Quest for the Best Tests

A retrospective on #TestingTheMerge

Parithosh Jayanthi

DevOps Engineer, Ethereum Foundation

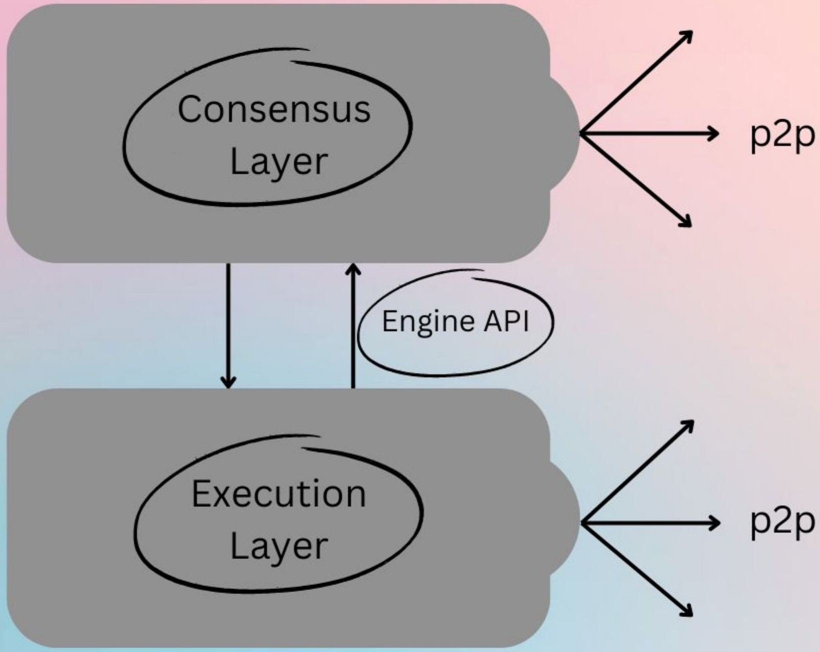
Section 1

#TestingTheMerge Assemble

Why is the Merge complicated?

- >20 client combinations need to be tested & regressions can sneak in very easily
- Specification is under active development -> Harder to track subtle differences
- Communicating and debugging various client combinations
- Figuring out how to test this in a reliable manner!
- All future upgrades will inherit some of the complexity - build once, use many
- Debug knowledge needed for ELs and CLs are quite different

Forked network
Slashing
Unhealthy network



Consensus Layer

p2p

Engine API

Execution Layer

p2p

Reorgs
Reduced throughput
State corruption

What tests can we have?

- Unit tests:
    - Handled by client teams internally
    - Usually runs on ever PR
    - Reduces chance of regressions
- Integration tests:
    - Handled partially by teams
    - Involves local testnets or interop tests
    - Ensures interop at a high level

What tests can we have?

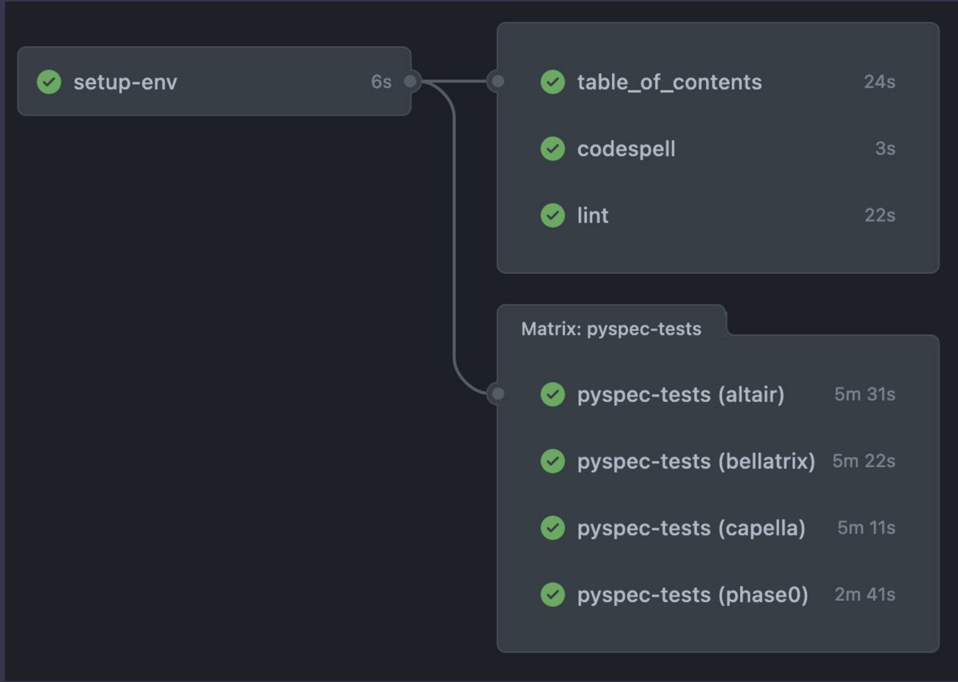
- System tests:
  - Tests end-to-end functionality
  - Involves external parties and the community
- Production tests:
  - Tests performance on a prod-like environment
  - Public testnets involving everyone
  - Finds issues that happen only at real-world loads

Section 2

#TestingTheMerge: The Infinity War

Spec tests

- The CI runs on every commit to the specs repo, ensuring that the specs pass tests
- Client teams import the specs and test it in their local CIs as well
- Acts as a sanity check to make sure client aren't implementing a spec that won't pass tests



✓ setup-env 6s

✓ table_of_contents 24s

✓ codespell 3s

✓ lint 22s

Matrix: pyspec-tests

✓ pyspec-tests (altair) 5m 31s

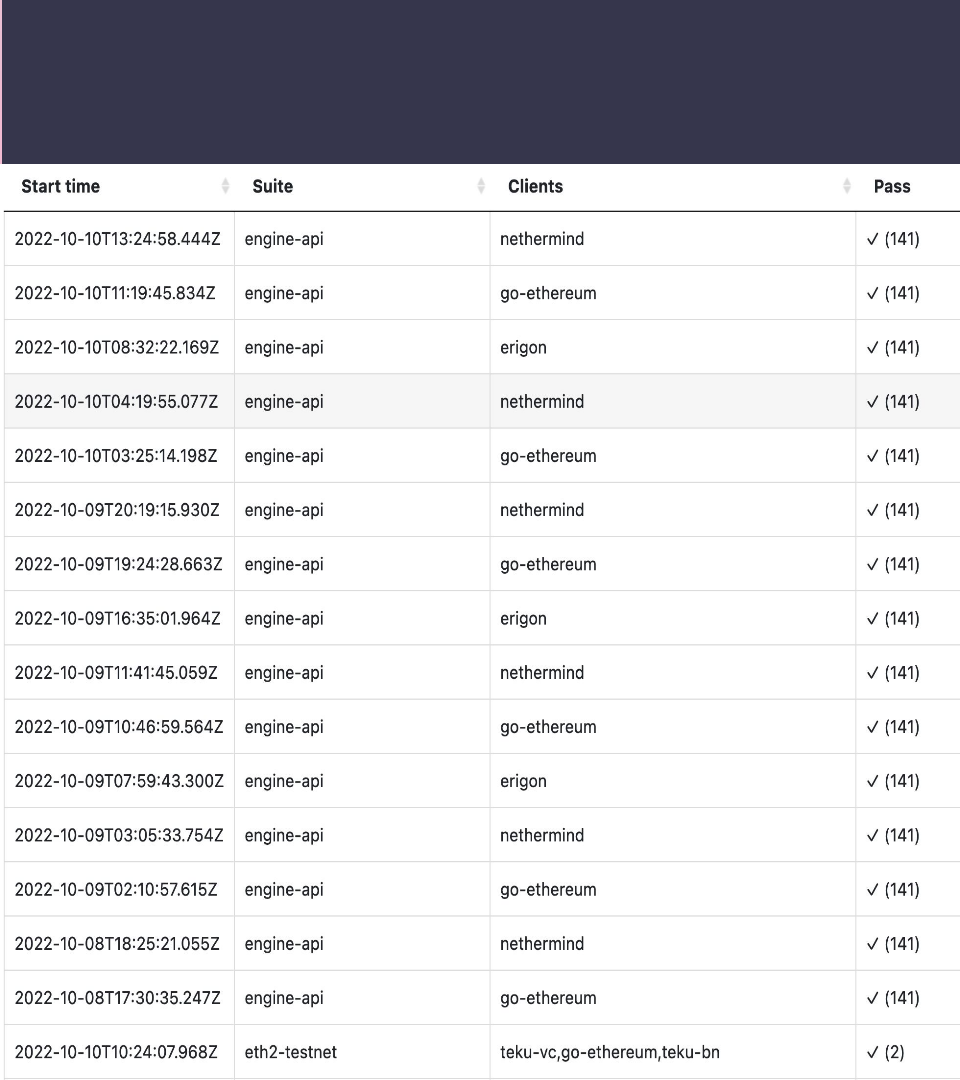
✓ pyspec-tests (bellatrix) 5m 22s

✓ pyspec-tests (capella) 5m 11s

✓ pyspec-tests (phase0) 2m 41s

Hive tests

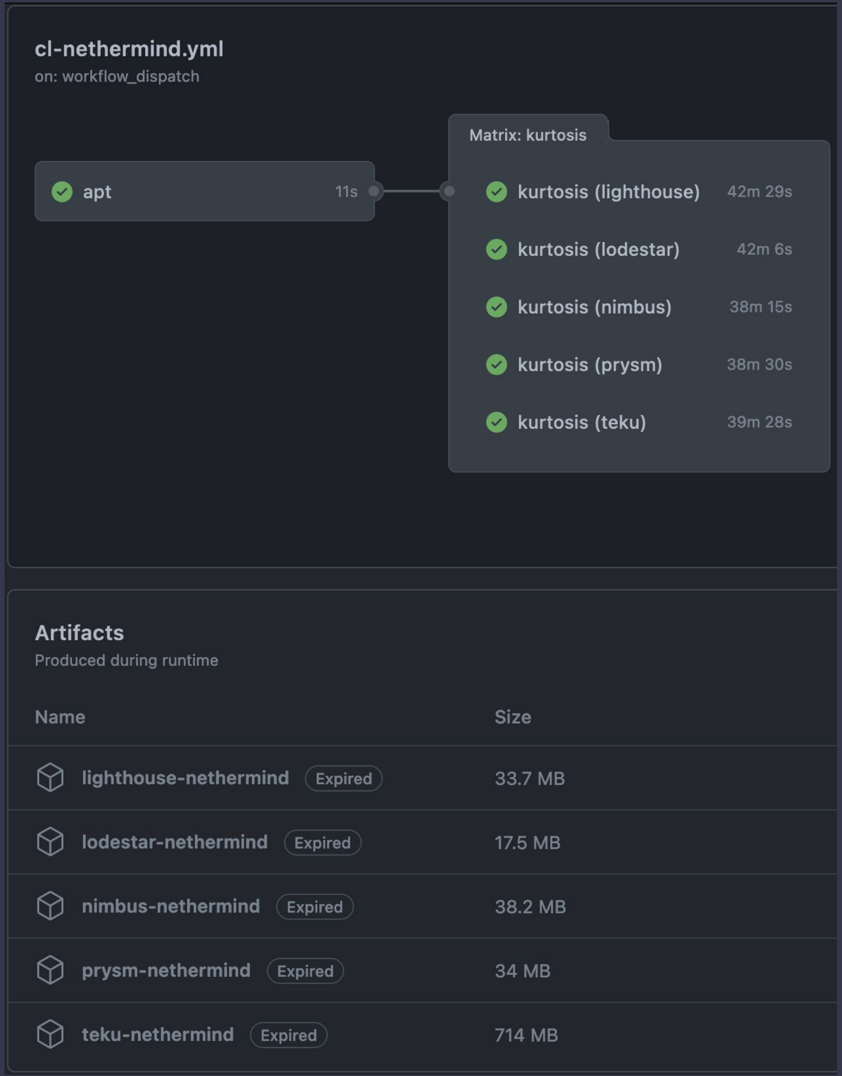
- Hive tests run using a simulator that starts up the clients and runs the tests against a pre-defined interface
- Acts as a integration and regression check to make sure client aren't failing defined edge cases
- e.g: Feed a Nethermind node two terminal blocks, assert how it transitions
- Shoutout to @elbuenmayini



Start time	Suite	Clients	Pass
2022-10-10T13:24:58.444Z	engine-api	nethermind	✓ (141)
2022-10-10T11:19:45.834Z	engine-api	go-ethereum	✓ (141)
2022-10-10T08:32:22.169Z	engine-api	erigon	✓ (141)
2022-10-10T04:19:55.077Z	engine-api	nethermind	✓ (141)
2022-10-10T03:25:14.198Z	engine-api	go-ethereum	✓ (141)
2022-10-09T20:19:15.930Z	engine-api	nethermind	✓ (141)
2022-10-09T19:24:28.663Z	engine-api	go-ethereum	✓ (141)
2022-10-09T16:35:01.964Z	engine-api	erigon	✓ (141)
2022-10-09T11:41:45.059Z	engine-api	nethermind	✓ (141)
2022-10-09T10:46:59.564Z	engine-api	go-ethereum	✓ (141)
2022-10-09T07:59:43.300Z	engine-api	erigon	✓ (141)
2022-10-09T03:05:33.754Z	engine-api	nethermind	✓ (141)
2022-10-09T02:10:57.615Z	engine-api	go-ethereum	✓ (141)
2022-10-08T18:25:21.055Z	engine-api	nethermind	✓ (141)
2022-10-08T17:30:35.247Z	engine-api	go-ethereum	✓ (141)
2022-10-10T10:24:07.968Z	eth2-testnet	teku-vc,go-ethereum,teku-bn	✓ (2)

Kurtosis tests

- Kurtosis spins up a local testnet with the required EL/CL combinations and then allows them to transition/merge. It then asserts some “happy case” conditions.
- An integration test make sure client are compatible
- Useful to rapidly iterate ideas
- e.g: Are blocks being produced, are there tx’s…



cl-nethermind.yml
on: workflow_dispatch

✓ apt 11s

Matrix: kurtosis

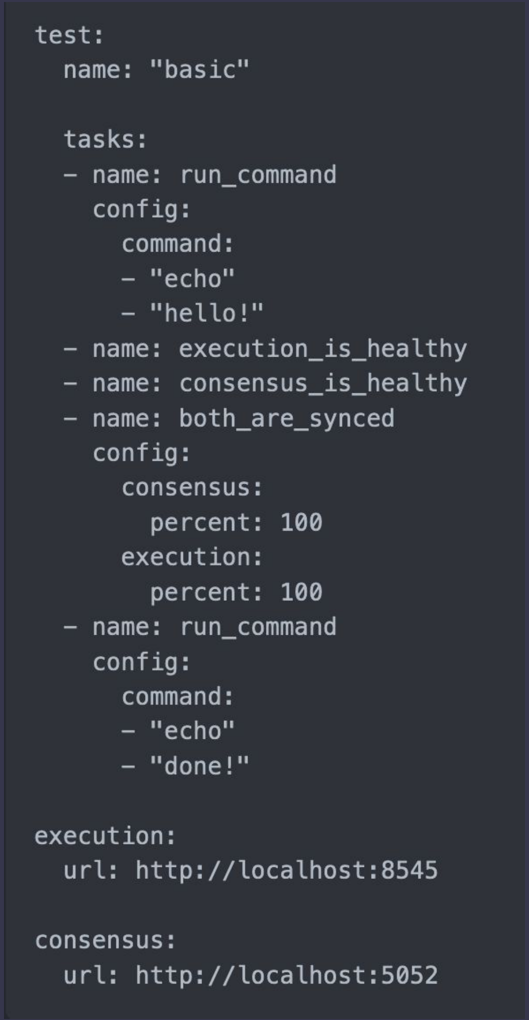
✓ kurtosis (lighthouse) 42m 29s
✓ kurtosis (lodestar) 42m 6s
✓ kurtosis (nimbus) 38m 15s
✓ kurtosis (prysm) 38m 30s
✓ kurtosis (teku) 39m 28s

Artifacts
Produced during runtime

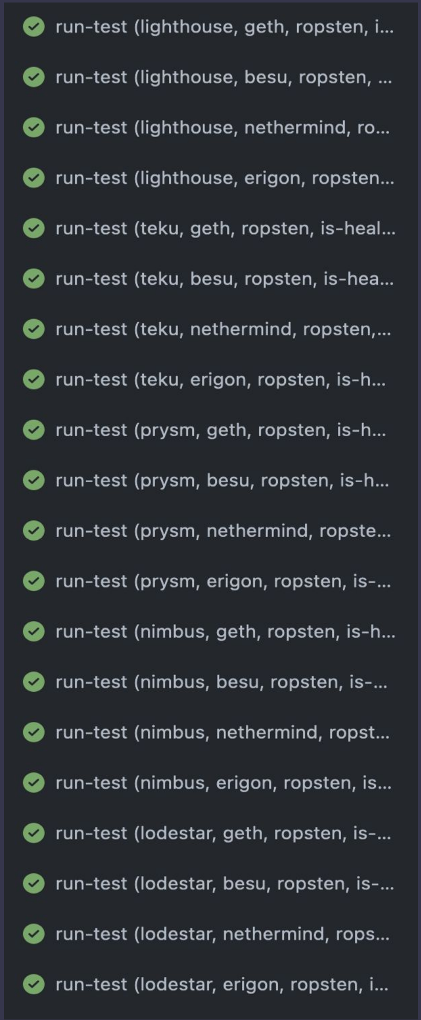
Name	Size
lighthouse-nethermind Expired	33.7 MB
lodestar-nethermind Expired	17.5 MB
nimbus-nethermind Expired	38.2 MB
prysm-nethermind Expired	34 MB
teku-nethermind Expired	714 MB

Sync tests

- The sync test co-ordinator spins up every client combination daily and syncs to head on various testnets. Both genesis sync as well as Checkpoint sync are performed.
- Edge case sync tests are also performed: EL down, CL down, etc
- Acts as a integration test make sure users can always sync the network
- Shoutout to @samcmAU



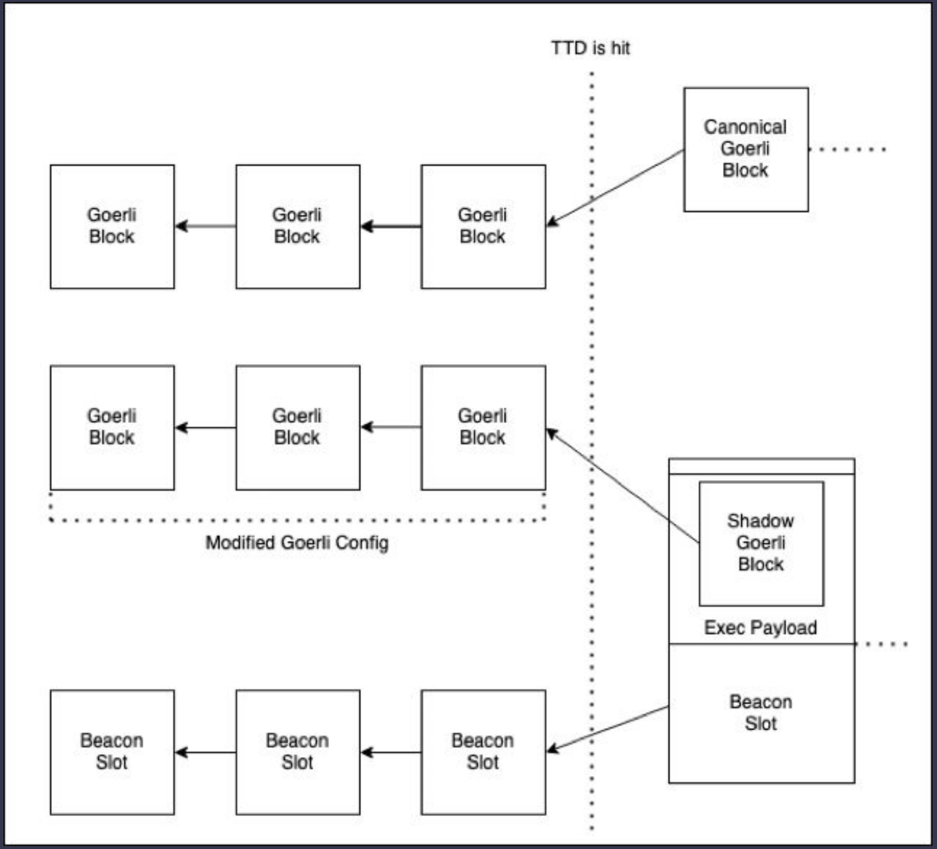
```
test:
  name: "basic"

  tasks:
  - name: run_command
    config:
      command:
      - "echo"
      - "hello!"
  - name: execution_is_healthy
  - name: consensus_is_healthy
  - name: both_are_synced
    config:
      consensus:
        percent: 100
      execution:
        percent: 100
  - name: run_command
    config:
      command:
      - "echo"
      - "done!"

execution:
  url: http://localhost:8545

consensus:
  url: http://localhost:5052
```



```
✓ run-test (lighthouse, geth, ropsten, i...
✓ run-test (lighthouse, besu, ropsten, ...
✓ run-test (lighthouse, nethermind, ro...
✓ run-test (lighthouse, erigon, ropsten...
✓ run-test (teku, geth, ropsten, is-heal...
✓ run-test (teku, besu, ropsten, is-hea...
✓ run-test (teku, nethermind, ropsten,...
✓ run-test (teku, erigon, ropsten, is-h...
✓ run-test (prysm, geth, ropsten, is-h...
✓ run-test (prysm, besu, ropsten, is-h...
✓ run-test (prysm, nethermind, ropste...
✓ run-test (prysm, erigon, ropsten, is-...
✓ run-test (nimbus, geth, ropsten, is-h...
✓ run-test (nimbus, besu, ropsten, is-...
✓ run-test (nimbus, nethermind, ropst...
✓ run-test (nimbus, erigon, ropsten, is...
✓ run-test (lodestar, geth, ropsten, is-...
✓ run-test (lodestar, besu, ropsten, is-...
✓ run-test (lodestar, nethermind, rops...
✓ run-test (lodestar, erigon, ropsten, i...
```

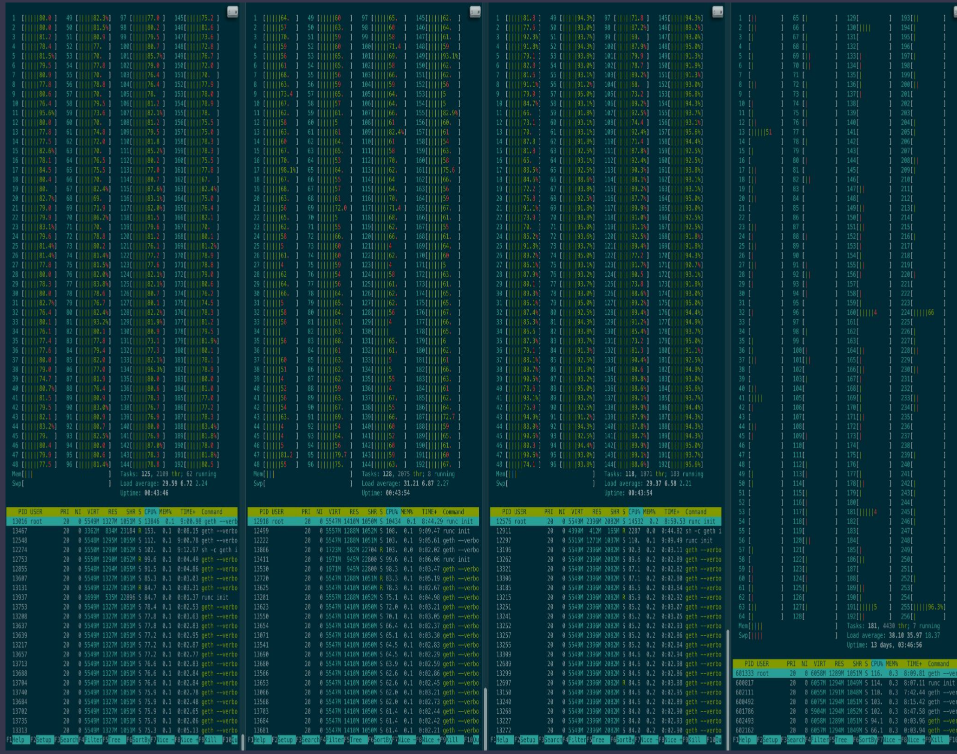
Shadow Forks & Testnets

- 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



Antithesis & Fuzzers

- Antithesis offers a deterministic hypervisor which allows us to perform network splits, packet loss while fuzzing clients. The deterministic hypervisor allows us to re-trigger the issue, allowing for capturing the state of the client and easier debugging.
- Various fuzzers are run against different layers of the stack to find bugs.
- These bugs also allow us to re-evaluate if changes need to be made in the specs or if the bug is an implementation level issue.



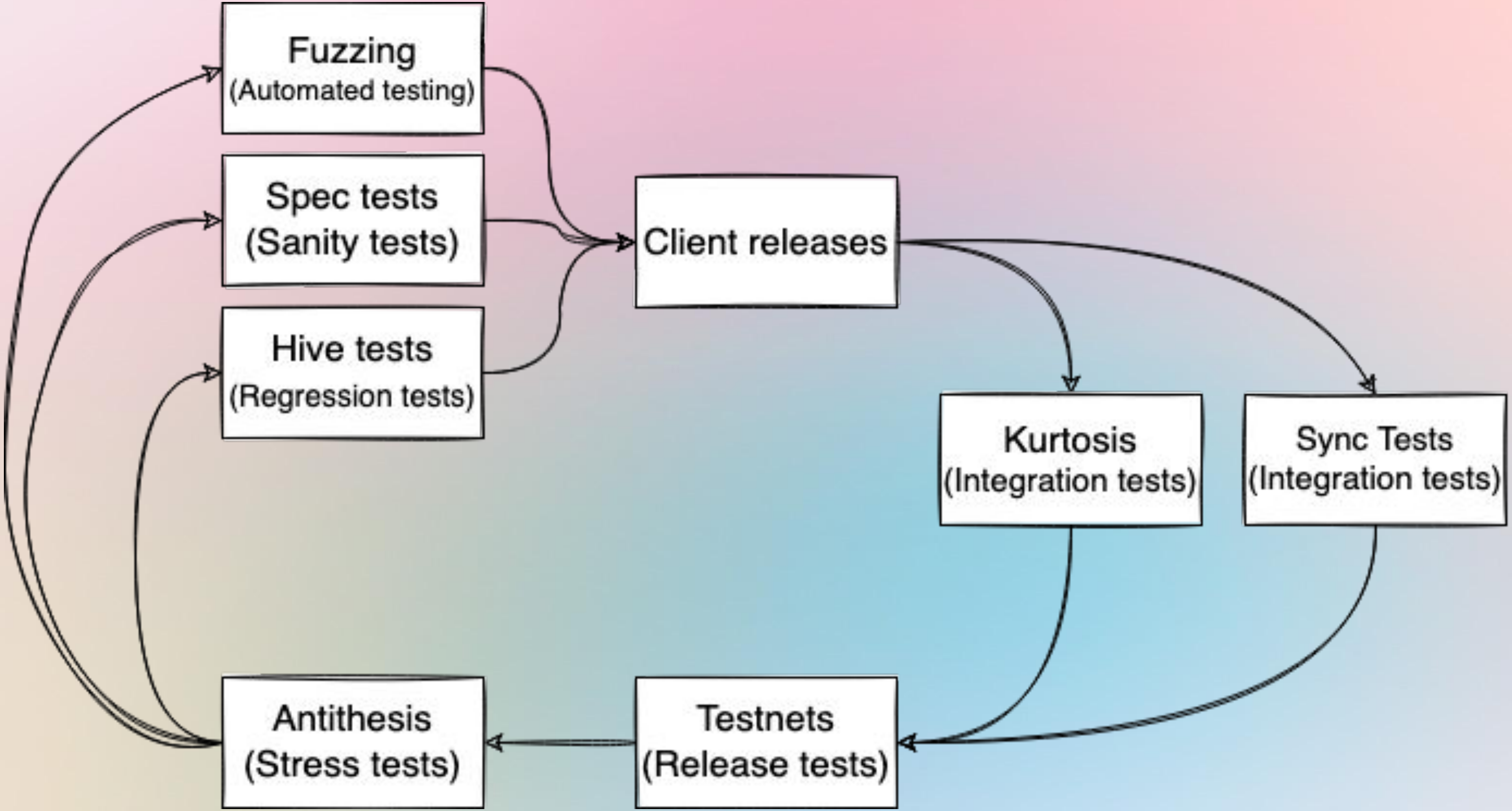
```
Tasks: 125, 2109 thr; 62 running
Load average: 29.59 6.72 2.24
Uptime: 00:43:46

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
13016 root       20   0 5549M 1327M 1051M S 13846  0.1  9:00.98 geth --verb
13467            20   0 3362M  834M 23184 R 153.  0.1  0:08.15 geth --verbo
12548            20   0 5548M 1295M 1055M S 112.  0.1  9:00.78 geth --verbo
12274            20   0 5550M 1290M 1052M S 102.  0.1  9:12.97 sh -c geth i
12753            20   0 5550M 1290M 1052M R 99.6  0.1  0:04.49 geth --verbo
12855            20   0 5548M 1294M 1055M S 91.5  0.1  0:04.86 geth --verbo
13607            20   0 5549M 1327M 1051M S 85.3  0.1  0:03.03 geth --verbo
13131            20   0 5549M 1327M 1051M R 84.7  0.1  0:03.31 geth --verbo
13937            20   0 1699M  535M 22896 S 84.7  0.0  0:01.37 runc init
13753            20   0 5549M 1327M 1051M S 78.4  0.1  0:02.53 geth --verbo
13208            20   0 5549M 1327M 1051M S 77.8  0.1  0:03.63 geth --verbo
13637            20   0 5549M 1327M 1051M S 77.8  0.1  0:02.83 geth --verbo
13639            20   0 5549M 1327M 1051M S 77.2  0.1  0:02.95 geth --verbo
13217            20   0 5549M 1327M 1051M S 77.2  0.1  0:02.87 geth --verbo
13657            20   0 5549M 1327M 1051M S 77.2  0.1  0:02.77 geth --verbo
13713            20   0 5549M 1327M 1051M S 76.6  0.1  0:02.83 geth --verbo
13688            20   0 5549M 1327M 1051M S 76.6  0.1  0:02.84 geth --verbo
13704            20   0 5549M 1327M 1051M S 76.6  0.1  0:02.84 geth --verbo
13740            20   0 5549M 1327M 1051M S 75.9  0.1  0:02.78 geth --verbo
13684            20   0 5549M 1327M 1051M S 75.9  0.1  0:02.48 geth --verbo
13702            20   0 5549M 1327M 1051M S 75.9  0.1  0:02.65 geth --verbo
13735            20   0 5549M 1327M 1051M S 75.9  0.1  0:02.06 geth --verbo
13313            20   0 5549M 1327M 1051M S 75.3  0.1  0:05.13 geth --verbo
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Qu
```

```
Tasks: 128, 2075 thr; 8 running
Load average: 31.21 6.87 2.27
Uptime: 00:43:54

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
12918 root       20   0 5547M 1410M 1050M S 10434  0.1  8:44.29 runc init
12499            20   0 5557M 1288M 1052M S 108.  0.1  9:09.47 runc init
12222            20   0 5547M 1288M 1051M S 103.  0.1  9:05.61 geth --verbo
13866            20   0 1723M  582M 22704 R 103.  0.0  0:02.02 geth --verbo
13411            20   0 1971M  945M 22800 S 99.6  0.1  0:06.06 runc init
13530            20   0 1971M  945M 22800 S 98.3  0.1  0:03.47 geth --verbo
12720            20   0 5547M 1288M 1051M R 83.3  0.1  0:05.19 geth --verbo
13625            20   0 5547M 1410M 1050M R 78.3  0.1  0:02.67 geth --verbo
13201            20   0 5557M 1288M 1052M S 75.1  0.1  0:04.98 geth --verbo
13623            20   0 5547M 1410M 1050M S 72.0  0.1  0:03.21 geth --verbo
13550            20   0 5547M 1410M 1050M S 70.1  0.1  0:03.05 geth --verbo
13654            20   0 5547M 1410M 1050M S 66.4  0.1  0:02.37 geth --verbo
13071            20   0 5547M 1410M 1050M S 65.1  0.1  0:03.30 geth --verbo
13541            20   0 5547M 1410M 1050M S 64.5  0.1  0:02.83 geth --verbo
13690            20   0 5547M 1410M 1050M S 64.5  0.1  0:02.29 geth --verbo
13680            20   0 5547M 1410M 1050M S 63.9  0.1  0:02.59 geth --verbo
13566            20   0 5547M 1410M 1050M S 62.6  0.1  0:02.86 geth --verbo
13653            20   0 5547M 1410M 1050M S 62.6  0.1  0:02.45 geth --verbo
13066            20   0 5547M 1410M 1050M S 62.0  0.1  0:03.21 geth --verbo
13568            20   0 5547M 1410M 1050M S 62.0  0.1  0:02.73 geth --verbo
13703            20   0 5547M 1410M 1050M S 61.4  0.1  0:02.44 geth --verbo
13684            20   0 5547M 1410M 1050M S 61.4  0.1  0:02.42 geth --verbo
13681            20   0 5547M 1410M 1050M S 61.4  0.1  0:02.21 geth --verbo
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Qu
```

```
Tasks: 118, 1971 thr; 183 running
Load average: 29.37 6.58 2.21
Uptime: 00:43:54

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
12576 root       20   0 5549M 2396M 2082M S 14532  0.2  8:59.53 runc init
12911            20   0 4390M  412M  165M R 2287  0.0  0:44.82 sh -c geth i
12297            20   0 5515M 1271M 1037M S 110.  0.1  9:09.49 runc init
13196            20   0 5549M 2396M 2082M S 90.3  0.2  0:03.11 geth --verbo
13262            20   0 5549M 2396M 2082M S 89.6  0.2  0:02.89 geth --verbo
13321            20   0 5549M 2396M 2082M S 87.1  0.2  0:02.82 geth --verbo
13306            20   0 5549M 2396M 2082M S 87.1  0.2  0:02.80 geth --verbo
13185            20   0 5549M 2396M 2082M S 86.5  0.2  0:03.64 geth --verbo
13215            20   0 5549M 2396M 2082M R 85.9  0.2  0:02.92 geth --verbo
13251            20   0 5549M 2396M 2082M S 85.2  0.2  0:03.07 geth --verbo
13241            20   0 5549M 2396M 2082M S 85.2  0.2  0:03.05 geth --verbo
13252            20   0 5549M 2396M 2082M S 85.2  0.2  0:02.93 geth --verbo
13257            20   0 5549M 2396M 2082M S 85.2  0.2  0:02.86 geth --verbo
13255            20   0 5549M 2396M 2082M S 85.2  0.2  0:02.84 geth --verbo
13309            20   0 5549M 2396M 2082M S 84.6  0.2  0:02.94 geth --verbo
12689            20   0 5549M 2396M 2082M S 84.6  0.2  0:02.98 geth --verbo
13299            20   0 5549M 2396M 2082M S 84.6  0.2  0:02.86 geth --verbo
12697            20   0 5549M 2396M 2082M R 84.6  0.2  0:03.88 geth --verbo
13150            20   0 5549M 2396M 2082M S 84.6  0.2  0:02.95 geth --verbo
13240            20   0 5549M 2396M 2082M S 84.6  0.2  0:02.89 geth --verbo
13268            20   0 5549M 2396M 2082M S 84.0  0.2  0:02.90 geth --verbo
13227            20   0 5549M 2396M 2082M S 84.0  0.2  0:02.93 geth --verbo
13272            20   0 5549M 2396M 2082M S 84.0  0.2  0:02.90 geth --verbo
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Qu
```

```
Tasks: 181, 4430 thr; 7 running
Load average: 38.10 35.97 18.37
Uptime: 13 days, 03:46:56

   PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
601333 root       20   0 6058M 1289M 1051M S 116.  0.3  8:09.81 geth --ver
600817            20   0 6057M 1294M 1049M S 114.  0.3  8:07.11 runc init
602111            20   0 6055M 1291M 1048M S 110.  0.3  7:42.44 geth --ver
600492            20   0 6075M 1294M 1051M S 103.  0.3  8:15.42 geth --ver
601786            20   0 5904M 1294M 1052M S 102.  0.3  8:47.58 geth --ver
602493            20   0 6058M 1289M 1051M S 94.1  0.3  0:03.96 geth --ver
602162            20   0 6057M 1294M 1049M S 66.1  0.3  0:03.94 geth --ver
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10
```

Testing lifecycle for the Merge



Fuzzing
(Automated testing)

Spec tests
(Sanity tests)

Hive tests
(Regression tests)

Client releases

Kurtosis
(Integration tests)

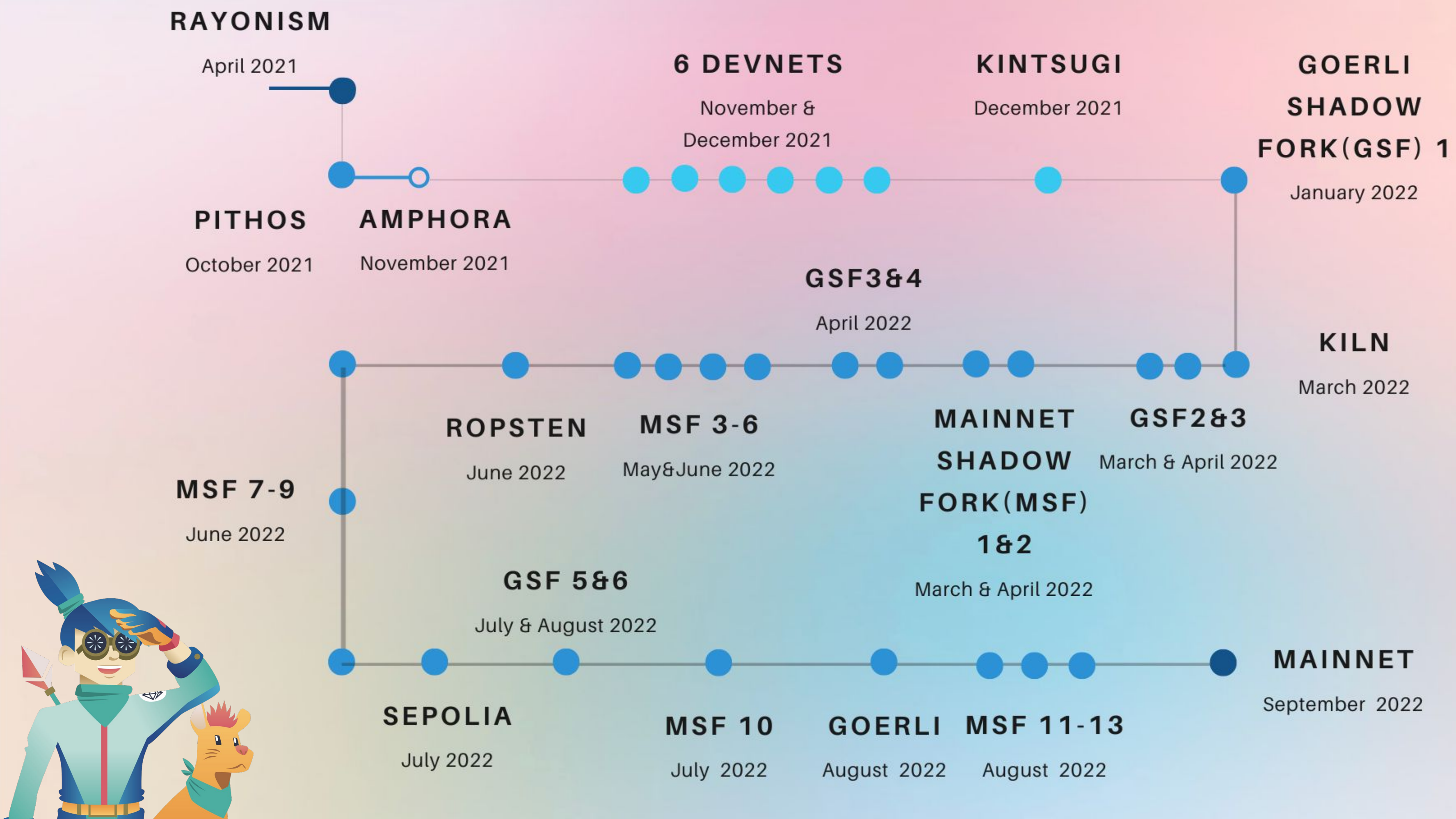
Sync Tests
(Integration tests)

Testnets
(Release tests)

Antithesis
(Stress tests)

Section 3

#TestingTheMerge: The Endgame



RAYONISM
April 2021

PITHOS
October 2021

AMPHORA
November 2021

6 DEVNETS
November &
December 2021

KINTSUGI
December 2021

GOERLI SHADOW FORK(GSF) 1
January 2022

KILN
March 2022

GSF2&3
March & April 2022

MAINNET SHADOW FORK(MSF) 1&2
March & April 2022

GSF3&4
April 2022

MSF 3-6
May&June 2022

ROPSTEN
June 2022

MSF 7-9
June 2022

SEPOLIA
July 2022

GSF 5&6
July & August 2022

MSF 10
July 2022

GOERLI
August 2022

MSF 11-13
August 2022

MAINNET
September 2022

So what did we still miss?

- In-memory database too low to process mainnet blocks
- Non-optimal block production: Random production of 0/few tx blocks
- Multiple terminal blocks (in a specific condition) caused missing receipts and caused failed proposals
- Lots of constant syncing nodes on mainnet led to unexpected performance degradation when compared to shadow forks
- Failover beacon node scenario -> some requests sent just to the primary

What can we reuse?

Running testnets helped show us tooling blind spots in the DevOps ecosystem:

- Metrics exporter: https://github.com/samcm/ethereum-metrics-exporter
- Sync testing: https://github.com/samcm/ethereum-sync-testing/actions
- Genesis gen.: https://github.com/skylenet/ethereum-genesis-generator
- Client automation: https://github.com/ethPandaOps/ethereum-helm-charts
- Scalable testnets: https://github.com/ethPandaOps/ethereum-k8s-testnets
- Easy testnets: https://github.com/ethereum/consensus-deployment-ansible
- Faucet: https://github.com/komputing/FaucETH
- Checkpoint Sync Provider: https://github.com/samcm/checkpointz
- PRs to explorers, validator key generation tools, load balancer

If you want to join the testing efforts contact mario.vega@ethereum.org

Thank you!
Join #TestingThe{Surge,Verge,Purge}!

Parithosh Jayanthi

DevOps Engineer

parithosh@ethereum.org

@parithosh_j