C1	Title of software: IoT Kubernetes MQTT Network Simulator
C2	Developer: Raj Singh Gaur
C3	Year first available: 2025
C4	Software code repository:
	`https://github.com/RajSinghGaur/IoT-Kubernetes-MQTT-Network-Simulator`
C5	License: MIT
C6	Programming language: Python, Shell, YAML
C7	Requirements: Docker, Kubernetes, Python 3.x
C8	Documentation:
	`https://github.com/RajSinghGaur/IoT-Kubernetes-MQTT-Network-Simulator`
C9	Version: v1.0.2 (tagged release)

Table 1: Code metadata

IoT Kubernetes MQTT Network Simulator: A Cloud-Native Platform for Realistic IoT Device and Network Behavior Simulation

Raj Singh Gaur

Independent Researcher

Abstract

The IoT Kubernetes MQTT Network Simulator is an open-source, cloud-native platform for simulating large-scale IoT deployments with realistic network conditions. By leveraging Kubernetes for orchestration and MQTT for device communication, the simulator enables researchers and educators to model, test, and analyze IoT systems under diverse scenarios, including network latency, packet loss, and node failures. The platform supports rapid prototyping, reproducible experiments, and extensibility, filling a gap in accessible, scalable IoT simulation tools. Example results demonstrate scalability and protocol fidelity, making this tool valuable for both research and teaching.

Keywords: IoT, Kubernetes, MQTT, network simulation, containerization, distributed systems, Python

1. Motivation and Significance

The proliferation of IoT devices has introduced new challenges in designing, deploying, and managing large-scale, distributed systems. Real-world testing of IoT solutions is often costly, time-consuming, and difficult to reproduce. Existing simulators frequently lack support for realistic network behaviors or cloud-native deployment. The IoT Kubernetes MQTT Network Simulator addresses these gaps by providing a scalable, containerized environment for simulating thousands of IoT devices, brokers, and network conditions using industry-standard tools. This platform enables researchers, developers, and educators to evaluate IoT protocols, architectures, and fault tolerance strategies in a controlled, reproducible manner.

2. Software Description

2.1. Software Architecture

The simulator consists of several core components orchestrated by Kubernetes:

- **IoT Device Pods**: Each device is a Kubernetes pod running a Python application (`mqtt_device.py`) that connects to a shared MQTT broker. Devices publish periodic heartbeats, support custom messages, and expose REST endpoints for status and metrics.
- **MQTT Broker**: Deployed as a containerized Eclipse Mosquitto instance, configured via a Kubernetes ConfigMap (`mosquitto-configmap.yaml`).
- **Network Simulation**: Each device pod simulates network latency, packet loss, and device failures in software, with parameters set via environment variables in `k8s.yaml`.
- **Resource Constraints**: Pods are limited in CPU and memory to mimic real IoT hardware, as specified in the deployment YAML.
- **Experiment Automation**: Shell scripts (e.g., `run_experiment.sh`) automate deployment, timed execution, log collection, and cleanup.
- **Result Aggregation**: Logs and metrics are saved in timestamped folders under `results/` for analysis. Users can manually analyze these logs or use their own scripts for further processing.

A system architecture diagram is provided in Figure 1.

2.2. Software Functionalities

The main features of the simulator are:

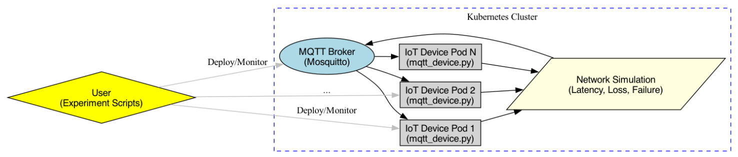

Figure 1: System architecture of the IoT Kubernetes MQTT Network Simulator. Each device pod simulates network conditions and communicates with a central MQTT broker.

- **Scalable Device Simulation**: Easily scale to dozens of pods (or more, depending on resources) by editing the `replicas` field in `k8s.yaml`.
- **Realistic Network and Failure Modeling**: Each device pod simulates message latency (10–500ms), packet loss (0–20%), and device failures (probabilistic downtime), with all parameters configurable.
- **Protocol Fidelity**: MQTT client inactivity is simulated by pausing the network loop, allowing broker-side disconnects and realistic reconnection behavior.
- **REST API and Metrics**: Each device exposes REST endpoints for status, publishing messages, and real-time metrics (sent, received, average latency, recent samples).
- **Automated Experimentation**: The `run_experiment.sh` script deploys resources, runs for a specified duration, collects logs, and cleans up, ensuring reproducibility.
- **Extensibility**: Users can modify device logic, broker configuration, or network parameters to model new scenarios or protocols.

2.3. Illustrative Examples

Example 1: Build, Deploy, and Run an Experiment

```
# Build and push the device image (replace <your-dockerhub-username>)
docker build -t <your-dockerhub-username>/iot-mqtt:latest .
docker push <your-dockerhub-username>/iot-mqtt:latest

# Deploy the MQTT broker
```

```
kubectl apply -f mqtt-broker.yaml

# Edit k8s.yaml to use your image, then deploy devices
kubectl apply -f k8s.yaml

# Run a timed experiment and collect logs
chmod +x run_experiment.sh
./run_experiment.sh 600  # Runs for 10 minutes
```

Listing 1: Build, push, and deploy the simulator, then run an experiment

Example 2: Interacting with Devices and Collecting Metrics

```
# Get a device pod name
kubectl get pods -l app=iot-device

# Port-forward to the pod
kubectl port-forward <pod-name> 5000:5000

# In another terminal, check status
curl http://localhost:5000/status

# Publish a custom message
curl -X POST -H "Content-Type: application/json" -d '{"message": "Hello
    network!"}' http://localhost:5000/publish

# Query device metrics
curl http://localhost:5000/metrics
```

Listing 2: Check device status, publish messages, and collect metrics

Example 3: Batch Metrics Collection

```
for pod in $(kubectl get pods -l app=iot-device -o
    jsonpath='{.items[*].metadata.name}'); do
  kubectl port-forward $pod 5000:5000 &
  sleep 2
  echo "$pod metrics:"
  curl -s http://localhost:5000/metrics
  kill %1
  sleep 1
done
```

Listing 3: Automate metrics collection from all device pods

Note: Users can analyze the collected logs in the `results/` *directory using their own scripts or tools.*

3. Impact

The IoT Kubernetes MQTT Network Simulator enables reproducible, scalable experiments for IoT research and education. It allows:

- Rapid prototyping and validation of IoT architectures and protocols.
- Reproducible experiments for academic research and publication.
- Training and education in cloud-native IoT deployment and testing.
- Extension to new protocols, device types, and network scenarios.

This platform lowers the barrier for realistic IoT experimentation, making it accessible to a wider community. It can be extended by integrating additional device models, brokers, or network emulation tools, and by connecting to real-world data sources or edge/cloud platforms.

4. Limitations and Future Work

While the simulator provides a flexible and reproducible platform, several limitations remain:

- **Scale:** Experiments are limited by local hardware and single-node Kubernetes by default. Larger-scale tests require a multi-node or cloud cluster.
- **Network Fidelity:** Network effects are simulated in software, not at the OS or physical network layer.
- **Device Heterogeneity:** All simulated devices use the same container image; real-world IoT deployments are more diverse.
- **Protocol Support:** Only MQTT is currently supported; other IoT protocols could be added.
- **Visualization:** No integrated dashboard for real-time monitoring or visualization is provided yet.

Future work includes scaling to larger clusters, supporting additional protocols (e.g., CoAP, HTTP), adding device heterogeneity, and integrating visualization tools for live monitoring and analysis.

5. Example Results

A typical experiment deploys 10 simulated IoT devices, each publishing periodic heartbeats and simulating network latency, packet loss, and device failures. Logs are collected for each device and the broker. Example summary metrics (extracted from device logs) are shown in Table **??**.

Exp.	**Nodes**	**Recv. (mean/min/max)**	**Avg Lat. (s)**	**Max Lat. (s)**
5	5	317/293/404	0.0023/0.0021/0.0025	0.058/0.011/0.096
10	10	809/595/1112	0.0023/0.0021/0.0025	0.080/0.017/0.104
20	20	1862/1319/2644	0.0028/0.0022/0.0031	0.120/0.106/0.188
50	50	4302/3064/6428	0.0038/0.0023/0.0050	0.120/0.095/0.196

Table 2: Scalability experiments: metrics vs. node count.

As shown in Table 2, the simulator delivers low-latency message transmission across all devices, with average latencies typically below 3 ms. The number of messages received per device is consistent, and the minimum and maximum latencies remain stable, even as the number of devices increases. This demonstrates the scalability and reliability of the platform for realistic IoT workloads.

Figure 2 further illustrates that as the number of simulated nodes increases from 5 to 50, the average message latency rises only modestly, remaining well below 5 ms even at the largest scale tested. This trend highlights the simulator's suitability for large-scale IoT studies, where maintaining low latency and high throughput is critical. The results confirm that the platform can be used to evaluate protocol performance and system behavior under realistic, scalable conditions.

Conflict of Interest

The author declares no conflict of interest.

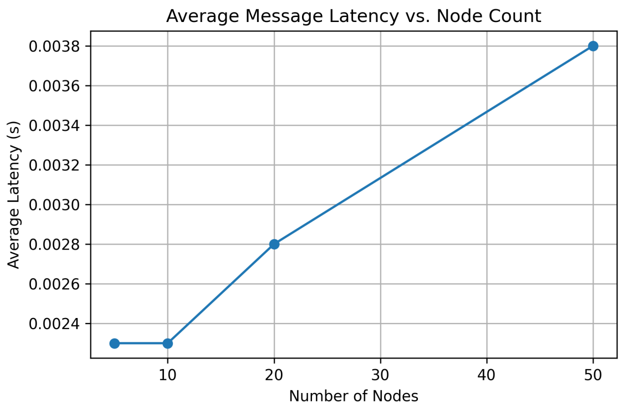

Figure 2: Average message latency as a function of simulated node count. The simulator demonstrates low-latency, scalable performance as the number of devices increases.

Acknowledgements

The author thanks the open-source community for tools and libraries used in this project and acknowledges the support of colleagues and beta testers who provided valuable feedback.

How to Cite

If you use this software in your research, please cite:

> Gaur, R. S. (2025). IoT Kubernetes MQTT Network Simulator: A Cloud-Native Platform for Realistic IoT Device and Network Behavior Simulation. *SoftwareX*. https://github.com/RajSinghGaur/IoT-Kubernetes-MQTT-Network-Simulator

References

[1] Eclipse Mosquitto, "Eclipse Mosquitto MQTT Broker," [Online]. Available: https://mosquitto.org/

[2] Kubernetes Authors, "Kubernetes: Production-Grade Container Orchestration," [Online]. Available: https://kubernetes.io/

[3] Eclipse Foundation, "Eclipse Paho MQTT Python Client," [Online]. Available: https://www.eclipse.org/paho/index.php?page=clients/python/index.php

[4] M. F. Usmani, "MQTT Protocol for the IoT," Int. J. Internet Things, vol. 12, no. 3, pp. 45-50, 2020.

[5] S. Cirani et al., "IoT-OAS: An OAuth-based Authorization Service Architecture for Secure Services in IoT Scenarios," IEEE Sensors J., vol. 15, no. 2, pp. 1224-1234, 2015.

[6] Minikube Authors, "Minikube: Run Kubernetes Locally," [Online]. Available: https://minikube.sigs.k8s.io/