Livepeer Documentation

Release 0.0.1

Doug Petkanics

May 11, 2020

Contents

Livepeer is a decentralized video broadcasting platform powered by a crypto token on the Ethereum blockchain. Livepeer is for:

- Developers who want to build applications that include live video.
- Users who want to stream video, gaming, coding, entertainment, educational courses, and other types of content..
- Broadcasters who currently have large audiences and high streaming bills or infrastructure costs can use the Livepeer network to potentially reduce costs or infrastructure overhead.

Use this documentation to learn how to broadcast video through Livepeer, participate in the Livepeer protocol as a transcoder or delegator, and build apps or DApps with video based features using Livepeer.

We suggest you start with *Quickstart*. If you are interested in testing out Livepeer without setting up your own nodes, sign up for early access to the pilots program.

CHAPTER 1

Contributing

The code for this documentation is open source and is available on Github. Updates and pull requests are much appreciated.

CHAPTER 2

Index

2.1 Overview

Livepeer is a live video streaming network protocol that is fully decentralized, highly scalable, crypto token incentivized, and results in a solution which is cheaper to an app developer or broadcaster than using traditional centralized live video solutions.

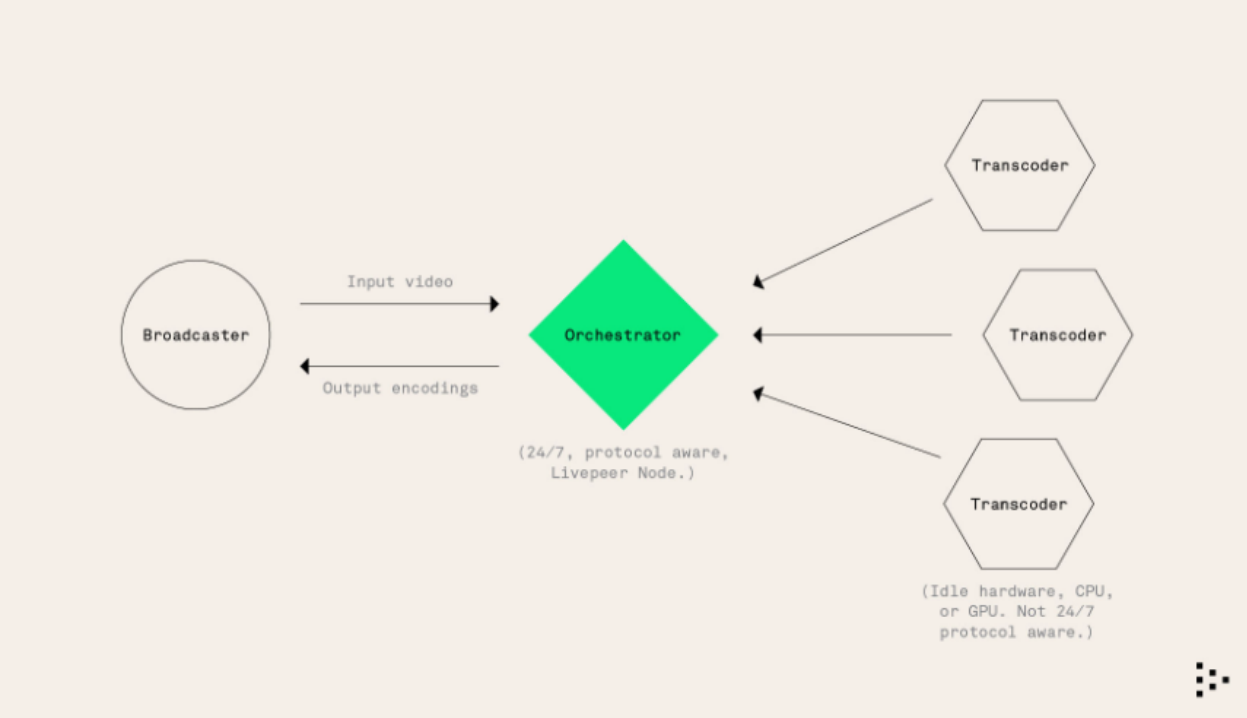
Users that participate in the Livepeer protocol have one of following roles:

1. Orchestrator
2. Transcoder
3. Broadcaster

An **orchestrator** is a protocol-aware, smart, 24/7 process that is responsible to the end user of the network for transcoding jobs being performed correctly. They stake LPT to secure the work that they perform, and ensure it is done correctly. They can be penalized if they maliciously cheat and mistranscode an end users content. When a user starts a Livepeer node on a CPU based server in `-orchestrator` mode, they are operating that process as an orchestrator.

A **transcoder** on the other hand, is a simple process that knows how to take an input segment of video, and transcode it to the desired outcome. It is not Livepeer protocol aware, it has no requirement for high reliability or being online 24/7, it makes no representation to the end user, and it has nothing at risk. While a user can start a Livepeer node on a server using the `-transcoder` flag to run a process in this role, they often will be running many transcoder processes, likely connected to many GPUs.

A **broadcaster** is a protocol-aware process that fulfills the demand side of the Livepeer network, it takes input streams from the end-user on its exposed RTMP interface to have them transcoded by the infrastructure providers running on Livepeer. The broadcaster takes care of splitting up streams into segments for transcoding and aggregating the transcoded results in a media playlist. Nodes running in `-broadcaster` mode will be able to determine the output renditions and maximum price per pixel for transcoding jobs it sends into the Livepeer network and pay for these jobs in ETH using probabilistic micropayments.



An orchestrator distributes work across one or many transcoders.

The most popular, and default setup, is that whomever is playing the role of orchestrator, is also running many transcoder processes, and they are distributing the work only to their own processes.

While it is possible to come up with constructions for "public transcoder pools" that allow orchestrators to distribute work to random transcoders, the design space for this sort of setup is outside the scope of this documentation.

This document will focus on teaching a user how to run their own orchestrator/transcoder setup to perform work on the Livepeer network. This includes:

- How to run an orchestrator
- How to scale transcoding on an orchestrator
- How to perform transcoding on a CPU and GPU
- How to run a broadcaster
- How to migrate your setup from the Livepeer alpha to Streamflow

2.2 Quickstart

2.2.1 Connecting to an Ethereum node

An Ethereum RPC provider needs to be specified via the `-ethUrl` flag when starting `livepeer` in order to connect to mainnet or Rinkeby.

If you do not want to run your own Ethereum node, you can use services such as Infura or Alchemy that host Ethereum nodes on behalf of users.

For example, you could use the following command to connect an orchestrator to Infura.

```
$ livepeer -network rinkeby -orchestrator -pricePerUnit 1 -ethUrl https://rinkeby.
↪infura.io/v3/<PROJECT_ID>
```

See the Infura docs for more details on how to obtain a <PROJECT_ID>.

If you would like to run your own Ethereum node you can use one of the clients mentioned here.

geth is recommended because it supports both mainnet and the Rinkeby testnet. Additionally, at this time livepeer has been the most thoroughly tested with geth.

Install geth using the instructions on the installing geth page.

Run geth:

```
$ geth -rinkeby -rpc -rpcapi eth,net,web3
```

If geth is running on a different machine than livepeer you will have to specify the -rpcaddr flag to indicate the interface to listen on.

Wait until geth is fully synced with the latest block on the Rinkeby testnet. You can check if geth is done syncing by using the Geth Javascript Console:

```
$ geth attach http://localhost:8545
Welcome to the Geth JavaScript console!

instance: Geth/v1.9.0-stable-52f24617/linux-amd64/go1.12.7
coinbase: 0x0161e041aad467a890839d5b08b138c1e6373072
at block: 583 (Wed, 23 Oct 2019 17:41:00 EDT)
 modules: debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

> eth.syncing
false
```

You could use the following command to connect an orchestrator to an Ethereum node running at localhost:8545.

```
$ livepeer -network rinkeby -orchestrator -pricePerUnit 1 -ethUrl http://
↪localhost:8545
```

2.2.2 Migrating to Streamflow

If you are upgrading from a go-livepeer < v0.5.3 you do not need to take any additional action - the node will create a fresh DB that is compatible with the changes included in the Streamflow protocol upgrade.

2.2.3 Run an orchestrator

Starting livepeer with the -orchestrator and -transcoder flags starts the node in orchestrator mode with solo transcoding. This is the simplest and fastest way to run an orchestrator and start transcoding video on the network. The transcoding section will describe the difference between solo and split transcoding as well as how to scale transcoding on your orchestrator with split transcoding.

The example commands in this document will use the -network rinkeby flag to connect the node to the Rinkeby public testnet. If you are connecting the node to mainnet, you should use the -network mainnet flag.

```
$ livepeer -network rinkeby -ethUrl <ETH_RPC_URL> -orchestrator -transcoder -
↪pricePerUnit 1
```

2.2.4 Run a broadcaster

Starting `livepeer` with the `-broadcaster` flag starts the node in broadcaster mode enabling you to stream video to be transcoded on the network.

```
$ livepeer -network rinkeby -ethUrl <ETH_RPC_URL> -broadcaster
```

Note that if you are already running an orchestrator node on the same machine, you will also have to pass additional flags into this command to specify unique ports so as not to conflict with your orchestrator node. See the below section on testing your transcoding setup for more detail.

2.2.5 Getting test ETH

If you are connecting to the Rinkeby public testnet you can get test ETH from the Rinkeby faucet.

2.2.6 Getting test LPT

You can get test LPT using `livepeer_cli` when running on Rinkeby. Before getting test LPT, make sure your account has test ETH.

In a separate terminal window other than the one that is running `livepeer`, run:

```
$ livepeer_cli
```

This command starts the CLI interactive wizard which can be used to issue commands to be executed by your node. The last few options of the terminal output should look something like this:

```
18. Get test LPT
19. Get test ETH
```

Select the option to get test LPT (note: the option numbering will be slightly different depending on if the wizard is connected to a node running in broadcaster or orchestrators mode). Upon entering the command in the wizard, you should see a transaction submitted by your node. After the transaction confirms, you can see your updated LPT balance by refreshing the wizard:

```
*----------------------------*-------------------------------------------*
|                ETH Account | 0xeb3F6d3adaA224aB84679b78376F3D96e8bF5781 |
*----------------------------*-------------------------------------------*
|                LPT Balance |                     10000000000000000000 |
*----------------------------*-------------------------------------------*
|                ETH Balance |                      9999925448000000000 |
*----------------------------*-------------------------------------------*
```

2.2.7 Open ports and networking

If you would like to test running an orchestrator or broadcaster node on the public testnet you may need certain ports open to the internet, or at least to specific IPs.

- Port **8935** (TCP) - Orchestrators need this port open to the world so that other broadcasters can discover and communicate with the orchestrator. Broadcasters should also open this port if they would like to serve their output video publicly from their node, or restrict access to this port for a specific audience or CDN.
- Port **1935** (TCP) - Broadcasters should open this port if they would like to stream into their node from a non-local source of video. You can restrict access to this port to the IP of the source of your video.

You now have a node running, and have the test ETH and LPT you need to begin interacting with the Livepeer network. Read on to learn how to activate your orchestrator node and confirm that it is transcoding video correctly.

2.3 Installation

First, download the latest mainnet compatible release for your platform from the releases page. If you are using OSX, download `livepeer-darwin-amd64.tar.gz`. If you are using Linux, download `livepeer-linux-amd64.tar.gz`.

To download the file using wget:

```
$ wget https://github.com/livepeer/go-livepeer/releases/download/<RELEASE_VERSION>/
↪livepeer-<YOUR PLATFORM>-amd64.tar.gz
```

After downloading the file, untar the archive and move the `livepeer` binary so that it is executable within your `$PATH`:

```
$ tar -zxvf livepeer-<YOUR_PLATFORM>-amd64.tar.gz
$ mv livepeer-<YOUR_PLATFORM>-amd64/livepeer /usr/local/bin
$ mv livepeer-<YOUR_PLATFORM>-amd64/livepeer_cli /usr/local/bin
```

2.4 Transcoding

Transcoding is the process of taking an input video in one format and bitrate, and converting it into many formats and bitrates to make it playable on the majority of devices on the planet at any connection speed.

In The Livepeer network, nodes who play the role of transcoder, perform this very important function, and as a result it's important that they have high bandwidth connections, sufficient hardware, and reliable devOps practices. These nodes are delegated towards and elected to perform this role, and they are rewarded with the ability to earn fees from the network.

Quicklinks:

Transcoder Megathread on Forum

Livepeer Explorer

Transcoder campaign thread

Livepeer Whitepaper

Transcoder chat

2.4.1 Install Livepeer

The following instructions assume that you have followed the installation and quickstart instructions.

Some of the `livepeer` commands in these instructions require an Ethereum node JSON-RPC URL to be provided via the `-ethUrl <ETH_RPC_URL>` flag. See the quickstart instructions for more details on obtaining an Ethereum node JSON-RPC URL.

2.4.2 Activation

In order to start transcoding video on the network and earning fees, your orchestrator node must be active.

An orchestrator (identified by its ETH account address) is active in a round if:

- It is registered on-chain. Registration consists of staking any amount of LPT and self-delegating. Anyone can register an orchestrator on-chain
- It is in the top 100 orchestrators with the most stake

The active orchestrator set is locked in at the beginning of each round. Any staking activity that occurs during a round will impact membership of the active set in the following round. So, if an orchestrator accumulates stake such that it is in the top 100 orchestrators with the most stake it will join the active set in the next round. If an orchestrator previously was in the top 100 orchestrators with the most stake, but no longer is in the top 100, it will exit the active set in the next round.

You can register an orchestrator and attempt to join the active set in the next round by selecting the following option in `livepeer_cli`:

```
13. Invoke multi-step "become an orchestrator"
```

Upon selecting the option, you should be prompted to:

- Set a `rewardCut` which is the percentage of inflationary LPT rewards that you will keep (the rest will be shared with your delegators)
- Set a `feeShare` which is the percentage of ETH transcoding fees that you will share with your delegators (the rest you will keep)
- Set a `pixelsPerUnit` which is the number of pixels you define to be in a unit, which will form the basis for your charging.
- Set a `pricePerUnit` which is the price that you charge per unit of transcoding (whole number, denominated in wei). See the *configuring payment parameters* for more details on pricing per pixel.
- Set an amount of LPT to stake and self-delegate
- Set a `serviceURI` which is the public accessible IP and port that broadcasters can send requests and video to be transcoded
  - The `serviceURI` is stored on-chain in the form `https://IP:port` (when registering you will just be asked for the IP and port). The IP should remain static since orchestrators are expected to provide consistent and reliable service, but a host (DNS) name can also be used for the `serviceURI` which provides orchestrators some flexibility. Orchestrators will not be able to serve the network if they are behind a NAT (i.e. a home router). If an orchestrator is behind a NAT, you will need to make special accomodations such as enabling port forwarding or putting the orchestrator in the DMZ. Be aware that there are many risks to running a public server. You should only run an orchestrator if you are comfortable managing these risks
  - The node will check if the current public IP matches the IP stored on-chain - if there is a mismatch, then your node might not be publicly accessible. You can override the inferred public IP by starting the node with the `-serviceAddr` and using the address stored on-chain, but you should make sure that your node is actually accessible using that address

After answering the wizard's prompt, you should see a few transactions submitted by your node. After the transactions confirm, you can see your orchestrator's registration status, stake, commission rates and pricing information by refreshing the wizard. If your orchestrator is in the top 100, it will join the active set at the beginning of the next round.

2.4.3 Testing your transcoding setup

You can test your orchestrator setup by setting up your own broadcaster and routing the broadcaster's requests directly to your orchestrator. Your orchestrator must be active before you can test the full transcoding/payment workflow - see the *activation section* for details on how to active your orchestrator.

First, make sure to turn on verbose logging on your orchestrator (transcoding/payment related logs will not be shown with the default logging level):

```
$ livepeer -network rinkeby -ethUrl <ETH_RPC_URL> -orchestrator -transcoder -
↪pricePerUnit 1 -v 99
```

Start a broadcaster that will connect directly to your orchestrator:

```
$ livepeer -network rinkeby -ethUrl <ETH_RPC_URL> -broadcaster -orchAddr <ORCH_
↪SERVICE_URI>
```

`<ORCH_SERVICE_URI>` should be the publicly accessible `serviceURI` that your orchestrator registered on-chain.

Follow the steps in the *broadcasting* section to deposit funds, configure broadcasting preferences and stream video into your broadcaster.

Look at the log output on your **orchestrator** node, and you should see your orchestrator start to transcode the incoming video. The broadcaster node will also receive the transcoded output back from the orchestrator and you can view your stream and each rendition in any web based or command line video player.

You can also test that your orchestrator is able to properly redeem *winning tickets* received from your broadcaster. Typically, the frequency of winning tickets in terms of clock time would be dynamically determined by the orchestrator based on the desired ticket expected value and price per pixel (these parameters are set by the orchestrator) as well as the current projected gas price required to redeem winning tickets and the amount/type of video that the orchestrator is transcoding. However, in this test scenario, since you are operating both the orchestrator and broadcaster, you can control all of the parameters mentioned to target a specific frequency of winning tickets.

First, check your orchestrator's logs to see the projected gas price for redeeming tickets (should see that the current gas price cached at a regular interval). Next, *select the renditions* to request for encoding. Now, run the PM calculator script using the following command:

```
$ python3 calc.py -t
```

You should be prompted for the desired ticket expected value, the projected gas price for redeeming tickets, the desired time (denominated in hours) for receiving a winning ticket and the set of renditions that will be encoded. After answering all of the prompts, the script should output the price per pixel that your orchestrator should set in order to receive a winning ticket in the desired time (this is just an approximation - if the desired time is 1 hour you will not necessarily receive a winning ticket exactly in 1 hour, but rather in 1 hour on average so the time elapsed might be more or less than 1 hour in practice).

Restart your orchestrator with the values used in the script and the price per pixel outputted by the script:

```
$ livepeer -network rinkeby -orchestrator -transcoder -ticketEV <TICKET_EXPECTED_
↪VALUE> -pricePerUnit <PRICE_PER_PIXEL> -v 99
```

Start streaming into your broadcaster and monitor the orchestrator's logs for a `redeemWinningTicket` log. Once this log is observed, you can use `livepeer_cli` and verify that the `Pending Fees` field in the wizard increased.

See the section on *configuring payment parameters* for a more detailed walkthrough of the payment parameter configuration process.

2.4.4 Configuring payment parameters

The Streamflow protocol upgrade introduces two main changes to the transcoding payment flow:

- A probabilistic micropayment protocol. Broadcasters send lottery tickets to orchestrators in exchange for transcoded results. Each lottery ticket is defined with a `faceValue`, the payout to the orchestrator if the ticket wins, and a `winProb`, the probability that the ticket will win. Each ticket is treated as a micropayment worth the expected value of the ticket (calculated as `faceValue * winProb`). Orchestrators will redeem winning tickets on-chain to receive the `faceValue` of tickets.
- A payment accounting protocol that meters the resources consumed by an orchestrator during transcoding using pixels. Orchestrators define a price per pixel (denominated in wei) which is the amount an orchestrator expects to be paid per pixel transcoded. The advertised price is used by broadcasters to filter the eligible orchestrators for selection (the default broadcaster implementation currently filters out orchestrators that advertise a price that exceeds the broadcaster's own max price). The video profiles requested by a broadcaster will impact the number of pixels that need to be transcoded by the orchestrator for each video segment. The more costly in terms of pixels it is to transcode a segment, the more tickets a broadcaster will need to send to compensate the orchestrator

An orchestrator can set its price per pixel by setting its price per unit, the amount of wei to charge for each unit of work, and its pixels per unit, the number of pixels that constitute a single unit of work. An orchestrator can set its price per pixel to be fractional wei by setting the number of pixels per unit to be greater than 1. The price per pixel can be set via:

- The `-pricePerUnit` and `-pixelsPerUnit` flags when starting the node
- The `livepeer_cli` wizard by selecting the set orchestrator config option

Note: At the moment, an orchestrator will only count the number of pixels *encoded*, but not the number of pixels *decoded* during transcoding. Metering of pixels decoded will be added in a future release.

An orchestrator can set its desired ticket expected value using the `-ticketEV` flag when starting the node.

While it is running, an orchestrator will monitor the expected gas price required to confirm a transaction on-chain. Using this gas price and the desired ticket expected value the orchestrator will dynamically set the `faceValue` and `winProb` to be used for tickets to target a 1% transaction cost overhead when a winning ticket is redeemed (1% of the `faceValue` of a winning ticket covers the cost of the on-chain transaction).

The following script can use an orchestrator's ticket expected value, price per pixel, a gas price for redeeming winning tickets and set of video profiles to be encoded to estimate:

- The value received per hour (in terms of ticket expected value)
- The frequency of winning tickets (in terms of hours)

Find below an example of using the script to project the value received per hour and frequency of winning tickets when transcoding 10 streams into 240p, 360p and 720p renditions:

```
  pm-params-calculator git:(master) python3 calc.py -f
DEFAULTS
---------
Ticket redemption gas cost: 100000
Transaction cost overhead: 0.01


Enter the desired ticket expected value (gwei): 1000


Enter the gas price (gwei) to use for ticket redemption transactions: 5
Ticket redemption gas price: 5.0 gwei
```

(continues on next page)

(continued from previous page)

```
Transaction cost to redeem a ticket: 0.0005000000 ether ($0.1000000000)
Ticket face value: 0.0500000000 ether ($10.0000000000)
Ticket winning probability: 0.000020000000000
1 out of 49999 tickets will win


Enter the price per pixel (wei) to charge: 1000
Price per pixel: 1000 wei


Would you like to add a rendition to be encoded? (y/n): y
Enter the output width: 426
Enter the output height: 240
Enter the output FPS (frames per second): 30
Enter the number of streams of this renditions: 10
Would you like to add a rendition to be encoded? (y/n): y
Enter the output width: 640
Enter the output height: 360
Enter the output FPS (frames per second): 30
Enter the number of streams of this renditions: 10
Would you like to add a rendition to be encoded? (y/n): y
Enter the output width: 1280
Enter the output height: 720
Enter the output FPS (frames per second): 30
Enter the number of streams of this renditions: 10
Would you like to add a rendition to be encoded? (y/n): n


Given the specified renditions you will:
Encode 1354579200000.0 pixels per hour
Receive 1354 tickets per hour
Receive 0.0013540000 ether ($0.2708000000) (in terms of ticket expected value) per␣
↪hour
Receive 1 winning ticket every 36.92688330871492 hours
```

In the above example, if an orchestrator transcodes 10 streams into 240p, 360p and 720p renditions, sets the ticket EV to 1000 gwei, redeems winning tickets on-chain using a gas price of 5 gwei and sets the price per pixel to 1000 wei then the orchestrator will receive approximately .001354 ETH of value per hour and 1 winning ticket every ~36.929 hours.

The script can also be used to calculate the price per pixel needed to target a desired frequency of winning tickets (in terms of hours) which can be useful for testing that your orchestrator can properly redeem winning tickets.

```
  pm-params-calculator git:(master) python3 calc.py -t
DEFAULTS
---------
Ticket redemption gas cost: 100000
Transaction cost overhead: 0.01


Enter the desired ticket expected value (gwei): 1000
Ticket expected value: 1000.0 gwei


Enter the gas price (gwei) to use for ticket redemption transactions: 5
```

(continues on next page)

(continued from previous page)

```
Ticket redemption gas price: 5.0 gwei


Transaction cost to redeem a ticket: 0.0005000000 ether ($0.1000000000)
Ticket face value: 0.0500000000 ether ($10.0000000000)
Ticket winning probability: 0.000020000000000
1 out of 49999 tickets will win


Enter the desired number of hours (i.e. 1, .5, etc.) until receiving a winning␣
↪ticket: 1


Would you like to add a rendition to be encoded? (y/n): y
Enter the output width: 426
Enter the output height: 240
Enter the output FPS (frames per second): 30
Enter the number of streams of this renditions: 10
Would you like to add a rendition to be encoded? (y/n): y
Enter the output width: 640
Enter the output height: 360
Enter the output FPS (frames per second): 30
Enter the number of streams of this renditions: 10
Would you like to add a rendition to be encoded? (y/n): y
Enter the output width: 1280
Enter the output height: 720
Enter the output FPS (frames per second): 30
Enter the number of streams of this renditions: 10
Would you like to add a rendition to be encoded? (y/n): n


Given the specified renditions you will and a target of 1 winning ticket every 1.0␣
↪hours you will:
Need to charge 36911.09386590315 wei per pixel
Encode 1354579200000.0 pixels per hour
Receive 49999.0 tickets per hour
Receive 0.0499990000 ether ($9.9998000000) (in terms of ticket expected value) per␣
↪hour
```

In the above example, if an orchestrator transcodes 10 streams into 240p, 360p and 720p renditions, sets the ticket EV to 1000 gwei, redeems winning tickets on-chain using a gas price of 5 gwei and targets 1 winning ticket every hour then the orchestrator should set the price per pixel to 36911.09386590315 wei.

In practice, the price set by orchestrators will likely also be influenced by the market rate for transcoding on the network and an orchestrator's own infrastructure cost. This information can be surfaced by tools that will be built in the future.

2.4.5 Scaling transcoding

The easiest way to setup an orchestrator to transcode video is to have the orchestrator perform solo transcoding by passing in both the `-orchestrator` and `-transcoder` flags when running `livepeer`. However, if you want to scale out your transcoding operation to support many concurrent streams, you will want to enable split transcoding.

Split transcoding consists of running many individual transcoder nodes (usually on separate remote machines) that connect to your orchestrator. When your orchestrator receives video from a broadcaster, it will then distribute the

incoming segments to the invidual transcoders instead of transcoding the video on its own. An orchestrator can increase its capacity and scale horizontally by adding more transcoders to its backend.

In order to enable split transcoding, you should first run an orchestrator with solo transcoding turned off:

```
$ livepeer -network rinkeby -orchestrator -pricePerUnit 1 -orchSecret <ORCH_SECRET>
```

<ORCH_SECRET> is a secret defined by the orchestrator that is used to authenticate requests from transcoders. The secret should be shared with all transcoders to be attached to the orchestrator.

Next, you should attach a transcoder to your orchestrator:

```
$ livepeer -transcoder -orchAddr <ORCH_SERVICE_URI> -orchSecret <ORCH_SECRET>
```

<ORCH_SECRET> should be the secret defined by your orchestrator. <ORCH_SERVICE_URI> should be the publicly accessible `serviceURI` that your orchestrator registered on-chain.

You can run the above command on any number of machines that you would like to dedicate to transcoding.

2.4.6 GPU transcoding

When you setup split transcoding for your orchestrator, you can run individual transcoders on not only CPUs, but GPUs as well. The GPUs that transcoders run on can be dedicated purely to transcoding or can also be mining cryptocurrencies simultaneously. See this guide for instructions on how to setup transcoders on GPUs.

2.4.7 FAQ

What does being 'publicly accessible' mean? Can I run a transcoder from home?

- The transcoder should be reachable by broadcasters via the public IP and port that is set during transcoder configuration. Transcoders will not be able to serve the Livepeer network if they are behind a NAT (eg, a home router). If this is the case, special accommodations must be made for the transcoder, such as port forwarding or putting the the transcoder in the DMZ. The only port that is required to be public is the one that was set during the transcoder registration step (default 8935). Be aware that there are many risks to running a public server. Only set up a transcoder if you are comfortable with managing these risks.

What is the Service URI? Does this need to be an IP?

The Service Registry acts as a discovery mechanism to allow broadcasters to look up the addresses of transcoders on the network. Transcoders register their Service URI at configuration time; this is submitted to the blockchain as a standalone transaction. While the configuration tool only asks for your IP:port, the URI stored on the blockchain in the form of `https://IP:port`. Transcoders are expected to provide a consistent and reliable service, so IPs here *should* remain static. However, a host (DNS) name is also allowed for the service URI to give transcoders some flexibility.

What does this error mean? "Service address https://127.0.0.1:4433 did not match discovered address https://127.1.5.10:8935; set the correct address in livepeer_cli or use -serviceAddr"

- When starting up, the transcoder checks if the current public IP matches the IP that is stored on the blockchain. If there is a mismatch, there is a possibility that your node is not publicly accessible. Override the locally inferred IP address by setting `-serviceAddr IP:port` to what is on the blockchain. Ensure your node is actually accessible at that address.

TODO: These documents could be expanded with far more information about the transactions that a Livepeer Transcoder has to submit on a regular basis to avoid being penalized and to earn their rewards and fees.

2.5 Broadcasting To A Livepeer Node

Broadcasting to Livepeer using existing broadcasting tools is easy. After a Livepeer node is running, it exposes an RTMP interface on port 1935. You can broadcast into Livepeer using this port.

2.5.1 Install Livepeer

The following instructions assume that you have followed the installation and quickstart instructions.

Some of the `livepeer` commands in these instructions require an Ethereum node JSON-RPC URL to be provided via the `-ethUrl <ETH_RPC_URL>` flag. See the quickstart instructions for more details on obtaining an Ethereum node JSON-RPC URL.

2.5.2 Deposit broadcasting funds

You will need to deposit funds (ETH) used to pay orchestrators on the network for transcoding video.

In `livepeer_cli`, select the following option:

```
1.  Invoke "deposit broadcasting funds" (ETH)
```

Upon selecting the option, you should be prompted to entire the amount of ETH to allocate for your deposit and reserve. Broadcasting funds are split into a deposit and a reserve. Deposit funds are used to pay any active orchestrator on the network. Reserve funds guarantee active orchestrators up to a fixed cap to ensure that orchestrators are paid fairly even if a broadcaster depletes its primary deposit. The distinction between the deposit and the reserve arises from the probabilistic micropayment protocol that broadcasters use to pay orchestrators - see this blog post for more details.

After answering the wizard's prompt, you should see a transaction submitted by your node. After the transaction confirms, you can see your updated deposit and reserve by refreshing the wizard.

2.5.3 Withdraw broadcasing funds

When you want to withdraw your broadcasting funds you will need to wait a fixed number of rounds before gaining access to your funds. The first step for withdrawal is to request to unlock your funds.

In `livepeer_cli`, select the following option:

```
13. Invoke "unlock broadcasting funds"
```

Upon selecting the option, you should be prompted to confirm that you would like to request to unlock your funds. After answering the wizard's prompt, you should see a transaction submitted by your node. After the transaction confirms, you will be able to withdraw your funds at the withdraw round that was specified in the prompt.

If you change your mind and do no want to withdraw your funds, you can cancel your unlock request in `livepeer_cli` by selecting the following option:

```
14. Invoke "cancel unlock of broadcasting funds"
```

Upon selecting the option, you should be prompted to confirm that you would like to cancel your request to unlock your funds. After answering the wizard's prompt, you should see a transaction submitted by your node. After the transaction confirms, your unlock request will be cancelled. Another way to cancel an unlock request is to deposit more funds.

If you want to withdraw your funds, in `livepeer_cli` select the following option:

```
15. Invoke "withdraw broadcasting funds"
```

If your unlock is not complete (i.e. your withdraw round is in the future), you will not be able to withdraw. Once your unlock is complete (i.e. your withdraw round is the current round or in the past), you should be prompted to confirm that you would like to withdraw your funds. After answering the wizard's prompt, you should see a transaction submitted by your node. After the transaction confirms, your withdrawal will be complete and can see your empty deposit and reserve by refreshing the wizard.

2.5.4 Configuring broadcasting preferences

You can configure the following broadcasting preferences using the wizard:

- The maximum price per pixel you are willing to pay. See the orchestrator section on *configuring payment parameters* for more details on how pricing per pixel works
- The set of video profiles that you want your input video to be transcoded into

In `livepeer_cli`, select the following option:

```
16. Set broadcast config
```

First, you will set the maximum transcoding price. You will be prompted for a maximum price relative to certain number of pixels. For example, you can set your maximum price to 0.1 wei per pixel if you set the maximum price to 1 wei for 10 pixels. The default number of pixels will be 1.

Then, you will pick a set of video profiles that you would like your input video to be transcoded to. The set of video profiles presented in the wizard correspond to the standard video profiles currently supported by the node. Future releases will offer greater flexibility around customizing video profiles to use for transcoding.

2.5.5 Broadcasting video

See the broadcasting guide for information on sending video into your broadcaster node and viewing the output transcoded video. In order to see more detailed logs, run your broadcaster with `-v 6` to enable verbose logging.

After receiving a stream, your broadcaster will try to connect to a set of orchestrators (ether based on the registered orchestrators on-chain or based on the orchestrators specified using the `-orchAddr` flag). Your broadcaster might reject certain orchestrators based on their required price or ticket parameters.

If you observe the `ticket faceValue higher than max faceValue` error in the broadcaster's logs, you can try:

1. Running the broadcaster with `-depositMultiplier 1`. The default value is 1000 (which is pretty high) meaning that the broadcaster will not be willing to use a ticket `faceValue` that exceeds the broadcaster's deposit divided by 1000. So, if the value is 1 then the broadcaster will be willing to use a ticket `faceValue` that equals the broadcaster's deposit. While this might not be desirable in other circumstances, it can be fine for testing purposes
2. If option 1 does not work, then the broadcaster's deposit is less than the `faceValue` required by an orchestrator - you should try depositing more ETH

If you observe the `ticket EV higher than max EV` error in the broadcaster's logs, you can try running the broadcaster with `-maxTicketEV <MAX_TICKET_EV>` where `MAX_TICKET_EV` is the maximum expected value (denominated in wei) for tickets sent by the broadcaster. The default value is 10 gwei so you could try using a higher value.

2.5.6 Transcoding verification (experimental)

The broadcaster can verify transcoded results received from orchestrators by sending verification requests to a verifier that runs alongside the broadcaster. The current default verifier implementation uses a machine learning classifier - refer to the verifier documentation for more details.

Transcoding verification is currently experimental and needs to be explicitly enabled by the user. R&D to improve its accuracy (in classifying correctly vs. incorrectly transcoded video) and performance (in terms of run time and computation cost) is underway, but in the short term, expect to see higher computational costs and transcoding latency after enabling verification for a broadcaster.

In order to enable transcoding verification, first make sure that you have Docker installed and setup the verifier.

```
git clone https://github.com/livepeer/verification-classifier
cd verification-classifier
./launch_api.sh
```

`./launch_api.sh` will start a verifier server running within a Docker container that accepts verification requests at the `/verify` endpoint. You should see log output that indicates that the verifier is listening on port 5000:

```
Sending build context to Docker daemon  5.904MB
Step 1/4 : FROM verifier:v1
 ---> fe5d1924dab5
Step 2/4 : COPY /api ./scripts
 ---> Using cache
 ---> bc21bfb203ef
Step 3/4 : ENTRYPOINT ["python"]
 ---> Using cache
 ---> 816b804bc7e7
Step 4/4 : CMD ["scripts/api.py"]
 ---> Using cache
 ---> 548f58689798
Successfully built 548f58689798
Successfully tagged verifier-api:v1
Verifier server listening in port 5000
```

If you are running the verifier on the same machine as the broadcaster, then you can run the below command in order to connect the broadcaster with the verifier:

```
livepeer -network rinkeby -ethUrl <ETH_RPC_URL> -broadcaster -verifierUrl http://
↪localhost:5000/verify -verifierPath ~/verification-classifier/stream
```

The broadcaster logs should indicate the address of the verifier being used:

```
Using the Epic Labs classifier for verification at http://localhost:5000/verify
```

The value of `-verifierUrl` should be the address (`http://<IP>:<port>/verify`) that the verifier is receiving requests at. In this case, since the verifier and broadcaster are on the same machine all requests can be received on `localhost`.

The value of `-verifierPath` should be the path to the volume that the verifier will read segment data from. By default, the verifier will read segment data from a volume mounted on the `/stream` directory within the `verification-classifier` project. So, if you started the verifier from within `~/verification-classifier` the value for `-verifierPath` should be `~/verification-classifier/stream`.

If you are running the verifier on a separate machine from the broadcaster, you will need to make sure that the verifier endpoint is accessible by the broadcaster and that the verifier's volume is accessible over the network.

One way to make the verifier endpoint accessible to the broadcaster without making it publicly accessible is to create an SSH tunnel from the broadcaster machine to the verifier machine.

Create an SSH tunnel with the following command run from the broadcaster machine:

```
ssh -N -L 5000:127.0.0.1:5000 -i <SSH_KEY_FILE> <USER>@<HOSTNAME>
```

Note that the command will not output anything if it is successfuly run.

One way to make the verifier's volume accessible over the network is to mount the verifier's volume on the broadcaster machine over an SSH connection using sshfs.

Mount the verifier's volume on the broadcaster machine by running the following command on the broadcaster machine:

```
sshfs -o IdentityFile=<SSH_KEY_FILE> <USER>@<HOSTNAME>:<FOLDER_ON_VERIFIER> <FOLDER_
↪ON_BROADCASTER>
```

After the SSH tunnel is setup and the remote volume is mounted, run the following command to start the broadcaster:

```
livepeer -network rinkeby -ethUrl <ETH_RPC_URL> -broadcaster -verifierUrl http://
↪localhost:5000/verify -verifierPath <FOLDER_ON_BROADCASTER>
```

The broadcaster also supports sharing segment data with a verifier using external cloud storage providers such as Amazon's S3 or Google's GCS. Documentation on using external cloud storage providers will be added in the future.

2.5.7 Broadcasting To A Local Node Using OBS

Start by reading our step by step guide

It is far more convenient to broadcast using existing tools that have features for screen capture, composites, overlays, multiple video and audio sources, etc. One such tool is OBS. To use OBS you have to change two settings:

- Settings -> Stream -> URL. Set it as `rtmp://localhost:1935`
- Settings -> Output -> Output Mode. Set it to Advanced. Ensure the following settings are enabled:
  - Encoder: x264
  - Rate Control: CBR
  - Keyframe Interval: 4
- Start streaming as usual.

If the broadcast is successful, you should see a log in the terminal like:

```

Video Created With ManifestID: 710ed610
```

If you have used `-currentManifest` to start your Livepeer node, you can verify that the broadcast is successful by running `curl http://localhost:8935/stream/current.m3u8`. It should return a valid HLS playlist like:

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-STREAM-INF:PROGRAM-ID=0,BANDWIDTH=4000000,RESOLUTION=1120x700
710ed610/source.m3u8
```

2.5.8 Playing The Local Video Stream

Make sure you have used `-currentManifest` to start your Livepeer node. You can watch your stream via:

- Playing the stream through the Livepeer media player. Use `http://localhost:8935/stream/current.m3u8` as the URL for the video stream.
- Playing the stream using `ffplay`

```
$ ffplay http://localhost:8935/stream/current.m3u8
```

2.5.9 Broadcasting To A Local Node Using FFMPEG

To broadcast using ffmpeg you can try the following command:

For Mac:

```
ffmpeg -f avfoundation -framerate 30 -pixel_format uyvy422 -i "0:0" -vcodec libx264 -
↪tune zerolatency -b 1000k -x264-params keyint=60:min-keyint=60 -acodec aac -ac 1 -
↪b:a 96k -f flv rtmp://localhost:1935/movie
```

For Linux:

```
ffmpeg -f dshow -framerate 30 -pixel_format uyvy422 -i "0:0" -vcodec libx264 -tune␣
↪zerolatency -b 1000k -x264-params keyint=60:min-keyint=60 -acodec aac -ac 1 -b:a␣
↪96k -f flv rtmp://localhost:1935/movie
```

You can now verify if the broadcast is successful.

2.5.10 Broadcasting To A Remote Node From Mobile

There is not currently a natively Livepeer aware mobile app, but much like *using OBS*, as described above, you can use any existing mobile broadcasting tool such as ManyCam on iOS or RTMPCamera on Android to broadcast into Livepeer.

Instead of setting the rtmp output to `localhost:1935`, you'll want to set it to a remote Livepeer node that you are running on a server. Replace `localhost` with the IP address of the server.

Make sure the node is started with `-currentManifest` to make this process easier.

2.5.11 Reaching Many Viewers at Scale

If you would like to take your output video and make it available via a conventional CDN, then you have the option to do so.

- Run a Livepeer node on a server, and expose ports `8935 and 1935`.
- Boot up the livepeer node with the `-rtmpAddr 0.0.0.0` and `-httpAddr 0.0.0.0` flags
- Configure your CDN to cache video content running at `http://hostname:8935/stream/{streamID}.m3u8`

Now any requests that come into your site or DApp for video streaming through Livepeer will pull the video from the network, but will be served off of a CDN. In the future, we would like to replace this option with the p2p network that Livepeer forms around a stream.

2.5.12 FAQ

Check out our Broadcasting Forum for frequently asked questions

If you have any questions, reach out to Chris Hobcroft on our community chat

2.6 Governance

Orchestrators and delegators can participate in protocol governance by voting in polls.

Refer to the following resources for more information on protocol governance:

- The Livepeer Governance Roadmap Proposal
- The Founder's Statement
- Livepeer Governance in a Nutshell
- Livepeer Governance Proposal Voting
- The Livepeer Improvement Proposal Process Repository

2.6.1 Voting with the explorer

Delegators with staked LPT can vote in a poll using the explorer and a web3 enabled wallet (i.e. Metamask).

- To vote on mainnet use https://explorer.livepeer.org/voting
- To vote on Rinkeby use https://rinkeby.explorer.livepeer.org/voting

2.6.2 Voting with livepeer_cli

Orchestrator operators can use `livepeer_cli` to vote in a poll without exporting their keys from the machine that their orchestrator node is running on.

In `livepeer_cli`, select the following option:

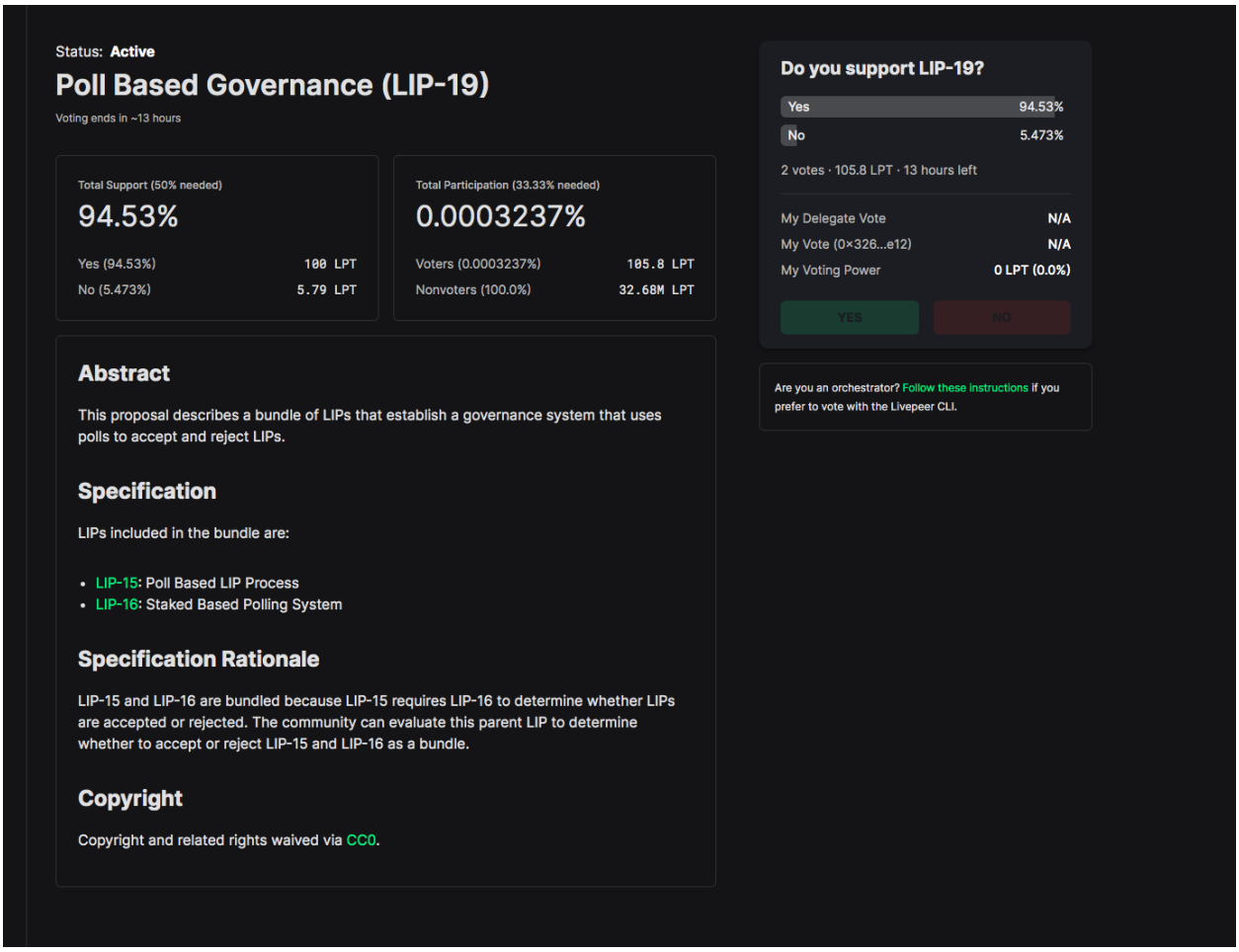
```
17. Vote on a poll
```

Note: On Rinkeby, the option will be number 19.

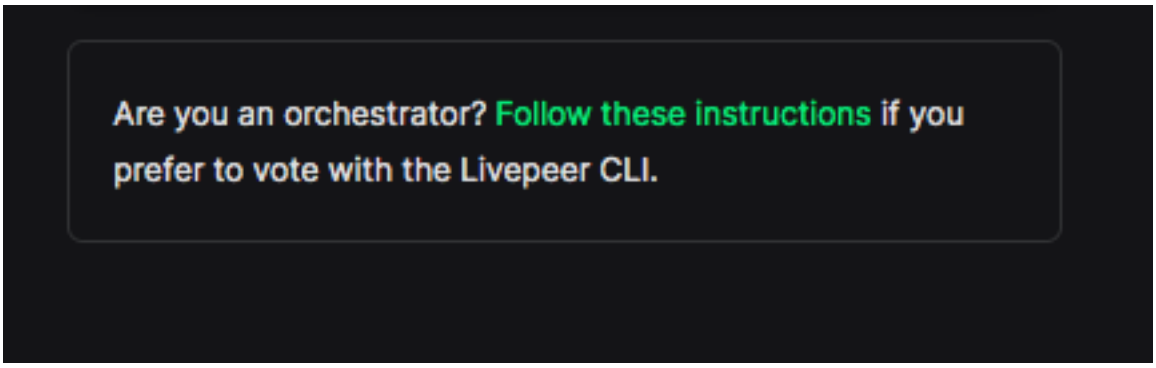
Upon selecting the option, you will be prompted to enter the contract address for the poll you want to vote on:

```
> 17
Enter the contract address for the poll you want to vote on - >
```

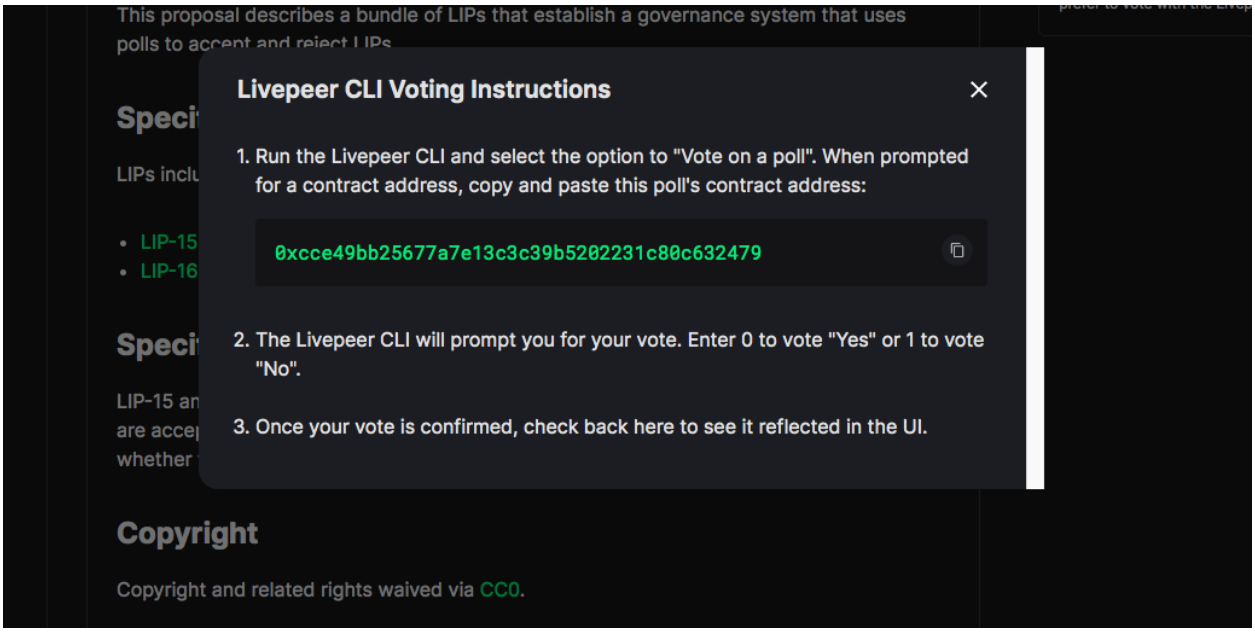
The contract address for the poll can be found in the explorer page for the poll.



The poll page will include the following box:



Click the "Follow the instructions" link to display the instructions for voting with `livepeer_cli`:



The contract address for the poll in these instructions should be entered in `livepeer_cli`.

```
Enter the contract address for the poll you want to vote on - >␣
↪0xcce49bb25677a7e13c3c39b5202231c80c632479
Identifier  Voting Options
0       Yes
1       No
```

Upon entering the poll contract address, you will be prompted to choose and confirm the the option you want to vote for:

```
Enter the ID of the option you want to vote for - > 0
Are you sure you want to vote "Yes" ? (y/n) - > y

 success
```

After choosing and confirming the option you want to vote for you will observe your node submitting the vote transaction:

```
I0422 03:30:44.191809   43457 backend.go:96]
*****************************Eth Transaction*****************************

Invoking transaction: "vote". Inputs: "_choiceID: 0"  Hash:
↪"0xf6957c190f1f16fc2ca4a93846903eb435c5e08fa7f6f40b6e159aab6d74905f".

*************************************************************************
```

Once the vote transaction is confirmed you will be able to see your vote reflected in the explorer poll page.

2.7 Developing on Livepeer

2.7.1 Building Video Dapps

- Video-based Dapps (for example, livepeer.tv)

- Infrastructure tools and services for broadcasters or live streamers (for example, SAAS services on top of Livepeer)
- Livepeer Player - A react component for playing live video - https://github.com/livepeer/livepeerjs/tree/master/packages/chroma

2.7.2 Building Livepeer Protocol Dapps

- Dapps for the Livepeer ecosystem. (for example, livepeer protocol explorer or Supermax)

2.7.3 Building Tools for Livepeer

- SDKs for Livepeer (for example, livepeerjs-sdk or livepeerjs-graphql)
- Client implementation for Livepeer (for example, go-livepeer)

2.7.4 Open Projects

Livepeer also posts open problems for discussion, ideas, and collaboration on Github. Check out:

- Open Project Proposals
- Open Research Areas

2.7.5 Contributing to Livepeer

For developers who are looking for interesting to problems to work on related to decentralized tech, blockchain, cryptocurrency, video engineering, and peer-to-peer networking, Livepeer may provide some interesting challenges. The three technical areas that Livepeer focuses on today are:

- Protocol implementation (Smart Contract)
- Livepeer Node (Distributed Systems / Networking)
- Livepeer Media Server (Video Engineering)

For the protocol , you can follow the protocol repo. It requires some background in Solidity and the Livepeer Whitepaper.

For the livepeer node, check out the go-livepeer repo. It requires some understanding of Golang and Geth. Setting up a development enviroment can be done by following **'these instructions'_**.

For the livepeer media server implementation, take a look at the LPMS repo. It requires some video engineering knowledge. The demuxed conf videos and the Apple Live streaming doc are good resources to start learning.

If you're interested in the any of the above challenges, or are building video features into an application, jump into our development chat room on Discord and join the conversation.

2.8 Need Help ?

The Livepeer team and community are available to help with any additional questions. You can find them on:

- Discord chat room
- Forum

- Twitter
- Reddit
- Github

Useful links:

- Livepeer Explorer
- Streamflow paper
- Broadcaster FAQ
- Transcoder campaign thread

2.9 License

2.9.1 Contact

Questions? Email contact@livepeer.org