Groot: Impact of Evolutionary Operators in XRPL Testing using Priority-Based Event Representation

Author:
Bryan Wassenaar (B.J.A.Wassenaar@student.tudelft.nl)
Supervisors:
Burcu Kulahcioglu Özkan, Mitchell Olsthoorn, Annibale Panichella

TU Delft

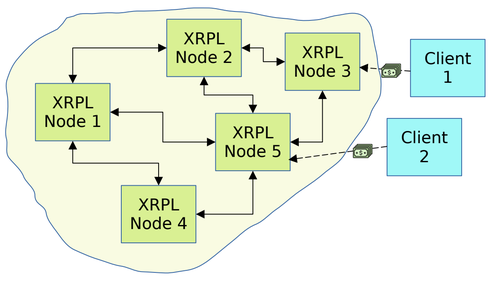
Background

Evolutionary Algorithm

Initial Population
An initial population is created based on random generation.

Selection
The best specimen are chosen using a mathematical fitness function.

Reproduction
The specimens are combined using a crossover and a mutation operator.



Distributed XRP Network

- All nodes keep track of all transactions.
- Clients can submit transactions to any node.
- Transactions will be communicated and validated with the network.

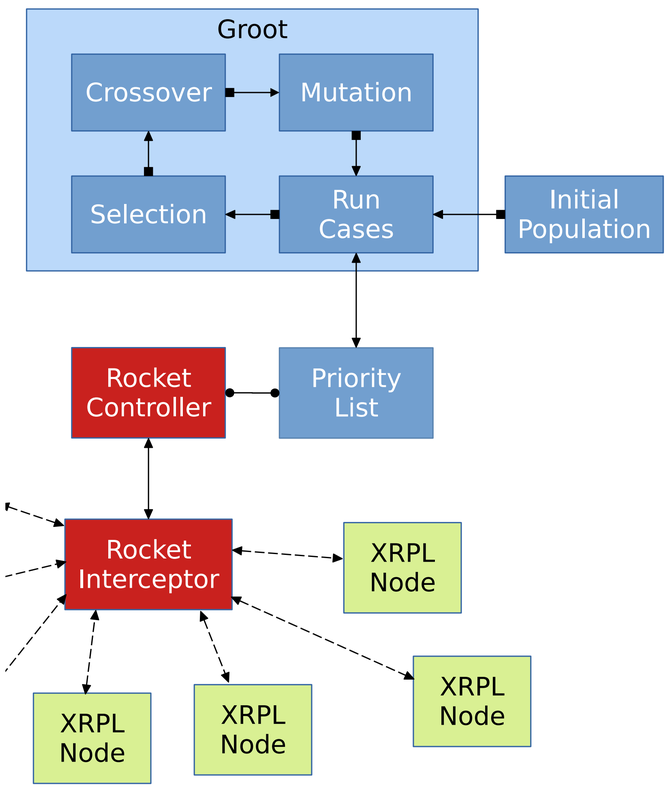
Problem

Crypto-assets importance
Crypto-assets are becoming more important in the modern day world with a market capitalization of 2.6 trillion USD in 2021.

Concurrency bugs
Testing is difficult because of the distributed nature of these systems, which causes difficult to detect concurrency bugs.

Diversity of operators
Current research which uses evolutionary algorithms for testing the XRPL Protocol did not diversify their selection of evolutionary operators.

Groot



For each combination of operators, Groot is used.

1) Random Baseline
2) SBX – Gaussian
3) Laplace - MPTM

- Groot creates an initial population of 10 test cases (encodings).
- The algorithm runs all test cases and uses selection and reproduction to create a new population.
- Each test case is represented as a priority list.
- The Rocket Framework runs each test case two times and checks for constraint violations.

Evaluation

XRPL Protocol Constraint violations

Termination
All nodes eventually finalize a ledger. This ensures the network makes progress and can process transactions.

Agreement
All nodes submit the same ledger for the same sequence number. This ensures no forks occur in the network.

Time-fitness

Test case run time
Long run time occurs when more messages were needed to reach consensus. This provides opportunities for faults.

Final evaluation factors

Effectiveness
Based on how many generations were able to find violations during the test.

Efficiency
Based on on the earliest generation that detected any violations and the number of violations identified within that generation.

Results

Total amount of runs having violations

	Baseline	SBX-Gaussian	Laplace-MPTM
Failed Agreement	68	69	67
Failed Termination	0	0	0

First generation that found a violation

	Baseline	SBX-Gaussian	Laplace-MPTM
First Violation	G2	G1	G2
Failed Agreement	4	3	5
Failed Termination	0	0	0

Conclusion

Effectiveness
all three setups had around the same amount of violations, the evolutionary algorithms are therefore not more effective than the random baseline at finding bugs in the XRP Ledger protocol.

Possible reasons:
- Noise in the fitness function causing to much variability.
- Time-fitness does not work well with priority-based event representation, because priorities don't cause larger delays.
- Priority-based event representation is not effective in detecting termination bugs, since they don't increase delays directly.

Efficiency
The three setups are close to each other in terms of efficiency, but SBX-Gaussian is slightly slower.

Possible reasons:
- The seeded bug was too easy to find, causing high probability that a random encoding finds it.
- High mutation probability causes generations to be diverse, but limits exploitation.