ConsenStress: A Framework to Torture Test Consensus Protocols

Pasindu Tennage*
EPFL
Switzerland

Shailesh Mishra
EPFL
Switzerland

Alberto Sonnino
Mysten Labs, UCL
United Kingdom

Lefteris Kokoris Kogias
Mysten Labs
Greece

Philipp Jovanovic
UCL
United Kingdom

Bryan Ford
EPFL
Switzerland

Abstract

Consensus protocols serve as the foundation for strongly consistent and fault-tolerant distributed systems. Despite the theoretical guarantees of safety and liveness, most consensus protocol implementations have robustness issues that often go unnoticed until a major failure occurs in production, causing significant financial losses.

We propose ConsenStress, a novel black-box testing framework tailored for detecting robustness issues in consensus protocols. Unlike existing frameworks that only allow testing of one or a few protocols, ConsenStress enables seamless testing of unmodified binaries of arbitrary consensus protocols. ConsenStress introduces a novel attack "interface" that allows users to write complex attack scenarios using a novel high-level API, eliminating the need to handle low-level implementation details. Finally, ConsenStress includes more than 30 concrete attack implementations and supports 16 consensus protocol integrations, enabling a wide range of attacks across different types of consensus protocols.

Our preliminary evaluation identifies previously undiscovered robustness issues in existing consensus protocol implementations, demonstrating ConsenStress's capability to detect complex robustness issues in consensus protocol implementations.

Keywords: Consensus, Robustness, Testing.

1 Introduction

Consensus allows a group of distributed replicas to agree on a single history of commands, providing the foundation for fault-tolerant systems [4]. Consensus protocols, however, experience outages, which contradict the liveness guarantees stated in their formal specifications. These outages are often caused by protocol implementation issues, such as deviations from the protocol specification, optimizations that focus only on the failure-free case while neglecting the correct implementation of subtle features like view change subroutines [21] and synchronizers [15]. These outages have caused significant problems, as seen in the Cloudflare incident [10], where a protocol bug in the Raft [23] implementation led to a cloud outage affecting many systems, and in the Solana blockchain [5], where failures resulted in over 150 hours of downtime, causing substantial financial losses.

Software testing has long held promise to detect robustness issues. Existing methods for detecting robustness issues, black-box and white-box testing, have several limitations that prevent them from effectively identifying robustness issues. First, black-box approaches, such as Jepsen[14], often have a limited set of tests that do not cover the full range of scenarios a consensus protocol may encounter and therefore, cannot detect subtle robustness problems in consensus algorithms. Second, white-box testing approaches [18, 32] often require programmers to write specifications in machine-proving languages, which limits its widespread adoption. Moreover, white-box approaches face state space explosion due to the many execution paths in complex consensus protocols, limiting the scope of testing. Due to these limitations, effectively testing the robustness of consensus protocol implementations remains an open research problem.

This paper proposes **ConsenStress**, a novel framework for detecting robustness issues in consensus protocol implementations. ConsenStress supports the seamless integration of any consensus protocol, making it a generic testing framework. ConsenStress enables testing of unmodified consensus protocol binaries without requiring any changes to the source code. ConsenStress introduces a novel attack interface that allows programmers to define arbitrary testing strategies using a high-level API.

Contributions

- We present the design of ConsenStress, a novel, generic, and extendable framework for evaluating the robustness of unmodified consensus protocol binaries.
- We design and implement 30 different attack scenarios, covering a wide range of failures observed in practical distributed system deployments.
- We provide a prototype of ConsenStress in Go [20] and evaluated 16 consensus protocols using a ConsenStress deployment running on Sphere-Testbed [26].
- We identify more than 10 previously undiscovered robustness issues in existing public consensus protocol implementations, demonstrating the effectiveness of ConsenStress.

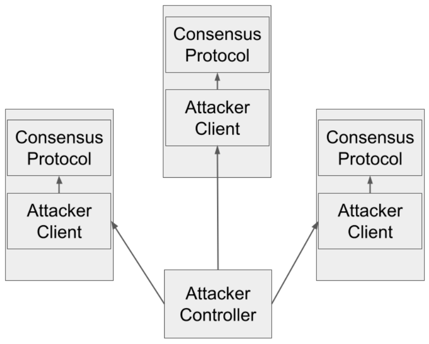

Figure 1. ConsenStress architecture.

2 ConsenStress design

ConsenStress is a distributed system, as shown in Fig. 1. ConsenStress employs "attacker client"s that are collocated with the consensus replica processes. Attacker client runs as a separate process and interacts with the consensus process solely through system interrupts and network manipulation calls. The "attacker controller" node runs the attack script and sends timely actions to attacker clients, which enforce them on the corresponding consensus protocol process. This design enables ConsenStress to remain oblivious to the protocol under test and allows seamless integration of unmodified consensus protocol binaries.

2.1 ConsenStress attack interface

ConsenStress abstracts the distributed consensus protocol deployment as a graph, with consensus replicas as vertices and network links (TCP/UDP connections) as edges. ConsenStress provides an "attack node interface" and an "attack link interface", which offer methods to manipulate both consensus replica processes and the network links connecting consensus processes, respectively. These interfaces enable users to write custom testing scenarios, foregoing low-level attack implementation details.

2.1.1 ConsenStress concrete attacks. As a first step in identifying factors affecting the robustness of consensus protocol implementations, we conducted a detailed study of network and node failures in the cloud [1, 2, 8, 25]. Based on this analysis, we identified key scenarios that affect the robustness of consensus protocols: (1) changing link properties (delay, bandwidth, jitter), (2) network partitions, (3) stragglers (slow nodes), (4) timeout and failure detector errors, and (5) node crashes. While we observed other factors influencing robustness, we determined that these five categories accurately summarize the most relevant root causes affecting the robustness of consensus protocols.

Using the ConsenStress attack interface, we implemented these attack scenarios and their variations. On average, each attack required only 20 lines of code, demonstrating that ConsenStress enables easy implementation of new attacks. At the time of writing this extended abstract, we have implemented over 30 concrete attack scenarios, uncovering more than 10 previously unknown issues in consensus protocol implementations.

2.2 Seamless integration of consensus protocols

ConsenStress enables the rapid integration of unmodified consensus protocol binaries written in any programming language. To integrate a new protocol into ConsenStress, a user must implement the following three methods.

```
copyConsensus(nodes)
bootstrap(nodes [], duration)
extractOptions() options
```

"copyConsensus" defines how the protocol is copied to each remote node, including all consensus-specific configuration files. "bootstrap" defines how the protocol should be started in n replicas, and "extractOptions" specifies how to interpret the logs generated by the protocol during execution. Currently, ConsenStress readily integrates 16 different consensus protocols, both from the crash fault tolerant domain (Raft [23], Multi Paxos [16], Baxos [28], Rabia [24], SADL-RACS [30], EPaxos [22, 31], Mencius [19], Generalized Paxos [17], QuePaxa [29], ETCD Raft [7], and ZooKeeper [11]) and byzantine fault tolerant domain (Mahi-Mahi [12], Codial-Miners [13], Mysticeti [3], Jolteon [9], HotStuff [33], Tusk [6], Bullshark [27]). These implementations include industry-used protocols (ZooKeeper [11] and ETCD [7]), publicly deployed blockchains (HotStuff [33], Mysticeti [3], Narwhal [6]), and academic prototypes. On average, ConsenStress requires approximately 250 lines of code to integrate a given consensus protocol, confirming its seamless integration capability.

3 Evaluation and Future Work

We implemented ConsenStress in Go [20] using approximately 3,000 lines of code. Our preliminary evaluation revealed several robustness issues across multiple consensus protocol implementations, including (i) a leader election problem in Raft [23], (ii) high bandwidth overhead and eventual execution halt in DAG-based protocols under stragglers and high-delay links [3, 12], (iii) high commit delay in HotStuff [33] even with just a single replica crash, and (iv) complete loss of liveness in Rabia [24] when the network topology is asymmetrical. In the future, we plan to extend our evaluation to cover all 16 consensus protocols.

References

[1] Michael Alicea and Izzat Alsmadi. 2021. Misconfiguration in firewalls and network access controls: Literature review. *Future Internet* 13, 11 (2021), 283.

[2] Ahmed Alquraan, Hatem Takruri, Mohammed Alfatafta, and Samer Al-Kiswany. 2018. An analysis of {Network-Partitioning} failures in cloud systems. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 51–68.

[3] Kushal Babel, Andrey Chursin, George Danezis, Lefteris Kokoris-Kogias, and Alberto Sonnino. 2024. Mysticeti: Low-Latency DAG Consensus with Fast Commit Path. arXiv:2310.14821 [cs.DC]

[4] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. 2011. *Introduction to Reliable and Secure Distributed Programming.* Springer Science and Business Media.

[5] CryptoManiaks. 2025. *Solana Outage: A List of Failures on the SOL Blockchain Mainnet.* https://cryptomaniaks.com/crypto-news/solana-outage-list-failures-sol-blockchain-mainnet Accessed: 2025-02-03.

[6] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022.* ACM, 34–50.

[7] etcd-io. 2025. etcd-io Raft: An implementation of the Raft consensus algorithm. https://github.com/etcd-io/raft Accessed: 2025-02-04.

[8] Peter Garraghan, Renyu Yang, Zhenyu Wen, Alexander Romanovsky, Jie Xu, Rajkumar Buyya, and Rajiv Ranjan. 2018. Emergent failures: Rethinking cloud reliability at scale. *IEEE Cloud Computing* 5, 5 (2018), 12–21.

[9] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. 2022. Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback. In *26th International Conference on Financial Cryptography and Data Security: (FC)* (Grenada). Springer, 296–315.

[10] Ittai Abraham Heidi Howard. 2020. Raft does not Guarantee Liveness in the face of Network Faults. https://decentralizedthoughts.github.io/2020-12-12-raft-liveness-full-omission/.

[11] Patrick Hunt, Mahadev Konar, Flavio P Junqueira, and Benjamin Reed. 2010. {ZooKeeper}: Wait-free Coordination for Internet-scale Systems. In *2010 USENIX Annual Technical Conference (USENIX ATC 10)* (Boston).

[12] Philipp Jovanovic, Lefteris Kokoris Kogias, Bryan Kumara, Alberto Sonnino, Pasindu Tennage, and Igor Zablotchi. 2024. Mahi-Mahi: Low-Latency Asynchronous BFT DAG-Based Consensus. *arXiv preprint arXiv:2410.08670* (2024).

[13] Idit Keidar, Oded Naor, Ouri Poupko, and Ehud Shapiro. 2023. Cordial Miners: Fast and Efficient Consensus for Every Eventuality. In *37th International Symposium on Distributed Computing (DISC 2023).*

[14] Kyle Kingsbury. 2013–2025. Jepsen: Distributed Systems Safety Research. https://jepsen.io/. Accessed: 2025-02-04.

[15] Mysten Labs. 2024. Mysticeti: Low-latency dag consensus with fast commit path. https://github.com/asonnino/mysticeti.

[16] Leslie Lamport. 2001. Paxos Made Simple. *ACM SIGACT News (Distributed Computing Column) 32, 4* 32 (12 2001), 51–58.

[17] Leslie Lamport. 2005. Generalized Consensus and Paxos. (2005).

[18] Jeffrey F Lukman, Huan Ke, Cesar A Stuardo, Riza O Suminto, Daniar H Kurniawan, Dikaimin Simon, Satria Priambada, Chen Tian, Feng Ye, Tanakorn Leesatapornwongsa, et al. 2019. Flymc: Highly scalable testing of complex interleavings in distributed systems. In *Proceedings of the Fourteenth EuroSys Conference 2019.* 1–16.

[19] Yanhua Mao, Flavio Junqueira, and Keith Marzullo. 2008. Mencius: Building Efficient Replicated State Machines for WANs. In *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI 08)* (San Diego).

[20] Jeff Meyerson. 2014. The Go programming language. *IEEE Software* 31, 5 (2014), 104–104.

[21] Iulian Moraru, David G Andersen, and Michael Kaminsky. 2013. EPaxos go-lang. https://github.com/efficient/epaxos/.

[22] Iulian Moraru, David G Andersen, and Michael Kaminsky. 2013. There Is More Consensus in Egalitarian Parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (Koblenz , Germany). 358–372.

[23] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *2014 USENIX Annual Technical Conference ATC14)* (Philadelphia). 305–319.

[24] Haochen Pan, Jesse Tuglu, Neo Zhou, Tianshu Wang, Yicheng Shen, Xiong Zheng, Joseph Tassarotti, Lewis Tseng, and Roberto Palmieri. 2021. Rabia: Simplifying State-Machine Replication Through Randomization. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event Germany). 472–487.

[25] Rahul Potharaju and Navendu Jain. 2013. When the network crumbles: An empirical study of cloud network failures and their impact on services. In *Proceedings of the 4th annual Symposium on Cloud Computing.* 1–17.

[26] SPHERE Project. 2025. SPHERE Merge Portal. https://launch.sphere-testbed.net/ Accessed: 2025-02-04.

[27] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. 2022. Bullshark: DAG BFT Protocols Made Practical. In *CCS '22: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security.*

[28] Pasindu Tennage, Cristina Basescu, Eleftherios Kokoris Kogias, Ewa Syta, Philipp Jovanovic, and Bryan Ford. 2022. Baxos: Backing off for Robust and Efficient Consensus. *arXiv preprint arXiv:2204.10934* (4 2022).

[29] Pasindu Tennage, Cristina Basescu, Lefteris Kokoris-Kogias, Ewa Syta, Philipp Jovanovic, Vero Estrada, and Bryan Ford. 2023. QuePaxa: Escaping the Tyranny of Timeouts in Consensus. *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)* (Oct. 2023).

[30] Pasindu Tennage, Antoine Desjardins, and Eleftherios Kokoris Kogias. 2022. Mandator and Sporades: Robust Wide-Area Consensus with Efficient Request Dissemination. *arXiv preprint arXiv:2209.06152* (2022).

[31] Sarah Tollman, Seo Jin Park, and John K Ousterhout. 2021. EPaxos Revisited. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI 21).* 613–632.

[32] Junfeng Yang, Tisheng Chen, Ming Wu, Zhilei Xu, Xuezheng Liu, Haoxiang Lin, Mao Yang, Fan Long, Lintao Zhang, and Lidong Zhou. 2009. MODIST: Transparent model checking of unmodified distributed systems. In *NSDI 2009.* 213–228.

[33] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing* (Toronto ON Canada). 347–356.