DISFI (Distributed Infrastructure Stress and Fault Injector) — User Manual

1. Overview

DISFI (Distributed Infrastructure Stress and Fault Injector) is a configurable fault-injection framework designed to induce controlled stress conditions and resource exhaustion in distributed, stateful systems. The tool generates sustained background telemetry traffic while selectively injecting pathological workload patterns, including connection storms, read/write amplification, intentional connection leaks, and CPU saturation. DISFI is specifically designed to trigger internal failure modes such as thread pool saturation, file descriptor exhaustion, request timeouts, and transient unavailability. All injected faults and execution parameters are explicitly logged, enabling precise temporal alignment between injected stress conditions and observed system anomalies. The framework supports reproducible experimental campaigns for anomaly detection, predictive maintenance, and reliability analysis in distributed databases and streaming platforms.

It is intended for:

- Testing predictive anomaly detection
- Producing labeled anomalies for ML pipelines
- Stress-testing observability dashboards (Prometheus, Grafana)
- Validation of alerting rules

2. Command Line Usage

```
python traffic_gen.py \
  --hosts 192.168.1.10,192.168.1.11 \
  --rate 200 \
  --threads 50 \
  --mode read \
  --duration-minutes 10 \
  --log logs/file.log
```

Mandatory parameters

Parameter	Meaning
`--hosts`	Comma-separated Cassandra IPs
`--mode`	read, write, mixed
`--threads`	Generator threads
`--rate`	Operations per second

Optional parameters

Optional	Meaning
`--duration-minutes`	Auto-stop timeout
`--storm`	Enable Connection Storm scenario
`--cpu-spike`	Enable artificial CPU saturation

Optional	Meaning
`--cluster-timeout`	ms timeout for connection
`--log`	Persistent JSON logging

If `duration-minutes` is not provided, the script **runs indefinitely**.

3. Observability and Logging

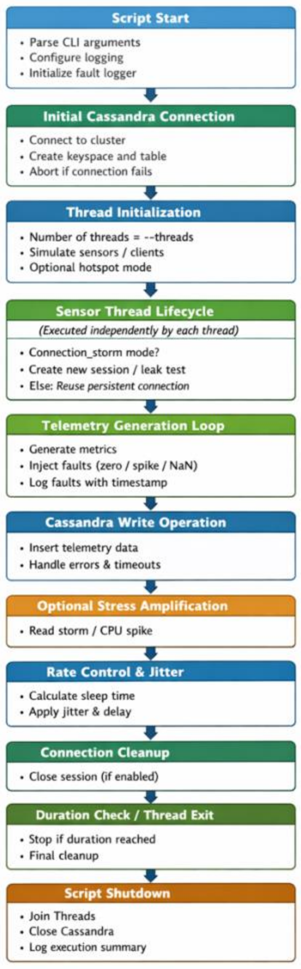
The generator writes extended diagnostic info:

- Start timestamp
- Stop timestamp
- Effective duration
- All CLI parameters
- Number of operations executed
- Exception count
- Performance counters

Useful for:

- auditability
- experiment reproducibility
- integration with notebooks or dashboards

Architecture and execution flow of the Cassandra stress traffic generator.


Script Start
• Parse CLI arguments
• Configure logging
• Initialize fault logger
Initial Cassandra Connection
• Connect to cluster
• Create keyspace and table
• Abort if connection fails
Thread Initialization
• Number of threads = --threads
• Simulate sensors / clients
• Optional hotspot mode
Sensor Thread Lifecycle
(Executed independently by each thread)
• Connection_storm mode?
• Create new session / leak test
• Else: Reuse persistent connection
Telemetry Generation Loop
• Generate metrics
• Inject faults (zero / spike / NaN)
• Log faults with timestamp
Cassandra Write Operation
• Insert telemetry data
• Handle errors & timeouts
Optional Stress Amplification
• Read storm / CPU spike
Rate Control & Jitter
• Calculate sleep time
• Apply jitter & delay
Connection Cleanup
• Close session (if enabled)
Duration Check / Thread Exit
• Stop if duration reached
• Final cleanup
Script Shutdown
• Join Threads
• Close Cassandra
• Log execution summary