

Conformance Testing of Consensus: generate forks from adversarial branches

Summary

This proposal is about enriching the generators for property based tests of the Haskell implementation of the Genesis protocol. With this proposal, the generators will produce more diverse inputs that should exercise more of the potential attacks to the implementation.

Problem statement

Property based tests (PBT) ensure that the implementation meets a particular expectation for randomly selected inputs. If there are some inputs where the software misbehaves, the PBT has a chance to find it. This chance, however, is influenced by the variety of the inputs produced by the input generator.

One of the inputs to Genesis tests is the tree of blocks that consensus nodes must navigate to discover the most dense branch with valid blocks ([link](#)). The tree generator, at the moment, produces trees where a single branch is sufficiently dense, and adversarial branches which are less dense fork from it. One shape of tree that we are missing is the one in which adversarial branches fork from other adversarial branches.

The solution

The core change that we propose is to add a phase to tree generation where we produce branches that are less dense than the densest branch and fork from an adversarial branch.

Once the tree is generated, we need to update the consumers of this tree. The first consumer is not the PBT directly but the point schedule generator ([link](#)). A point schedule indicates the time to send blocks or headers to each simulated peer of the node under test. The generator of point schedules uses the generated tree to produce sensible point schedules. For instance, there should be a delay to prevent headers from being sent before the simulation has reached the slot in which the corresponding block was minted.

With the point schedule in hand, we can run the property based tests that use them, and we will diagnose and fix test failures.

Core Objectives

- Extend the Conformance Testing of Consensus (CTC) framework to generate adversarial fork scenarios, including branches that violate expected consensus behavior.
- Develop test cases that simulate Genesis-specific edge cases and adversarial conditions to improve test coverage.
- Enable early detection of consensus flaws by validating node behavior against adversarial fork scenarios.
- Integrate these tests into existing testing workflows to ensure continuous validation of the consensus implementation.

Expected Value

This proposal improves the safety of the Cardano consensus layer by enabling systematic testing of adversarial fork scenarios, including those that expose edge cases in Genesis and related mechanisms. By validating node behavior under adversarial conditions prior to deployment, it enables early detection of consensus flaws that could otherwise lead to incidents on mainnet.

This directly supports network reliability by reducing the likelihood of consensus failures and unsafe upgrades (**Monthly uptime**). By making these tests available across implementations, the proposal also supports client diversity and operational resilience, ensuring consistent behavior across node clients.

Metric of Success

- No identified flaky tests remain in the Genesis test suite, with all tests demonstrating stable results across repeated CI executions.

How this work package will be delivered

Milestones

Milestone	Deliverables	Acceptance criteria	Weeks
T7.0 Project Kickoff	Public project setup including repository, documentation, and tracking. Initial description of adversarial fork testing	Repository and project materials are publicly accessible. Scope and objectives of adversarial fork testing are clearly documented.	2

	goals and scope. Public announcement/demo introducing the project.	A public demo or announcement is available.	
T7.1 Adversarial Fork Generator Design	Design of enhancements to: - block tree generator (supporting adversarial branching). - point schedule generator (timing of block propagation). Documentation describing how adversarial scenarios are constructed.	The design is accepted by the consensus team.	12
T7.2 Implementation of Adversarial Fork Generation	Implementation of: - adversarial branch generation in block tree generator. - updated point schedule generator supporting complex fork scenarios. Initial integration into testing framework.	Changes are reviewed and accepted for merge as pull requests to the ouroboros-consensus repository. All review feedback is addressed. Generator produces adversarial fork scenarios as defined in the design. Initial tests execute (even if not all pass yet).	8
T7.3 Test Stabilization and Coverage Validation	Stabilization of all tests using adversarial fork scenarios. Analysis of test coverage and identification of previously untested cases. Improvements to ensure deterministic and reliable test execution.	All tests pass consistently across repeated CI runs. Test coverage analysis is documented and published. All changes are reviewed and accepted for merge in the ouroboros consensus repository.	14