

FaultRay: In-Memory Cascade Simulation with Formal LTS Semantics and Multi-Layer Availability Limits

Yutaro Maeda
Independent Researcher
Chigasaki, Japan
contact@faultray.com

Abstract—Production chaos engineering tools inject real faults into running systems, creating risk for regulated operators. Classical reliability analysis (FTA, RBD) assumes component independence, which does not hold for cloud infrastructure where a single incident can simultaneously degrade multiple layers. We present FAULTRAY, a research prototype contributing: (1) a cascade propagation engine formalized as a Labeled Transition System (LTS) over a dependency graph, with proven termination and $O(|V|+|E|)$ complexity; and (2) an N -layer availability limit model composing categorical ceilings via $\min$ rather than the classical series-product. An illustrative backtest on 54 real-world cloud incidents reproduces known cascade paths. FAULTRAY is open source (Apache 2.0, PyPI).

I. PROBLEM AND MOTIVATION

Modern cloud infrastructure cascading failures—the 2021 Meta BGP outage, the 2017 AWS S3 incident, the 2022 Cloudflare BGP leak—demonstrate how a single fault propagates across dozens of dependent services within minutes. Two gaps remain in current approaches:

Gap 1: Production risk. Chaos engineering tools (Gremlin, Steadybit, AWS FIS, LitmusChaos) inject *real* faults. For regulated financial institutions under DORA (EU 2022/2554), touching production to discover weaknesses is often unacceptable.

Gap 2: Independence assumption. Classical Reliability Block Diagrams compose availabilities via series-product $A = \prod_i A_i$ [1], assuming independent failures. Cloud infrastructure violates this: a region outage simultaneously degrades compute, network, and storage layers.

Concurrent work by Krasnovsky [2] (ICSE-NIER 2026) proposes in-memory Monte-Carlo simulation on service-dependency graphs. FAULTRAY addresses the same positioning with a different technical approach: formal LTS semantics with proven properties and multi-layer availability decomposition.

II. APPROACH

A. Cascade Engine (LTS Formalization)

FAULTRAY models cascade propagation as a Labeled Transition System $\mathcal{M} = (S, s_0, Act, \rightarrow, F)$ over an infrastructure dependency graph $G = (C, E, dep, w, \tau, cb)$. Each state is a 4-tuple (H, L, T, V) : health map, latency map, elapsed time, and visited set.

Eight transition rules govern fault injection, propagation through typed dependencies ($//$), circuit breaker trips, timeout cascades, and termination. Proven properties include:

- **Termination:** the visited set V grows monotonically and is bounded by $|C|$.
- $O(|V|+|E|)$ **complexity:** each component is visited at most once; each edge is examined once.
- **Monotonicity:** health can only worsen during a run.
- **Bounded blast radius:** affected components are bounded by the reverse-reachable set from the fault origin.

B. N -Layer Availability Limit Model

Rather than a single $A = \text{MTBF}/(\text{MTBF}+\text{MTTR})$, FAULTRAY decomposes availability into five categorical layers:

$$A_{\text{sys}} = \min(A_{L1}, A_{L2}, A_{L3}, A_{L4}, A_{L5}) \quad (1)$$

where $L1$ =software (deploy, drift), $L2$ =hardware (MTBF/MTTR, redundancy), $L3$ =theoretical (packet loss, GC), $L4$ =operational (incident response), $L5$ =external SLA ($\prod a_i$). The $\min$ operator replaces the series-product because cloud layers exhibit *correlated* failure modes, making the weakest-link composition more appropriate than independent-product composition.

III. PRELIMINARY RESULTS

Backtest. We performed a post-hoc backtest against 54 real-world cloud incidents (2017–2024: AWS, GCP, Azure, Meta, Cloudflare, GitHub, and 16 others). For each, a topology graph was constructed from the public post-mortem and the documented root cause was injected. The cascade engine achieved $F_1 = 0.87$ (precision = 1.00, recall = 0.81, severity accuracy = 0.58). We stress this is *illustrative*, not predictive: topologies were constructed with knowledge of the outcome.

Scale. The engine processes a 50-component topology in <10 ms on commodity hardware (Apple M1). The implementation comprises 512 source files and 33,841 tests.

Positioning. Table I summarizes key differences from representative tools and concurrent work.

IV. STATUS AND FUTURE WORK

FAULTRAY is released as open source under Apache 2.0 on GitHub¹ and PyPI. A US provisional patent has been filed

¹<https://github.com/mattyopon/faultray>

TABLE I
POSITIONING OF FAULTRAY.

	FaultRay	Krasnovsky	Chaos tools
In-memory	✓	✓	×
Formal proofs	✓	×	×
N -layer avail.	✓	×	×
Monte-Carlo	×	✓	×
Real injection	×	×	✓
Open source	✓	✓	partial

(No. 64/010,200, 2026-03-19). A Zenodo preprint is available (DOI: 10.5281/zenodo.19139911).

Future work includes: (1) prospective validation on live infrastructure with blinded topologies; (2) integration with DeathStarBench [3] for quantitative comparison with Krasnovsky’s Monte-Carlo approach; (3) extending the N -layer model to capture temporal correlation between layers.

Disclaimer. FAULTRAY is a research prototype, not a certified compliance tool.

REFERENCES

- [1] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, 2nd ed. Wiley, 2001.
- [2] A. A. Krasnovsky, “Model Discovery and Graph Simulation: A Lightweight Gateway to Chaos Engineering,” in *Proc. ICSE-NIER*, 2026. arXiv:2506.11176.
- [3] Y. Gan *et al.*, “An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems,” in *Proc. ASPLOS*, 2019.