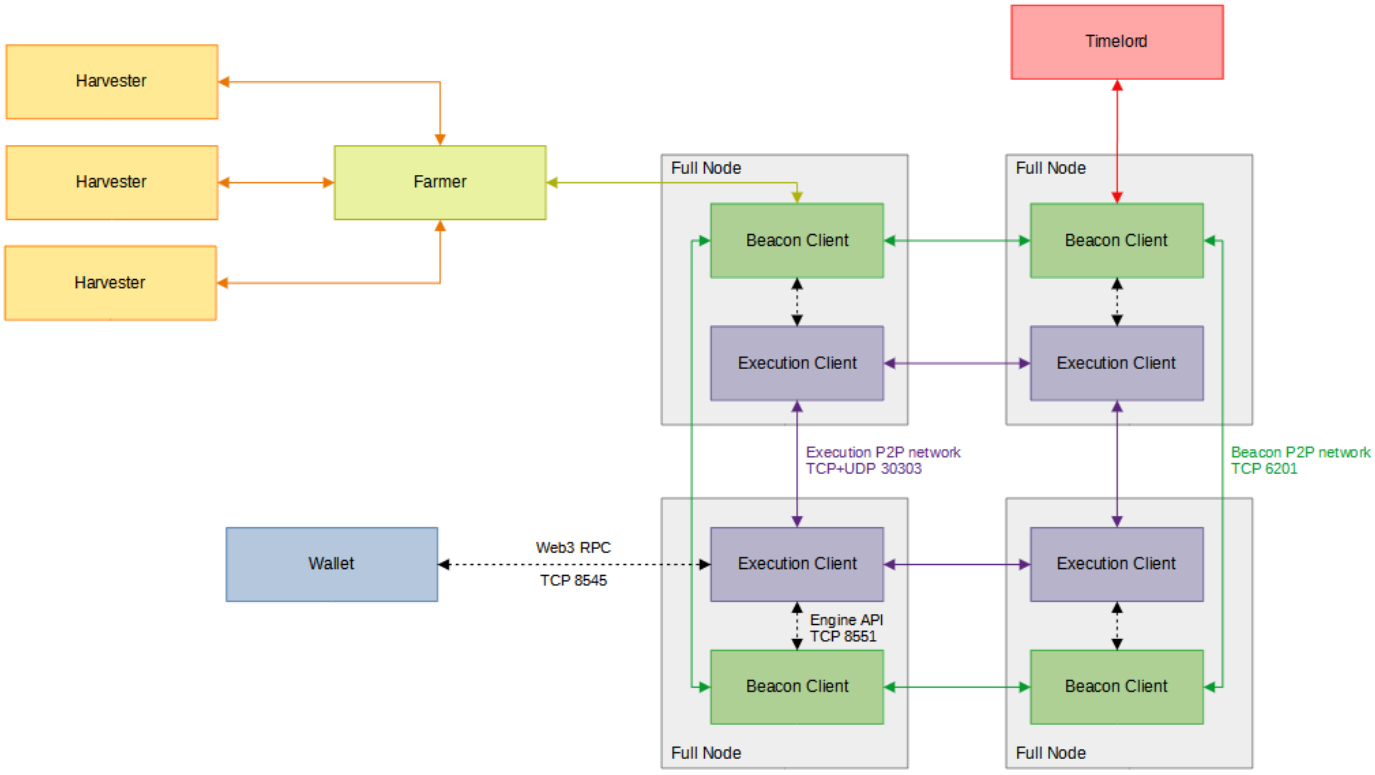
Architecture

Architecture Overview

The above diagram shows BPX network architecture. A single machine can run more than one of these processes. Typical BPX network full node consist of two processes: Beacon Client and Execution Client. Additionally, Farmer and Harvester are activated to farm new blocks.



Beacon Client

The **Beacon Client** serves as a full node for the BPX Beacon Chain with several key responsibilities:

1. Maintaining a complete copy of the Beacon Chain.
2. Validating Beacon Chain blocks.
3. Propagating new beacon blocks and proofs across the peer-to-peer network.
4. Providing consensus updates to the Execution Client.

Beacon Clients are continually connected to a random subset of other Beacon Clients within the network. They distribute new blocks to peers, enabling all nodes to maintain a full copy of the blockchain.

Beacon Clients communicate with one another over **TCP port 6201**.

Execution Client

The **Execution Client** functions as a full node for the **BPX Execution Chain**. It listens for new transactions broadcast on the network, maintains the mempool, executes transactions within the EVM, creates and validates execution blocks, and holds the latest state and database of the Execution Chain.

The Execution Chain lacks an internal consensus algorithm. Instead, it exposes the **Engine API** on **TCP port 8551**, allowing the Beacon Client to connect and provide consensus updates, including new block candidates, chain head updates, and requests to generate new blocks.

Execution Clients communicate with each other over a listener **TCP port** and a discovery **UDP port**, both defaulting to **30303**. Each Execution Client also provides access to the **Web3 RPC API** on **TCP port 8545**, which supports interaction with wallets, developer tools, and dApps.

Farmer

BPX farmers are analogous to Bitcoin's miners. They earn block rewards and fees by finding valid proofs of space inside their stored plots. The farmer processes don't maintain a copy of the Beacon Chain, but they trust a beacon client to provide updates. The beacon client and farmer processes communicate with each other using the farmer protocol.

Farmers communicate with harvesters (individual machines, including the farmer machine, that actually store the plots) through the harvester protocol.

Farmers operate by waiting for updates from a beacon client, which gives them new signage points (equivalent to a lottery's winning numbers) approximately every nine seconds. Farmers then send the signage point to each harvester, to check whether any winning proofs of space exist. If the harvester finds any valid proofs, it sends them to the farmer. Farmer has a private key, which is used for signing blocks. After receiving signed block from the farmer, beacon client creates a new beacon chain block and propagates it accross the network.

Harvester

Harvesters are individual machines controlled by a farmer. In a large farming operation, a farmer may be connected to many harvesters.

Harvesters control the actual plot files by retrieving qualities or proofs from disk. The minimum plot size (and by far the most common) is k32, which corresponds to around 100 GiB. With each increment of a k-value, the plot size roughly doubles, so a k33 plot is around 200 GiB, k34 is around 400 GiB, etc.

The difficulty level automatically adjusts every 4608 beacon chain blocks to target one proof of space - across the entire network - for every two signage points. This is the targeted average value - there can also be zero or multiple proofs per signage point. This leads to a difficulty adjustment approximately every 12 hours.

Given a plot, the harvester must perform two tasks to find a valid proof:

1. Fetch the initial quality - this requires around seven random disk seeks, or 70 milliseconds on a slow HDD.
2. (Only performed if the initial quality is sufficiently high) Fetch the full proof - this requires around 64 disk seeks, or 640 milliseconds on a slow HDD.

For most challenges, the quality (step 1) will be very low, so fetching the entire proof (step 2) will not be necessary. A node has 28 seconds to return a proof, so disk I/O will not be a limiting factor, even when proofs are stored on slow HDDs.

> Tape drives are too slow for farming. The protocol was designed to support hard disks, but nothing slower. It is possible to use tape for long-term plot storage, only transferring the plots to disks for occasional farming, but this is likely a very rare use case.

Finally, harvesters also maintain a private key for each plot. The blocks are signed with these keys.

Timelord

Timelords support the network by creating sequential proofs of time (using a Verifiable Delay Function) and broadcasting them approximately every nine seconds. This provides "deterministic randomness", which is used to decide the winning proofs of space.

Since this computation is sequential, very little energy is consumed, as opposed to proof-of-work systems, where computation is parallelizable. For example, if 100 timelords are doing the same computation on a proof of time, they will all create the exact same output.

A timelord is required to connect to exactly one beacon client, typically on the same machine. This connection is verified with a certificate. This 1:1 architecture has a large security benefit: it keeps the timelord sandboxed in its own private network. That way, the beacon client protocol is the only protocol that requires total security. If more than one beacon client could connect to the same timelord, it would add a potential attack vector to the network.

Timelords do not directly earn rewards. Furthermore, only the fastest timelord on the network will broadcast proofs at any given time. Therefore, only one timelord is required to keep the network running, and most farmers will not feel compelled to run one. However, farmers with multi-PiB farms may want to run a timelord, both for redundancy and for protection against temporary local latency issues.

If someone controls the fastest timelord in the world, it doesn't give them much of an advantage at winning rewards. However, they could potentially orphan or censor other farmers, depending on how much faster their timelord is.

Furthermore, an attacker with a significantly faster timelord than anyone else could potentially run a long-range attack against the network with less than 42.7% of the total netspace. For security purposes, it is very important to maintain open designs of VDF hardware.

Types of Timelords

There are two primary types of Timelords: Regular and Blueboxes.

The first is the core Timelord that takes in Proofs of Space and uses a single fastest core to perform repeated squaring in a class group of unknown order as fast as possible. Beside each running VDF (referred to as a vdf_client in the application and source) is a set of proof generation threads that accumulate the proof that the time calculation's number of iterations was done correctly.

The second are Bluebox Timelords. Blueboxes are most any machine - especially things like old servers or gaming machines - that scour the historical chain looking for uncompressed proofs of time. So that the chain moves quickly, the regular Timelords use a faster method of generating proofs of time but the proofs are larger, which takes your Raspberry Pi a lot more time and effort to validate and sync the blockchain. A Bluebox picks up an uncompressed Proof of Time and recreates it, but this time with the slower and more compact proofs generated at the end. Those are then gossiped around to everyone so they can replace the large and slow to verify Proofs of Time with the compact and much quicker to validate version of exactly the same thing.

Running a Timelord

The network only requires one running Timelord to keep moving (liveness.) The way Timelords race is like they are on a series of 100 meter dashes. Each one takes off with the last good Proof of Space and tries to get to the total number of iterations required to complete a given Proof of Space. Better Proofs of Space require less iterations to prove. When the fastest Timelord announces the Proof of Time for this Proof of Space all of the other Timelords stop racing and are magically teleported to the starting line of the next 100 meter dash to start it all over again.

It's good to have a few Timelords out there. There can be things like routing flaps or the overzealous backhoe that takes large swaths of the internet offline. If the fastest Timelord was just about to win the current dash when its internet blinked off in a fury of construction misadventure, then the second fastest will win that dash and the next dashes - until the fastest returns. One of the key qualities about Proofs of Time is that given the same Proof of Space, their output and proof are always the same (though the proofs can be larger or smaller and harder or easier to validate - they all end up with the same outcome.)

BPX developers plans to run a few Timelords around the world - and some backups too - to ensure that all Farmers and nodes can hear the beat that the Timelords are calling.

Installing a Timelord

If you want to run a Timelord on Linux/MacOS, first follow the Install from Source instructions here. Then run:

```
. ./activate
sh install-timelord.sh
bpx start timelord
```

Timelords execute sequential verifiable delay functions (proofs of time or VDFs), that get added to blocks to make them valid. This requires fast CPUs and a few cores per VDF.

> Due to restrictions on how MSVC handles 128 bit numbers and how Python relies upon MSVC, it is not possible to build and run Timelords of all types on Windows.

Regular Timelords

On MacOS x86_64 and all Linux distributions, building a Timelord is as easy as running `bpx start timelord` in the virtual environment. You can also run `./vdf_bench square_asm 400000` once you've built Timelord to give you a sense of your optimal and unloaded ips. Each run of `vdf_bench` can be surprisingly variable and, in production, the actual ips you will obtain will usually be about 20% lower due to load of creating proofs. The default configuration for Timelords is good enough to just let you start it up. Set your log level to INFO and then grep for "Estimated IPS:" to get a sense of what actual ips your Timelord is achieving.

Bluebox Timelords

Once you build the Timelord with `sh install-timelord.sh` in the virtual environment, you will need to make two changes to `~/.bpxchain/beacon/config/config.yaml`. In the `timelord`: section, set `bluebox_mode` to `True`. Then you need to proceed to the `beacon` section and set `send_uncompact_interval` to something greater than 0. We recommend 300 seconds there so that your Bluebox has some time to prove through a lot of the un-compacted Proofs of Time before the node drops more into its lap. The default settings may otherwise work but if the total effort is a little too much for whatever machine you are on you can also lower the process_count: from 3 to 2, or even 1, in the `timelord_launcher` section. You know it is working if you see `VDF Client: Sent proof` in your logs at INFO level.

Timelords and Attacks

One of the things that is great about BPX consensus is that it makes it almost impossible for a Farmer with a maliciously faster Timelord to selfishly Farm. Due to the way the consensus works, a Farmer with a faster Timelord is basically compelled to prove time for all the farmers winning blocks around him also. Maliciously running a faster Timelord can give a benefit when attempting to 51% attack the network, so it is still important that over time we push the Timelord speeds as close to the maximum speeds of the silicon processes available. We expect to have the time and the resources to do that right and make open-source hardware versions widely available.

Wallet

The wallet is responsible for managing private keys, as well as generating, storing and sending transactions. Wallets communicate with the execution clients through the Web3 RPC API. Each execution client exposes the API on TCP port 8445.

You can also connect your wallet to a public RPC node provided by us, using the configuration parameters below:

> **Network name:** BPX Chain
> **New RPC URL:** https://rpc.bpxchain.cc
> **Chain ID:** 279
> **Currency symbol:** BPX
> **Block explorer URL:** https://explorer.bpxchain.cc

BPX Chain is compatible with all popular wallets dedicated for EVM chains, e.g. **Metamask** or **Trust Wallet**.