Thoth: A Lightweight Framework for End-to-End Consumer IoT Rapid Testing

Salma Roshdy Aly[1], Sherif Saad[1] and Mohammad Mamun[2]
[1]*School of Computer Science, University of Windsor, Canada*
[2]*National Research Council, Canada*

Keywords: End-to-End Testing, Internet of Things (IoT), Consumer IoT, Automated Testing, Rapid Testing, Quantifiable Metrics, Cascading Failure Simulation.

Abstract: The rapid expansion of consumer IoT devices has increased the need for scalable, automated testing solutions. Manual methods are often slow, error-prone, and inadequate for capturing real-world IoT complexities. Existing frameworks typically lack comprehensiveness, quantifiable metrics, and support for cascading failure scenarios. This paper introduces Thoth, a lightweight, end-to-end IoT testing framework that addresses these limitations. Thoth enables holistic evaluation through integrated support for performance, reliability, recovery, security, and load testing. It also incorporates standardized metrics and real-time failure simulations, including cascading faults. We evaluated Thoth using eight test cases in a real-world health-monitoring setup involving a smartwatch, edge gateway, and cloud infrastructure. Key metrics—such as fault detection time, recovery speed, data loss, and energy usage—were logged and analyzed. Results show that Thoth detects faults in as little as 2.5 seconds, recovers in under 1 second, limits data loss to a few points, and maintains sub-1% energy overhead. These findings highlight its effectiveness for low-intrusion testing in resource-constrained environments. By combining scenario-driven design with reproducible, metrics-based evaluation, Thoth fills key gaps in IoT testing.

1 INTRODUCTION

The number of IoT devices continues to grow rapidly (Statista, 2024), embedding themselves in homes, businesses, and institutions as interconnected, heterogeneous systems. This evolution demands scalable and realistic testing methods (Minani et al., 2024a). Manual testing is no longer viable—it is slow, error-prone, and lacks the scalability to address network instability and security issues. These limitations highlight the need for automated testing frameworks tailored to IoT deployment realities (Vemula, 2024).

Recent work has proposed various automated frameworks focusing on robustness, scalability, and coverage (Minani et al., 2024a). However, most fall short in three critical areas for practical deployment: comprehensiveness (Minani et al., 2024b), quantifiability (Poess et al., 2018), and cascading failure testing (Xing, 2021). Many lack end-to-end validation, standardized metrics, and realistic (non-simulated) cascading failure scenarios, instead relying on simplified models (Dayalan et al., 2022).

This study addresses these gaps through three pillars:

1. **Comprehensiveness** – Evaluating frameworks by their support for diverse test types, including performance, reliability, recovery, load, and security.
2. **Quantifiability** – Capturing standardized performance metrics (e.g., execution time, resource usage) for realistic benchmarking (Poess et al., 2018).
3. **Cascading Failure Testing** – Incorporating real, non-simulated cascading failure scenarios for accurate analysis of failure propagation (Amal et al., 2023).

To this end, we present Thoth, a lightweight, end-to-end IoT testing framework designed to address these gaps. Thoth supports a wide range of test types, integrates quantifiable metrics, and simulates real cascading failures for practical system evaluation—all with low overhead.

This study explores the following research questions:

RQ1: How comprehensively does Thoth support different IoT test scenarios?

Aly, S. R., Saad, S. and Mamun, M.
Thoth: A Lightweight Framework for End-to-End Consumer IoT Rapid Testing.
DOI: 10.5220/0013650200003964
In *Proceedings of the 20th International Conference on Software Technologies (ICSOFT 2025)*, pages 430-437
ISBN: 978-989-758-757-3; ISSN: 2184-2833

RQ2: Does Thoth provide reproducible metrics for test case evaluation?

RQ3: How effectively can Thoth simulate and mitigate cascading failures?

RQ4: What is the trade-off between threshold-based and event-based failure detection?

RQ5: What overhead does Thoth introduce, particularly on battery usage?

Named after the Egyptian deity of wisdom, Thoth embodies rigorous evaluation and structured testing. It offers a novel approach to scalable, intelligent, and scenario-aware IoT testing.

This paper is structured as follows: Section 2 motivates Thoth through comparison with existing frameworks. Section 3 details the architecture. Section 4 introduces the test scenarios and design rationale. Section 5 covers implementation. Section 6 evaluates Thoth's performance, and Section 7 concludes the paper.

2 CASE FOR THOTH

This section presents three case studies to illustrate the motivation for Thoth. Our comparative analysis focuses on existing end-to-end IoT testing frameworks that are aligned with our work, have been applied to real-world IoT environments, and have been evaluated using standard evaluation metrics. Thoth aims to 1. Provide a comprehensive light-weight end–to-end IoT testing framework through including various test types. 2. Measure each test case scenario included by quantifiable metrics for transparent evaluation. 3. Analyze and mitigate cascading failure events in IoT environments and develop approaches to mitigate them.

2.1 Comprehensive End-to-End IoT Testing

The diverse and interconnected nature of Internet of Things (IoT) systems requires testing frameworks that evaluate system behavior holistically rather than in isolated components. However, as summarized in Table 1, existing frameworks tend to focus narrowly on specific aspects. For instance, (Siboni et al., 2016) and (Akhilesh et al., 2022) focus solely on security. Others like (Behnke et al., 2019) and (Kim et al., 2018) extend testing to network performance and interoperability, while (Dayalan et al., 2022) supports broader simulation scenarios including performance and failure events. Despite these contributions, none integrate many critical test types—performance, reliability, recovery, security, and load—within a single scalable framework. Thoth addresses this gap by unifying these dimensions under one lightweight, end-to-end architecture.

Table 1: Comparison of IoT frameworks based on comprehensiveness of test types.

Paper Reference	Comprehensive Test Types
(Siboni et al., 2016)	Security only
(Akhilesh et al., 2022)	Security only
(Kim et al., 2018)	Interoperability, Conformance, Semantics
(Dayalan et al., 2022)	Performance, Load, Failure Simulation, Network
(Behnke et al., 2019)	Performance, Resource Utilization

2.2 Quantifiable End-to-End IoT Testing

Quantifiable metrics are essential for assessing IoT testing frameworks (Poess et al., 2018), enabling transparent, reproducible comparisons through indicators like execution time, throughput, and recovery duration. As shown in Table 2, most existing frameworks lack such metrics or apply them inconsistently. For instance, (Siboni et al., 2016) and (Akhilesh et al., 2022) emphasize vulnerability detection—using CVSS in the latter—but omit performance metrics like runtime or load handling.

(Kim et al., 2018) addresses conformance and semantics but offers limited performance data, while (Dayalan et al., 2022) and (Behnke et al., 2019) provide latency or resource metrics in isolated cases. However, none deliver comprehensive, cross-category evaluations. This inconsistency hampers comparability and real-world applicability. Thoth overcomes this by applying standardized, quantifiable metrics—such as execution time, detection latency, and recovery duration—across all test types, enabling rigorous and objective evaluation.

Table 2: Comparison of IoT frameworks based on use of quantifiable evaluation metrics.

Paper Reference	Quantifiable Metrics
(Siboni et al., 2016)	Context-based only
(Akhilesh et al., 2022)	CVSS scoring
(Kim et al., 2018)	Limited metric focus
(Dayalan et al., 2022)	Latency & Overhead
(Behnke et al., 2019)	Partial execution time

2.3 Cascading Failure Simulation

Cascading failures—where the malfunction of one IoT device triggers a chain of failures—pose serious risks in real-world systems such as smart cities or healthcare. As summarized in Table 3, existing frameworks either ignore cascading scenarios entirely

or treat them passively. For instance, (Dayalan et al., 2022) simulates events like a fire-in-a-room but only measures failure impact without providing active mitigation. Frameworks such as (Akhilesh et al., 2022), (Kim et al., 2018), and (Behnke et al., 2019) do not address cascading failure at all. Thoth fills this critical gap by not only simulating cascading failures but also implementing automated mitigation strategies, including failure isolation and adaptive recovery, thereby improving system resilience and operational continuity.

Table 3: Comparison of IoT frameworks based on cascading failure testing support.

Paper Reference	Cascading Failure Testing
(Siboni et al., 2016)	None
(Akhilesh et al., 2022)	None
(Kim et al., 2018)	None
(Dayalan et al., 2022)	Failure simulation only
(Behnke et al., 2019)	None

3 ARCHITECTURE

In this section, we provide the rationale behind the design of each layer in the architecture, as depicted in Figure 1.

The proposed architecture is a modular, pluggable framework for end-to-end IoT testing, designed to adapt across diverse deployments. It abstracts critical components into five primary layers to support extensibility, reusability, and interoperability across different IoT ecosystems. These layers include the sensing layer, which standardizes device integration; the gateway layer, which manages multi-protocol data transmission; the storage layer, offering flexible data persistence options; the processing layer, which supports various testing methodologies such as anomaly detection; and the application layer, which oversees alerts, notifications, and failure handling. This modular design allows components to be seamlessly integrated, replaced, or extended without altering the core system.

Sensing Layer: Provides an abstract interface for heterogeneous devices (e.g., heart rate, motion, environmental sensors), allowing seamless integration without altering core logic despite hardware-specific adjustments.

Gateway Layer: Bridges sensing and downstream layers using protocols like Bluetooth, WiFi, and HTTP APIs, ensuring adaptability to diverse network conditions and deployment scenarios.

Storage Layer: Supports cloud (e.g., Google Cloud, AWS S3) and local (e.g., SQLite, MongoDB) backends via a unified interface, enabling backend swaps without impacting upper layers.

Processing Layer: Executes test strategies (e.g., anomaly detection) using cloud and edge platforms. It accommodates various methodologies while maintaining modularity and scalability.

Application Layer: Handles failure mitigation, alerts, and notifications through email, SMS, dashboards, or auto-recovery. Its event-driven design supports dynamic logic updates.

The modular, pluggable architecture enables flexible integration of new devices, protocols, and testing strategies without changing core logic. By abstracting each component, the framework ensures reusability, scalability, and adaptability across diverse IoT domains, while maintaining consistency and supporting rapid experimentation.

4 THOTH DESIGN

In this section, we discuss Thoth's design details that we followed for the implementation. To be able to fulfill these design objects, we explore test case scenarios included in Thoth.

4.1 Test Case Scenarios

This subsection outlines the test scenarios in Thoth, each representing a distinct failure type to evaluate system behavior under varied fault conditions. Failures may occur independently or sequentially. For clarity, test cases are grouped by failure type and labeled hierarchically (e.g., TC2.1), supporting traceable analysis of detection, recovery, and resilience.

4.1.1 Standard Operation

TC1.1: No Failure. Represents normal system behavior. Timestamped sensor data is logged locally before cloud upload to ensure robustness. Once a threshold is reached, anomaly detection is performed, and responses are triggered as needed.

4.1.2 Infrastructure Failures

TC2.1: Gateway Crash. Evaluates detection and recovery from gateway failure based on activity thresholds. Recovery includes re-establishing connections and processing locally buffered data.

TC2.2: Cloud Crash. Assesses response to unresponsive cloud services. When inactivity exceeds a threshold, operations shift to a backup service, ensuring continued data processing and storage.

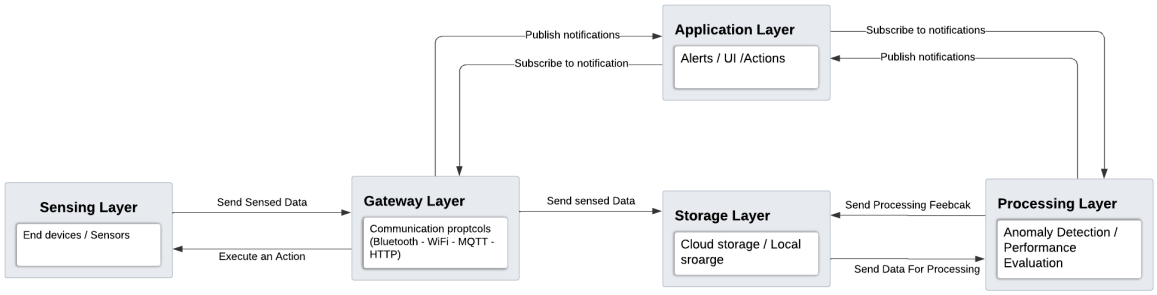

Figure 1: Thoth architecture.

4.1.3 Communication Failures

TC3.1: Watch-Gateway Bluetooth Drop. Tests the system's response to Bluetooth disconnection. Recovery involves reconnection attempts and alerting, followed by fallback strategies if the issue persists.

4.1.4 Sensor Failures

TC4.1: No Data Received. Monitors for missing data. If no input is received beyond a set threshold, the system attempts to reset the sensor and issues alerts.

TC4.2: Erroneous Data Received. Detects incorrect readings using validation and anomaly detection. Upon detection, the system resets the sensor and applies corrective actions.

4.1.5 Security Failures

TC5.1: Unauthorized Gateway Access. Triggers alerts, access restrictions, and event logging upon failed authentication attempts.

4.1.6 Resource Constraints

TC6.1: Battery Consumption. Measures energy use under normal and intensive scenarios (TC3.1, TC4.1, TC4.2). TC5.1 is excluded as it blocks gateway access, preventing TC6.1 execution.

4.2 Comprehensive End-to-End IoT Testing

This subsection explains how our test cases achieve comprehensive end-to-end IoT testing. We incorporate various test types—performance, reliability, recovery, security, and load testing—to address the limited scope of prior work. This broad coverage ensures a more complete evaluation of IoT systems. Below, we briefly define each test type for reference.

Performance Testing: This test evaluates the system's speed, responsiveness, and stability under a given workload.

Reliability Testing: This test assesses the system's ability to function correctly over time without failure.

Recovery Testing: This test verifies how well the system recovers from failures, crashes, or unexpected interruptions.

Security Testing: This test ensures the system is protected against threats, vulnerabilities, and unauthorized access.

Load Testing: This test measures system performance under expected or peak load conditions to check for bottlenecks.

In Table 4, we list the test types included in every test case. Please note that we refer to each test case by its number.

Table 4: Test types included in every test case in our framework.

Test Type	1.1	2.1	2.2	3.1	4.1	4.2	5.1	6.1
Performance	✓	✓	✓	✓	✓	✓	✓	✓
Reliability	✓	✓	-	✓	-	-	-	-
Recovery	-	✓	✓	-	✓	✓	-	-
Security	-	-	-	-	-	-	✓	-
Load	-	-	-	-	-	-	-	✓

4.3 Quantifiable End-to-End IoT Testing

This section demonstrates how Thoth enables quantifiable IoT testing to support reproducible and transparent evaluation. Each test case is paired with relevant monitoring methods and metrics selected to reflect real performance and facilitate benchmarking. **TC1.1:** 1. **End-to-end latency (s)**: Time from watch to cloud and back. 2. **Transmission success rate (%)**: Successful transmissions over total attempts. 3. **Cloud upload success rate (%)**: Successful uploads over total attempts. 4. **Anomaly detection time**

(s): Time from file upload to detection result. 5. **Message delivery time (s)**: Time from alert creation to delivery.

TC2.1: 1. **Gateway failure detection time (s)**: Time to detect gateway outage. 2. **Recovery time after WiFi reconnection (s)**: Time to resume after WiFi recovery.

TC2.2: 1. **Service failover time (s)**: Time to switch to replica bucket. 2. **Primary service recovery time (s)**: Time to restore primary service. 3. **Alert notification latency (s)**: Time from fault detection to alert delivery.

TC3.1: 1. **Disconnection duration (s)**: Time Bluetooth remains disconnected. 2. **Reconnection attempts (count)**: Number of reconnection tries. 3. **Reconnection success rate (%)**: Successful reconnects over total attempts. 4. **Lost data points (count)**: Missed sensor readings during outage.

TC4.1: 1. **Time to detect missing data (s)**: Time to detect missing sensor updates. 2. **Time to reset after detection (s)**: Time to trigger sensor reset.

TC4.2: 1. **Error detection time (s)**: Time to detect erroneous data. 2. **System recovery time (s)**: Time to recover normal sensor behavior.

TC5.1: 1. **Unauthorized access detection time (s)**: Time to detect intrusion. 2. **Blocked attempts (count)**: Unauthorized attempts successfully blocked. 3. **Response execution time (s)**: Time to complete security measures.

TC6.1. 1. **Watch battery drain – normal vs. intensive (%)**: Battery usage under normal and stress modes. 2. **Gateway battery drain – normal vs. intensive (%)**: Gateway battery usage under different loads. 3. **Battery logging interval (s)**: Time between battery status recordings. Each test case includes time-based, count, or percentage metrics to enable realistic benchmarking. By embedding these into its design, Thoth provides transparent, reproducible, and quantifiable evaluation for real-time IoT testing.

4.4 Cascading Failure Simulation and Mitigation

This section shows how Thoth addresses a key limitation in existing frameworks by supporting cascading failure analysis. While not all test cases involve cascading faults, those that do are described, analyzed, and mitigated to evaluate system resilience under fault propagation.

TC2.1: A gateway fails to upload data to the cloud, causing the cloud layer to detect a missing update after five minutes. The Watchdog pub/sub triggers a recovery, instructing the gateway to reconnect and re-upload queued data. This minimizes data loss and improves reliability.

TC2.2: A simulated cloud function failure halts data processing. The gateway detects inactivity on the Anomaly Detection pub/sub and activates a failover, redirecting uploads to a replica bucket. This ensures continuous operation and prevents system failure.

TC3.1: A forced Bluetooth disconnection between the watch and gateway stops data transmission, leading to TC4.1. The gateway attempts reconnection and alerts the Watchdog pub/sub, enabling recovery tracking or escalation.

TC4.1: Triggered by TC3.1, this test covers the failure to receive sensor data. The gateway tries to reset the watch and logs the event via Watchdog, supporting further recovery if needed.

TC4.2: The sensor reports anomalous data (e.g., 200+ bpm), risking false alarms. The gateway detects this via anomaly detection, resets the watch, and publishes a failure alert to prevent faulty data propagation.

These scenarios demonstrate Thoth's ability to simulate, detect, and mitigate cascading failures, enhancing resilience in real-world IoT deployments.

5 THOTH IMPLEMENTATION

This section presents the implementation details of Thoth across all layers. The full implementation is open-source and available on GitHub (Aly, 2025).

Sensing Layer: We use the Bangle.js 2 smartwatch (Ltd, 2023) to collect heart rate and accelerometer data. It supports Bluetooth and allows custom JavaScript applications, enabling wireless data transmission to the gateway.

Gateway Layer: The gateway is a laptop running three key components: (1) a watch-gateway interface (HTML + JavaScript) for real-time data visualization and simulating failures (TC3.1, TC4.1, TC4.2, TC5.1), (2) an integrated-monitor (Python) for uploading data, monitoring logs, and publishing failure messages to the Watchdog pub/sub, and (3) a local storage directory used as a file queue for uploads. The interface is launched via Chromium (Google, 2025a), which also triggers the Python script for monitoring and metric computation (TC1.1, TC2.1, TC2.2).

Storage Layer: We use Google Cloud Storage (Google, 2025b) with two buckets—anomaly-data-bucket and its replica—both storing anomaly results under a dedicated "Results" directory. The replica ensures redundancy and failover.

Cloud Layer: This layer runs two serverless cloud functions (Google, 2025c), each triggered by uploads to its respective bucket. Both functions, writ-

ten in Python, perform anomaly detection and save results to the bucket, while publishing to the Anomaly Detection pub/sub.

Application Layer: Implemented using Google Cloud Pub/Sub (Google, 2025d), this layer uses two topics: Anomaly-detection (for cloud-based results) and Watchdog (for general and failure-related events), enabling asynchronous communication between layers.

6 EVALUATION

This section analyzes Thoth's performance across the test cases. Raw data is available in the GitHub repository (Aly, 2025). Table 5 presents the averaged results, followed by a comparative analysis of detection times, recovery durations, disconnection intervals, and data loss. These results directly address **RQ1** and **RQ2**, demonstrating Thoth's test coverage and reproducible metric generation.

6.1 Evaluation Methodology

Thoth was evaluated in a real-world IoT setup: a Bangle.js 2 smartwatch connected via Bluetooth to a laptop gateway, with cloud backend services (Google Cloud Functions and Pub/Sub), emulating a remote health monitoring use case.

Each test case was executed 10 times, with faults injected as follows:

- **TC2.1, TC2.2:** Gateway and cloud failures simulated via service disablement.
- **TC3.1:** Bluetooth manually toggled off during data transmission.
- **TC4.1, TC4.2:** Watch stopped sending or sent invalid heart rate data.
- **TC5.1:** Simulated unauthorized access.
- **TC6.1:** Intensive 15-minute sessions assessed battery impact.

Metrics were logged and exported as structured CSV files for analysis.

6.2 Application Context and Test Case Illustration

The tests simulate a smart home health monitoring scenario where the smartwatch transmits biometric data to a cloud pipeline via an edge gateway.

TC2.1 – Gateway Failure: WiFi loss is detected via missing uploads (48.6 s), followed by automated reconnection (0.76 s).

Table 5: Average metric results across all test cases.

Test Case	Key Metrics
TC1.1	Latency: 0.904 s Transmission success rate: 81.02% Upload success rate: 0.25%
TC2.1	Time to detect failure: 48.64 s Recovery time: 0.76 s
TC2.2	Time to detect failure: 40.74 s Failover time: 40.75 s Primary recovery time: 39.16 s Alert latency: 40.74 s
TC3.1	Disconnection time: 42.24 s Reconnects attemps: 9.8 Data packets lost: 9.8
TC4.1	Time to detect failure: 3.90 s Time to Reset: 2.64 s Data packets lost: 1
TC4.2	Time to detect failure: 2.52 s Recovery time: 2.59 s Data packets lost: 1
TC5.1	Time to detect failure: 0.04 s Blocked attempts number: 1.0 Response time: 1.46 s
TC6.1	Watch drain (normal): 0% Gateway drain (normal): 0.4% Watch drain (intensive): 0.3% Gateway drain (intensive): 1% Logging interval: 902.7 s

TC4.1 – Sensor Malfunction: The watch stops sending heart rate data. The gateway detects the issue in 3.9 s and resets the sensor in 2.6 s, minimizing data loss.

These results confirm Thoth's ability to handle diverse fault conditions with quantifiable metrics and demonstrate its effectiveness in real-world IoT deployments.

6.3 Detection and Recovery Times

A primary observation emerges when comparing detection times among infrastructure failures (TC2.1 and TC2.2) and sensor issues (TC4.1 and TC4.2). Specifically, gateway and cloud crashes rely on threshold-based detection logic, resulting in detection times on the order of tens of seconds (e.g., 48.641 s for the gateway crash and 40.731 s for the cloud crash). By contrast, anomalies at the sensor level—namely, absence of readings (3.896 s) or erroneous sensor data (2.523 s)—are discovered substantially faster because they use event-based checks on incoming data streams, as shown in Figure 2. This distinction underscores a key design trade-off in Thoth: threshold-based strategies prevent false alarms

but delay the overall detection of service-level failures, which in turn can exacerbate cascading failure effects.

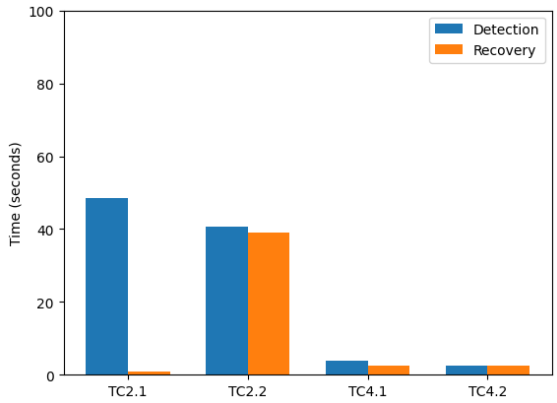

Figure 2: Detection vs. recovery times for TC2.1, TC2.2, TC4.1, and TC4.2.

Delayed detection significantly impacts cascading failures. In TC2.1, slow gateway failure detection delays data uploads and anomaly detection (TC2.2), disrupting system-wide monitoring. Similarly, cloud function failure in TC2.2 halts the anomaly pipeline, leading to undetected faults. These cases show how delayed infrastructure-level detection can trigger multi-layer disruptions.

This supports **RQ3**, confirming Thoth's ability to simulate and mitigate cascading failures using proactive detection and automated recovery (e.g., WiFi reconnection, cloud failover, sensor resets in TC2.1–TC6). These mechanisms reduced data loss and restored functionality with minimal manual input.

Recovery trends show that localized recoveries (e.g., WiFi: 0.762s; sensor resets: 2–3s) are rapid, while cloud failover (TC2.2: 40.751s) is slower due to multi-step redirection and function activation, increasing risk during that window.

TC3.1 illustrates this: a 42.237s detection delay causes a watch sensor failure (TC4.1), halting data flow and delaying TC2.2 detection—demonstrating full-layer cascade.

Overall, these findings highlight a resilience trade-off: threshold-based infrastructure detection risks propagation, while localized event-based detection curbs it. Thoth's hybrid model mitigates critical failures through self-healing and failover strategies, though optimizing cloud-level failover remains key for large-scale resilience.

6.4 Communication Reliability and Data Loss

Beyond temporal metrics, communication resilience varies notably between Bluetooth disruptions (TC3.1) and sensor malfunctions (TC4.1, TC4.2). In TC3.1, a forced Bluetooth disconnection causes 42.237s of downtime, averaging 9.8 reconnection attempts and resulting in 9.8 data points lost—highlighting the impact of prolonged communication loss on data continuity.

In contrast, sensor malfunctions trigger faster detection and intervention. Since the watch remains Bluetooth-connected, the gateway can quickly respond to missing data (TC4.1) or anomalies (TC4.2), minimizing data loss. This comparison underscores a key strength of Thoth: maintaining partial connectivity enables quicker recovery and better data preservation than total disconnection.

6.5 Security Mechanisms vs. Other Failures

Among all test scenarios, TC5.1 shows the fastest response, with near-instant detection at 0.040 s. Since authentication checks occur with each access request, Thoth blocks unauthorized logins with 100% success and initiates security measures within 1.5 s. Unlike threshold-based detection, these real-time, event-driven triggers enable rapid threat response, highlighting Thoth's emphasis on protecting consumer IoT devices from misuse and intrusion.

6.6 Energy Consumption Trade-Offs

TC6.1 shows that Thoth imposes minimal energy overhead. The watch consumes 0% in normal mode and only 0.3% under intensive tasks; the gateway draws 0.4% and 1% respectively. These results answer **RQ5**, confirming Thoth's suitability for resource-constrained IoT deployments.

Battery logging every 15 minutes (902.7 s) balances data granularity with efficiency. Shorter intervals offer finer analytics but slightly increase power use, especially on the gateway, while longer intervals conserve energy at the cost of less frequent updates. Overall, Thoth supports robust monitoring with minimal resource impact.

6.7 Comparison with Prior Work

Existing frameworks like (Dayalan et al., 2022), (Behnke et al., 2019), and (Kim et al., 2018) often

rely on simulations or protocol testing and lack standardized fault metrics such as detection latency or recovery time. Thoth addresses this gap by defining reproducible metrics across diverse real-world failure types, laying the groundwork for future benchmarking. Re-implementing scenarios from prior work within Thoth can enable direct, fair comparisons.

7 CONCLUSION

Thoth offers a lightweight, comprehensive framework for end-to-end IoT testing, integrating performance, reliability, recovery, security, and load testing into a unified system. It enables holistic evaluation across diverse fault scenarios and introduces standardized, quantifiable metrics to ensure objective and reproducible assessments.

A key contribution is Thoth's ability to simulate cascading failures, addressing resilience under multi-layer faults—an often overlooked aspect in existing frameworks. Real-world deployment shows that Thoth achieves fast detection and recovery, minimizes data loss, and maintains low resource overhead, validating its practicality for lightweight IoT testing.

Limitations and Scalability. The current evaluation is limited to a single edge device and controlled fault injections, without large-scale deployment. Comparisons with prior work remain qualitative due to the absence of standardized metrics in existing frameworks. While Thoth's modular, event-driven design supports scalability, further testing in distributed, high-load environments is needed.

Future Work. We plan to (1) integrate AI-assisted analysis to recommend optimizations based on failure patterns, and (2) expand support for large-scale, distributed IoT testing by improving coordination, communication, and fault isolation.

These extensions aim to make Thoth a scalable, intelligent IoT testing platform.

REFERENCES

Akhilesh, R., Bills, O., Chilamkurti, N., and Chowdhury, M. J. M. (2022). Automated penetration testing framework for smart-home-based iot devices. *Future Internet*, 14(10):276.

Aly, S. R. (2025). Thoth: E2e-iot-testing-framework.

Amal, G., Aïssaoui, F., Bolle, S., Boyer, F., and De Palma, N. (2023). *Solving the IoT Cascading Failure Dilemma Using a Semantic Multi-agent System*, pages 325–344.

Behnke, I., Thamsen, L., and Kao, O. (2019). Héctor: A framework for testing iot applications across heterogeneous edge and cloud testbeds. In *Proceedings of the 12th IEEE/ACM international conference on utility and cloud computing companion*, pages 15–20.

Dayalan, U. K., Fezeu, R. A., Salo, T. J., and Zhang, Z.-L. (2022). Kaala: scalable, end-to-end, iot system simulator. In *Proceedings of the ACM SIGCOMM Workshop on Networked Sensing Systems for a Sustainable Society*, pages 33–38.

Google (2025a). The chromium projects.

Google (2025b). Google cloud bucket.

Google (2025c). Google cloud function.

Google (2025d). Google cloud pub/sub.

Kim, H., Ahmad, A., Hwang, J., Baqa, H., Le Gall, F., Ortega, M. A. R., and Song, J. (2018). Iot-taas: Towards a prospective iot testing framework. *IEEE Access*, 6:15480–15493.

Ltd, P. (2023). Bangle.js 2 website.

Minani, J., Sabir, F., Moha, N., and Guéhéneuc, Y.-G. (2024a). A systematic review of iot systems testing: Objectives, approaches, tools, and challenges. *IEEE Transactions on Software Engineering*, PP:1–29.

Minani, J. B., Sabir, F., Moha, N., and Guéhéneuc, Y.-G. (2024b). A systematic review of iot systems testing: Objectives, approaches, tools, and challenges. *IEEE Transactions on Software Engineering*, 50(4):785–815.

Poess, M., Nambiar, R., Kulkarni, K., Narasimhadevara, C., Rabl, T., and Jacobsen, H.-A. (2018). Analysis of tpcx-iot: The first industry standard benchmark for iot gateway systems. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1519–1530.

Siboni, S., Shabtai, A., Tippenhauer, N. O., Lee, J., and Elovici, Y. (2016). Advanced security testbed framework for wearable iot devices. *ACM Transactions on Internet Technology (TOIT)*, 16(4):1–25.

Statista (2024). Internet of things (iot) connected devices worldwide 2019–2030. Accessed: 2025-03-27.

Vemula, S. (2024). Exploring challenges and opportunities in test automation for iot devices and systems. *International journal of computer engineering and technology*, 15:39–52.

Xing, L. (2021). Cascading failures in internet of things: Review and perspectives on reliability and resilience. *IEEE Internet of Things Journal*, 8(1):44–64.