**Livepeer Whitepaper**

**Protocol and Economic Incentives For a Decentralized Live Video Streaming Network**

Doug Petkanics doug@livepeer.org
Eric Tang eric@livepeer.org

## Abstract

The Livepeer project aims to deliver a live video streaming network protocol that is fully decentralized, highly scalable, crypto token incentivized, and results in a solution which can serve as the live media layer in the decentralized development (web3) stack. In addition, Livepeer is meant to provide an economically efficient alternative to centralized broadcasting solutions for any existing broadcaster. In this document we describe the Livepeer Protocol - a delegated stake based protocol for incentivizing participants in a live video broadcast network in a game-theoretically secure way. We present solutions for the scalable verification of decentralized work, as well as the prevention of useless work in an attempt to game the token allocations in an inflationary system.

## Table of Contents

Note: This paper was originally published in April, 2017. A scaling proposal called "Streamflow" has been proposed in December, 2018 which outlines some iterations and enhancements on some of the ideas presented below. Read the Streamflow Proposal here.

**Introduction and Background**

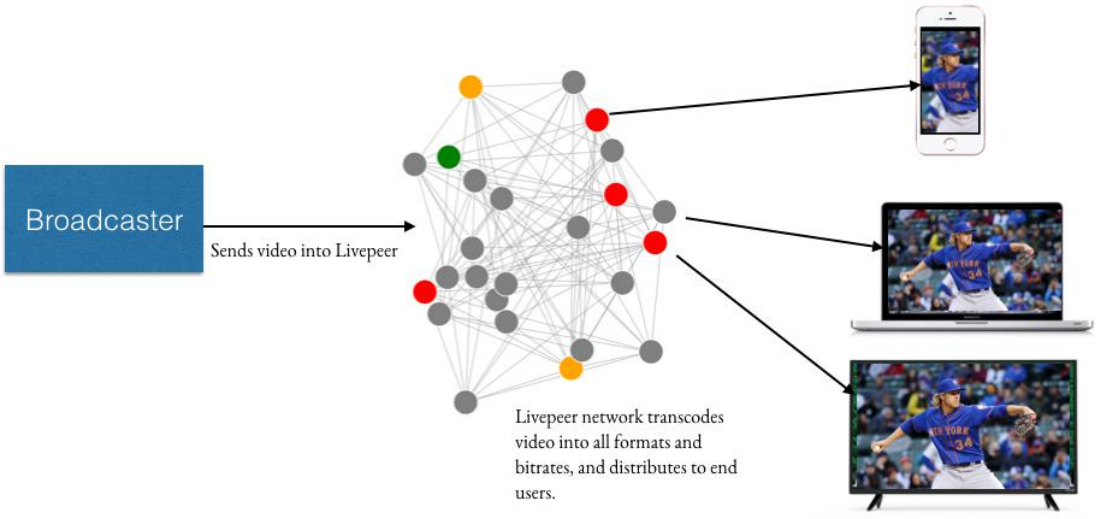
The vision of the decentralized web has begun to be realized over the past couple years with the emergence of networks like Ethereum to enable trustless computing, Swarm and IPFS/Filecoin to enable decentralized storage and content distribution, Bitcoin and various token projects to facilitate p2p transfer of value, and decentralized name registries like Blockstack and ENS to provide human accessible names to content and identities. These elements form the basis for decentralized applications (DApps) to be built in the form of largely static or infrequently updated web or mobile content, but at the moment DApps still lack the ability to include streaming media and data in an open and decentralized way. The goal of the Livepeer project is to decentralize live video broadcast over the internet.

The Livepeer Project Overview provides a nice introduction to the current state of live video on the internet. This whitepaper will largely focus on the cryptoeconomic protocol details of Livepeer, rather than the business case, but in summary the overview describes the current state of live streaming as growing at a rapid pace, centralized, and expensive. On the other hand, a fully decentralized P2P solution, where nodes contributed their own computation and bandwidth in service of streaming live video would be more open and scalable, as there would be no limit to the number of connections that could be served.

This technology is certainly available to a certain extent, but to date there has been no incentive to get users to run nodes that provide this functionality, nor has there been proper funding for the development of an open protocol that can facilitate this in a way that benefits the entire internet rather than one centralized company. However, with the recent emergence of crypto token powered protocols [2, 3], there is now an opportunity to simultaneously incentivize users to contribute computation and bandwidth towards live video broadcast, in a way that aligns with financing the development of an open media server solution capable of delivering live streamed video according to all the latest standards and formats required to reach the full range of devices. Additionally, the economic actions traditionally seen as a result of token powered protocols indicate that the cost to the broadcaster in order to use the Livepeer network could be cheaper than the cost of any centralized solution.

As the Livepeer technology and protocol are delivered, it will enable users to participate in the following flow:

1. Capture a video on your camera, phone, screen, or web cam and send it into the Livepeer network.
2. Nodes running within the network will encode it into all the necessary formats to reach every supported device. Users running these nodes will be incentivized via fees paid by the broadcaster in ETH, and the opportunity to build reputation through the protocol token to earn the right to perform more work in the future.
3. Any user on the network can request to view the stream, and it will automatically be distributed to them in near realtime.




**The Live Video Stack**

The technology stack for broadcasting live video has evolved over many years and contains many layers. Broadcasters need to capture video at the source, interface with a media server to process and transcode the video into many different formats, distribute the video across a network, and then allow the video to be played in high perceived quality by the end consumer. There are also economic questions that are introduced when one thinks through this stack, such as whether it should be the broadcaster or consumer who should be paying for the bandwidth to transfer the video.

A typical live streaming platform today needs to support RTMP, HLS, Mpeg-Dash video formats in H.264 and VP8 codec. New codecs like H.265/HEVC, VP9, and AV1 will become more popular in the near future as consumers become more accustomed to higher video quality. For HLS alone, Apple suggests bitrates from 145kb/s all the way up to 7800kb/s, in order to serve the

different types of devices under different conditions. All of this adds a significant amount of complexity and cost to live video broadcasting.

The existing decentralized development stack (web3) contains solutions for some of the layers required for a live video platform, like file transfer and payments, but currently there are no solutions for the capture and interface, transcoding and processing, and serving layers of live video. For this, Livepeer introduces the Livepeer Media Server (LPMS) - an open source implementation of a media server which provides all of the live video specific functionality necessary for DApp developers and existing broadcasters to build live functionality into their applications. Read more about it here.

As a standalone application, any developer could build a live application on top of the LPMS, but it would still be centralized and would need to be scaled through traditional means. However when every node on the Livepeer network is running the LPMS, and the protocol's economic incentives ensure that those nodes will contribute their processing power and bandwidth in service of transcoding and distributing live video, **a self-scaling, pay-as-you-go network is made available to developers, who can simply send their live stream into the network, and have the implementation details of scaling, payment, and media hosting abstracted away**.

### Livepeer Protocol

The Livepeer Protocol defines how the various actors in a live streaming ecosystem participate in a secure and economically rational way. The two major areas that the protocol needs to address are the actual distribution of live video from the source to a large number of consumers in a performant and scalable way, and the economic incentives for encouraging participation in the network in a secure and game-theoretic manner. While this whitepaper will touch on the live video distribution itself where overlapping with the economic protocol, it will largely focus on the latter in order to demonstrate security and economic alignment. At the highest level, the protocol is designed to:

- Allow any node to send a live video into the network, and optionally pay to have it transcoded into various formats and bitrates.
- Allow any node to request the video from the network.
- Allow participants to contribute their processing power and bandwidth in service of transcoding and distribution of video, and to be compensated accordingly.

In a decentralized network where participants are rewarded in proportion to the amount of work that they contributed, the two big challenges that need to be addressed to ensure security are:

- Can it be verified that the work that the nodes did was done correctly?
- Are the nodes being awarded for real work that contributed value to the network, as opposed to fake work done in an attempt to gain token allocations unfairly?

The Livepeer protocol is designed to address both the verification of work and the prevention of fake work, while also offering solutions for automatic scalability of the network and baked in governance for protocol evolution over time.

#### Video Segments

The core unit of media within Livepeer is what we will call a `segment`. A segment is a time sliced chunk of multiplexed audio and video of time length `t`. Every segment in the Livepeer network is unique, and contains the cryptographic evidence to verify

that the broadcaster intended this specific data for this specific segment. Each stream is made up of many consecutive segments, each containing a sequence number identifying their proper ordering. A segment contains the following fields:

| Video Segment Field | Description |
|---|---|
| **StreamID** | Identifies the origin node and stream that this segment belongs to. |
| **SequenceNumber** | The sequential order that this segment belongs in the original stream. |
| **DataPayload** | The binary metadata and data representing the audio/video in this segment. |
| **DataHash** | The hash of the data payload. |
| **BroadcasterSignature** | A signature from the broadcaster of `Priv(StreamID, SequenceNumber, hash(StreamID, SequenceNumber, DataHash` that the broadcaster claims this to be the true data for this unique segment. |

The Livepeer protocol generally uses segments as the unit of work for transcoding, distribution, and payments.

**Livepeer Token**

The Livepeer Token (LPT) is the protocol token of the Livepeer network. But it is not the medium of exchange token. Broadcasters use Ethereum's Ether (ETH) to broadcast video on the network. Nodes who contribute processing and bandwidth earn ETH in the form of fees from broadcasters. LPT is a staking token that participants who want to perform work on the network stake in order to coordinate how work gets distributed on the network, and to provide security that the work will get done honestly and correctly. LPT has the following purposes:

- It serves as a bonding mechanism in a delegated proof of stake system, in which stake is delegated towards transcoders (or validators) who participate in the protocol to transcode video and validate work. The token, and potential slashing that occurs due to protocol violation, is necessary in order to secure the network against a number of attacks. More below.
- It routes work through the network in proportion to the amount of staked and delegated token, essentially serving as a coordination mechanism.
- It is a unit of account that is specific to the Livepeer ecosystem, which forms the basis of a SectorCoin concept, applicable to additional functionality to be introduced in the future [4]. Services such as DVR, closed captioning, ad insertion/monetization, and analytics can all plug into the Livepeer ecosystem and potentially make use of the security provided by staking LPT.

An initial allocation of Livepeer Token will be distributed so that stakeholders can fulfill various roles in, and use the network, and then additional token will be issued according to algorithmically programmed issuance over time. See the Token Distribution section.

Following the conventions of Ethereum and many popular ERC20 tokens [16], LPT will be divisible by 10 ^ 18, with larger denominations such as the LPT itself intended to be used for user level transactions such as staking, and smaller denominations intended to be used for protocol accounting.

**Protocol Roles**

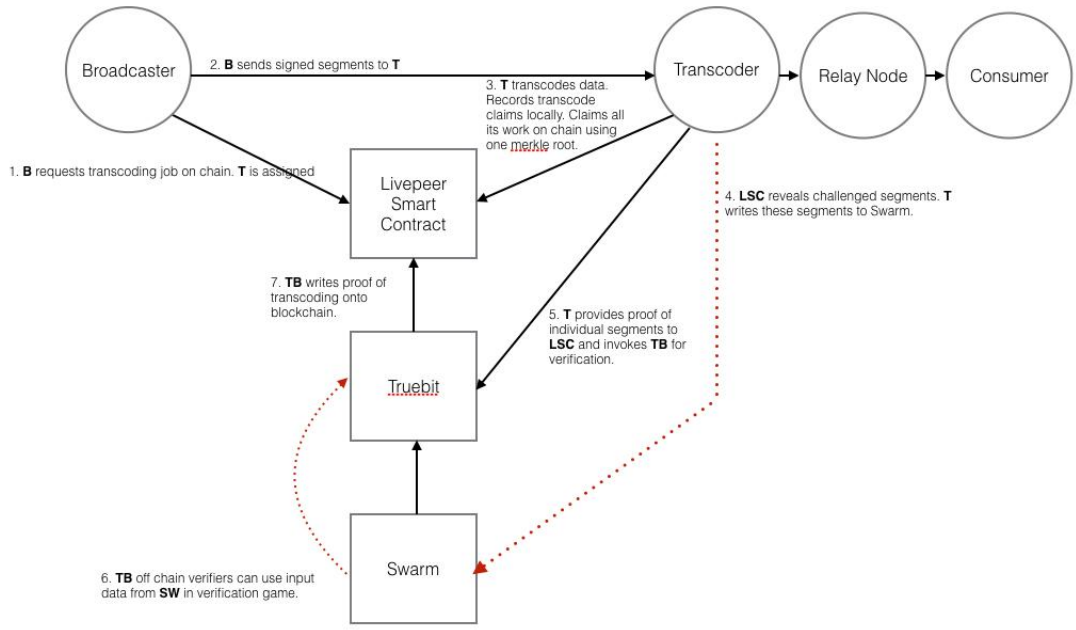
Before going forward, let's define the roles in the network so that there is a common vocabulary for discussing the protocol. A Livepeer node is any computer running the Livepeer software.

| Node Role | Description |
|---|---|
| **Broadcaster** | Livepeer node publishing the original stream |
| **Transcoder** | Livepeer node performing the job of transcoding the stream into another codec, bitrate, or packaging format. |
| **Relay Node** | Livepeer node participating in the distribution of live video and passing of protocol messages, but not necessarily performing any t |
| **Consumer** | Livepeer node requesting the stream, likely to view it or serve it through a gateway to their app or DApp's users. |

In addition to the above roles played by users running Livepeer nodes, the protocol also will refer to the following systems. While we use certain specific systems to make reference to a possible implementation, alternative systems can also be swapped in if they provide similar functionality and cryptoeconomic guarantees:

| System Role | Description |
|---|---|
| **Swarm** | Content addressed storage platform. Data can be guaranteed to be available there temporarily during the verification process via document we refer to Swarm, but other content addressed storage platforms can be substituted if data availability can be guarant |
| **Livepeer Smart Contract** | Smart contract running on the Ethereum network [1]. |
| **Truebit** | Blackbox verification protocol that guarantees correctness of computation placed on chain (at a hefty cost) [6]. (http://truebit.io) |

Here is a visual overview of the roles, and the ways in which they communicate with one another in the work verification process described below.



Segments flowing from the broadcaster to the transcoder and eventually to the consumer. The transcoder ensures they have signatures and proof of work to participate in the work verification procedure.

**Note on Transcoders:** Transcoders play the most critical role in the Livepeer ecosystem. They are the ones who are taking an input stream and converting it into many different formats in a timely manner for low latency distribution. As such they benefit from high availability, efficient, powerful hardware (potentially with GPU accelerated transcoding), high bandwidth connections, and solid DevOps practices. Transcoders should churn far less than other network participants, as when they take on the job of transcoding a stream, it's less than ideal if they drop off the network. While the network can scale to support many participants playing the role of transcoder (and earning the requisite token allocations), this is a special role that's delegated from most network participants, in order to ensure that a reliable network that provides value to broadcasters is maintained. More below on this delegation.

**Consensus**

Livepeer has a two layer consensus system. The LPT ledger and transactions are secured by the underlying blockchain, such as Ethereum. Any transfer of the LPT token or any transaction in the system can be considered to have been confirmed with the same security as the underlying proof of work or proof of stake blockchain. The second layer however, dictates the distribution of newly generated LPT. This is governed by the Livepeer Smart Contract, and participation in the protocol by various actors. While there is no consensus required per say, in terms of acceptance and validation of previous blocks, the protocol defines rules for participation and conditions upon which actors will be penalized (slashed) for failing to fulfill their role.

This second level of consensus governing the newly generated token is based upon Delegated Proof of Stake (DPOS), as inspired by systems like Bitshares, Steem, Tendermint, and Casper [5, 9, 10, 11]. The role of validators in the network is played by Transcoders. Any user can delegate their stake towards a transcoder, who then needs to perform transcoding jobs in the network, participate in the work verification protocol, and invoke functions on chain at specific intervals to validate this work. The protocol will distribute fees and newly generated token, and it will slash the stake of badly behaved actors. The validation result will be recorded on-chain via Truebit after it performs the validation, so there will be no room for disputes between the broadcaster and the transcoder.

**Bonding + Delegation**

In Livepeer, in order to indicate stake in the network, nodes must bond some amount of their LPT. They do this through the `Bond()` transaction, which will tie up their stake in the smart contract until they `Unbond()`, at which point they will enter an unbonding state which will last for `UnbondingPeriod` time. Upon completion of the `UnbondingPeriod` they can then withdraw their LPT.

The bonded amount is used to delegate stake towards a Transcoder. The network supports `N` active transcoders at any one time, which is a moveable network parameter. Any node can indicate that it wishes to be a Transcoder with a `Transcoder()`transaction, and the protocol will select the `N` transcoders with the most cumulative stake (their own + delegated from other nodes) at the start of each round, along with one random transcoder from the waitlist.

Newly generated token in Livepeer is distributed to bonded nodes in relative proportion to the amount of work that they have bonded (minus fees), as long as they've delegated towards transcoding nodes that behave according to the protocol. Bonds can be slashed (reduced by a certain percentage) if the nodes that they've delegated towards do not behave and violate one of the slashing conditions. Nodes who have bonded and delegated towards a Transcoder also receive a portion of the fees

that the Transcoder generates through transcoding jobs on the network. In essence, nodes who perform work, earn the fees that broadcasters paid for that work.

Going forward, when this document uses the term "delegator", it is referring to bonded nodes who have delegated their stake towards a transcoder candidate, instead of delegating it towards themselves as a transcoder.

In summary, participants choose to bond their stake for the following reasons:

- Participate in delegating towards effective transcoders who will provide great service to the network, ensuring its value to broadcasters.
- Build reputation and future-work allocation in form of allocated token in proportion to stake.
- Earn fees generated from transcoders.
- They may wish to be a Transcoder.

**Transcoder() Transaction**

A node indicates their willingness to be a transcoder by submitting a `Transcoder()` transaction, which publicizes the following three properties:

- `PricePerSegment`: the lowest price they are willing to accept to transcode a segment of video
- `BlockRewardCut`: The % of the block reward that bonded nodes will pay them for the service of transcoding. (Example 2%. If a bonded node were to receive 100 LPT in block reward, then 2 LPT to the transcoder).
- `FeeShare`: The % of the fees from broadcasting jobs that the transcoder is willing to share with the bonded nodes who delegate towards it. (Example 25%. If a transcoder were to receive 100 ETH in fees, they would pay 25 ETH to the bonded nodes).

The Transcoder can update their availability and information up until `RoundLockAmount` time before the next transcoding round. This is offered as a % of the round. (Example 10% == 2.4 hours. They can change this information until 2.4 hours before the next transcoding round which lasts for `RoundLength` 1 day). This gives bonded nodes the chance to review the fee splits and token reward splits relative to other transcoders, as well as anticipated fees based upon the rate they're charging and network demand, and move their delegated stake if they wish. At the start of a transcoding round (triggered by a call to the `InitializeRound()` transaction), the active transcoders for that round are determined based upon the total stake delegated towards each transcoder, and stakes and rates are locked in for the duration of that round.

There is one change that is allowed during the `RoundLockPeriod`: The lowest offered price/segment for any of the candidate transcoders is locked in and can't be moved, but other transcoder candidates can adjust their price/segment downwards. This allows them to match the lowest offered price on the network if they wish in order to guarantee their stake-weighted share of work on the network. They are not allowed to move their offered price upwards during this period.

Here is an example state of Transcoder options that a delegator can review when deciding whom to delegate towards.

| Transcoder ID | PricePerSegment | BlockRewardCut |
|---|---|---|
| 1 | 22 wei | 1% |

| Transcoder ID | PricePerSegment | BlockRewardCut |
|---|---|---|
| 2 | 30 wei | 2% |
| 3 | 10 wei | 4% |
| ... | ... | ... |
| N | 14 wei | 0% |

Note on price: In this document we list price/segment. In reality, Livepeer plans to use a gas accounting inspired model where there is a notion of units of gas required for certain job parameters of a segment such as bitrate, encoding, frame size, etc. Price/segment is a stand in, where the incentives are the same, but in reality they'll likely be communicating price/gas.

**Broadcast + Transcoding Job**

Transcoders who are open for business on the network, throw their hat into the ring for transcoding work by submitting a `TranscodeAvailability()` transaction. This indicates their availability and places them into a pool of transcoders available to take a newly submitted job.

When a broadcaster submits their stream into the Livepeer network it is given a `StreamID`. This serves as both a unique identifier, and it also contains the origin node address so that nodes know how to request and route requests to consume this stream towards the origin. The stream contains many consecutive `Segments`, as described in the Video Segments section. If the broadcaster would like the network to take care of transcoding their stream into all the formats and bitrates necessary to reach every user on every device, then the first step is submitting a transcoding job transaction on chain. Jobs are given a unique ID as well, and the input data to job consists of:

```
Job(StreamID, TranscodingOptions, PricePerSegment)
```

The `TranscodingOptions` define the output bitrates, formats, encodings, etc, and the `PricePerSegment` lists the price that the broadcaster will offer.

As soon as this transaction is mined, the next blockhash will be used to pseudo-randomly determine the transcoder selected for this job. All transcoders with a price that's lower than or equal to the price offered will be considered, and the blockhash modulus the number of candidate transcoders (weighted by their stakes) will determine the index of the selected transcoder.

At this point the broadcaster can begin streaming video segments towards the transcoder, and they'll participate in the following protocol. The protocol also makes use of a persistent storage solution, for example Swarm, as part of the work verification process.

**Preprocessing**

1. **Broadcaster** -> **Livepeer Smart Contract**: submits a deposit on chain to cover the cost of the full transcoding job. This can be refilled later at any point, but the Transcoder may stop work if the deposit runs out as they gradually cash in for work done.

**The Job**

1. **Broadcaster** -> **Livepeer Smart Contract**: Job(streamID, options, price/segment)
    - Creates the job request on chain and places some ETH in escrow to pay for the work.
2. The protocol can use the next block hash to deterministically select the correct Transcoder for this job.
3. **Transcoder** -> **Broadcaster**: send output streamID and receipt that the job is accepted.
4. **Broadcaster** -> **Transcoder**: send stream segments, which contain signatures verifying the input data.
5. **Transcoder** performs transcoding and makes new output stream available on network
6. **Transcoder**: Store a transcode receipt for each segment of transcoding work. A transcode receipt has the following fields.

| Transcode Receipt Field | Description |
|---|---|
| **StreamID** | Identifies the origin node and stream that this segment belongs to. |
| **Sequence Number** | The sequential order that this segment belongs in the original stream. |
| **Input Data hash** | The hash of the input segment data payload. |
| **Transcoded Data hash** | The hash of the output data after transcoding this segment. |
| **Broadcaster segment signature** | A signature from the broadcaster of Priv(StreamID, Seq#, Dhash) which can be used to attest and verify that the broa this unique segment. |
| **Transcoder segment signature** | A signature of all of the above fields from the transcoder attesting to the claim that this specific output transcoding v |

Whenever the transcoder observes that they are no longer receiving segments, they can call `ClaimWork()` to claim their work.

**End Job**

1. **Transcoder** -> **Livepeer Smart Contract**: Call `ClaimWork(JobID, StartSegmentSeq#, EndSegmentSeq#, MerkleRoot)`. Transcoder is claiming on chain they have performed work on the claimed segment range, with a merkle root of all of the transcode receipt data to commit to the content of these encoded segments.
2. Wait for this transaction to be mined, and observe the next blockhash. The protocol can then determine which segments will be verified based upon the `VerificationRate`.
3. **Transcoder** -> **Swarm**: Write input data payloads for the segments that will be challenged via verification, using SWEAR params to ensure the data will be there long enough for verification (`VerificationPeriod` time).
4. **Transcoder** -> **Livepeer Smart Contract**: Provide transcode claims on chain for each segment that needs to be verified, along with merkle proofs for the receipts for each segment in the transcode claims. The smart contract can verify the signatures from Broadcaster and **Transcoder** to ensure all data necessary is available to conduct verification, and can verify the merkle proofs against the committed merkle root from `ClaimWork()`.
5. **Transcoder** -> **Truebit**: `Verify()`. This is an onchain call to the Truebit smart contract, where the Transcoder provides the Swarm input hash for the challenged segment. (More on verification in the following section)
6. **Truebit** -> **Livepeer Smart Contract**: The result of the job is written on chain. This is compared to the transcoding claim result that the Transcoder provided.

7. **Livepeer Smart Contract**: at this point the Livepeer smart contract has all the information it needs to determine if the Transcoder's work is verified. - If verified correct, then use as input to token allocation algorithm and release of escrowed fees. - If incorrect, then Transcoder and its stakers get slashed `FailedVerificationSlashAmount` and the Broadcaster is refunded.

The Broadcaster can stop sending segments at any point, which effectively is an `EndJob()`.

At this point the transcoding has been performed, proof of the work has been claimed on the chain, and failure or success of the verification of the work has been reported. All the info is on chain to determine allocation of fees and token allocations to transcoders and delegators, or slashing in the case of failed verification. Let's take a look at how work is actually verified.

**Verification of Work**

In order to allocate fees to transcoders who claim that they have performed a transcoding job, it's necessary that the protocol can determine that the job was actually performed correctly with high probability. For this, Livepeer extends the research of, and makes use of, the Truebit Protocol [6].

Truebit works by having one participant (the solver) perform the actual work for the fee, in this case transcoding, and then having additional participants (verifiers) verify the work in order to detect mistakes, errors, or cheating. The task is broken down into very small steps, and the verifiers check the work of the solver to find the first step that differs from what they expected it to be. Then, only this one very small step needs to be played out on chain by a smart contract (judge), who can tell which party did the work correctly. The economic incentives, including forced errors to incentivize checking on the part of verifiers, ensure that it is not profitable to cheat or challenge incorrectly, but it is profitable to play the role of checking the work.

The downside of this protocol is that it costs between 5x-50x the cost of the original work in order to verify all work. Livepeer uses Truebit as a black box to verify segments, but it gets around having to pay this very high verification tax by only verifying a small percentage of segments randomly, and using slashing in the case of failed verifications. The `VerificationRate` set within Livepeer determines how frequently a specific segment is to be selected for challenge within Truebit, and the randomness of a future block hash after the work has been committed to the blockchain, determines which segments specifically are selected.

If work is committed via an `ClaimWork()` call in block `N`, then

If `Sha3(N, BlockHash(N), Seg#) % VerificationRate == 0` then the segment # must be verified.

The Transcoder provides Transcode Claims on chain for the candidate segments by invoking the `Verify()` transaction. The Livepeer Smart Contract can verify the authenticity of these claims using the internal signatures and provided merkle proofs, and then invoke a call to Truebit to verify only these segments.

Truebit solvers and verifiers access the input data for a segment from a persistent content addressed storage system, such as Swarm. The Transcoder is responsible for verifying that the segment data is available in Swarm, and can optionally look for receipts from the SWEAR protocol [5] guaranteeing persistence for a certain period of time, which is long enough for Truebit to play out. Additionally, they can take it upon themselves to run a Swarm node ensuring that the data is available to Truebit verification. If they have reason to believe that data is not available in Swarm, they can provide it, or just call `ClaimWork()` on the previously available data.

Truebit will write the results of the computation (succeeded or failed) back to the Livepeer Smart Contract, which can then be used in the reward and slashing calculations within the protocol. A transcoding node can not predict in advance which segments will be verified, and the following penalties will be felt in the case of cheating or failing to transcode correctly:

- `FailedVerificationSlashAmount` will be slashed if they fail a verification from Truebit.
- `MissedVerificationSlashAmount` will be slashed if they fail to provide transcode claims and invoke Truebit on segments they were required to do so.
- Lost fee from the broadcaster.
- Not only will the Transcoder be slashed, but all their delegators will be slashed as well. They will take this account into their decision of who to delegate towards, and the Transcoder could lose the lucrative job they hold.

It is important that it be more profitable to simply stake LPT towards a valid, honestly performing transcoder, than it can be to cheat and take slashing penalties while still collecting fees and token allocations for dishonest work. Careful selection of the slashing params and verification rate can ensure this.

**A Note On Truebit**

While the protocol makes use of Truebit in order to provide fully trustless verification of work, it may be necessary in practice to use available solutions that provide verification without the degree of trustlessness that Truebit can offer while Truebit is still under development and testing. Some options, ordered by degree of trustlessness, include:

1. Livepeer API Based Oracle - Trust Livepeer to verify computation. Very centralized, not ideal for anything beyond testing.
2. Oraclize Computation Service - Trust a company who provides proofs of computation and who's entire reputation relies upon putting external data on chain with proofs that it wasn't tampered with.
3. Secure hardware enclaves - Services like Intel SGX or TownCrier provide trusted computing environments. Trust that their hardware implementation is correct and secure. This can be decentralized and audited.

**Token Generation**

Livepeer is inflationary in that new tokens will be generated and allocated over time according to the schedule communicated below in Token Distribution. If all roles in Livepeer behave according to the protocol, then newly generated tokens will be allocated to users in proportion to their bonded stake (minus fees). Transcoders have the role of calling the `Reward()` function in order to trigger the new token allocation or slashing which can be computed from all data available on chain.

Each transcoder will be required to call `Reward()` once per round.

- Ensure that an active Transcoder is calling `Reward()`.
- Ensure that the Transcoder has not called `Reward()` yet in this round.
- Compute the number of token to mint based upon the `InflationRate`. Mint this many token.
- Calculate the Transcoder's cut based upon their `BlockRewardCut`.
- Distribute this into the Transcoder's bonded stake.
- Distribute the remainder into the delegators reward pool.
- Update the bonded amount of token to this Transcoder.

Failure to invoke `Reward()` results in the direct consequence of losing a portion of token allocations, and showing up as a ding on one's Transcoder reputation when it comes to being elected by Delegators for the role.

**Slashing**

As previously mentioned, the conditions for slashing are:

- Failing a verification
- Failing to invoke verification when required to do so
- Not performing a proportional share of the required work within the platform based upon delegated stake

One of the benefits of building within the Ethereum ecosystem are the network effect benefits you receive from being able to build on top of other protocols such as Truebit and Swarm/SWEAR. Unfortunately, with reliance on these external systems, which themselves have external dependencies and incentives, it's possible that a flaw or weakness in one of those protocols could result in slashing within Livepeer.

For example, if a Truebit verification job sat in their queue for a long period of time without any solver or verifier claiming it, Livepeer would fail to see the result of that verification in time before `Reward()` was called. Or if the Swarm network suffered a partition and couldn't propagate the file to the Truebit verifier in time, then this could also create an issue.

These risks can be mitigated by incentivizing these roles to be played in house by participants in the Livepeer protocol, who may find it in their best interest to serve as Truebit verifiers or Swarm nodes. But there's also another approach which is introducing the concept of probability thresholds on the slashing parameters. Optional protocol variables such as `VerificationFailureThreshold` could be set to indicate that as long as the node passes 99% of verifications they won't be slashed for example. This will remain a further area of research to be worked out prior to network deployment.

The failure to invoke verification slashing condition can be checked and invoked by any Livepeer protocol participant. There is a `FinderFee` which specifies the percent of the slash amount which the finder will receive as a reward for successfully invoking this slashing condition.

The remainder of the slashed funds will enter the `CommonPool`, which can be burned or allocated to common uses such as further ecosystem development, according to the governance mechanism of the protocol.

**Token Distribution**

As a token that represents the ability to participate and perform work in the network through a DPoS staking algorithm, the initial Livepeer token distribution will follow the patterns of other DPoS systems which require a widely distributed genesis state.

An initial allocation of the token will be distributed to the community at genesis and over the early stages of the network. Receipients can use it to stake into the role of Transcoder or Delegator. A portion will be allocated to groups who contributed prior work and money towards the protocol before the genesis, and a portion will be endowed for the long term development of the core project.

At the launch of the network, token issuance will continue according to an inflationary schedule with token being generated at `InflationRate` per round relative to the outstanding float of token. As token is issued in proportion to stake of all bonded

participants in the protocol, it serves to incentivize active participation. Participants are "protected" from this inflation, due to earning their proportional share. It is only inactive participants who are sitting on token without bonding it for participation, who will see their proportional network ownership dilluted by this inflation.

The initial target for `InflationRate` will be set such that it aims to incentivize approximately `ParticipationRate` of the LPT to be bonded and actively participating [19). For example, if `ParticipationRate` is 50% then incentives will exist to have half the oustanding token bonded. The inflation rate will move algorithmically each round to incent the participation target. A higher inflation rate would incent more token to be bonded, and a lower rate would lead to more people choosing liquidity rather than participation. It's this liquidity preference vs network ownership percentage tradeoff which should find equilibrium due to a number of economic factors in the network.

#### Governance

The role of governance within the Livepeer protocol is intended to be three fold:

1. Determine the burning or appropriation of common funds which were slashed from misbehaving nodes.
2. Adjust network parameters to ensure a healthy, thriving network which is valuable to broadcasters.
3. Invoke proposed protocol updates in a decentralized fashion.

Many of the network parameters referenced in this document such as `UnbondingPeriod`, `RoundLength`, `ParticipationRate`, and `VerificationRate` are adjustable. Proposals for adjustments to these parameters can be submitted, and the governance process, including voting by transcoders in proportion to their delegated stake, will determine adoption of these changes automatically within the protocol. The detailed spec for governance is left for another document. See more here.

### Attacks

This section contains a survey of the various ways that malicious actors may try to attack the Livepeer network. We use a rational attacker model in which the attacker makes decisions based upon their own economic self interest. A number of attacks are mitigated via it being unprofitable to conduct such attacks, but we also strive to ensure that at the worst the network suffers decrease of efficiency in the case of a sustained unprofitable attack, and doesn't suffer a failure.

#### Consensus Attacks

As mentioned previously, consensus in the Livepeer ecosystem is provided by the underlying blockchain platform (Ethereum for example). 51% attacks, double spends of Livepeer Token, and forks of the network would require the same resources and cost-of-attack as Ethereum itself.

Livepeer is a staked based protocol, and while Transcoders have the role of participating in the work verification process and the token reward distribution process, they actually do not have the role of validating or accepting other Transcoders' work. There is no concept of a chain, nor is there validation of previous blocks. There simply exists the economic incentives to verify one's own work and distribute one's own portion of token allocations when it is one's turn. As such, attacks that are seen in a proof of stake protocols such as the Long Range Attack, the Nothing at Stake problem, and The Bribe Attack don't apply, as there is no opportunity to attempt to sign multiple blocks or attempt to create a longer chain from an earlier state. However, one should be aware that as the underlying blockchain migrates to proof of stake, these attacks do threaten to undermine Livepeer if the benefit of carrying them out on Livepeer were to exceed the cost of attack on Ethereum itself.

While relying on the security of the underlying blockchain is nice for prevention of consensus attacks, there still exists a class of quality and efficiency attacks that can harm the Livepeer network.

**DDoS**

Denial of Service in Livepeer can go two ways:

1. A Transcoder can try to prevent or slow down a Broadcaster from getting their encoded stream out to the network by accepting a job but refusing to transcode.
2. A Broadcaster can prevent a Transcoder from being able to do the job that they believe they were assigned by refusing to send them segments.

Both attacks have a cost and can be mitigated, with slight annoyance.

In the first case, a Transcoder has to pay to claim their availability on chain. If they are not going to receive a fee because they didn't do the work, then they're throwing ETH away. The Broadcaster can just resubmit the job and be assigned a new Transcoder. One potential option for scalability is that the protocol can identify a number of valid Transcoders in priority order instead of just one, and this way the Broadcaster can just move on without another on chain transaction. Additionally, all stats about accepted jobs and average # of segments transcoded/job, etc, can be calculated from on-chain data, and delegators would use this as input into their decision about whom to delegate towards. Behave poorly and lose your role.

In the case of a Broadcaster preventing a Transcoder from doing work, this is merely a capacity planning calculation. A Transcoding node can maintain records of its capacity for concurrent jobs, likelihood of a job being active/inactive, and ensure that it always believes it will have capacity for the work that it claims. Simply ignoring or calling `EndJob()` on a node that's refusing to send segments hardly hurts the Transcoder.

**Useless or Self Dealing Transcoder**

If a Transcoder has enough stake to maintain their position, they could theoretically list a 100% `BlockRewardCut`, 0% `FeeShare`, and charge a high `PricePerSegment` such that they would never have to do any work, yet could collect their token allocation. This is prevented by the `CompetitivenessTolerance` which requires them to contribute some amount of valid work. Additionally, because of the transaction costs of participating in the protocol incurred by Transcoders, it would be more profitable for them to simply stake their token toward a valid Transcoder who was sharing fees with them, than it would be to act as a useless Transcoder who would receive no fees to speak of.

A misbehaving Transcoder who is outputting invalid output would quickly get slashed down to the point of their stake being reduced too low to actually keep their job and receive any work.

**Transcoder Griefing**

If a Broadcaster wanted to make the protocol very expensive to operate for a transcoder, it could send transcoders non-consecutive segment numbers. This is because transcoders can claim work for a continuous range of segment numbers in a single transaction, but would have to make many transactions to claim work across random segment number ranges. This can be defended against by the following options:

1. Transcoder calls `EndJob()` and doesn't bother doing the work or attempting to collect the fees.
2. Protocol implements on chain parsing or better segment claim encoding in order to reduce fees associated with claiming non-consecutive segments in a single call.
3. Simply ignore the segments and never claim the work.

This attack has a high cost to a broadcaster since they must have a deposit and submit jobs on chain in order to even get assigned to a transcoder in the first place. They have the ability to make life annoying for a transcoder and potentially lose efficiency, but not cause damage to the network.

**Chain Reorg**

When a broadcaster submits a job to the Livepeer Smart Contract, the protocol uses the current block hash to determine which transcoder will be assigned the job. Reorganizations of the underlying blockchain can cause confusion in this scenario. While this is not "an attack" directly, a transcoder will be valid one second, and then upon reorganization, will no longer be valid. When a reorg is detected the broadcaster can either redirect the stream towards the new valid transcoder, or the protocol can detect uncle blocks that are included in the main chain, and consider a transcoder to be valid if an uncle block within a given threshold would have made them valid.

**Live Video Distribution**

This whitepaper has largely focused on the economic incentives and protocol for ensuring proper transcoding of live video, which is necessary to support adaptive bitrate streaming and reach every device. But equally important is the distribution of video throughout the network so that it can be consumed with high quality and low latency. The economics of distribution rely on tit-for-tat bandwidth accounting as popularized by Bittorrent, and extended via protocols like SWAP [13]. As a simplification, nodes pay to request a segment of video, and nodes get paid to serve a segment of video. If a node already has a segment and can serve it to multiple requestors, it is profitable. We call this type of node, a Relay node.

Different incentives exist when it comes to bandwidth for nodes playing different roles in the network.

- Consumers may be willing to exchange upstream bandwidth to serve the content to additional Consumers in exchange for being able to consume the video themselves free of charge. See systems like Webtorrent [14].
- Broadcasters serve as origin nodes and may want to charge for consumption of the video, or may want to subsidize the cost of bandwidth so that everyone can access their video for free.
- Transcoders and Relay nodes are willing to provide bandwidth in service of distributing video as long as it is profitable. This is similar to the role of traditional CDNs.

With `Segments` as the core unit of data flowing through the network, it is possible to do tit-for-tat bandwidth accounting using ETH as the basis for settlement. We borrow the Chequebook Contract abstraction from Swarm [6] as a method of offchain payment passing with on chain settlement. Future developments in the ecosystem including the Raiden Network [15] may allow of payment channels to be used for this purpose as well. Since token transfer is native to the protocol, it is also possible to embed pricing associated with content directly into the protocol. A broadcaster can charge for their time or content directly, and nodes will opt into this transfer of value by paying a higher price/segment which will flow back to the broadcaster.

What's important to note is that while bandwidth accounting can be used to make it profitable to run Relay Nodes which just pass video segments around the network to add capacity, a-la a CDN, these nodes are purely incentivized by demand for the content, and not incentivized by new token allocations. In fact, the output of Livepeer can be inserted into a traditional CDN (like Amazon S3, Cloudflare, etc) or decentralized CDN (like IPFS or Swarm). Development of this peer-to-peer protocol for video segment distribution itself will be an ongoing opportunity for optimization and improvement in performance.

Peer-to-peer CDNs have been shown to reduce 80-98% of bandwidth requirements on an origin CDN server [17], and the token mechanics seen in decentralized networks can align stakeholders for the development and maintenance of an open version of the proprietary P2P CDNs that exist today. The PPSPP Protocol [18] serves as a viable candidate for an open implementation that focuses on delivery of live content.

As non-critical to the cryptoeconomics of the Livepeer protocol, the details are spared from this document, but the interested can follow along here with the development, and look for a future document addressing purely the video distribution protocol.

## Use Cases

The Livepeer project is concerned with decentralizing one-to-many live video broadcast (multicast). This is the truest form of media distribution, as it allows a broadcaster to connect directly with their audience in a first-hand manner, free from alterations, after-the-fact interpretation, and spin. It gives everyone a platform to have a voice. Existing centralized solutions can suffer from censorship, third party control over user data/relationship/monetization, and inefficient cost structures around payment for the service. Here are some of the logical use cases for applications and services to be built on top of Livepeer.

### Pay-As-You-Go Content Consumption

With a transfer of value transaction baked into the protocol, it is now possible for broadcasters to charge viewers directly for the consumption of their live broadcast, without requiring a credit card, account, or control over user identity via a centralized platform. This has applications in education (pay to attend an online course), events (pay to view a concert or live sporting event), entertainment (pay to watch a gamer or performer's live stream), and many other use cases - all while preserving the privacy of the viewer, and allowing them to pay for only what they consume directly to the broadcaster.

### Auto-scaling Social Video Services

One of the challenges of building consumer video services today is scaling infrastructure to support the demand for the growing number of streams and growing number of consumers as new users are added. A service layer that easily lets developers begin building their video solution on top of the Livepeer Network, which will automatically scale to support any number of streams and viewers as they go, will be a welcome solution to infrastructure developers who would otherwise have to continue provisioning servers, licensing media servers, and efficiently manage resources to handle spikes.

### Uncensorable Live Journalism

Current platforms such as Twitter and Facebook provide amazing live video solutions for reaching a large audience, but they're also the first to get blocked or censored in a variety of political conflict situations. Use of a decentralized network such as Livepeer would render it nearly impossible to prevent the word from getting out as to what is really going on on the ground in realtime.

### Video Enabled DApps

Decentralized apps (DApps) are beginning to emerge, driven largely by the Ethereum ecosystem. However, to date there hasn't been a viable solution for embedding live video within a DApp without using a centralized solution or limiting the number of consuming clients based on the constraints of WebRTC. By introducing Livepeer to the stack, an application can be fully decentralized, yet still contain live video, at scale, to as many users as wish to consume it.

## Summary

In summary, the Livepeer protocol incentivizes nodes to contribute their processing and bandwidth to the network in service of transcoding and distributing live video. The verification of work is solved by a scalable extension on top of the Truebit protocol which incentivizes nodes to perform transcoding operations correctly in order to earn their fees and token allocations and preserve their value earning role as a transcoder. The gamification of the network and false work problem is solved via the economics of the delegated proof of stake block reward accounting. It becomes more economically rational to simply stake one's tokens towards a value adding node than to pay fees into the network to be distributed to other delegators when performing work that there wasn't actually real demand for.

The end result is a scalable, pay-as-you-go network for decentralized live video broadcast - a missing layer in the web3 stack that Livepeer seeks to fill.

## Appendix

### Livepeer Protocol Parameter Reference

| Parameter Name | Description | |
|---|---|---|
| `T` | Segment length in seconds | 2 seco |
| `N` | Number of active transcoders | 144 |
| `RoundLength` | Length of time between election of a new round of transcoders | 1 day |
| `InflationRate` | The current target inflation rate per round of LPT. (Moves algorithmically). | .04% |
| `ParticipationRate` | The target percent of token bonded vs liquid. | 50% |
| `RoundLockAmount` | Transcoders rates lock in for this percentage of a round at the end of a round so that delegators can review and delegate accordingly without worrying about last minute rate changes. | 10% = |
| `UnbondingPeriod` | Time between entering unbonding state, and ability to withdraw the funds. | 1 mon |
| `VerificationPeriod` | The deadline for verifying a job claim after submission of the job claim. This also serves as the minimum period that a receipt of data persistence must be provided in the decentralized storage solution. | 6 hou |
| `VerificationRate` | The % of segments that will be verified. | 1/500 |

| Parameter Name | Description | |
|---|---|---|
| `FailedVerificationSlashAmount` | % to slash in the case of a failed verification (beyond the potential allowed failure threshold) | 5% |
| `MissedRewardSlashAmount` | % to slash in the case of missing a block reward round (Maybe only do this in the case of n consecutive misses) | 3% |
| `MissedVerificationSlashAmount` | % to slash in the case the transcoder didn't call verification | 10% |
| `CompetitivenessTolerance` | If all transcoders were always available and set the same price and fees, they would receive work in proportion to their stake. This parameter sets a % that they have to be within this target work % to be eligible for token allocation. This prevents transcoders from doing very little share of work relative to their stake. | 90% ( and 1 only c suppo |
| `*SlashingThresholds` (TBD) | Placeholder to indicate that we may not slash on all failures, only if they exceed some threshold % of failure rate. | |
| `VerificationFailureThreshold` | % of verifications you can fail without being slashed. Useful because of external dependencies like Swarm/Truebit that could cause sporadic failure. | 1% |
| `FinderFee` | % of slash amount that the finder will receive as compensation. | 5% |
| `SlashingPeriod` | The deadline for invoking a slashing condition after the `VerificationPeriod` has completed. | 1 hou |

**Livepeer Protocol Transaction Types**

| Transaction | Description |
|---|---|
| `Bond()` | Bond stake towards a transcoder. |
| `Unbond()` | Enter the unbonding state for the fixed `UnbondingPeriod`. |
| `Transcoder()` | Declare your intentions as a transcoder. |
| `ResignAsTranscoder()` | Resign your intentions as a transcoder. |
| `TranscodeAvailability()` | This transcoder is currently open to accepting another job. They're in the pool to be assigned randomly on new job s |
| `Job()` | Submit a transcoding job on chain. |
| `EndJob()` | End the job to relinquish transcoding responsibility. |
| `Deposit()` | Submit a deposit on chain that will be used and drawn against to pay for jobs. |
| `Withdraw()` | Withdraw from deposit and unbonded stake. |
| `ClaimWork()` | End the transcode job and make the claim of which segments you can prove you've transcoded via segment range a |
| `DistributeFees()` | Transcoder claims the fees for a particular claim after verification. |

| Transaction | Description |
| --- | --- |
| `Reward()` | Does all the verifications on chain to either slash or distribute token allocations. Can only be invoked by a transcoder per round. |
| `Verify()` | Transcoder provides the transcode claims for segments which will be verified along with merkle proofs for comparis Explicitly call Truebit to perform verification. |
| `InitializeRound()` | This transaction needs to be invoked once after the new round's start block to initialize the new active transcoder po |
| `UpdateDelegatorStake()` | This allows a delegator to claim their fees + token allocation from previous rounds. It's invoked automatically throug as a failsafe in case the delegator would like to update without changing state. |
| `*GovernanceTransactions()` | TBD |

**References**


1. Ethereum White Paper - Vitalik Buterin - Ethereum Wiki - https://github.com/ethereum/wiki/wiki/White-Paper
2. Fat Protocols - Joel Monegro - USV Blog - http://www.usv.com/blog/fat-protocols
3. Crypto Tokens and the Coming Age of Protocol Innovation - Albert Wenger - http://continuations.com/post/148098927445/crypto-tokens-and-the-coming-age-of-protocol
4. The Case For SectorCoins - Eric Tang - https://medium.com/@ericxtang/case-for-sectorcoins-b70a7c820c2d#.7892n4a57
5. Delegated Proof-of-Stake Consensus - Daniel Larimer - https://bitshares.org/technology/delegated-proof-of-stake-consensus/
6. Truebit - Jason Teutsch and Christian Reitweisner - https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf
7. swap, swear and swindle, incentive system for swarm - viktor trón, aron fischer, dániel a. nagy, zsolt felföldi, nick johnson - http://swarm-gateways.net/bzz:/theswarm.eth/ethersphere/orange-papers/1/sw%5E3.pdf
8. Kademlia: A Peer-to-peer Information System Based On The XOR Metric - Petar Maymounkov and David Mazieres https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf
9. Steem Whitepaper - Daniel Larimer, Ned Scott, Valentine Zavgorodnev, Benjamin Johnson, James Calfee, Michael Vandeberg - https://steem.io/SteemWhitePaper.pdf
10. Introducing Casper "the Friendly Ghost" - Vlad Zamfir - https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/
11. Tendermint Docs - Jae Kwon and Ethan Buchman - https://tendermint.com/docs
12. Swarm - http://swarm-gateways.net/bzz:/theswarm.eth/
13. Incentives Build Robustness in BitTorrent - Bram Cohen - http://bittorrent.org/bittorrentecon.pdf
14. WebTorrent - https://webtorrent.io/
15. Raiden Network - http://raiden.network/
16. ERC20 Token Standard - https://github.com/ethereum/EIPs/issues/20
17. Peer5 leverages viewers' devices for a P2P approach to streaming video - https://techcrunch.com/2017/01/26/peer5-y-combinator/

18. Peer-to-Peer Streaming Peer Protocol - https://tools.ietf.org/html/rfc7574
19. Inflation and Participation in Stake Based Protocols - Doug Petkanics - https://medium.com/@petkanics/inflation-and-participation-in-stake-based-token-protocols-1593688612bf