HYPERPARAMETER-TUNED RANDOMIZED TESTING FOR BYZANTINE FAULT-TOLERANCE OF THE XRP LEDGER CONSENSUS PROTOCOL

Author: Aistė Macijauskaitė <a.macijauskaite@student.tudelft.nl>
Supervisors: Dr. Burcu Kulahcioglu Ozkan, Dr. Mitchell Olsthoorn, Dr. Annibale Panichella

XRP LEDGER

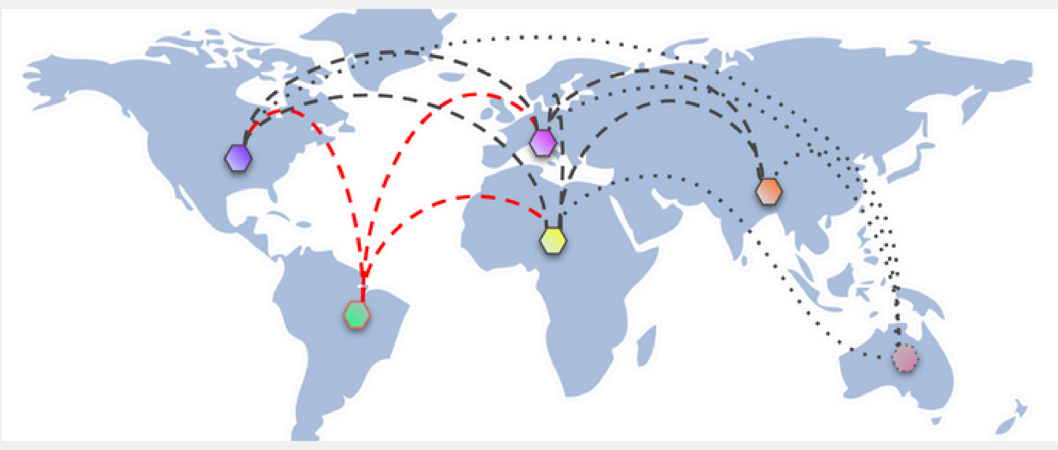
A public, decentralized blockchain that processes transactions for the popular cryptocurrency XRP.

$460.831.398.702

XRP Trading volume (December 2024)

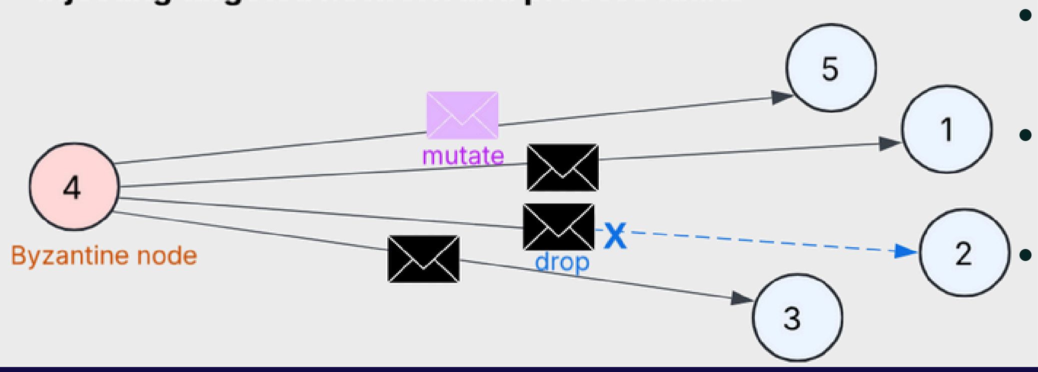
XRPL CONSENSUS PROTOCOL

- Validator nodes are decentralized and distributed worldwide. They must reach consensus to keep the blockchain **consistent** and **trustworthy**.
- Blockchain systems rely on **consensus protocols** to ensure agreement among nodes, even in the presence of malicious or faulty participants.
- The XRPL Consensus Protocol whitepaper [1] describes the design and implementation of **Byzantine Fault-Tolerant (BFT)** protocols that guarantees **safety** and **liveness**.
- In practice, these protocols are difficult to implement correctly and often suffer from **subtle logic errors**.



PROBLEM

Injecting targeted network and process faults



- Exhaustive manual testing of BFT protocols is infeasible as the number of possible execution scenarios is too large to test manually and node communication is inherently non-deterministic.
- **ByzzFuzz** search algorithm [2] uses a **randomized fault-injection** strategy that systematically injects **network faults** and **process faults** while preserving protocol semantics.
- However, there has been little focus on **optimizing its hyperparameters** or understanding how they impact the performance of the testing method.

BYZZFUZZ SEARCH ALGORITHM

Fault-Bounded Testing

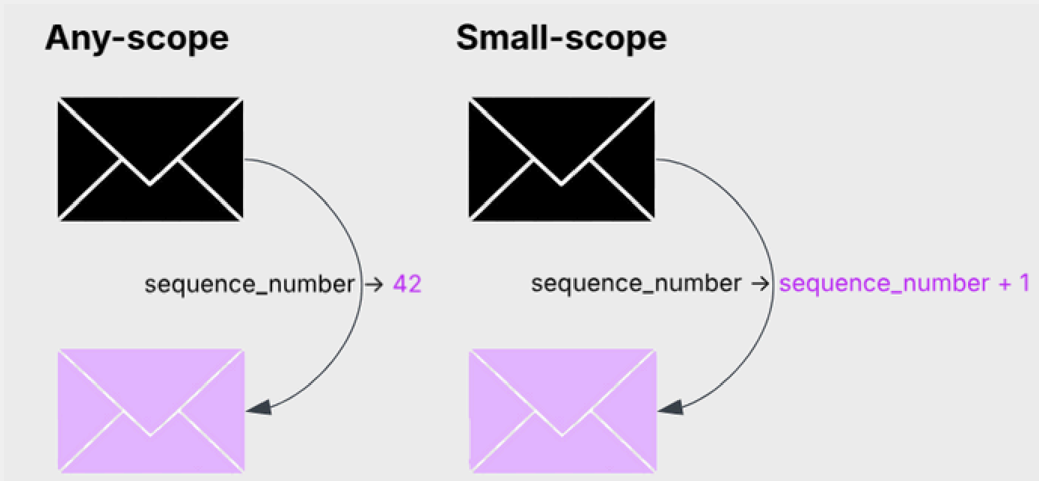
Restrict the total faults per run (e.g., at most *d* network faults and *c* process faults).

Round-Based Testing

Structure fault injection into specific protocol rounds.

Structure-Aware Message Mutations

- **Any-Scope (as) Mutations**: Apply mutations that deviate significantly from the original values while remaining syntactically valid.
- **Small-scope (ss) Mutations**: Mutate message fields to values close to the original message either in value or in time.



EXPERIMENTAL SETUP

RQ:
- Can the Byzzfuzz search algorithm detect bugs in the XRPL Consensus Protocol or its variants?
- How does the Byzzfuzz search algorithm compare to a baseline algorithm in bug detection?
- How does the selection of test parameters affect the performance of the Byzzfuzz search algorithm?

Evaluation

After each test run, we checked four consensus properties:
- **Termination**: Every honest node decides on a value within the time limit.
- **Validity**: Honest nodes decide only on values proposed by other honest nodes.
- **Integrity**: No honest node decides on the same value twice.
- **Agreement**: No two honest nodes validate different ledgers for the same round.

Test Design
- Designed an XRPL network of seven nodes, one of which is randomly chosen to be Byzantine (satisfying ≤ $\lfloor (n-1)/5 \rfloor$ Byzantine nodes for safety and liveness)
- Configured three Unique Node Lists describing trust relationships between nodes, with 60% overlap.
- Executed naive random testing as a baseline and ran the ByzzFuzz algorithm with varied test parameters: c ∈ {0, 1, 2} (number of process faults) and d ∈ {0, 1, 2} (number of network faults) within first (r=8) rounds.

RESULTS & CONCLUSIONS

Table 1. XRP LCP v2.4.0 with a seeded bug

Faults	T		V		I		A		Total	
baseline	35		0		0		3		35	
	ss	as	ss	as	ss	as	ss	as	ss	as
$c = 0, d = 1$	0	1	0	0	0	0	4	6	4	7
$c = 0, d = 2$	2	2	0	0	0	0	13	8	15	8
$c = 1, d = 0$	2	1	0	0	0	0	4	1	4	2
$c = 1, d = 1$	0	0	0	0	0	0	6	3	6	3
$c = 1, d = 2$	4	5	0	0	0	0	10	14	13	17
$c = 2, d = 0$	0	0	0	0	0	0	2	1	2	1
$c = 2, d = 1$	2	0	0	0	0	0	3	7	4	7
$c = 2, d = 2$	3	3	0	0	0	0	7	8	9	10

A modified version of the XRPL source code, where the threshold to validate proposals is set to 40% agreement instead of 80% agreement.

- Both **naive random testing** (baseline) and **ByzzFuzz** testing can uncover bugs in the protocol, particularly agreement violations in the seeded XRP LCP version.
- The ByzzFuzz search algorithm significantly improves testing efficiency, detecting more critical violations per unit time compared to the baseline method.
- Configurations with **c = 1, d = 2** and **c = 0, d = 2** consistently yield the most effective testing results.
- Further experiments are needed to determine which mutation scope, **small-scope** or **any-scope**, provides more efficient testing.

REFERENCES

[1] Schwartz, D., Youngs, N., & Britto, A. (2014). The Ripple protocol consensus algorithm (White Paper No. 8). Ripple Labs Inc. https://xrpl.org/whitepaper.pdf

[2] Winter, L. N., Busè, F., de Graaf, D., von Gleissenthall, K., & Kulahcioglu Ozkan, B. (2023). Randomized testing of Byzantine fault tolerant algorithms. *Proceedings of the ACM on Programming Languages, 7*(OOPSLA 1), 757–788. https://doi.org/10.1145/3622840