By:
Parithosh Jayanthi
DevOps engineer
Ethereum Foundation

Testing and Prototyping Ethereum upgrades

Intro

What was complicated about it?

- >20 client combinations need to be tested & Regressions can sneak in very easily
- Communicating and debugging various client combinations
- Figuring out how to test this in a reliable manner! - We just had single client focussed tests till now
- All future upgrades will inherit some of the complexity - build once, use many
- Competences for ELs and CLs are quite separate

What are devnets?

- A testing mirror of the Ethereum base layer
- Contains EL/CL/Validators, setup in a configuration that we want to test
- Allows devs to deploy forks and changes without affecting mainnet
- Devnets are public, allowing the community to test with us
- A fully in control validator set allows us to design edge case scenarios

What does testing look like today?

- Current upcoming fork (Pectra)
- Upcoming future fork (Verkle)
- Features that are proposed for future forks:
  - ILs
  - EIP-7441 (Whisk)
- Client optimisations:
  - EthereumJS Snap sync testnet
  - Bigboi beaconchain tests for blob/validator limits

Local testing

Hard to test changes quickly!
Fix:

- Earlier we needed full fledged testnets to test changes
- Moving to local devnets enables faster iterations: **enter Kurtosis!**
- Able to work async on features, knowing it'll work with full interop
- Configurable locally: 3s slot times, quick forks, mev workflow
- Scalable: To whatever extent kubernetes/docker allows
- Docs: https://github.com/kurtosis-tech/ethereum-package

"kurtosis run --enclave <name> github.com/kurtosis-tech/ethereum-package --args-file <filename>"

Local testing

Configured with YAML:

```
participants:
  - el_type: geth
        cl_type: teku
  - el_type: nethermind
        cl_type: prysm
additional_services:
  - tx_spammer
  - blob_spammer
  - dora
  - prometheus_grafana
snooper_enabled: true
keymanager_enabled: true
```

MEV Local testing

```
mev_type: full
additional_services:
  - tx_spammer
  - blob_spammer
  - custom_flood
  - el_forkmon
  - dora
  - prometheus_grafana
mev_params:
  launch_custom_flood: true
  mev_relay_image: flashbots/mev-boost-relay:latest
network_params:
  seconds_per_slot: 3
```

How do I prototype?

- Kurtosis works on the concept of “allow everything to be overridden”
- Outside of some network basics, you can change anything in a kurtosis network
- To test protocol changes, we can override the client images
- To test new tools, run an existing network and connect your tool to it
- To test quick forks, we can override the network params

Prototype testing

```
participants:
  - el_type: geth
    el_image: ethpandaops/geth:gballet-transition-post-genesis
    cl_type: lodestar
    cl_image: ethpandaops/lodestar:g11tech-verge
    count: 3
network_params:
  electra_fork_epoch: 1
  genesis_delay: 100
snooper_enabled: true
persistent: true
launch_additional_services: true
additional_services:
  - assertoor
  - dora
```

What comes after local testing?

- Remote/public devnets
- We needed a way to set them up easily
- Devnets used to be error prone and time consuming
- Easy drift between setup configs of various testnets due to customizations

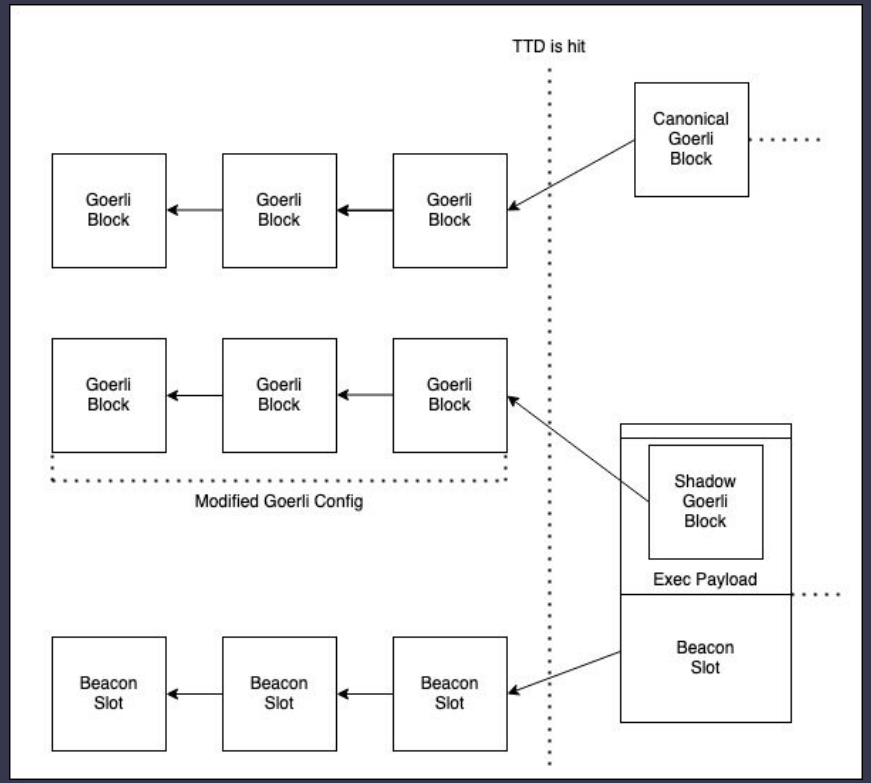
Fix:

- Move barebones logic upstream into role: https://github.com/ethpandaops/ansible-collection-general
- Move generic components into its own tool, e.g: Genesis: https://github.com/ethpandaops/ethereum-genesis-generator/
- Make tooling independent of repo/testnet (With GitOps): https://github.com/ethpandaops/ethereum-helm-charts/
- Generalize setups for all testnets: https://github.com/ethpandaops/template-testnet/

Shadowforks

- Allows us to check compatibility across all clients through the entire lifecycle
- Fresh testnets allow us to check assumptions across client pairs without much overhead
- Shadow forks allow us to stress test the clients with real state and transaction load
- We can invite participants in a controller manner to take part in the tests
- Acts as release test which triggers real world edge cases, before we recommend the releases to the general public

- Simple in principle:
  - Take a network genesis file
  - Modify file to add a fork timestamp
  - Setup a new beaconchain with validators and same fork timestamp
  - Connect new beaconchain ONLY to ELs with modified config
  - At fork timestamp, the modified ELs and all CLs will shadowfork into a new chain
  - New chain continues to build on top of the canonical chain
- Sprinkle some peering and mempool complexities on top to get a shadowfork



Handy tools to know about

Overview

- Snoop around this website: https://ethpandaops.io/
- All codebases can be found here: https://github.com/ethpandaops/
- Spend some time reading through: https://notes.ethereum.org/@parithosh/testing-overview-doc
- If you're interested in testing, find a project and start contributing or keep an eye out on `Eth R&D > interop` channel for ideas

Kurtosis (covered earlier)

Template-devnets

- Repo: https://github.com/ethpandaops/template-testnet/
- Contains everything you need to configure for any type of testnet
- It uses Terraform to spin up cloud instances and Ansible to deploy the network
- Ansible configs are reliant on these roles: https://github.com/ethpandaops/ansible-collection-general
- Example usage: https://github.com/ethpandaops/verkle-devnets
- Useful if you want to run nodes on a larger scale and local testing tools are inadequate

Assertoor

- Repo: https://github.com/ethpandaops/assertoor
- Tool to assert network level expectations
- E.g: can a network handle deposits, can it handle every opcode being called, can it handle a reorg
- As its a general testing tool, you can use it for any assertion on any network
- E.g: On verkle networks: was the transition a success
- Similar to hive, Hive -> single node, Assertoor -> Network wide
- Can be run locally, via kurtosis or integrated into a CI: https://github.com/ethpandaops/assertoor-github-action

```
participants:
    - el_type: geth
    el_image: ethpandaops/geth:gballet-transition-post-genesis
    cl_type: lodestar
    cl_image: ethpandaops/lodestar:g11tech-verge
    count: 3
network_params:
  electra_fork_epoch: 1
  network: holesky-shadowfork-verkle
  network_sync_base_url: http://10.10.101.7:9000/snapshots/
persistent: true
global_node_selectors: {"kubernetes.io/hostname": "lenovo-berlin-02"}
assertoor_params:
  image: "ethpandaops/assertoor:verkle-support"
  run_stability_check: false
  run_block_proposal_check: true
  tests:
    -
https://raw.githubusercontent.com/ethpandaops/assertoor-test/master/assertoor
-tests/verkle-conversion-check.yaml
```

Forky

- Repo: https://github.com/ethpandaops/forky
- Ethereum forkchoice visualizer
- Can display the forkchoice of a live node or you can upload your own
- Helps debug forkchoice related issues
- Can be run standalone or via kurtosis
- Forkchoice of mainnet: https://forky.mainnet.ethpandaops.io/

Tracoor

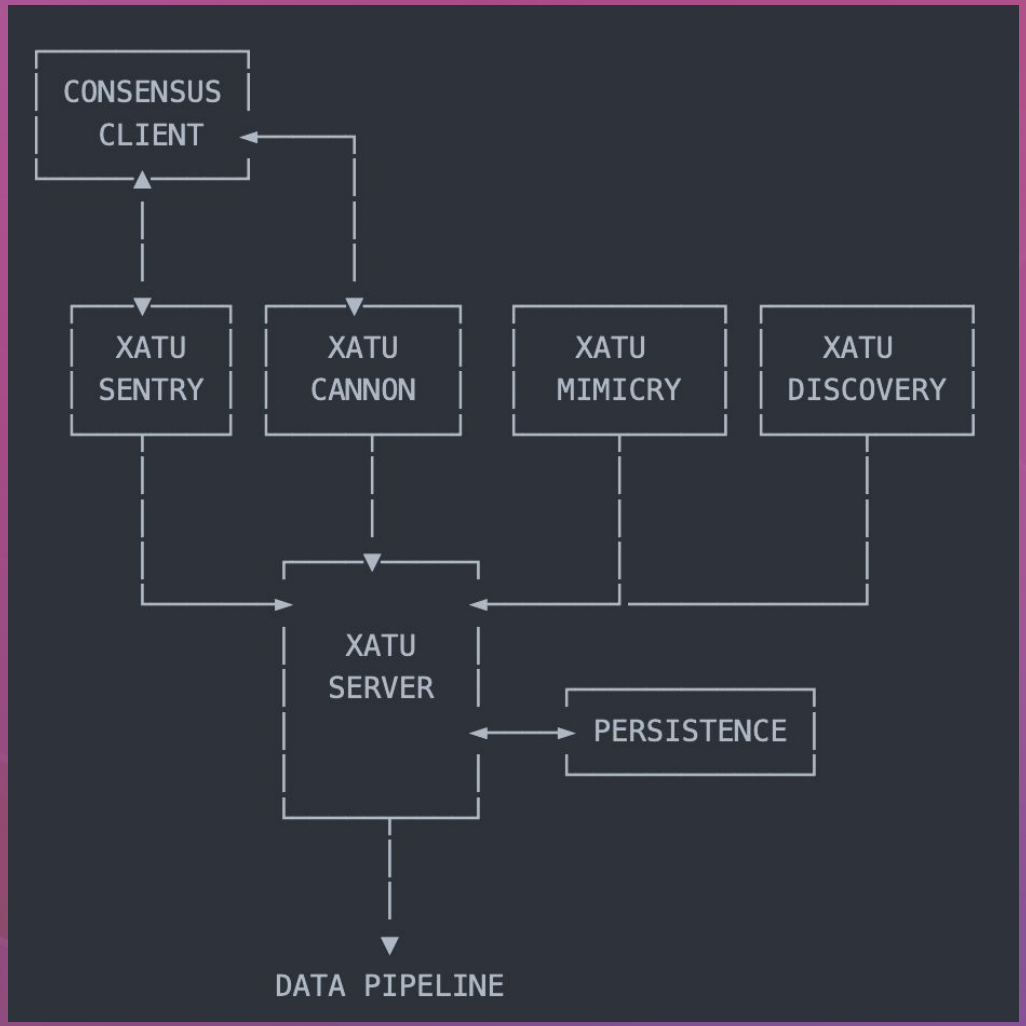
- Repo: https://github.com/ethpandaops/tracoor
- Ethereum Trace explorer
- Can display a collection of traces and states of EL and CL blocks/slots
- Helps debug network related issues
- Traces of mainnet: https://tracoor.mainnet.ethpandaops.io/

Dora

- Repo: https://github.com/ethpandaops/dora
- Lightweight slot explorer for the Ethereum beaconchain
- Extremely extendable, low level access to DB also possible for analysis usage
- Example extendability: https://dora.verkle-gen-devnet-6.ethpandaops.io/
- Example standard use: https://dora.holesky.ethpandaops.io/

Xatu

- Repo:
    - https://github.com/ethpandaops/xatu
    - https://github.com/ethpandaops/analytics-pipeline
- Ethereum p2p layer is hard to get visibility about
- Xatu data is then fed into an analysis pipeline to get data we care about
- The visualization is handled by Grafana, but the DB can directly be queried as well
- Data is all open sourced:
    - https://ethpandaops.io/posts/open-source-xatu-data/
    - https://esp.ethereum.foundation/data-challenge-4844



Xatu Server: central server collecting events from various sentries

Xatu Sentry: client that runs alongside a CL node

Xatu Discovery: client that uses discv4 and discv5 to crawl the network for metadata

Xatu Mimicry: client that collects transaction data from EL p2p layer

Xatu Canon: client that backfills canonical beaconchain data

https://notes.ethereum.org/@ethpandaops/xatu-overview

Attestation propagation

Blob sidecar propagation vs number of blobs for slot - all clients

Blob sidecar propagation vs number of blobs for slot - all clients

Blob sidecars	max	p95	average	p50	p05	min
1	7.39 s	2.76 s	767 ms	512 ms	230 ms	118 ms
2	6.60 s	3.00 s	830 ms	553 ms	243 ms	124 ms
3	8.00 s	3.00 s	850 ms	563 ms	259 ms	127 ms
4	8.00 s	2.53 s	859 ms	618 ms	278 ms	130 ms
5	5.00 s	3.48 s	1.38 s	993 ms	409 ms	319 ms
6	7.26 s	2.40 s	852 ms	628 ms	308 ms	162 ms

What next?
Run your own local devnet!
Get involved in writing tests, best way to learn how Ethereum works imo

Thank you!
#TestingThePurge

https://github.com/ethpandaops/

@parithosh_j