Testing Byzantine Fault Tolerant Algorithms

Evaluating the correctness of Tendermint protocol using ByzzFuzz

1. Background

- Consensus protocols run the world. They are the backbone for services like online banking, e-commerce etc.
- Testing is hard, but is necessary for ensuring that systems are safe
- Tendermint[1] is a very widely used protocol
- Tendermint has the theoretical guarantees, however implementations are still susceptible to bugs
- ByzzFuzz[2], a fuzzing-based testing approach, was used to evaluate the robustness of Tendermint. It injects structured faults in the schedules to simulate real-world conditions

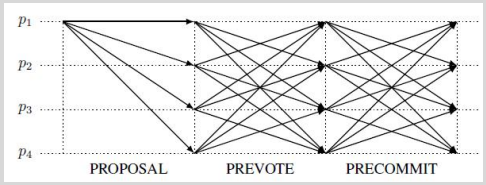

Figure 1: An overview of the Tendermint consensus protocol

2. Research Questions

- Can ByzzFuzz find any bugs in the implementation of Tendermint?
- How does the bug detection performance of ByzzFuzz compare to a baseline testing method that arbitrarily injects faults?
- How do small-scope and any-scope message mutations of ByzzFuzz compare?

3. Methodology

- Implement Tendermint protocol in ByzzBench framework
- Create structure aware mutations, namely small-scope and any-scope, which mutate values incrementally and arbitrarily respectively
- Test the protocol using the baseline testing approach
- Test the protocol using ByzzFuzz, while trying out different numbers of process and network faults

Message	Mutation
PROPOSAL(id, h, r, vr, v)	PROPOSAL(id, h', r, vr, v)
	PROPOSAL(id, h, r', vr, v)
PREVOTE(id, h, r, v)	PREVOTE(id, h', r, v)
	PREVOTE(id, h, r', v)
PRECOMMIT(id, h, r, v)	PRECOMMIT(id, h', r, v)
	PRECOMMIT(id, h, r', v)

Table 1: Structure aware mutations

- Test the implementation against known potential violation[2]

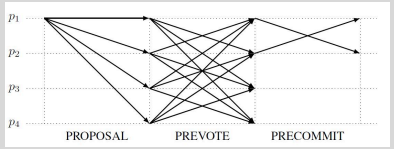

Figure 2: The known violation, as reported by Winter et al.[2]

References

[1] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on BFT consensus," CoRR, vol. abs/1807.04938, 2018.

[2] L. N. Winter, F. Buse, D. de Graaf, K. von Gleissenthall, and B. Kulahcioglu Ozkan, Randomized testing of byzantine fault tolerant algorithms," Proc. ACM Program. Lang., vol. 7, Apr. 2023.

4. Results

Faults	Liveness		Agreement		Total
baseline	0		0		2000
	SS	AS	SS	AS	
$\mathbf{P}=0,\mathbf{N}=1$	0	0	0	0	2000
$\mathbf{P}=0,\mathbf{N}=2$	0	0	0	0	2000
$\mathbf{P}=1,\mathbf{N}=0$	1	0	0	0	2000
$\mathbf{P}=1,\mathbf{N}=1$	1	0	0	0	2000
$\mathbf{P}=1,\mathbf{N}=2$	1	2	0	0	2000
$\mathbf{P}=2,\mathbf{N}=0$	4	0	0	0	2000
$\mathbf{P}=2,\mathbf{N}=1$	8	1	0	0	2000
$\mathbf{P}=2,\mathbf{N}=2$	4	2	0	0	2000

Table 2: Faults found by ByzzFuzz on the implementation fulfilling both synchronous message delivery and gossip protocol assumptions

D	M	Liveness		Agreement		Total
		SS	AS	SS	AS	
0	0	0	0	0	0	2000
0	1000	0	0	0	0	2000
0	2000	1	0	0	0	2000
1000	0	2	2	0	0	2000
1000	1000	7	2	0	0	2000
1000	2000	8	10	0	0	2000
2000	0	31	34	0	0	2000
2000	1000	35	37	0	0	2000
2000	2000	34	30	0	0	2000

Table 3: Faults found by the baseline testing approach

Faults	Liveness		Agreement		Total
	SS	AS	SS	AS	
$\mathbf{P}=0,\mathbf{N}=0$	0	0	0	0	2000
$\mathbf{P}=0,\mathbf{N}=1$	441	441	0	0	2000
$\mathbf{P}=0,\mathbf{N}=2$	669	701	0	0	2000
$\mathbf{P}=1,\mathbf{N}=0$	1	0	0	0	2000
$\mathbf{P}=1,\mathbf{N}=1$	397	420	0	0	2000
$\mathbf{P}=1,\mathbf{N}=2$	657	666	0	0	2000
$\mathbf{P}=2,\mathbf{N}=0$	5	0	0	0	2000
$\mathbf{P}=2,\mathbf{N}=1$	381	421	0	0	2000
$\mathbf{P}=2,\mathbf{N}=2$	644	685	0	0	2000

Table 4: Faults found by ByzzFuzz on the implementation fulfilling synchronous delivery of messages and using unreliable gossip protocol

Faults	Liveness		Agreement		Total
	SS	AS	SS	AS	
$\mathbf{P}=0,\mathbf{N}=0$	92	83	0	0	2000
$\mathbf{P}=0,\mathbf{N}=1$	81	98	0	0	2000
$\mathbf{P}=0,\mathbf{N}=2$	93	87	0	0	2000
$\mathbf{P}=1,\mathbf{N}=0$	90	79	0	0	2000
$\mathbf{P}=1,\mathbf{N}=1$	90	67	0	0	2000
$\mathbf{P}=1,\mathbf{N}=2$	73	100	0	0	2000
$\mathbf{P}=2,\mathbf{N}=0$	65	90	0	0	2000
$\mathbf{P}=2,\mathbf{N}=1$	66	77	0	0	2000
$\mathbf{P}=2,\mathbf{N}=2$	83	78	0	0	2000

Table 5: Faults found by ByzzFuzz on the implementation with asynchronous delivery of messages and using reliable gossip protocol

5. Conclusion

- ByzzFuzz successfully identified two bugs in the implementation of Tendermint
- ByzzFuzz performs better than the baseline approach
- Small-scope mutations perform better than any-scope mutations
- This implementation was resilient to the known potential violation
- Additionally an extension of the known vulnerability was found

6. Limitations

- Baseline should be configured to not violate Tendermint assumptions
- More mutations could be introduced
- An implementation making use of power voting could be introduced and tested to attempt to find more violations that stem from a Proof-of-Stake design
- Twins could be used to compare performance of ByzzFuzz to another BFT focused testing approach

Delft

Author: Antoni Nowakowski – a.f.nowakowski@tudelft.student.nl Supervisors: João Neto, Dr. Burcu Kulahcioglu Ozkan Affiliations: EEMCS Faculty Delft University of Technology