hour.

The below table shows an approximation of how much music a user can stream with different data plans.

Data Plan	Low Quality	Medium Quality	High Quality
2 GB	47 hours	28 hours	17 hours
5 GB	117 hours	70 hours	42.5 hours
10 GB	234 hours	140 hours	85 hours

## Opportunity

Our mission is to leverage blockchain technology to create the first Decentralized Live Audio Streaming and Delivery Network whereby audio listeners are incentivized to share redundant computing and bandwidth resources to address today's live audio streaming challenges. Using the Ethereum EVM "World Computer" metaphor, the Allaudio Network can be viewed as the "World Cache" formed by the memory and bandwidth resources contributed by its listeners.

Specifically, listeners around the globe can contribute their devices as "caching nodes" whereby they form an audio delivery mesh network that is responsible for delivering any given live audio stream to listeners around the world optimized for local delivery. The Allaudio Network can effectively address the technical challenges discussed in the previous section. First, all listener's devices are much closer than to the CDN PoPs. This reduces packet round-trip time and improves the stream delivery quality, and thus addresses the "last-mile" delivery issue. Second, with sufficient amount of caching nodes, most listeners will receive the stream from caching nodes, which will help streaming sites reduce their CDN bandwidth cost. Third, caching nodes also reduce round-trip time making next generation streaming technology practical.

To encourage listeners to contribute their computing and bandwidth resources, Allaudio token is introduced as an incentive mechanism. Caching nodes can earn tokens as they relay audio streams to other listeners. Not only does the Allaudio Token motivate listeners to join the network as caching nodes, it also greatly improves the streaming market efficiency by streamlining the audio delivery process. Also within the Allaudio Network, advertisers can take advantage of the live streaming content category into consideration and submit both text and audio based advertisements, and since the network uses a key-based authentication for every user within the system it is virtually impossible to run targeted ad-campaigns based on the user's data.

The full launch of the Allaudio Protocol will introduce a new blockchain and a native token structure where:

- Caching nodes earn tokens for caching and relaying audio streams to other geographically near listeners in the network.
- Listeners optionally can earn tokens from advertisers as engagement rewards which can be gifted to live streamers of their choice.
- Streaming sites and platforms can drive incremental new revenues through sales of premium goods and services, and deepen user engagement through Allaudio Network.
- Advertisers may fund advertisement campaigns with tokens to support influencers, streaming sites and listeners.
- Streaming sites and platforms can offload 80% of CDN costs.

The Allaudio protocol adopts and builds upon the following novel concepts:

- **Multi-Level BFT:** A modified BFT consensus mechanism which allows thousands of nodes to participate in the consensus process, while still supporting very high transaction throughput (1000+ TPS). The core idea is to have a small set of nodes, which forms the validator committee, to produce a chain of blocks as fast as possible using a PBFT-like process. Then, the thousands of consensus participants, called the guardians, can finalize the chain generated by the validator committee at regular checkpoint blocks. The name multi-level BFT consensus mechanism reflects the fact that the validator/guardian division provides multiple levels of security guarantee. The validator committee provides the first level of protection - with 10 - 20 validators, the committee can come to a consensus quickly. The guardian pool forms the second line of defence. With thousands of nodes, it is substantially more difficult for attackers to compromise, and thus provides a much higher level of security. This mechanism achieves a good balance among transaction throughput, consistency, and level of decentralization, the three pillars of the so-called "impossible triangle".
- **Aggregated Signature Gossip Scheme:** A naive all-to-all broadcasting of the checkpoint block hash could work between guardian nodes, but it yields quadratic communication overhead, and so cannot scale to 1000+ nodes. Instead an Aggregated Signature Gossip Scheme can significantly reduce messaging complexity. Each guardian node keeps combining the partially aggregated signatures from all its neighbors, and then gossips out the aggregated signature. This way the signature share of each node can reach other nodes at exponential speed, thanks to the gossip protocol. In addition, the signature aggregation keeps the size of the node-to-node message small, and thus further reduces the communication overhead.
- **Resource Oriented Micropayment Pool:** An off-chain "Resource Oriented Micropayment Pool" that is purpose built for audio streaming. It allows a user to create an off-chain micropayment pool that any other user can withdraw from using off-chain transactions, and is double-spend resistant. It is much more flexible compared to off-chain payment channels. In particular, for the audio streaming use case, it allows a listener to pay for audio data pulled from multiple caching nodes

without on-chain transactions. By replacing on-chain transactions with off-chain payments, the built-in "Resource Oriented Micropayment Pool" significantly improves the scalability of the blockchain.

- **Key-Based Authentication and Profile Management:** As an alternative to password authentication, instead of requiring a user's password, it is possible to confirm the identity of each node by using asymmetric cryptography algorithm, RSA, with public and private keys. This empowers the user to own their private information and contradict any forms of targeted advertisement.

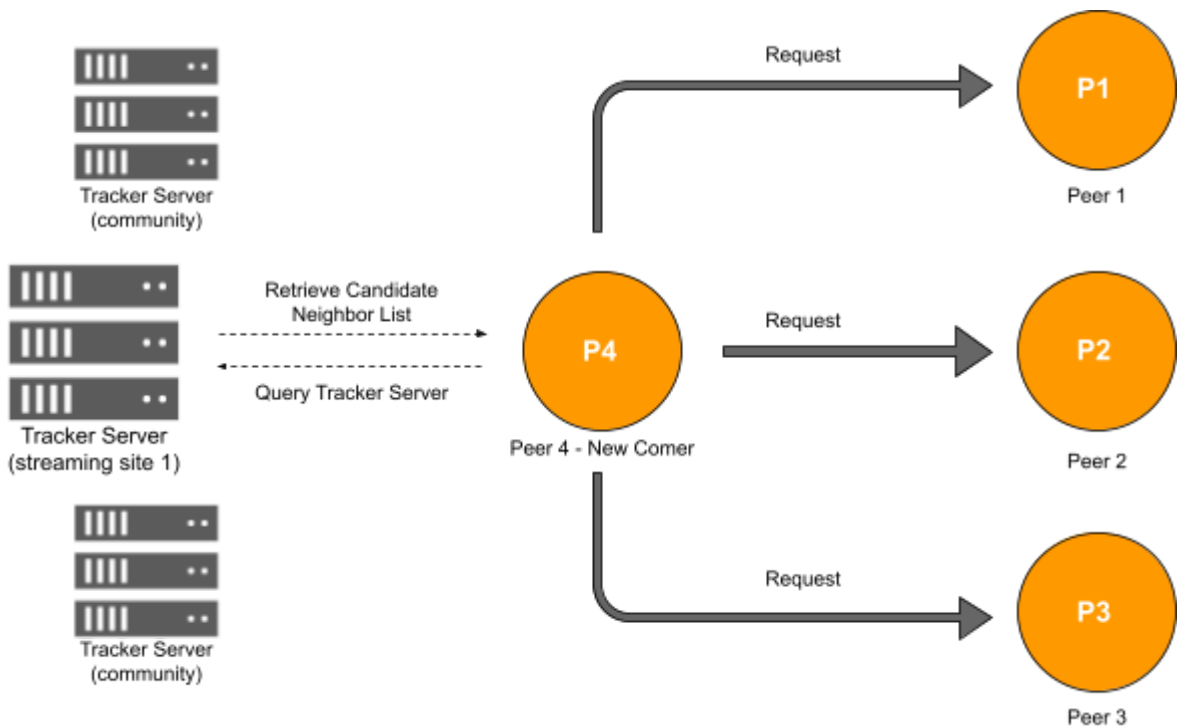
## Allaudio Mesh Delivery Network

Peer-to-peer streaming focuses on timely delivery of audio data under strict, near real-time parameters. Peer-to-peer livestream delivery works best when many users tune in for the same stream at the same time. High concurrent user count means more peering resources are available, and thus the peer nodes can pull the stream from each other more effectively. The whole system capacity increases as more peer nodes become available. Moreover, robustness of the system is increased in a peer-to-peer network, as nodes do not need to rely on a centralized CDN-based delivery, high concurrent users instead place scalability pressures on the CDN servers.

However, the shortcoming of pure peer-to-peer streaming is availability. Peers come and go at any time, which makes it difficult to predict the availability of any given peer node. There are also uncontrollable differences of nodes, such as upload and download capacities. On the other hand, a CDN service is more reliable and robust, and hence it can serve as a reliable "backup" when the stream is not available from peer nodes.

Our goal is to achieve maximum CDN bandwidth reduction without sacrificing the quality-of-service (QoS) which is critical to established streaming platforms. This means whenever possible we want the peer nodes to pull the stream from each other instead of from the CDN. To achieve this goal, it's crucial for the peer nodes to be able to identify the neighbouring nodes efficiently. If a node can identify multiple peers in close proximity, chances are that it can find peers that can provide the audio stream data segments much more consistently. On the contrary, if the identified peers are "further away" in terms of network hops, nodes might not be able to pull stream from peers consistently and cause degraded user experiences such as stuttering, frequent buffering, etc.

To address this problem, Allaudio Network has designed and is currently implementing a strategy which combines both a hyper-optimized tracker server and a player client-side intelligence. Essentially, the tracker server provides high level guidance (e.g. a list of candidate peers) for a player client, while the player client implements a peer filtering algorithm at a finer granularity based on multiple variables to find the neighbouring nodes that can best serve them.



{figure q: Interactions between the tracker servers and player clients}

## Geo-Optimized Tracker Server

In order to provide a list of candidate peer nodes to each client, the tracker server records the location information whenever a peer joins the network, including its IP address, latitude/longitude, and a number of other performance parameters. With this information the server can organize the nodes in a spatial database. A "hyper-optimized" spatial database is optimized for storing and querying data that represents objects defined in geometric space. As a peer node joins the network, the server can perform a spatial query to retrieve a list of candidate peers that are in close proximity very quickly and efficiently, see figure: q. The tracker servers and the spatial databases can be maintained by live audio streaming sites that use the Allaudio network and/or by community peers for content delivery.

As mentioned earlier, a peer node might leave the network at any time. Hence the tracker server also needs to be aware of which nodes are active. To achieve this, an active peer node needs to maintain a socket connection with the server and send heartbeat signals consistently. If the server does not receive a heartbeat for a certain amount of time, it considers that peer node as having left the network, and updates the spatial database accordingly.

An important distinction is that the "distance" between two peer nodes is measured by the number of router hops between them rather than the geographical distance. Typically network distance and geographical distance are highly correlated, but aren't necessarily equivalent. For instance, two computers could sit next to each other physically, but connect to different ISPs so there might be many hops between them. Hence, aside from geographical information, the tracker server also utilizes the connectivity between the IP addresses collected in the past to analyze and select neighbour candidates. For example, candidates returned by the spatial query can go through another filter to exclude those that are not connected to the same ISP as the listener's.

## Intelligent Player Client

Each peer node may act both as a listener, a caching node or both. As the node launches, during the handshake step, it retrieves a list of candidate peers from the tracker server for the livestream it's playing. Then, it performs a speed and availability test to select a subset that has optimized performance, connectivity and can reliably provide the audio stream data segments. The client performs the speed and availability tests regularly during a live stream session and continuously refines its neighbor list.

{figure p: Player stream data buffer handling}

To avoid QoS degradation, local buffer management is critical. The client player maintains a local cache to buffer the downloaded stream data as in figure p. If the duration of the cached stream data is less than a certain threshold, the player checks with the neighboring peers to see if they have the desired audio data segment. In the event when none of the neighbors has that particular audio data segment, the player intelligently falls back to the CDN. To achieve the best QoS possible, the player fetches an updated candidate list from the tracker server on a regular basis during a typical stream session.

The first version of the client player is a web/HTML5 based player which employs the WebRTC protocol for stream delivery among peers. Deploying web-based players requires minimal effort. Streaming sites and platforms simply embed the player onto their webpages, and it instantly has access and "launches" millions of end user nodes in the Allaudio mesh network. Thus, the deployment of Allaudio's mesh streaming technology is very light-weight and frictionless.

Allaudio Network also plans to release client dApps for all major desktop and mobile operating systems including Apple's Mac, Microsoft's Windows, Linux distributions, Android and iOS. Other methods of proposed client interaction are based on browser plugins or extensions supporting Google's Chrome, Mozilla Firefox and Apple's Safari.



# Allaudio Blockchain Ledger

The Allaudio Ledger is a decentralized ledger designed for the audio streaming industry. It powers the Allaudio token ecosystem which encourages end users to share their redundant bandwidth and storage resources, and enables them to engage more actively with streaming platforms and content creators. To realize these goals, a number of challenges, many of which are unique for audio streaming applications, need to be tackled.

One of such challenges is to support ultra high transaction throughput. Although many blockchain projects are facing transaction throughput problems, scaling for live audio streaming is different and probably even more complex. Typically, audio segments are a few seconds long. To achieve the finest granularity of a token reward - one micropayment per audio segment - even a livestream with a moderate ten thousand concurrent listeners could generate a couple of thousand microtransactions per second, which far exceeds the maximum throughput of today's public chains, such as Bitcoin and Ethereum. Popular live streams like major news channels can attract more than one million listeners listening to one stream simultaneously, not to mention multiple concurrent live streams, which could potentially push the required transaction throughput to the range of millions per second.

A byproduct of the high throughput is rapidly growing storage consumption. Storing the micropayment transactions is highly storage demanding. With tens of thousands of transactions added to the ledger every second, the storage space of an ordinary computer could run out quickly.

Any streaming applications typically require fast consensus. For bandwidth sharing rewards, the users that contribute redundant bandwidth, in our case every listener could allow upto a maximum of 100 peer connections and the caching memory required for the same, typically want the payment to be confirmed before sending the next one. Other use cases, such as virtual gift donations to live stream hosts, also require short confirmation times to enable real-time interactions between the hosts and audience.

Finally, as in any blockchain, security of the ledger is critical. Security is highly correlated with the level of decentralization. In a Proof-of-Stake (PoS) based consensus mechanism, decentralization means an even stake distribution among consensus participants. Ideally, the consensus mechanism should allow thousands of independent nodes, each with similar amounts of stake and each processing a local copy of the blockchain, to participate in the block finalization process. To compromise such a system, a significant amount of independent nodes would need to be controlled by the attackers, which is difficult to achieve.

To achieve these goals, we need to design and implement our PoS consensus algorithm based on the Byzantine Fault Tolerance (BFT) protocols, which offers good guarantees such as consistency (a.k.a. safety) when more than  $\frac{2}{3}$  of nodes running the ledger software are honest. However, the traditional BFT algorithms do not allow a high level of decentralization. They typically incur  $O(n^2)$  messaging complexity even for the normal (non-faulty proposer) case, where  $n$  is the number of nodes participating in the consensus

protocol. When we have thousands of nodes, it will take a considerable amount of time to reach agreement. To allow mass participation we adopt a multi-level BFT consensus mechanism to achieve an approximate of 1000+ TPS throughput with the transaction time as short as a few seconds.

Such level of transaction throughput, although already much higher than Bitcoin and Ethereum, is still not sufficient to handle the micropayments for the “pay-per-byte” granularity. To further increase the throughput, the Allaudio ledger provides native support for off-chain scaling, with a “resource oriented micropayment pool” which further amplifies the supportable throughput by several orders of magnitudes.

The off-chain payment support not only boosts the throughput, but also decreases the number of the transactions that need to be stored in the blockchain. On top of that, we also adopt the technique of state and block history pruning to further reduce the storage space requirement. Moreover, we have adopted the microservice architecture for the storage system, which can adapt to different types of machines and storage backends, be it powerful server clusters running in data centers, or commodity desktop PCs.

## The Consensus Mechanism

### Multi-Level BFT

The Allaudio ledger, built on a proven multi-level BFT consensus mechanism which allows thousands of nodes to participate in the consensus process, while still supporting very high transaction throughput (1000+ TPS).

The core idea is to have a small set of nodes, which form the validator committee, produce a chain of blocks as fast as possible using a PBFT-like process. With a sufficient number of validators (e.g. 10 to 20), the validator committee can produce blocks at a faster rate, and still retain a higher degree of difficulty to prevent an adversary from compromising the integrity of the blockchain. Hence, it is reasonable to expect that there is a very high probability the validators will produce a chain of blocks without forking. Then, the thousands of consensus participants, called the guardians, can finalize the chain generated by the validator committee. Here “finalization” means to convince each honest guardian that more than  $\frac{2}{3}$  of all the other guardians see the same chain of blocks.

Since there are many more guardians than validators, it could take a longer time for the guardians to reach consensus than the validator committee. In order for the guardians to finalize the chain of blocks at the same speed that the validator committee produces new blocks, the guardian nodes can process the blocks at a much coarser grain. To be more specific, they only need to agree on the hash of the checkpoint blocks, i.e. blocks whose height are a multiple of some integer  $T$  (e.g.  $T = 100$ ). This “leapfrogging” finalization strategy leverages the immutability characteristic of the blockchain data structure - as long as two guardian nodes agree on the hash of a block, with overwhelming probability,

they will have exactly the same copy of the entire blockchain up to that block. Finalizing only the checkpoint blocks gives sufficient time for thousands of guardians to reach consensus. Hence, with this strategy the two independent processes, i.e., block production and finalization, can advance at the same pace.

Under the normal condition, finalizing a checkpoint block is similar to the “commit” step of the celebrated PBFT algorithm since each guardian has already stored the checkpoint block locally. Moreover, the checkpoint block has been signed by the validator committee, and hence it is highly likely that all the honest guardians have the same checkpoint. Thus, we only need a protocol for the honest guardians to confirm that indeed more than  $\frac{2}{3}$  of all guardians have the same checkpoint hash.

To implement this protocol, a naive all-to-all broadcasting of the checkpoint block hash could work, but it yields quadratic communication overhead, and so cannot scale to large numbers of guardians. Instead a proposed aggregated signature gossip scheme which could significantly reduce messaging complexity, combines the partially aggregated signatures from its neighbors, and then gossips out the aggregated signature, along with a compact bitmap which encodes the list of signers. This way the signature share of each node can reach other nodes at exponential speed utilizing the gossip protocol. Within  $O(\log n)$  iterations, with high probability, all the honest guardian nodes should have a string which aggregates the signatures from all other honest nodes if there is no network partition. In addition, the signature aggregation keeps the size of the node-to-node messages small, and thus further reduces the communication overhead.

As mentioned above, the validator committee consists of a limited set of validator nodes, typically in the range of ten to twenty. They can be selected through an election process, or a randomized process, and may be subject to rotation to improve security. To be eligible to join the validator committee, a node needs to lock up a certain amount of stake for a period of time, which can be slashed if malicious behavior is detected. The blocks that the committee reaches consensus on are called the settled blocks, and the process to settle the blocks is called the block settlement process.

The guardian pool is a *superset* of the validator committee, i.e. a validator is also a guardian. The pool contains a large number of nodes, which could be in the range of thousands. With a certain amount of tokens locked up for a period of time, any node in the network can instantly become a guardian. The guardians download and examine the chain of blocks generated by the validator committee and try to reach consensus on the checkpoints with the above described “leapfrogging” approach. By allowing mass participation, we can greatly enhance the security of transactions. The blocks that the guardian pool has reached consensus on are called finalized blocks, and the process to finalize the blocks is called the block finalization process.

The name multi-level BFT consensus mechanism reflects the fact that the guardian/validator division provides multiple levels of security guarantee. The validator committee provides the first level of protection - with 10 to 20 validators, the committee can come to consensus quickly. Yet it is resistant enough to attacks - in fact, it already provides a similar level of security compared to the DPoS mechanism if each validator

node is run by an independent entity. Thus, a transaction can already be considered safe when it has been included in a settled block, especially for low stake transactions. The guardian pool forms the second line of defence. With thousands of nodes, it is substantially more difficult for attackers to compromise blockchain integrity, and thus provides a much higher level of security. In an unlikely event that the validator committee is fully controlled by attackers, the guardians can re-elect the validators, and the blockchain can restart, advancing from the most recent block finalized by the guardians. A transaction is considered irreversible when it is included in a finalized block. This mechanism is said to achieve a good balance among transaction throughput, consistency, and level of decentralization, the three corners of the so-called “impossible-triangle”.

This multi-level security scheme suits audio streaming applications well. For live streaming platforms, most of the transactions are micropayments (e.g. payments for peer bandwidth, virtual gifts to hosts/influencers, etc.) which typically have low value, but require fast confirmation. For such low stake payments, the users only need to wait for the block settlement, which is very fast, in a matter of seconds. For high stake transfers, the user can wait longer until the block containing the transaction is finalized, which could take slightly more time, but should happen within a few minutes.

## System Model

Before diving into the details of the block settlement and finalization process, we first list our assumptions of the system. For ease of discussion, without loss of generality, below we assume each node (be it a validator or a guardian) has the same amount of stake. Extending the algorithms to the general case where different nodes have different amounts of stake is straightforward.

**Validator committee failure model:** There are  $m$  validator nodes in total. Most of the time, at most one-third of them are byzantine nodes. They might be fully controlled by attackers, but this happens only rarely. We also assume that between any pair of validator nodes there is a direct message channel (e.g. a direct TCP connection).

**Guardian pool failure model:** There are  $n$  guardian nodes in total. At any moment, at most one-third of them are byzantine nodes. We do not assume a direct message channel between any two guardians. Messages between them might need to be routed through other nodes, some of which could be byzantine nodes.

**Timing model:** We assume the “weak synchrony” model. To be more specific, the network can be asynchronous, or even partitioned for a bounded period of time. Between the asynchronous periods there are sufficiently long periods of time where all message transmissions between two honest nodes arrive within a known time bound  $\Delta$ . As we discuss later in this paper, during the asynchronous period, the ledger simply stops producing new blocks. It will never produce conflicting blocks even with a network partition. During synchronous phases, block production will naturally resume, and eventual liveness can be achieved.

Attacker model: We assume powerful attackers. They can corrupt a large number of targeted nodes, but no more than one-third of all the guardians simultaneously. They can manipulate the network at a large scale, and can even partition the network for a bounded period of time. Yet they are computationally bounded. They cannot forge fake signatures, and cannot invert cryptographic hashes.

## The Block Settlement Process

Block settlement is the process in which the validator committee reaches agreement and produces a chain of blocks for the guardian pool to finalize. Inspired by the Proof-of-Stake research works including Tendermint, Casper FFG, and Hot-Stuff, we have designed and implemented the block settlement algorithm described below. It employs a rotating block proposer strategy where the validators take turns to propose new blocks. Then the committee votes on the blocks to determine their order using a protocol similar to Casper FFG and Hot-Stuff.

### Block Proposal

The validators rotate in a round robin pattern to play the role of block proposer, which is responsible for proposing the next block for the validator committee to vote on. To enable the round robin rotation, each proposer maintains a local logical clock called epoch. Assuming there are  $m$  validators, during epoch,  $t$ , the validator with index  $(t \bmod m)$  is elected as the proposer for that epoch. We note it is important that:

1. The epoch  $t$  should not be stalled so that liveness of the proposer rotation is guaranteed; and
2. The epoch  $t$  of different validators should be mostly in sync. I.e. most of the time all the validators have the same  $t$  value, so they can agree on which node should produce the next block.

Below is our protocol for proposer election and block proposal.

### Algorithm 1: Round Robin Block Proposal

```
 $t \leftarrow 0, proposer \leftarrow 0$   
 $voted \leftarrow false, received \leftarrow false, timeout \leftarrow false$   
loop begin  
   $proposer \leftarrow t \bmod m$   
  if( $proposer == self.index$ ) and (not proposed yet) begin //node elected as the proposer  
    propose one block  
  end  
  
   $voted \leftarrow$  the node has proposed or voted for a block for epoch  $t$   
   $received \leftarrow$  the node has received  $m/3 + 1 EpochChange(t + 1)$  messages  
   $timeout \leftarrow$  timeout reached  
  if  $voted$  or  $received$  or  $timeout$  begin  
    broadcast message  $EpochChange(t + 1)$   
  end  
  
  if the node has received  $2m + 3 EpochChange(t + 1)$  messages begin  
     $t \leftarrow t + 1$  //enters epoch  $t + 1$   
     $voted \leftarrow false, received \leftarrow false, timeout \leftarrow false$   
  end  
  
  sleep for some time  
end
```

#### Algorithm 1. The round robin block proposal protocol

The protocol defines a message  $EpochChange(t + 1)$ , which can be viewed as a synchronization message passed among the validators to assist them to advance to the next epoch  $t + 1$  together. Essentially, a validator broadcasts message  $EpochChange(t + 1)$  to all other validators if any of the following conditions are met:

1. the node proposed or voted for a block in epoch  $t$ , or
2. the node has received  $m/3 + 1 EpochChange(t + 1)$  messages from other validators, or
3. the node timed out for epoch  $t$  (the timeout is set to  $4\Delta$ ).

On the other hand, the validator enters epoch  $t + 1$  when it has received  $2m/3 EpochChange(t + 1)$  messages from other nodes.

Here we show that this protocol meets the above two requirements.

Eventual Progression: All the honest will eventually enter epoch  $t + 1$ . In the worst case, all the honest nodes (at least  $2m/3 + 1$  nodes) reach timeout and broadcast the  $EpochChange(t + 1)$  messages. Under the timing model assumption, all these messages will be delivered within time  $\Delta$  after being sent out. Thus each honest node will receive at least  $2m/3 EpochChange(t + 1)$  messages, and it then enters  $t + 1$ .

Epoch Synchrony: Intuitively, this means the epochs of all the honest nodes "move together". More precisely, we claim that the time any two honest nodes enter epoch  $t + 1$  differ by at most  $2\Delta$ . To prove this, we note that since there are at most  $f$  faulty nodes, for the first honest node to enter epoch  $t + 1$ , at least  $m/3$  other honest nodes must have broadcasted the  $EpochChange(t + 1)$  messages. This honest node then also broadcasts an  $EpochChange(t + 1)$  message following the protocol. After at most  $\Delta$ , any honest node should have received at least  $m/3 + 1$   $EpochChange(t + 1)$  messages, which triggers them to also broadcast the  $EpochChange(t + 1)$  message. After  $\Delta$ , all the honest nodes receive  $2m/3$   $EpochChange(t + 1)$  messages and enter epoch  $t + 1$ . Thus, at most  $2\Delta$  after the first honest node enters epoch  $t + 1$ , the last honest node will enter the same epoch.

In practice, when the network latency is small enough, all the honest nodes should enter epoch  $t + 1$  at almost the same time. As a result, they can agree on who is the next proposer. Also we note that for an actual implementation, the  $EpochChange(t + 1)$  messages can be combined with other types of messages (e.g. block votes) to improve the efficiency. So that in the normal case (no proposer failure), no additional synchronization overhead is added to the system for epoch changes.

## Block Consensus Among Validators

The protocol to settle proposed blocks involves a PBFT-based voting procedure among all validators, similar to Casper FFG and Hot-Stuff. In the Allaudio ledger blockchain, the header of each block contains a hash pointer to its parent block (i.e. the previous block in the chain), similar to Bitcoin and Ethereum. Two blocks are conflicting if neither block is an ancestor of the other. If there are multiple, conflicting block proposals for the same epoch, an honest validator keeps all of them until one becomes settled, and then it discards all conflicting blocks.

The block settlement protocol operates epoch by epoch. The proposer for the current epoch sends to all validators a block proposal. A validator reacts by broadcasting a vote for the block. All messages are signed by their senders.

The header of the proposed block might carry a commit-certificate, which consists of at least  $(2m/3 + 1)$  signed votes for its parent block. We note that under the assumption that no more than  $m/3$  validators are faulty, at most one block per height can obtain a commit-certificate. A commit-certificate for a block indicates this block and all its predecessors are committed. The proposed block may carry no commit-certificate, if its parent block did it get  $\geq 2m/3 + 1$  signed votes.

For the validators that are not the current proposer, their job is to vote on the proposed blocks. Once a validator receives the new block, it broadcasts a signed vote to all validators, so it can be collected by the proposer of the next epoch to form the commit-certificate. If two consecutive blocks A and B both receive a commit-certificate, then the parent block A and all its predecessors are considered settled. To ensure safety,

we require that honest nodes never vote for a block that conflicts with a settled block. When there are forks (either due to faulty proposer or asynchrony), the honest nodes should vote for the blocks on the largest fork.

The figure below illustrates the block settlement process. Assume that the proposer for height 101 is faulty, and it proposed two conflicting blocks  $X_{101}$  and  $Y_{101}$ , which leads to two branches. Assuming neither  $X_{101}$  nor  $Y_{101}$  gets  $\geq 2m/3 + 1$  votes, then, neither the header of  $X_{102}$  and  $Y_{102}$  contains the commit-certificate (denoted by *nil* in the figure). However, at some point branch X grows faster, and the two consecutive blocks  $X_{103}$  and  $Y_{103}$  both obtain  $\geq 2m/3 + 1$  votes. After that the upper branch X up to block  $X_{102}$  is considered settled. And the lower branch Y can be discarded.

*{figure b: The block settlement process}*

The above example also illustrates one advantage of our implementation compared to other PBFT based protocols like Tendermint - a block that does not receive a commit-certificate can also be included in the settled chain, as long as one of its successor blocks is settled. For instance, block  $X_{101}$  in the example did not get a commit-certificate, but after block  $X_{102}$  is settled, it is also considered settled. This reduces the waste of computation power and helps increase the transaction throughput.

## Analysis

**Safety:** Safety means all honest validators agree on the same chain of blocks. More precisely, if one honest validator accepts a block A, then any future blocks accepted by other honest validators will appear in a chain of blocks that already contains A. The argument for safety is similar to Casper FFG and Hot-Stuff and is omitted here, but pointing out that safety stems from the requirement that honest nodes never vote for a block that conflicts with a settled block.

**Liveness:** Liveness means the validator committee always makes progress, i.e., always able to produce and agree on new blocks. Here we show that under our timing model, during the synchronous periods, the committee can always achieve the liveness goal. First, in the "Block proposal" section, we have proved that the epoch can always advance, and all the honest validators march forward together. In an epoch where the proposer is an honest validator, it will propose a new block. For the block settlement process, liveness depends on that during the synchronous periods, there are infinitely many epochs where two proposers in a row are honest, and wait sufficiently long to form the commit-certificate. We note this is guaranteed to happen infinitely often with the round robin rotation, since at least  $\frac{2}{3}$  of the validators are honest.

**Transaction:** With ten to twenty validators, the committee can produce and settle the chain of blocks rather quickly. Average block production and settlement time is in the order of seconds, and this leads to high throughput of as much as 1000+ transactions per second.



## The Block Finalization Process

In this section, we will discuss the "leapfrogging" block finalization process in detail. As mentioned above, the guardians only need to reach consensus on the hashes of the checkpoint blocks, which are the blocks whose heights are multiple of some integer  $T$  (e.g.  $T = 100$ ).

To see why it is sufficient to finalize just the checkpoint blocks, we note that the transaction execution engine of the blockchain software can be viewed as a "deterministic state machine", whereas a transaction can be viewed as a deterministic state transfer function. If two nodes run the same state machine, then from an identical initial state, after executing the same sequence of transactions, they will reach an identical end state. Note that this is true even when some of the transactions are invalid, as long as those transactions can be detected by the state machine and skipped. For instance, assume there is a transaction that tries to spend more tokens than the balance of the source account. The state machine can simply skip this transaction after performing a sanity check. This way the "bad" transactions have no impact on the state.

In the context of blockchain, if all the honest nodes have the same copy of the blockchain, they can be ensured to arrive at the same end state after processing all the blocks in order. But with one caveat - the blockchain might contain a huge amount of data. How can the two honest nodes compare whether they have the same chain of blocks efficiently?

Here the immutability characteristic of the blockchain data structure becomes highly relevant. Since the header of each block contains the hash of the previous block, as long as two nodes have the same hash of the checkpoint block, with overwhelming probability, they should have an identical chain of blocks from genesis up to the checkpoint. Of course each guardian node needs to verify the integrity of the blockchain. In particular, the block hash embedded in each block header is actually the hash of the previous block. We note that a node can perform the integrity checks on its own, no communication with other nodes is required.

Interestingly, the immutability characteristic also enhances the tolerance to the network asynchrony or even partition. With network partition, the guardians may not be able to reach consensus on the hash of a checkpoint. However, after the network is recovered, they can move on to vote on the next checkpoint. If they can reach agreement, then all the blocks up to the next checkpoint are finalized, regardless of whether or not they have consensus on the current checkpoint.

To provide byzantine fault tolerance, an honest node needs to be assured that at least two-thirds of the guardians have the same checkpoint block hash. Hence it needs to receive signatures for a checkpoint hash from at least two-third of all guardians before the node can mark the checkpoint as finalized. This is to ensure safety, which is similar to the "commit" step in the celebrated PBFT protocol.

Since the guardians only need to vote on the checkpoint hashes every  $T$  blocks, they have more time to reach consensus. A straightforward implementation of checkpoint finalization is thus to follow the PBFT "commit" step where each guardian broadcasts its signature to all other guardians. This requires each node to send, receive and process  $O(n)$  messages, where each message can be a couple kilobytes long. Even with  $T$  blocks time, this approach still cannot scale beyond a couple hundred guardian nodes, unless we select a large  $T$  value, which is undesirable since it increases the block finalization latency.

## Scaling to Thousand of Guardians

To reduce the communication complexity and scale to thousands of guardians, we have designed an aggregated signature gossip scheme inspired by the BLS signature aggregation technique and the gossip protocol. The scheme requires each guardian node to process a much smaller number of messages to each consensus, which is much more practical. Below are the steps of the aggregated signature gossip protocol. It uses the BLS algorithm for signature aggregation.

### Algorithm 2: Aggregated Signature Gossip

```

finalized  $\leftarrow$  false,  $\sigma_i \leftarrow \text{SignBLS}(sk_i, \text{height}_{cp} || \text{hash}_{cp})$ ,  $c_i \leftarrow \text{InitSignerVector}(i)$ 
for  $t = 1$  to  $L$  begin
    send  $(\sigma_i, c_i)$  to all its neighboring guardians
    if finalized break
    wait for  $(\sigma_j, c_j)$  from all neighbors until timeout
    verify each  $(\sigma_j, c_j)$ , discard if it is invalid
    aggregate valid signatures  $\sigma_i \leftarrow \sigma_i \cdot \prod_j \sigma_j$ ,  $c_i \leftarrow \left( c_i + \sum_j c_j \right) \bmod p$ 
    calculate the number of uniquesigners  $s \leftarrow \sum^n I(c_i[k] > 0)$ 
    if  $s \geq \frac{2}{3}n$  finalized  $\leftarrow$  true
end

```

### Algorithm 2: The aggregated signature gossip protocol

The core idea is rather simple. Each guardian node keeps combining the partially aggregated signatures from its neighbors, and then gossips this newly aggregated signature

out. This way the signature share of each node can reach other nodes at exponential speed by using the gossip protocol. In addition, the signature aggregation keeps the size of the messages small, and thus reduces the communication overhead.

In the above diagram,  $i$  is the index of the current guardian node. The first line of the protocol uses function **SignBLS()** to generate its initial aggregated signature  $\sigma_i$ . It essentially signs a message which is the concatenation of the height and hash of the checkpoint block using the BLS signature algorithm, with multiplicative cyclic group  $G$  of prime order  $p$ , and generator  $g$ :

$$h_i \leftarrow H(pk_i, height_{cp} || hash_{cp})$$

$$\sigma_i \leftarrow (h_i)^{sk_i}$$

In the first formula above, function  $H: G \times \{0, 1\}^* \rightarrow G$  is a hash function that takes both the public key  $pk_i$  and the message as input. This is to prevent the rogue public-key attack.

The protocol also uses function **InitSignerVector()** to initialize the **signer vector**  $c_i$ , which is an  $n$  dimensional integer vector whose  $j^{th}$  entry represents how many times the  $j^{th}$  guardian has signed the aggregated signature. After initialization, its  $i^{th}$  entry is set to 1, and the remaining entries are all set to 0.

After initialization, the guardian enters a loop. In each iteration, the guardian first sends out its current aggregated signature  $\sigma_i$  and the signer vector  $c_i$  to all its neighbors. Then, if it has not considered the checkpoint as finalized, it waits for the signature and signer vector from all its neighbors, or waits until timeout. Upon receiving all signature and signer vectors, it checks the validity of  $(\sigma_j, c_j)$  using the BLS aggregated signature verification algorithm.

$$h_u \leftarrow H(pk_u, height_{cp} || hash_{cp})$$

$$\text{check if } e(\sigma_j, g) = \prod_u^n (e(h_u, pk_u))^{c_j[u]}$$

where  $e: G \times G \rightarrow G_T$  is a bilinear mapping function from  $G \times G$  to  $G_T$ , another multiplicative cyclic group also of prime order  $p$ . All the invalid signatures and their associated signer vectors are **discarded** for the next aggregation step. It is worth pointing out that besides  $height_{cp}, hash_{cp}$ , the above check also requires the public key  $pk_u$  of the relevant guardians as input. All this information should be available locally, since when a guardian locks up its stake, its public key should be attached to the stake locking transaction which has already been written into the blockchain. Hence, no communication with other nodes is necessary to receive these inputs.

The aggregation step aggregates the BLS signature  $\sigma_j$ , and updates the signer vector  $c_j$ . Note that for the vector update, we take  $\text{mod } p$  for each entry. We can do this because  $e(h_u, pk_u) \in G_T$ , which is a multiplicative cyclic group of prime order  $p$ . This guarantees that the entries of vector  $c_j$  can always be represented with a limited number of bits.

$$\sigma_i \leftarrow \sigma_i \cdot \prod_j \sigma_j, c_i \leftarrow \left( c_i + \sum_j c_j \right) \text{mod } p$$

The algorithm then calculates the number of unique signers of the aggregated signature.

$$s \leftarrow \sum^n I(c_i[k] > 0)$$

Here function  $I: \{\text{true}, \text{false}\} \rightarrow \{1,0\}$  maps a true condition to 1, and false to 0. Hence the summation counts how many unique signers have contributed to the aggregated signature. If the signature is signed by more than two-third of all the guardians, the guardian considers the checkpoint to be finalized.

If the checkpoint is finalized, the aggregated signature will be gossiped out in the next iteration. Hence within  $O(\log(n))$  iterations all the honest guardians will have an aggregated signature that is signed by more than two-third of all guardians if the network is not partitioned.

The loop has  $L$  iterations,  $L$  should be in the order of  $O(\log(n))$  to allow the signature to propagate through the network.

## Analysis

**Aggregated Signature Gossip Correctness:** To prove the correctness of the aggregated signature gossip protocol, we need to prove two claims. First, if an aggregated signature is correctly formed by honest nodes according to Algorithm 2, it can pass the check given by Formula (4). Second, the aggregated signature is secure against forgery. Stated more formally, forging a fake aggregated signature in the context of Algorithm 2 means to find  $\sigma \in G$  and integers  $c_1, c_2, \dots, c_n$  which satisfy the equation below

$$e(\sigma, g) = \prod_{u=1}^n (e(h_u, pk_u))^{c_u}$$

for randomly chosen  $pk_1 = g^{sk_1}, \dots, pk_n = g^{sk_n} \in G$ , and random message hashes  $h_1, \dots, h_n \in G$ . It can be shown that this is as hard as the Computational Diffie-Hellman (CDH) problem. For the proof of these two claims, please refer to our multi-level BFT technical report.

**Finalization Safety:** Safety of the block finalization is easy to prove. Under the  $\frac{2}{3}$  supermajority honesty assumption, if two checkpoint hashes for the same height both get aggregated signatures from at least  $\frac{2}{3}$  of all guardians, at least one honest guardian has to sign different hashes for the same height, which is not possible.

**Finalization Liveness:** Without network partition, as long as  $L$  is large enough, it is likely that after  $O(\log(n))$  iteration, all the honest nodes will see an aggregated signature that combines the signatures of all honest signers. This is similar to how the gossip protocol can robustly spread a message throughout the network in  $O(\log(n))$  time, even with up to  $\frac{1}{3}$  byzantine nodes. When there is network partition, consensus for a checkpoint may not be able to reach finalization. However, after the network partition is over, the guardian pool can proceed to finalize the next checkpoint block. If consensus can then be reached, all the blocks up to the *next* checkpoint are considered finalized. Hence the finalization process will progress eventually.

**Messaging Complexity:** The aggregated signature gossip protocol runs for  $L$  iterations, which is in the order of  $O(\log(n))$ . In each iteration, the guardian needs to send message  $(\sigma_i, c_i)$  to all its neighboring guardians. Depending on the network topology, typically it is reasonable to assume that for an average node, the number of neighboring nodes is a constant (i.e. the number of neighbors does not grow as the total number of nodes grows). Hence the number of message a node needs to send or receive to finalize a checkpoint is in the order of  $O(\log(n))$ , which is much better than the  $O(n)$  complexity in the naive all-to-all signature broadcasting implementation. We do acknowledge that each message between two neighboring guardians contains an  $n$  dimensional signer vector  $c_i$ , where each entry of  $c_i$  is an integer smaller than prime  $p$ . However, we note that this vector can be represented rather compactly since most of its entries are smaller integers ( $\ll p$ ) in practice.

To get a more concrete idea of the messaging complexity, let us work out an example. Assume that we pick a 170-bit long prime number  $p$  for the BLS signature, which can provide security comparable to that of a 1024-bit RSA signature. And there are 1000 guardians in total. Under this setting,  $c_i$  can be represented with about twenty kilobytes without any compression. Since most of the entries of  $c_i$  are far smaller than  $p$ ,  $c_i$  can be compressed very effectively to a couple kilobytes long. Plus the aggregated signature, the size of each message is typically in the kilobytes range. Moreover, if we assume on average a guardian connects to 20 other guardians, then  $L$  can be as small as 5 (more than twice of  $\log_{20}(1000) = 2.3$ ). This means finalizing one checkpoint just requires a guardian to send/receive around 100 messages to/from its neighbors, each about a couple kilobytes long. This renders the aggregated signature gossip protocol rather practical to implement

and can easily scale to thousands of guardian nodes. For further analysis, please also refer to the multi-level BFT technical report in this paper.

## Reward and Penalty for Validators and Guardians

The token reward and penalty structure is essential to encourage nodes to participate in the consensus process, and not to deviate from the protocol.

Both the validators and guardians can obtain a token reward. Each block includes a special Coinbase transaction that deposits newly minted tokens to the validator and guardian addresses. All the validators can get a share of tokens for each block. For guardians, rewarding every guardian for each block might not be practical since their number is large. Instead, the following algorithm is proposed to randomly pick a limited number of guardians as the reward recipient for each block.

Denote the height of the newly proposed block by  $l$ , and  $cp$  is the most recently finalized checkpoint. The proposer should have received the aggregated signature  $\sigma_{cp}$  and corresponding signer vector  $c_{cp}$  for checkpoint  $cp$ . Upon validating  $(\sigma_{cp}, c_{cp})$ , the proposer can check the following condition for each guardian whose corresponding entry in vector  $c_{cp}$  is not zero (i.e. that the guardian signed the checkpoint).

$$H(pk_i, \sigma_{cp} || B_{l-1}) \leq \tau$$

where  $B_{l-1}$  is the hash of the block with height  $l - 1$ , and  $H: G \times \{0, 1\}^* \rightarrow G$  is the same hash function used in the BLS signature algorithm. If the inequality holds, the proposer adds the guardian with public key  $pk_i$  to the Coinbase transaction recipient list. Threshold  $\tau$  is chosen properly such that only a small number of guardians are included. The proposer should also attach  $(\sigma_{cp}, c_{cp})$  to the Coinbase transaction as the proof of reward.

The Allaudio ledger also enforces a token penalty should any malicious behavior be detected. In particular, if a block proposer signs conflicting blocks for the same height, or if a validator votes for different blocks of the same height, they should be penalized. Earlier we mentioned that to become either a validator or a guardian, a node needs to lock up a certain number of tokens for a period of time. The penalty will be deducted from their hashed tokens. The node that detects the malicious behavior can submit a special Slash transaction to the blockchain. The proof of the malicious behavior (e.g. signatures for conflicting blocks) should be attached to the Slash transaction. The penalty tokens will be pulled from the malicious node and awarded to the node that submitted the first Slash transaction.

In the unlikely event that more than one-third of the validators are compromised, the malicious validators can attempt to perform the double spending attack by forking the blockchain from a block that is settled but not yet finalized. However, this is detectable by the guardian pool, since forking will generate multiple blocks with the same height, but signed by more than two-third of the validators. In this case, the validators that conducted the double signing will be penalized, and the entire validator committee will be re-elected. After the validator committee is reinstated, the blockchain can continue to advance from the most recent finalized checkpoint.

## Turing-Complete Smart Contract Support

This Allaudio ledger offers a smart contract runtime environment fully compatible with the Ethereum Virtual Machine. It provides full-fledged support for Turing-Complete smart contracts. Solidity-based Ethereum smart contracts can be ported to the Allaudio ledger with little effort. Solidity has grown a large developer community and the prospect of allowing that proven talent pool to also contribute to Allaudio without reinventing the wheel was a prime consideration in enabling compatibility with Ethereum Virtual Machine (EVM).

Smart contracts enable rich user experiences and new attribution models for audio platform dApps built on Allaudio ledger. For instance, audio platforms can make use of smart contracts for loyalty programs to engage users. Based on users' activity, or the volume of audio segment data they have cached and relayed, platform dApps may promote users to a higher tier, which unlocks certain privileges or exclusive capabilities. As another example, audio platforms can issue virtual items backed by the ledger blockchain (e.g. a virtual heart) for gifting to their favourite artist. To expand on such a concept, built on the "non-fungible token" standard, the virtual items could be rare or unique, such that they are essentially "crypto collectibles", which can be kept as trophies or traded for other sought after collectibles, all without additional permissions from 3rd parties.

Moreover, audio platforms are able to write smart contracts that enable more fluid payment-consumption models, such as pay-as-you-go or per-use models. Instead of traditional annual or monthly subscriptions, user consumption can be priced at a bite-sized granularity, such that users only need to pay for what they use. This is a feasible way to allow low-priced, short-form content to be transacted in an economically sensible way, that accrues benefits to both the audio platform and user. Allaudio ledger's properties of tracking micropayments and audio segments enables such smart contracts to be executed.

Smart contracts can also be designed to the benefit of content creators (e.g. user-generated content producers, large production studios) as a way to fairly and transparently distribute royalties. The traditional royalty settlement processes, with all their complexities and obscurities, can be accommodated with clear smart contract terms

that are mutually agreed upon by creators and distributors - and made available to users that consume the content.

Leveraging smart contracts on the Allaudio ledger to enable fully digitized item ownership, innovative payment-consumption models, and transparent royalty distributions provide an additional layer of social and economic interactivity that supplements the core functionality of the audio content delivery.

## Off-Chain Micropayment Support

As discussed in the introduction section, support for high transaction throughput is a must for an audio streaming focussed blockchain. Building the support for off-chain payment directly into the ledger to facilitate high volumes of transactions.

### Resource Oriented Micropayment Pool

We have designed and implemented an off-chain "Resource Oriented Micropayment Pool" that is purpose-built for live audio streaming. It allows a user to create an off-chain micropayment pool that any other user can withdraw from using off-chain transactions, and is double-spend resistant. It is much more flexible compared to off-chain payment channels. In particular, for the audio streaming use case, it allows a listener to pay for a live audio stream pulled from multiple caching nodes without on-chain transactions. By replacing on-chain transactions with off-chain payments, the built-in "Resource Oriented Micropayment Pool" significantly improves the scalability of the blockchain.

The following scenario and diagram provide a comprehensive walkthrough of how the Resource Oriented Micropayment Pool works in application.

*{Figure 1: Resource Oriented Micropayment Pool shows listener Ann making off-chain transactions to cachers Bob and Carol for audio chunks}*

- **Step 1. Micropayment pool creation:** As the first step, Ann publishes an on-chain transaction to create a micropayment pool with a time-lock and slashable collateral.

*CreatePool(resourceID, deposit, collateral, duration)*

A couple things to be noted. To create the pool, Ann needs to specify the "Resource ID" *resourceID* that uniquely represents the digital content she intends to retrieve. It refers to a live stream.



The *deposit* amount needs to be at least the total value of the resource to be retrieved. For example, if the resource is an audio file which is worth 10 tokens, then the deposit has to be at least 10 tokens.

The *collateral* is required to discourage Ann from double spending. If a double spending attempt from Ann is detected by the validators of the blockchain, the collateral will be slashed. Later in the document we will show that if  $collateral > deposit$ , the net return of a double spend is always negative, and hence any rational user will have no incentive to double spend.

The duration is a time-lock similar to that of a payment channel. Any withdrawal from the payment pool has to be before the time-lock expires.

The blockchain returns Ann the Merkle proof of the *CreatePool* transaction after it has been committed to the blockchain, as well as *createPoolTxHash*, the transaction hash of the *CreatePool* transaction.

- **Step 2. Initial handshake between peers:** Whenever Ann wants to retrieve the specified resource from a peer (Bob, Carol, or David, etc.). She sends the Merkle proof of the on-chain *CreatePool* transaction to that peer. The recipient peer verifies the Merkle proof to ensure that the pool has sufficient deposit and collateral for the requested resource, and both parties can proceed to the next steps.
- **Step 3. Off-chain micropayments:** Ann signs *ServicePayment* transactions and sends them to the peers off-chain in exchange for parts of the specified resource (e.g. a live stream segment). The *ServicePayment* transaction contains the following data

$$\begin{aligned} &targetAddress, transferAmount, createPoolTxHash, targetSettlementSequence, \\ &\text{Sign}(SK_A, targetAddress || transferAmount || createPoolTxHash || \\ &\quad targetSettlementSequence) \end{aligned}$$

The *targetAddress* is the address of the peer that Ann retrieves the resource from, and the *transferAmount* is the amount of token payment Ann intends to send. The *targetSettlementSequence* is to prevent a replay attack. It is similar to the "nonce" parameter in an Ethereum transaction. If a target publishes a *ServicePayment* transaction to the blockchain (see the next step), its *targetSettlementSequence* needs to increment by one.

The recipient peer needs to verify the off-chain transactions and the signatures. Upon validation, the peer can send Ann the resource specified by the *CreatePool* transaction.

Also, we note that the off-chain *ServicePayment* transactions are sent directly between two peers. Hence there is no scalability bottleneck for this step.

- **Step 4. On-chain settlement:** Any peer (i.e. Bob, Carol, or David, etc.) that received the *ServicePayment* transactions from Ann can publish the signed transactions to the blockchain anything before the timelock expires to withdraw the tokens. We call the *ServicePayment* transactions that are published as "on-chain settlement" transactions.

Note that the recipient peers need to pay for the gas fee for the on-chain settlement transaction. To pay less transaction fees, they would have the incentive to publish on-chain settlements only when necessary, which is beneficial to the scalability of the network.

We note that no on-chain transaction is needed when Ann switches from one peer to another to retrieve the resource. In the audio streaming context, this means the viewer can switch to any caching node at any time without making an on-chain transaction that could potentially block or delay the audio stream delivery. As shown in the figure, in the event that Bob leaves, Ann can switch to Carol after receiving  $k$  chunks from Bob, and keep receiving audio stream segments without an on-chain transaction.

Moreover, the total amount of tokens needed to create the micropayment pool is (*collateral* + *deposit*), which can be as low as twice the value of the requested resource, no matter how many peers Ann retrieves the resource from. Using computational complexity language, the amount of reserved token reduces from  $O(n)$  to  $O(1)$  compared to the unidirectional payment channel approach, where  $n$  is the number of peers Ann retrieves the resource from.

## Double Spending Detection and Penalty Analysis

To prevent Ann, the creator of the micropayment pool from double spending, we need to 1) be able to detect double spending, and 2) ensure that the net value Ann gains from double spending is strictly negative.

Detecting double spending is relatively straightforward. The validators of the Allaudio Network check every on-chain transaction. If a remaining deposit in the micropayment pool cannot cover the next consolidated payment transaction signed by both Ann and another peer, the validators will consider that Ann has conducted a double spend.

Next, we need to make Ann worse off if she double spends. This is where the collateral comes in. Earlier, we mentioned that the amount of collateral tokens has to be larger than the deposit. And here is why.

In Figure u below, Bob, Carol, and David are honest. Ann is malicious. Even worse, she colludes with another malicious peer, Edward. Ann exchanges partially signed transactions

with Bob, Carol, and David for the specified resource. Since Ann gains no extra value for the duplicated resource, the maximum value she gets from Bob, Carol, and David is at most the *deposit* amount. As Ann colludes with Edward, she can send Edward the full *deposit* amount. She then asks Edward to commit the settlement transaction before anyone else and return her the *deposit* later. In other words, Ann gets the resource which is worth at most the *deposit* amount for free, before the double spending is detected. Later when Bob, Carol, or David commit the settlement transaction, the double spending is detected, and the full *collateral* amount will be slashed. Hence, the net return for Ann is

$$net_{Ann} = deposit - collateral$$

Therefore, we can conclude that for this scenario, as long as *collateral* > *deposit*, then Ann's net return is negative. Hence, if Ann is rational, she would not have any incentive to double spend.

We can conduct similar analysis for other cases. The details are omitted here, but it can be shown that in all cases Ann's return is always negative if she conducts a double spend.

Another case is that Ann is honest, but some of her peers are malicious. After Ann sends a micropayment to one of those peers, it might not return Ann the resource she wants. In this case, Ann can turn to another peer to get the resource. Since each incremental micropayment can be infinitesimally small in theory, Ann's loss can be made arbitrarily small.

{Figure u. Malicious Actor Detection and Penalty shows malicious actor Ann attempting to make a double spend and the resulting penalty she receives}

## Ledger Storage System

Using a public ledger to facilitate the micropayments for streaming is challenging, not only because of high transaction throughput, but also for storage space management. To achieve the "pay-per-byte" granularity, each listener could spend out a payment every few seconds. With even a moderate ten thousand concurrent users, it could generate a couple thousands of transactions per second. Even with the off-chain payment pool which already reduces the amount of on-chain transactions dramatically, the block and state data could still balloon rather quickly.

Our storage system is built to address this problem, and can adapt to different types of machines, be it a powerful server cluster running in data centers, or a commodity desktop PC.

# Storage Microservice Architecture

To harness the processing and storage power of server clusters, the key design decision is to adopt the popular microservice architecture commonly seen for modern web service backends, where different modules of the ledger can be configured to run on different machines. In particular, the consensus module and the storage module can be separated. Potentially the consensus module can run on multiple machines using MapReduce framework to process transactions in parallel.

The Allaudio ledger stores both the transaction blocks and the account state history, similar to Ethereum. The bottom layer of the storage module is a key value store. The Allaudio ledger implements the interface for multiple databases, ranging from single machine LevelDB to cloud based NoSQL database such as MongoDB, which can store virtually unlimited amounts of data. Thus the ledger can run on one single computer, and can also be run on server clusters.

## History Pruning

While the microservice architecture suits the powerful server clusters well, we still face storage space constraints when running the ledger on a lower-end home PC. We have designed several techniques to reduce the storage consumption.

Similar to Ethereum, the Allaudio ledger stores the entire state for each block, and the state tree root is saved in the header of the corresponding block. To reduce the space consumed by the state history, the Allaudio ledger implements state history pruning, which leverages a technique called reference counting illustrated in the figure below.

*{Figure m. State history pruning with reference counting}*

The ledger state (i.e. the token balance of each account, etc.) is stored using Merkle-Patricia trie. Figure m(a) depicts the initial state tree, whose root is denoted by the State 0. Each node in the tree has an attribute called the "reference count", which is equal to the number of parents of the node. In the initial state tree, each node has only one parent, so the reference counts are all set to 1.

In Figure m(b), account A is updated to A\* after applying the transactions in the newly settled block. Hence a new Merkle state root State 1 is created, along with the Merkle branch connecting the new root State 1 and A\* (the blue nodes and edges). Since new nodes are added, we update the reference count of direct children of these new nodes from 1 to 2.

At some point we decided to delete State 0 to save some storage space. This is done by deleting the nodes whose reference count is zero recursively starting from the root State 0, until no node can be deleted. Whenever a node is deleted, the reference count of all its

children will be decremented by one. Figure m(c) illustrates the process, and Figure m(d) shows the result of the pruning. To achieve the maximum level to state storage compaction, once a block is finalized by the guardian pool, we can delete all the history prior to that block. The ledger can also be configured to keep a limited history of states, for example, the state trees of the latest 1000 blocks, depending on the available space.

It can be shown that with the reference counting technique, pruning a state tree has the time complexity of  $O(k \log N)$ , where  $k$  is the number of accounts updated by the transactions in one block, and  $N$  is the total number of accounts. Typically,  $k$  is in the range of a couple of hundreds to a thousand. Hence, pruning a state tree should be pretty efficient and should not take much time.

Managing the space consumed by the transaction blocks is even simpler, after a block is finalized, we can simply delete all its previous blocks, or keep a limited history similar to the state trees.

With these techniques, common PCs and laptops are sufficient to run the guardian nodes.

## State Synchronization

One of the pain points using earlier generation blockchains is the state synchronization time. After spinning up a new node, typically it needs to download the full block history all the way from genesis block. This could take days to complete, and already becomes a hurdle for user adoption.

The state and block history stored by the full nodes can help reduce the synchronization time dramatically. After a new node start, the first step is to download all the validator and guardian join/leave transactions and the headers of the blocks that contain these special transactions up to the latest finalized block. With these special transactions and the headers which contain the validator and guardian signatures, the new node can derive the current validator committee and guardian pool. Since the validator and guardian set changes are relatively infrequent, the amount of data needed to be downloaded and verified for this step should be minimal.

In the second step, the new node downloads the state tree corresponding to the latest and finalized block. And it needs to confirm that the root hash of the tree equals the state hash stored in the latest finalized block. Finally, the new node verifies the integrity of the state tree (e.g. the validity of the Merkle branches). If all the checks are passed, the new node can start listening to new blocks and start participating in the consensus process.

# A Dual Currency System and Token Mechanics

In the interest of securing the network, installing proper governance, and managing the usage of the network, the Allaudio blockchain will use a dual currency system. The Allaudio token will be used to stake, secure, and govern the Allaudio Network, while individual operations (live audio segment transactions, smart contract operations, etc.) will be paid for with the operational token, Tansen.

There are two reasons to introduce a second token:

First, this allows the utility and purpose of each token to be separated. Allaudio is used strictly for staking and securing the network, while Tansen is used to power utility-based operations of the network. This is necessary because staking inherently decreases circulation supply, but audio segment transactions and smart contracts will require a highly liquid token that can facilitate millions of daily transactions.

Second, two tokens are needed to solve possible consensus issues that arise from using the same token for staking and operations. Because the token used for operations must be liquid, it would be easier for a malicious actor to accumulate a significant number of that frequently-traded token on the open market. If that same token is also used for staking, they could potentially threaten the security of the Allaudio Network. By separating the two functions (staking and operations) into different tokens, that risk is greatly decreased.

## **Allaudio Token Supply and Mechanics**

As an ERC20 token, the Allaudio token, currently can be fixed at 1 billion. The supply of Allaudio tokens will be permanently fixed at 1 billion, meaning no new Allaudio tokens will ever be created.

The primary reason for fixing the Allaudio token supply is to make it prohibitively expensive for a malicious actor to acquire enough tokens to threaten the network. Since new Allaudio tokens will never be created, the only way to acquire more is by purchasing existing tokens and overtime making it more expensive to amass the controlling amount of Allaudio tokens.

## **Tansen Token Supply and Mechanics**

Tansen is the operational token of the Allaudio blockchain, used to pay for the audio segment microtransactions and smart contract operations. The Tansen token is also built on the Allaudio blockchain and 5 billion Tansen will be generated at the launch of Allaudio Network.

## **Validator and Guardian Nodes**

The validator set will initially be made up of nodes operated by Allaudio Network Inc, to be followed by additional validator nodes operated by key strategic partners.

website: <https://allaudio.network/>