Ethereum Eth1.x WG3: Simulation

https://github.com/ethereum/ethsim

This document is to capture the infrastructure, platform, and requirements for simulation towards an Eth1.x feature list for inclusion in an upcoming hard fork. The simulation platform is intended to resolve questions that the community has about where we are at, where we are headed, and what we need to know and do to get there. The structure of the document will capture as-is state, to-be state, and potential transition paths from the as-is to the to-be state.

Key concepts:

- Simulation: a mathematical model to provide probabilistic data sets for performance or whatever else.
- Emulation: functionally replaces these processes to provide a practical and functional model of a system
- Testnets: This term covers platforms running one or more chains and clients with a wide spectrum of properties, from long-running, unpermissioned chains to permissioned chains lasting only seconds.
- Test plan: A plan for a testnet run, describing properties like configuration and starting state. See here for an example: https://notes.ethereum.org/Q_kQKXZUQD29YCshej1qPQ
- Infrastructure: machine instances to be used for testing.
- Platform: code + infrastructure that is to be used for testing.
- Eth1.x Features: See the overview document here.
- POC: point of contact

Minutes from Dec 7

Updates after some comments and reviews (Nov 27, 2018):

General Instructions:

1. Fill in your name and email address (plus any other contact info) in the participant section.
2. Indicate your role in the notes, such as which feature you're helping to coordinate.
3. Review the document and comment as a way to suggest changes. There are various areas on which to comment: testnet platforms, client tests, simulation platforms, simulation tests, integration tests.
4. Those who manage bootnodes and instances should report their role in the participants table.
5. If you are a contact for bootnodes and instances and are running a specified feature, provide the relevant addresses, docker links, info in the table at the end of this doc.
6. For support, comments and questions, contact Shahan.

Participants:

Name	Notes
Shahan Khatchadourian (PegaSys), shahan.khatchadourian@consensys.net, Telegram: @shahankhatch	WG POC
Casey Detrio	eWASM
Alexey Akhunov	State
Piper Merriam	My team is adopting `ethereum/tests` and cross client testing within the EF. This seems potentially related.
Zak Cole (Whiteblock) zak@whiteblock.io Telegram: @zcole	Testnet Platforms – Whiteblock testing environment and open source framework.

Summary of target properties to test (discussion is below):

1. Block propagation, to reduce uncle rate
2. If yes, raise gas block limit
3. If yes, consider increasing cost of storage wrt compute
4. Determining what the next bottleneck is
    a. Assume compute time is reduced
    b. Assume bandwidth limit results in many uncles

The plan is to work towards the following setups

1. Simulation framework that develops a mathematical model given some datasets and can produce output that might estimate properties (currently Wittgenstein is suggested)
2. Emulation framework that launches functional instances and alters environment conditions in order to test properties (currently being designed by Whiteblock)
3. Testnet that launches instances together on the same network to ensure there are no unexpected hiccups in client interop (the idea of a testnet should be reviewed).

In order for a simulation/emulation framework to be useful, datasets (assumed to be accurate) are being collected here in order to determine appropriate parameters to test:

1. Whiteblock's prior experience with testing uncle rates.
2. Data from external sources, e.g., etherscan

3. Properties drawn from various sources, like research papers, a number of these properties have already been incorporated into this simulation framework

Old notes below here

Important notes:

1. Since the simulation framework does not yet exist, we are starting with the coordination of non-simulated testnets.
2. Please update the document as per the general instructions below.
3. As soon as possible, please update the list of relevant docker containers and EIPs that they integrate as part of understanding the clients to to be launched for test.

Goals:

1. Support clients and core developers in two mainnet needs, for (1) being a peer on the Ethereum mainnet network, such as by reducing state storage requirements, and (2) for supporting novel community applications, by supporting deployment and execution of ewasm-based smart contracts.
2. Determine the best way to serve peer and end user needs using Eth1.x simulations and one or more testnets for each feature.
3. Open collaboration of clients and participants by sharing information and obtaining feedback.

As-is:

Platforms:
There are a few testnets running, but they aren't planned to be upgraded or used for this work (as far has been mentioned so far, this can change). The plan is to have a new simulation framework and platform, with the intent to reuse as much of existing code and simulation frameworks as possible.

To-be:
Platform Options (can do one or more of these):

1. Run a testnet as usual, with no simulation platform. Add tests into the GeneralState tests framework however much possible. This ensures that the clients operate in a "backwards compatible" way.
2. Run a testnet with the baseline simulation platform to test various network latencies and expectations. This will require definitions and configurations that are usable by the framework and understood by the various teams. As with open source, there will be a

mechanism for collaborating on the open source project. The current timeline is beginning of January, but this should be discussed.

Infrastructure: Nodes running clients that wish to participate in the testnet. The code initially will be Docker for running an Ethereum for a particular feature.
With 2 features being considered, we initially aim to have (at least) 2 testnets:

(1) Statement management testnet(s)
    (a) Determine the baseline client for this configuration. The experts should fill in the exact details, including genesis, environment configuration, etc. If possible, have the tests as part of the general state tests.
    (b) Part of this work is applying the state pruning code / ideas into the clients. This will take time and the sequencing of which parts of the code need to be updated needs to go through EIPs. There are several views on whether some of the proposals are actually viable and semantically correct for the mainnet.
(2) eWASM testnet(s)
    (a) Determine the baseline client for this configuration. The experts should fill in the exact details, including genesis, environment configuration, etc. If possible, have the tests as part of the general state tests.
    (b) ewasm already has its changes in geth, and a testnet (which I could not access), though the team says that they will give an update on access details very soon.
        (i) There are some environment configurations that differ from mainnet expectations. See here.

Platform features that aim to be addressed through this document:

1. Bootnodes will be managed by testnet POCs for each stream of work.
2. Instances not yet running on the simulation platform are to be self-managed by participating clients (the "backwards compatible" approach).
3. When a simulation platform is to be used, the instances will be testnet-managed instances, such that the instances are launched, simulated given set parameters, then shutdown. The results are collected and shared by simulation testnet POCs.
4. Links to Docker images containing the features to be explored, for each of the clients.
5.

The process of merging the multiple features will be discussed based on their dependencies, e.g., gas limits vs storage requirements.

Immediate Solution

While the development of this open-source framework may take some time, an immediately available solution is the implementation of a permissioned testnet by Whiteblock and made available to the Ethereum Core Dev team. The network and primary nodes will be provisioned and managed by Whiteblock. Transactional behavior will be automated and individuals who

wish to participate by provisioning their own nodes within the network need only provide their enode address and ensure their client is running on the appropriate network ID.

The parameters, configuration, and other behaviors of the testnet can be modified in real-time based on the consensus of the network participants in order to implement particular test cases and conditions, allowing for the simulation of various scenarios.

Some good notes, comments, and views from other testnets and hardforks (relevant parts of the notes will be referenced in this document):
https://ethereum-magicians.org/t/issues-we-discovered-in-the-ropsten-constantinople-hard-fork/1598/7

The following table is intended to capture the testnets for this working group. This table can also be used to capture other testnets if you wish (and the table can be restructured for clarity as needed):

Testnet	Bootnode	Clients	Simulated?	EIPs
ExampleTestnet1 POC: Shahan	Does not exist yet	Parity, Geth	No	eWASM
ExampleTestnet2 POC: Shahan	127.0.0.1:124	Pantheon	Yes	None
			No	State
			Yes	State

There will be additional tables that will be added here soon to capture the simulation parameters of note for each feature. This will cover, system configs as well as inputs. Like simulation framework links, network latency, disk/memory usage, ethereum configs, tx sources, result output.

Simulation references (more info on platform coming soon):
1. Ethereum info
https://theethereum.wiki/w/index.php/Network_Ports,_Files_And_Directories

2. TCP Proxy
https://github.com/Shopify/toxiproxy
TCP-based proxy, which suffices for non-UDP components of Ethereum networking, i.e., peer

discovery is the only UDP-based component. This can be resolved through manual configuration of peer addresses.

3. Docker container chaos management:
https://github.com/alexei-led/pumba
https://hackernoon.com/chaos-testing-for-docker-containers-bc6e9d66645

4. netem module
https://wiki.linuxfoundation.org/networking/netem

Notes from Alexey

HI! I don't know how to integrate my comment into this document yet, but here are my thoughts on the first things we could simulate.
I assume that there is currently a simulation being worked on to test whether the block propagation fix will bring the uncle rate down.
Assuming that it will, the next thing everyone would likely to want doing to raise gas block limit. However, we might want to do that, and simultaneously making storage operations more expensive, effectively reducing the cost of computation compared to the status quo. That would be a hard fork, but a simple one.
Then, the next thing we might want to ponder, where is our next bottleneck lies. Imagine that we keep increasing block gas limit. What will happen first - our nodes will start lagging behind the chain because the computation of a block will be less than 1/10 th of the average time (I remember Peter called it a rough safety margin), or bandwidth will get exhausted and we will have lots of uncles again. Based on that, we would need to establish a recommended safe block gas limit for the network to have.

More from Alexey
Something more controversial, but aso maybe more exciting. Inspried by this (stress test on Bitcoin Cash networks):
https://www.reddit.com/r/btc/comments/a1bp0b/please_send_me_your_debuglog_files_from_the_bsv/
I am suggesting to implement stress test mode in the mainnet clients. It could be transactions marked by a special flags, so that they have two special properties. Although they are included into the blocks and pay miners reward, they update a "shadow state", which is add-on to the existing state at the start of the stress test. After the end of the stress test, the "shadow state" gets evicted from the state completely. There is a sub-protocol allowing anyone to download logs (with proper anonymisation). Etc… Do you think this is a dangerous idea?

Shahan's summary of target properties to test from above thoughts:

5. Block propagation, to reduce uncle rate
6. If yes, raise gas block limit

7. If yes, consider increasing cost of storage wrt compute
8. Determining what the next bottleneck is
    a. Assume compute time is reduced
    b. Assume bandwidth limit results in many uncles