Design and Implementation of a Chaos Injector to Improve Resilience in MQTT-Based IoT Systems

Pham Van Ha[1], Vu Viet Thang[1], Le Hong Vinh[2], Nguyen Phuc Quynh Nhi[3], and Nguyen Thi Dieu Linh[1] *

[1] Hanoi University of Industry
{hapv,vuvietthang,nguyen.linh}@haui.edu.vn
[2] Lam Son High School for the Gifted, Vietnam
lhvblog@gmail.com
[3] Tanh Linh High School, Vietnam
nguyenphucqnhi@gmail.com

Abstract. The growing reliance on Internet of Things (IoT) systems for smart homes, cities, industries, and healthcare has made resilience a critical concern. MQTT, a lightweight messaging protocol, is widely adopted for these systems, but failures—particularly at the publisher level—can lead to cascading system disruptions. This paper presents the design and implementation of a chaos injector framework that mimics the concept of vaccination in biological systems by introducing controlled chaos into MQTT-based IoT systems. Our approach autonomously scans MQTT topics and performs replay attacks, simulating data failures at the publisher level to strengthen the overall resilience of the system. The chaos injector acts as a "vaccine" for IoT, enabling the system to recognize, respond to, and recover from real-world failures. By testing the tool in a real smart office deployment, we uncovered system weaknesses and identified the underlying causes. Through this work, we contribute to the development of more fault-tolerant and resilient MQTT infrastructures using chaos engineering..

Keywords: MQTT, IoT, Chaos Engineering, Failure Simulation, Chaos Injection

1 Introduction

MQTT (Message Queuing Telemetry Transport) has become a core communication protocol for modern IoT systems due to its lightweight, simple, and effective design, as well as the ability to work on resource-constrained devices [1]. As a result, MQTT has been used for a wide range of IoT systems, including smart homes, industrial automation, smart cities, and environmental monitoring, where reliable and scalable communication between devices is essential. However,

* Corresponding author

MQTT is also vulnerable to failure, where a misbehaving device can trigger a chain of malicious behavior of other devices and disrupt entire systems. Therefore, it is crucial to make MQTT-based IoT systems resilient to such failures.

Resilience refers to the ability of a system to maintain normal operations and recover quickly in the event of failure [2]. Ensuring resilience requires extensive tests under both normal and failure conditions. Traditional testing methods focus on predefined test cases that assume certain normal conditions, but are unable to handle unexpected failures. In contrast, chaos engineering [3] is a methodology that injects failures into a system to discover potential weaknesses (inject-and-learn) and has been successfully applied in large-scale commercial systems such as Netflix, Amazon Web Services, and other cloud-based and microservice architectures.

The objective of this work is to bring the concept of chaos engineering to IoT systems by proposing a Chaos Injector that simulates and injects a range of failure scenarios in MQTT-based IoT environments. The original idea of this paper is similar to how vaccines help the human body build immunity. Just as a vaccine introduces a harmless version of a virus to train the immune system to fight future infections, the chaos injector strengthens the resilience of IoT systems in the same manner.

The contributions of this paper include: (i) an analysis of common data failures in IoT devices and the simulation of such failures, (ii) a chaos engineering tool for MQTT that can simulate and inject various types of data failure and failure scenarios, and (iii) an evaluation of the proposed solution in a real-world deployment.

2 Background and Related Work

2.1 MQTT and IoT Resilience

MQTT is a publish/subscribe (Pub/Sub) messaging protocol that enables asynchronous communication between clients (devices). It uses a broker to distribute messages from publishers, who connect to the broker and publish messages to specific topics, to subscribers, who subscribe to topics of interest. As a result, publishers and subscribers do not need to be aware of each other's existence to operate, which enhances the system's flexibility and scalability.

- **Publisher** is an MQTT client that publishes a message to a topic. It plays the role of a data producers.
- **Subscriber** is also an MQTT clients that subscribes to certain topics in order to receive the data. It plays the roles of a data consumer.
  Note: An MQTT client can be both publisher and subscriber.
- **Broker** is a central node that maintain the topics, receives messages from publishers and distributes to all subscribed based on pre-registered topics.

The broker is often seen as the critical point for resilience and security, because of its central role in communication. Recent studies have focused primarily on securing the MQTT broker to mitigate threats such as denial-of-service

(DoS) attacks, unauthorized access, and broker overloads. For example, authors in [4] proposed a fault-tolerant MQTT architecture with redundant brokers to enhance resilience. Other works, such as in [5] have explored adaptive security mechanisms for broker protection. However, failures at MQTT clients, especially publishers, have not been extensively addressed. In a typical system, applications or services rely on data from publishers to create logic for tasks, like "If This, Then That" scenarios. If a publisher fails, it can trigger a chain reaction of faults across the entire IoT system. Therefore, applying chaos engineering to IoT devices that function as MQTT publishers is essential for enhancing the overall resilience of IoT the system.

2.2 Chaos Engineering

Chaos engineering has proven highly effective in improving the reliability of cloud-native systems [6]. This success has led to multiple efforts to extend chaos engineering practices to IoT environments. The following table provides a summary of tools that support chaos engineering for IoT systems.

Table 1. Non-Exhaustive List of Tools for Chaos Engineering

Tool	Layer	Host Faults	Network Faults	Application Faults	Works with
Chaos Toolkit [4]	Cloud	Yes	Yes	No	All cloud platforms
AWS FIS[5]	Cloud	Yes	Yes	Yes	AWS-Only
Azure Chaos Studio[6]	Cloud	Yes	Yes	Yes	Azure-Only
Gremlin[7]	Cloud	Yes	Yes	Yes	All coud platforms
Barebone MQTT Broker [7]	Edge	No	Yes	No	MQTT clients
MicroChaos [8]	IoT Device	Yes	Yes	No	RTOS only
IoT Simulator [9]	IoT Device	Yes	Yes	No	ECHONET Lite Protocol
Chaos injector	**IoT Device**	**Yes**	**Yes**	**Yes**	**All MQTT brokers**

Overall, most tools focus on providing Failure as a Service (FaaS) at the cloud layer. Major cloud service providers offer tools that allow system developers using these cloud services to perform chaos engineering on their systems. The work in [7] introduces a broker that simulates failure scenarios, such as injecting delays or randomly dropping incoming and outgoing messages.

In contrast, the main focus of this work is at the device layer. The work in [8] focuses on chaos engineering for a specific real-time operating system (RTOS).

[4] https://chaostoolkit.org/
[5] https://aws.amazon.com/fis/
[6] https://azure.microsoft.com/en-us/products/chaos-studio
[7] https://www.gremlin.com/chaos-engineering

On the other hand, the approach in [9] is operating system-independent, which is a significant advantage. However, it is tied to the ECHONET Lite protocol[10], a protocol mainly used in Japanese smart homes only. The **chaos injector** tool developed in this work will be independent of any operating system and will utilize the widely used MQTT protocol, making it compatible with a broad range of IoT use cases.

3 Proposed Solution: Chaos Injector Tool

In IoT systems that use the publish-subscribe paradigm (e.g., MQTT, Robot Operating System), services consume data published by others to perform tasks, activate actuators, or publish new data to relevant topics for further processing by other services. Without safeguards, a single faulty data point can trigger a chain of failures across the system. The primary goal of the proposed tool is to scan all active topics and simulate replay attacks with chaotic data. This allows developers to conduct chaos engineering for their systems. Similar to how a vaccine strengthens the immune system, this tool injects chaos to enhance the resilience and realiability of MQTT-based IoT systems.

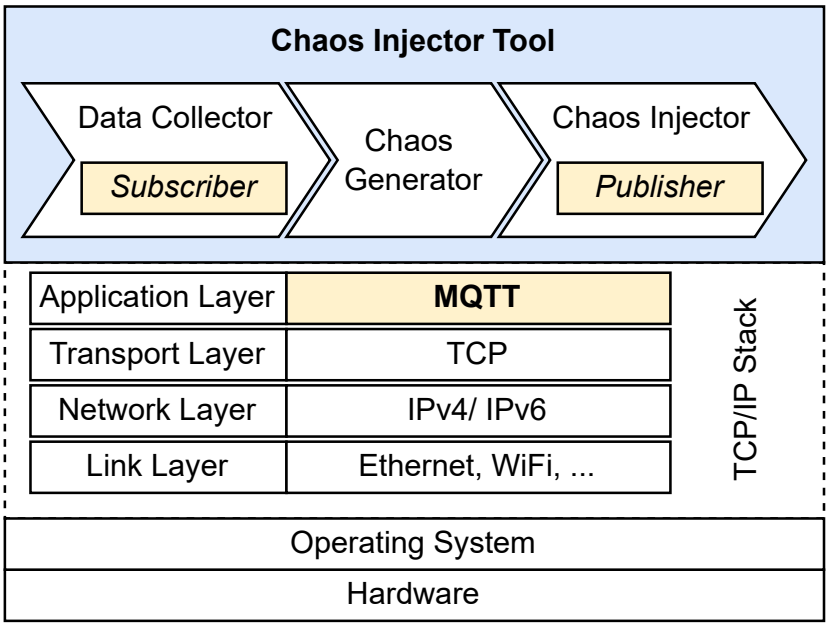

Fig. 1. Overall architecture of the Chaos Injector Tool. The tool is built on top of the TCP/IP protocol stack, making it independent of both the operating system and hardware

The overview of the main components of the chaos injector, as shown in Fig. 1, includes:

- **Data Collector** implements an MQTT subscriber that subscribes to all topics on the broker and keeps track of the following information
  - `Active topic list`: The list of active topics helps in targeting which topics are currently being used to ensure that the tool works only on the most relevant areas.
  - `Data published to each topic`: By keeping a record of the data being published to each topic, we can simulate realistic failure scenarios by introducing anomalies based on real data.
  - `Frequency of data published in each topic`: The frequency of messages on each topic allows us to introduce chaos that simulate high or low message traffic, testing the system's resilience under both normal and stressful conditions.
- **Chaos Generator** targets data failures that can result from a range of causes, such as technical malfunctions, human mistakes, software defects, cyberattacks, natural disasters or hardware problems. The selected failure scenarios [11] are detailed in Table 2.
- **Chaos Injector** implement an MQTT publisher that injects chaos into active topics according to generated failure scenarios.

The injector has been implemented using Python and the `mqtt_paho` library, which supports MQTT version 5.

Table 2. Data Faults and Their Generation Strategies

Fault	Description	Injection Method
Noise/ Outlier	A single, unexpected value that occurs frequently compared to normal measurements may be the result of hardware issues, such as an unstable sensor connection, or external influences, like an electromagnetic pulse	A pseudo-random value, generated randomly and added to the original measurement, appears at a pseudo-random frequency
Offset/ Rotation	A constant value is either added to or subtracted from the output	The sample is increased or decreased each time by a constant or random percentage of the actual value
Spike	Set of datapoints contains values differing from expectations, typically due to supply issues or connection failures	A peak in the value of data that rises and falls symmetrically over a number of samples randomly
Stuck at value	Values remain constant for a certain period, which may indicate a malfunction of the sensor.	The value of data remains constant and is based on the first sample obtained after the command is executed

4 Usecase and Evaluation

4.1 Experimental Setup

To validate the effectiveness of the chaos injector, we deployed the tool into a real smart environment that uses MQTT as the main communication protocol. The smart office is currently working normally and has not been subjected to a chaos engineering test before. The overview of the smart office is shown in Fig.2.

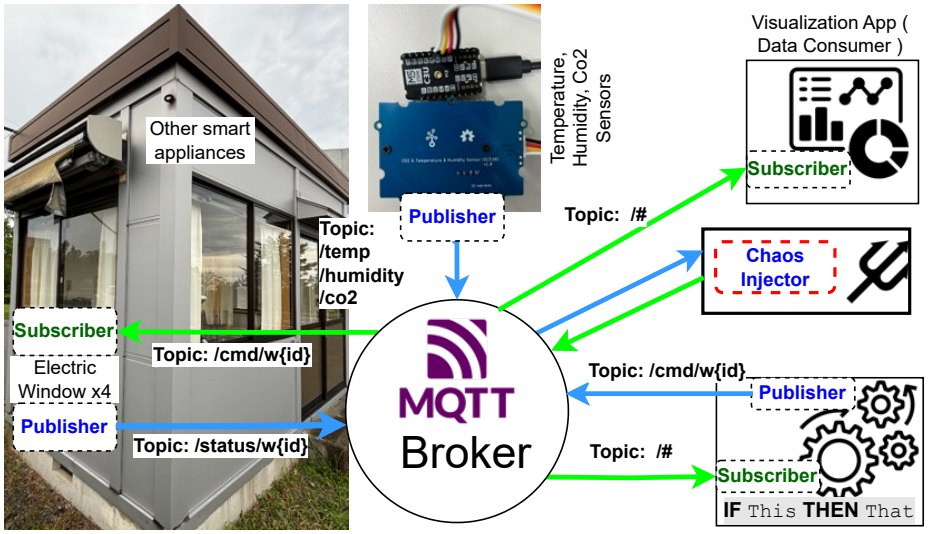

Fig. 2. Smart office experimental environment equipped with IoT devices communicating via MQTT

The office is equipped with environmental monitoring sensors that measure room temperature, humidity, and CO2 levels. These sensors publish data every 30 seconds to the topics `/office204/temp`, `/office204/humidity`, and `/office204/co2`, respectively. Additionally, there are four electrically operated windows that can be controlled via the topic `/office204/cmd/wid`, where `id` ranges from *0 to 3*. The windows also report state changes (e.g., from open to closed and vice versa) to the topic `/office204/status/wid`.

A dashboard application monitors the overall office status, displaying sensor readings and the current state of the windows. There is also an automation application that consumes sensor data (temperature, humidity, and CO2 levels) and sends commands to open or close the windows to maintain a comfortable environment for anyone in the room. For example, the system will open a window if CO2 levels are high to improve ventilation or close the windows to retain warmth in cooler conditions.

The chaos injector is deployed on a Raspberry Pi 4, configured with the address and credentials of the MQTT brokers used by the smart office. Although the broker serves other offices, only the topic `/office204/#` is targeted for chaos engineering in this experiment.

4.2 Experiment Result: Autonomous Chaos Generation

The strategies for generating data faults based on collected data from active topics are outlined in Table 2. For a data point **d** published to a topic **t** at at frequency **f**, two types of random `Noise/Outliers` can be selected, along

with four types of `Offset/Rotation` faults, one type of `Spike`, and one `Stuck At Value` fault. These faults are then injected into the broker at five different frequencies, ranging from slow to very fast. In total, 40 variations can be generated from a single data point published to any topic. In this experiment, **440** chaos scenarios (11 topics) were generated via the autonomous chaos generation methodology.

4.3 Chaos Injection

The results of the chaos injection experiment are summarized in Fig.3. For clearer visualization, one chaos scenario per sensor data was selected for display in the dashboard monitoring application. The CO2 level was simulated with both Spike and Offset faults, the humidity value with an outlier fault, and the temperature remained fixed at 26.2°C due to a 'stuck at value' fault.

This visualized result confirms the correct operation of the chaos injector. Despite the collected data being quite messy, the dashboard application functioned as expected. However, the noisy data could not be used in its raw form, leading to temporary data loss. To prevent similar issues in the future, developers could implement AI/ML models to detect anomalies and block abnormal data from being injected into the database.

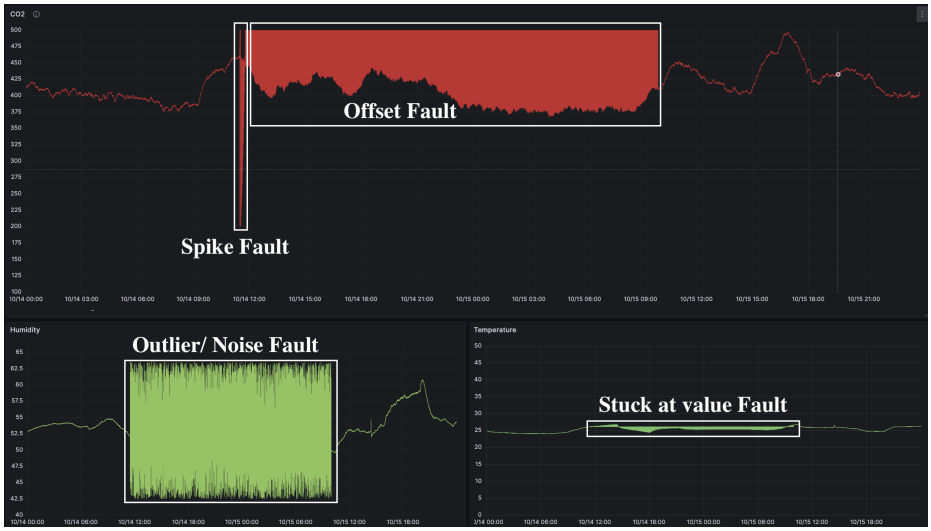

Fig. 3. Chaos injection results as displayed in the dashboard application

However, the automation application that consumes sensor data to control the windows was severely affected. Initially, the chaos injector failed to recognize the active topics related to window control. But after injecting chaos into the system with abnormally high CO2 levels, the automation application sent a

command to open the windows. This action was recorded by the chaos injector, causing the window controller to open and close the windows in a dangerously erratic manner, which could potentially damage the windows. Even after removing the topics related to window control from the tool, the automation application continued to perform the same harmful actions. By altering the CO2 data, especially using Spike data faults, the application repeated the erratic behavior, potentially causing damage to the windows.

In summary, data faults have a minor impact on sensors but can severely affect actuators. This highlights the need for system developers to prioritize safeguarding actuators in MQTT-based IoT systems.

5 Conclusion

This paper presents a novel approach to improving resilience in MQTT-based IoT systems through the design and implementation of a chaos injector tool. By autonomously scanning active topics and injecting replay attacks, the proposed solution mimics the concept of biological vaccination, exposing the system to controlled chaos to build its "immunity" against real-world malfunctions. The fault injector provides valuable insights into publisher-side vulnerabilities and contributes to the development of more robust, fault-tolerant IoT infrastructures. As IoT systems continue to grow in complexity and scale, our work offers a proactive approach to ensuring their resilience in the face of uncertainty.

References

1. Biswajeeban Mishra and Attila Kertesz. The use of mqtt in m2m and iot systems: A survey. *IEEE Access*, 8:201071–201086, 2020.
2. Christian Berger, Philipp Eichhammer, Hans P. Reiser, Jörg Domaschka, Franz J. Hauck, and Gerhard Habiger. A survey on resilience in the iot: Taxonomy, classification, and discussion of resilience mechanisms. *ACM Comput. Surv.*, 54(7), sep 2021.
3. Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. Chaos engineering. *IEEE Software*, 33(3):35–41, 2016.
4. Edoardo Longo and Alessandro E.C. Redondi. Design and implementation of an advanced mqtt broker for distributed pub/sub scenarios. *Computer Networks*, 224:109601, 2023.
5. Dan Dinculeană and Xiaochun Cheng. Vulnerabilities and limitations of mqtt protocol used between iot devices. *Applied Sciences*, 9(5), 2019.
6. Amro Al-Said Ahmad, Lamis F. Al-Qora'n, and Ahmad Zayed. Exploring the impact of chaos engineering with various user loads on cloud native applications: an exploratory empirical study. *Computing*, 106(7):2389–2425, May 2024.
7. Miguel Duarte, Joao Pedro Dias, Hugo Sereno Ferreira, and Andre Restivo. Evaluation of iot self-healing mechanisms using fault-injection in message brokers. In *Proceedings of the 4th International Workshop on Software Engineering Research and Practice for the IoT*, SERP4IoT '22, page 9–16, New York, NY, USA, 2023. Association for Computing Machinery.

8. Wojciech Kalka and Tomasz Szydlo. uchaos: Moving chaos engineering to iot devices. In Leonardo Franco, Clélia de Mulatier, Maciej Paszynski, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot, editors, *Computational Science – ICCS 2024*, pages 239–254, Cham, 2024. Springer Nature Switzerland.
9. Van Cu Pham, Yoshiki Makino, Khoa Pho, Yuto Lim, and Yasuo Tan. Iot area network simulator for network dataset generation. *Journal of Information Processing*, 28:668–678, 2020.
10. Van Cu Pham, Toan Nguyen-Mau, Marios Sioutis, and Yasuo Tan. Matter and echonet lite: Similarities, differences, and a bridge solution for interoperability. *Internet of Things*, 27:101265, 2024.
11. Ghaihab Hassan Adday, Shamala K. Subramaniam, Zuriati Ahmad Zukarnain, and Normalia Samian. Fault tolerance structures in wireless sensor networks (wsns): Survey, classification, and future directions. *Sensors*, 22(16), 2022.