IoT-Zoo: A Container-Based Framework for Heterogeneous IoT Device Profiles and Reproducible Traffic Capture

Vagner E. Quincozes[1], Diego Kreutz[2], Silvio E. Quincozes[2]

[1] Universidade Federal Fluminense - UFF

[2]AI Labs | PPGES, Universidade Federal do Pampa - UNIPAMPA

vequincozes@midiacom.uff.br,

{diegokreutz, silvioquincozes}@unipampa.edu.br

Abstract. *The validation of networking and security solutions for the Internet of Things (IoT) requires realistic and reproducible experimental data. However, existing platforms often achieve scalability by replicating a limited set of device types, which restricts profile diversity and fails to capture the heterogeneity of real-world IoT environments. In this paper, we present IoT-Zoo, a container-based testbed designed to support reproducible experimentation through heterogeneous, dataset-driven IoT device profiles. Built upon Containernet, IoT-Zoo automates the deployment of multi-domain scenarios and supports real application protocols such as MQTT and RTSP. The platform provides a single-command interface for environment provisioning and automated traffic capture (PCAP), enabling the generation of consistent traffic baselines and reducing the operational effort required to evaluate networking and security solutions.*

1. Introduction

Internet of Things (IoT) research increasingly depends on reproducible experimental environments to evaluate networking, management, and security solutions under realistic traffic conditions [Sah et al. 2025, De Keersmaeker et al. 2023]. However, building practical IoT testbeds remains challenging due to the inherent heterogeneity of real-world deployments, which vary significantly in terms of devices, application domains, and communication patterns [Lima and Pinto 2025, De Keersmaeker et al. 2023, Alex et al. 2023].

Most platforms address this challenge by scaling the number of emulated nodes, often reaching hundreds of devices. However, scale is frequently achieved by replicating only a few device types, which limits profile diversity and may underrepresent heterogeneous IoT behaviors (e.g., different sensing modalities, update rhythms, payload structures, and protocol stacks) [Mishra et al. 2025, Lima and Pinto 2025, Alex et al. 2023]. In practice, scale is easier to achieve than heterogeneity.

This gap is clearly reflected in existing IoT testbeds and traffic generation frameworks, as summarized in Table 1. While several platforms achieve large-scale deployments, they do so with limited device diversity. For instance, Gotham reports 100 emulated devices but relies on only 11 distinct profiles, replicated to reach scale [Sáez-de Cámara et al. 2023]. GothX extends this approach to 450 nodes, yet still inherits the same 11-profile structure [Poisson et al. 2024]. Similarly, IoT-Flock models 18 devices using only four basic sensor types [Ghazanfar et al. 2020], and even large-scale generators such as STGen reach up to 6,000 nodes with only five sensor categories [Islam et al. 2025]. Across the literature, scale is consistently achieved by replication, whereas diversity remains constrained to a small set of profiles.

Table 1. IoT testbeds and traffic generation frameworks.

Testbed / Project	Infrastructure	Diversity (Profiles & Sensors)	Reproducibility
IoT-Flock [Ghazanfar et al. 2020]	Virtual (Traffic Generator / GUI)	4 sensor types (temperature, humidity, motion, light)	High
Automated IoT Testbed [Waraga et al. 2020]	Physical	2 device types (1 camera and 1 smart bulb).	Low
MQTT DoS Testbed [Syed et al. 2020]	Physical / Virtual mixed	5 sensor types (motion, air quality, clock, temperature, gas).	Low
TON_IoT [Moustafa 2021]	Hybrid (SDN, NFV, Fog/Edge)	7 simulated sensors (via Node-RED) + 3 physical devices.	Medium
DoS/DDoS MQTT [Alatram et al. 2023]	Physical	13 sensor types (analog and digital sensors).	Low
ICSSIM [Dehlaghi-Ghadim et al. 2023]	Emulated / Physical	1 industrial process (bottle-filling factory with 2 PLCs).	High
Gotham [Sáez-de Cámara et al. 2023]	Emulated (GNS3 + Docker/VMs)	11 distinct profiles (e.g., smart home, motor, hydraulics)	High
GothX [Poisson et al. 2024]	Emulated (GNS3 + Docker)	11 distinct profiles based on Gotham	High
STGen [Islam et al. 2025]	Virtual (Central core / CLI)	5 sensor types (temperature, humidity, camera, GPS, switch)	High
Physical Smart Office [Farag et al. 2025]	Physical	3 sensor types (climate, sound, air) + 3 smart device types.	Low
IoMT Security [Zachos et al. 2025]	Physical (Medical focus)	1 sensor type (producing temperature and humidity).	Low
IoT-Zoo **Our Proposal**	Virtual / Hybrid-ready (Containernet & Docker)	**43 distinct profiles** (e.g., environmental, smart building, industrial, smart agriculture, human-centric, cameras)	High

Reproducibility is classified as **High** (software-based, automated deployment via open-source scripts/templates), **Medium** (requires robust enterprise virtual infrastructure), or **Low** (requires purchasing and assembling physical hardware).

This imbalance reveals a fundamental limitation: existing testbeds prioritize instance scale over device-level heterogeneity, underrepresenting the variability of real-world IoT ecosystems, which encompass diverse sensing modalities, temporal patterns, payload structures, and protocol stacks. As a result, many experimental environments fail to capture the richness and variability required for realistic evaluation of networking and security solutions.

We address this limitation with IoT-Zoo, a container-based framework that treats profile diversity as a first-class design principle. As shown in Table 1, IoT-Zoo provides 43 distinct IoT profiles, representing the highest diversity among the compared testbeds. In contrast, the second most diverse platform supports 13 profiles [Alatram et al. 2023], meaning that IoT-Zoo achieves more than $3.3\times$ higher diversity, corresponding to an increase of approximately 231%. This substantial improvement shifts the focus from predominantly replication-based scalability to diversity-driven modeling, without compromising scalability or reproducibility.

IoT-Zoo achieves this through three key design elements: (i) a profile-first architecture that models IoT devices as dataset-driven, containerized profiles; (ii) an extensible catalog of heterogeneous profiles grounded in real data characteristics; and (iii) a time-bounded execution workflow that automates deployment, orchestration, and traffic capture. This combination enables the construction of experimental environments that

are both scalable and intrinsically heterogeneous, better reflecting the complexity of real-world IoT deployments.

2. Architecture and Components

The IoT-Zoo architecture is designed to enable reproducible, extensible, and data-driven emulation of heterogeneous IoT environments, with explicit emphasis on device-level diversity. Unlike existing approaches that achieve scale by replicating a small set of device types, IoT-Zoo treats heterogeneity as a first-class architectural concern, enabling the systematic integration and execution of a large number of distinct device profiles within a unified experimental pipeline.

To support this goal, the architecture is organized into four layers: Build, Orchestration, Emulation, and Data Collection, as illustrated in Figure 1. These layers are loosely coupled and interact through well-defined interfaces, enabling modular evolution while preserving a consistent execution model.

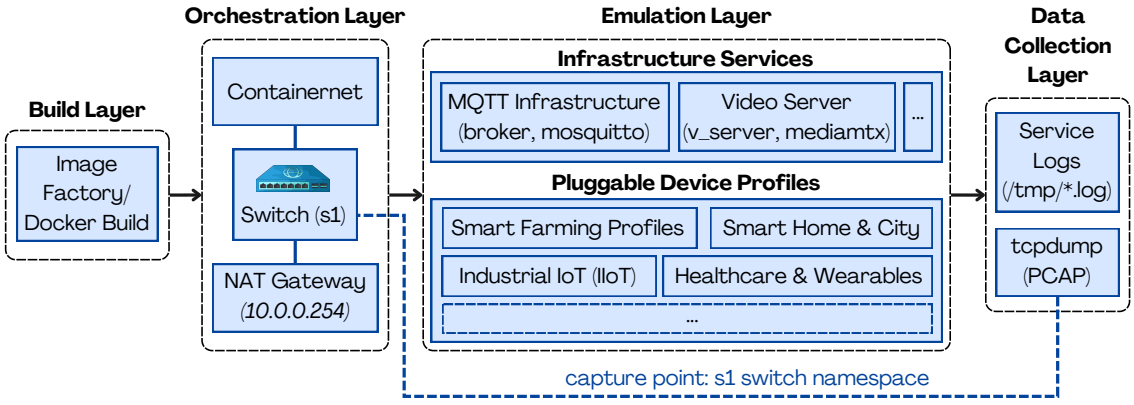


Figure 1. Architecture of the IoT-Zoo emulation testbed, illustrating the separation between build, orchestration, emulation, and data collection layers.

In IoT-Zoo, heterogeneity is defined by the number and diversity of device profiles, rather than by the number of replicated instances. Each profile encapsulates a unique combination of data source, temporal dynamics, protocol stack, and payload structure. This abstraction allows the system to represent diverse IoT behaviors in a composable and extensible manner.

The *Build Layer* is responsible for preparing all containerized components required by the testbed. It implements a Docker-based image factory where each device profile and infrastructure component is packaged as an independent image. This includes not only application logic but also dataset bindings and protocol configurations, enabling each profile to preserve its behavioral semantics across executions. By encapsulating profiles as self-contained artifacts, this layer supports consistent deployment and facilitates the dynamic inclusion of new device types without affecting existing components.

The *Orchestration Layer* manages the lifecycle of experiments and the logical network topology. It is implemented using Containernet[1], which enables Docker containers to be deployed as network nodes interconnected through a virtual switch. Beyond standard responsibilities such as container instantiation, IP assignment, and routing

[1] `https://containernet.github.io/`

configuration, this layer enables the dynamic composition of heterogeneous scenarios by instantiating arbitrary combinations of device profiles. It also controls execution parameters such as startup order, experiment duration, and termination, ensuring reproducible and time-bounded experiments. A central switch (`s1`) aggregates traffic, while a NAT gateway provides controlled external connectivity.

The *Emulation Layer* is the core of the diversity-driven design and comprises Infrastructure Services and Pluggable Device Profiles. While infrastructure components (e.g., MQTT brokers and streaming servers) provide stable shared functionalities, device profiles are modular and dataset-driven, enabling the representation of diverse IoT behaviors across multiple domains.

Each profile is implemented as a container that transforms raw telemetry data into protocol-compliant traffic (e.g., MQTT and RTSP) via a streaming pipeline that preserves temporal dynamics. This design enables dynamic heterogeneity, allowing new device types to be incorporated by simply adding new datasets and profiles, without modifying the underlying infrastructure. Consequently, IoT-Zoo supports continuous expansion of its profile space, overcoming the limitations of static or template-based approaches.

The *Data Collection Layer* captures all emulation outputs in a consistent, analysis-ready format. Network traffic is collected at the virtual switch level via traffic mirroring within the `s1` namespace, ensuring full visibility of inter-device communication without impacting execution. Device-level logs are also collected and temporally aligned with packet traces, enabling precise correlation between application events and network behavior. This integrated view supports downstream tasks such as intrusion detection, traffic classification, and behavioral analysis.

Overall, the architecture of IoT-Zoo enables a shift from replication-based scalability to diversity-driven modeling. By combining containerized profiles, dataset-driven behavior, and dynamic orchestration, the system supports large-scale yet inherently heterogeneous experimental environments, better reflecting the complexity of real-world IoT deployments.

3. Reference Implementation of the IoT-Zoo Tool

Our reference implementation instantiates the proposed architecture using a star-tree topology centered on the virtual switch `s1`, as shown in Figure 2. The deployment comprises 46 containers organized into functional categories and includes 43 distinct IoT device profiles, excluding infrastructure services such as the broker and media server. These profiles span multiple domains, including urban observability, human-centric monitoring, industrial and building automation, smart agriculture, and multimedia traffic.

The Urban Observatory-driven cluster (highlighted in grey in Figure 2) forms the core of the emulation, using 2025 data from the Newcastle Urban Observatory[2]. It integrates telemetry from 662 sensors, where containerized profiles act as aggregators (e.g., NO_2 from 34 stations and *Pedestrian Count* from 218 sensors). This cluster generates continuous MQTT traffic representative of city-scale telemetry, including air quality, meteorology, water management, and mobility.

[2]`https://urbanobservatory.ac.uk/`

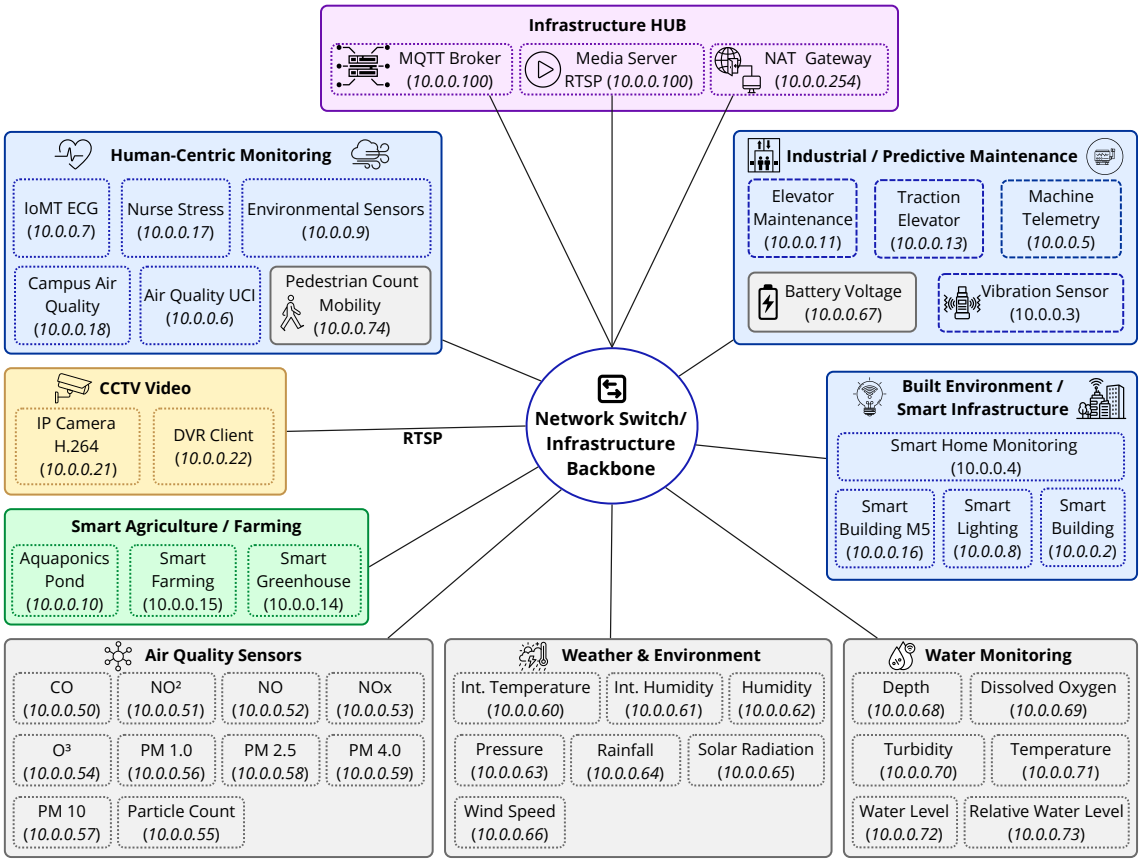


Figure 2. IoT-Zoo testbed topology highlighting device heterogeneity.

Beyond the Urban Observatory-driven core, IoT-Zoo incorporates additional profile categories to generate heterogeneous traffic patterns. Each containerized profile is driven by a distinct dataset, with representative examples discussed here due to the breadth of the instantiated profiles. For instance, Industrial / Predictive Maintenance includes profiles based on the AI4I 2020 Predictive Maintenance Dataset [Matzka 2020], while Human-Centric Monitoring includes physiological profiles such as IoTMT ECG from the mHealthDroid framework [Banos et al. 2014]. The remaining categories cover smart-building automation, smart agriculture / farming, and CCTV streams, combining lightweight MQTT telemetry with high-bandwidth RTSP traffic.

4. Experimental Evaluation

4.1. Setup

The evaluation was conducted in a virtualized environment (VMware Workstation Player) running Ubuntu 20.04.6 LTS with 4 vCPUs, 8 GB RAM, and 50 GB storage. The software stack includes Containernet (Mininet 2.3.1b1), Docker Engine 28.1.1, and Open vSwitch 2.13.8, enabling reproducible IoT emulation on commodity hardware.

4.2. Methodology and Metrics

Experiments were executed over a fixed duration of $T = 600$ seconds, with traffic captured at the virtual switch using `tcpdump`. Shorter datasets are automatically looped to ensure continuous traffic generation throughout the experiment. Analysis was performed

Table 2. Newcastle Urban Observatory subset (2025) for Smart City profiles.

Domain	Variable / Profile	Sources
Air Quality	Carbon Monoxide (CO)	36
	Nitrogen Dioxide (NO_2)	34
	Nitric Oxide (NO)	34
	Nitrogen Oxides (NO_x)	28
	Ozone (O_3)	28
	Particle Count	26
	Particulate Matter (PM 1.0)	26
	Particulate Matter (PM 10)	20
	Particulate Matter (PM 2.5)	12
	Particulate Matter (PM 4.0)	8
Weather & Indoor	Humidity	38
	Atmospheric Pressure	31
	Rainfall	14
	Wind Speed	11
Weather & Indoor	Internal Temperature	9
	Internal Humidity	9
	Solar Radiation	2
Water	Water Level	26
	Relative Water Level	25
	Battery Voltage (Sensors)	17
	Water Temperature	3
	Turbidity	3
	Depth	2
	Dissolved Oxygen	2
Mobility	Pedestrian Count	218
Total	**Aggregated Physical Sensors**	**662**

using `tshark`, focusing exclusively on MQTT `PUBLISH` packets to isolate application-level telemetry. TCP control traffic and keep-alive messages were excluded, and duplicate frames ($\Delta t < 100\ \mu s$) were removed.

Three metrics were computed from packet arrival timestamps (t_i): Packet Count (N), Inter-Arrival Time ($IAT = \mu(\Delta t)$), and Jitter ($J = \sigma(\Delta t)$), capturing traffic volume, temporal behavior, and variability.

4.3. Heterogeneity and Traffic Fidelity

We quantify heterogeneity by analyzing per-profile message volume and temporal dynamics (IAT and jitter), as distinct IoT domains exhibit different periodicity and burstiness patterns. Table 3 summarizes the network metrics collected over a 600-second execution window. The results demonstrate the ability of IoT-Zoo to generate highly heterogeneous traffic profiles concurrently within the same topology, reflecting the diversity of real-world IoT environments. The analysis of inter-arrival times further reveals multiple coexisting behavioral patterns, ranging from stable periodic transmissions to bursty and irregular dynamics.

First, several industrial and infrastructure-oriented profiles, such as *Ind. Motor (Cooler)*, *Predictive Maint.*, *Ind. Pred. Maint. GW*, and *Ind. Traction Elevator*, exhibited near-periodic behavior, with IAT values close to $5.0s$ and low jitter. This confirms the testbed's ability to reproduce stable telemetry patterns typical of monitoring and predictive maintenance scenarios.

In contrast, Urban Observatory-driven profiles, represented by streams such as *Weather: Press*, *Urban Air: NO2*, and *Urban Air: PM10*, generated sustained high-volume traffic. These devices produced between roughly 3000 and 3800 packets during the execution window, with low IAT values (down to $0.1560s$) and stable jitter, effectively stressing the network's throughput capacity.

Finally, profiles derived from more irregular real-world processes showed higher variability. For example, the *Mobility Sensor* combined a low average IAT ($0.2605s$) with comparatively high jitter ($1.8041s$), reflecting bursty transmission dynamics associated with human activity. Other profiles, such as *Building Rooms* and *Water Q: Depth*, also exhibited higher timing dispersion, reinforcing that IoT-Zoo captures heterogeneous application behavior rather than relying on uniform synthetic traffic generation.

Table 3. Network metrics for representative IoT device profiles.

Device	Pkts	IAT(s)	Jit(s)	Device	Pkts	IAT(s)	Jit(s)
Building Rooms	269	2.2111	2.5412	Urban Air: PM10	2989	0.2003	0.3415
Build: Int. Hum.	301	1.9987	1.6460	Urban Air: PM2.5	3055	0.1960	0.3390
Build: Int. Temp.	302	1.9857	1.6513	Urban Air: PM4	2462	0.2433	0.4289
Ind. Motor (Cooler)	121	4.9588	0.4547	Urban Air: Particle	2450	0.2445	0.4297
Legacy Air Sensor	74	7.9919	6.2948	Water L: Abs	1812	0.3292	1.2396
Mobility Sensor	2292	0.2605	1.8041	Water L: Rel	1380	0.4322	1.4048
Predictive Maint.	120	4.9999	0.0993	Water Q: DO	180	3.3274	2.3576
Smart Home (Domotic)	78	7.6615	4.9212	Water Q: Depth	238	2.5134	2.5001
Smart Lighting	125	4.7996	0.9390	Water Q: Temp	208	2.8774	2.4701
Urban Air: CO	2338	0.2562	0.4364	Water Q: Turb	308	1.9403	2.4356
Urban Air: NO	762	0.7829	1.8172	Weather: Hum	302	1.9920	1.6473
Urban Air: NO2	3519	0.1702	0.3228	Weather: Press	3843	0.1560	0.3623
Urban Air: NOx	597	0.9996	2.0002	Weather: Rain	1352	0.4411	1.4181
Urban Air: O3	2894	0.2070	0.4050	Weather: Solar	592	1.0137	0.9991
Urban Air: PM1	2999	0.1997	0.3411	Weather: Wind	1010	0.5940	0.8909
e-Health ECG	1184	0.5037	0.3834	Human: Nurse Stress	124	4.8572	1.0376
Aquaponics	117	5.1495	0.9955	Smart Building M5	123	4.8955	0.9522
Environmental Sensors	121	4.9954	0.9436	Farming Sensor	120	4.9711	0.9409
Greenhouse	237	2.5146	1.3538	Ind. Pred. Maint. GW	119	5.0256	0.9107
				Ind. Traction Elevator	122	4.9423	0.9407

4.4. Scalability & Resource Efficiency

To assess the feasibility of deploying heterogeneous IoT topologies on commodity hardware, we monitored the resource footprint of all 46 containerized profiles throughout the experiment. Table 4 summarizes memory usage and CPU peak load grouped by the device categories defined in the instantiated topology. The results demonstrate that IoT-Zoo maintains a lightweight and predictable resource footprint even when combining diverse telemetry and multimedia workloads.

Table 4. Resource Consumption Profile by Device Category ($T = 600s$).

Device Category	Total Profiles	Mem. Usage (MiB) ($\mu \pm \sigma$)	CPU Peak (%)
Air Quality Sensors	10	48.65 ± 1.98	11.75
Weather & Environment	5	48.65 ± 2.02	12.04
Water Monitoring	6	48.62 ± 2.03	11.26
Built Environment / Smart Infra.	8	20.82 ± 16.24	38.41
Human-Centric Monitoring	3	27.18 ± 16.37	7.71
Industrial / Predictive Maintenance	6	16.83 ± 12.15	7.67
Smart Agriculture / Farming	3	14.58 ± 4.43	9.93
CCTV Video	2	8.94 ± 6.53	5.27
Infrastructure HUB	3	12.44 ± 10.01	1.60
Total / Average	**46**	≈ 27.42 **MiB**	-

The Urban Observatory-driven categories, namely *Air Quality Sensors*, *Weather & Environment*, and *Water Monitoring*, exhibited a highly consistent memory profile, with average usage close to 48.6 MiB and low intra-category dispersion, with a standard deviation of approximately 2 MiB. This uniformity across profiles processing different telemetry variables indicates that the stream-based ingestion approach keeps memory consumption bounded and predictable, regardless of the size of the underlying datasets.

The remaining categories showed lower average memory usage overall, including *Industrial / Predictive Maintenance* (16.83 ± 12.15 MiB), *Smart Agriculture / Farming* (14.58 ± 4.43 MiB), and *CCTV Video* (8.94 ± 6.53 MiB). *Built Environment / Smart Infrastructure* displayed the highest variability and CPU peak (38.41%), reflecting the coexistence of lightweight automation profiles and more heterogeneous building-oriented workloads within the same category.

Finally, the *Infrastructure HUB* maintained a modest footprint (12.44 ± 10.01 MiB) and low CPU peak (1.60%), indicating that host resources are predominantly devoted to the emulated endpoints rather than orchestration overhead. Across the Urban Observatory-driven categories, CPU peaks remained around 11%–12%, showing that data-driven replay can scale without saturating the allocated resources of a standard virtual machine (4 vCPUs).

5. Availability and Demonstration Plan

To support validation and reuse, the IoT-Zoo source code, including Containernet scripts and Dockerfiles, is publicly available at `https://github.com/GT-IoTEdu/sbrc2026-IoT-Zoo`. The repository provides a `README` with Ansible-based installation and automated build instructions (`build_images.sh`). The fully containerized testbed can be reproduced via a single configurable command (e.g., `sudo python3 run_experiment.py --time 600`), enabling automated topology provisioning and traffic capture without manual intervention. An example of captured traffic in Wireshark is presented in Appendix A.

For the SBRC Tools Lounge, the demonstration will run on a standard laptop (4 vCPUs, 8GB RAM) using an Ubuntu 20.04 VM, requiring no specialized hardware or network infrastructure. The demo will showcase automated setup, real-time testbed execution, and live traffic analysis with Wireshark, highlighting protocol and payload heterogeneity (JSON, XML, Binary).

6. Conclusions and Future Work

This work introduced IoT-Zoo, a container-based testbed designed to enable reproducible and diversity-driven IoT experimentation. In contrast to existing platforms that rely on replication of a few device types, IoT-Zoo supports 43 distinct profiles within a unified pipeline, achieving the highest diversity among the evaluated testbeds. The architecture combines dataset-driven modeling, containerized execution, and automated orchestration to generate realistic multi-domain traffic with preserved temporal dynamics.

Experimental results show that heterogeneous patterns, from periodic telemetry to bursty human-driven behavior, can be reproduced concurrently with a lightweight and predictable resource footprint on commodity hardware. By bridging static datasets and live emulation, IoT-Zoo enables the systematic generation of reproducible datasets for traffic analysis, intrusion detection, and machine learning evaluation.

Future work will focus on increasing realism and applicability. We plan to incorporate benign user-driven behaviors, expand the profile catalog across domains, and integrate attack scenarios with ground-truth labeling. Additionally, we aim to couple IoT-Zoo with machine learning pipelines for end-to-end evaluation and to explore large-scale deployments with thousands of nodes.

Acknowledgments

This work was partially supported by the Brazilian National Research and Education Network (RNP) through the GT-IoTEdu project, approved under the 2025 Advanced Services R&D Program; by the Brazilian National Council for Scientific and Technological Development (CNPq) under Grant 409743/2025-9; by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Finance Code 001; and by the Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS) under Grants 24/2551-0001368-7 and 24/2551-0000726-1.

References

Alatram, A., Sikos, L. F., Johnstone, M., Szewczyk, P., and Kang, J. J. (2023). DoS/DDoS-MQTT-IoT: A dataset for evaluating intrusions in IoT networks using the MQTT protocol. *Computer Networks*, 231:109809.

Alex, C., Creado, G., Almobaideen, W., Alghanam, O. A., and Saadeh, M. (2023). A Comprehensive Survey for IoT Security Datasets Taxonomy, Classification and Machine Learning Mechanisms. *Computers & Security*, 132:103283.

Banos, O., Garcia, R., Holgado-Terriza, J. A., Damas, M., Pomares, H., Rojas, I., Saez, A., and Villalonga, C. (2014). mHealthDroid: A Novel Framework for Agile Development of Mobile Health Applications. In *Int. Workshop on Ambient Assisted Living*, pages 91–98. Springer.

De Keersmaeker, F., Cao, Y., Ndonda, G. K., and Sadre, R. (2023). A Survey of Public IoT Datasets for Network Security Research. *IEEE Communications Surveys & Tutorials*, 25(3):1808–1840.

Dehlaghi-Ghadim, A., Balador, A., Moghadam, M. H., Hansson, H., and Conti, M. (2023). ICSSIM—A framework for building industrial control systems security testbeds. *Computers in Industry*, 148:103906.

Farag, W., Wu, X.-W., Ezekiel, S., Rado, D., and Lassinger, J. (2025). Development and evaluation of a novel iot testbed for enhancing security with machine learning-based threat detection. *Sensors*, 25(18):5870.

Ghazanfar, S., Hussain, F., Rehman, A. U., Fayyaz, U. U., Shahzad, F., and Shah, G. A. (2020). IoT-Flock: An Open-source Framework for IoT Traffic Generation. In *Int. Conf. on Emerging Trends in Smart Technologies*, pages 1–6. IEEE.

Islam, H., Nath, S., Rahman, M., Shahriar, N., Khan, M., and Islam, R. (2025). STGen: A Novel Lightweight IoT Testbed for Generating Sensor Traffic for the Experimentation of IoT Protocol and its Application in Hybrid Network. *arXiv preprint arXiv:2504.17725*.

Lima, B. and Pinto, R. (2025). Current challenges and future perspectives in testing IoT systems: A comprehensive review. *IEEE Sensors Reviews*.

Matzka, S. (2020). Explainable Artificial Intelligence for Predictive Maintenance Applications. In *Int. Conf. on Artificial Intelligence for Industries*, pages 69–74. IEEE.

Mishra, S. R., Shanmugam, B., Yeo, K. C., and Thennadil, S. (2025). SDN-enabled IoT security frameworks—a review of existing challenges. *Technologies*, 13(3):121.

Moustafa, N. (2021). A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets. *Sustainable Cities and Society*, 72:102994.

Poisson, M., Carnier, R., and Fukuda, K. (2024). GothX: a generator of customizable, legitimate and malicious IoT network traffic. In *Cyber Security Experimentation and Test Workshop*, pages 65–73.

Sáez-de Cámara, X., Flores, J. L., Arellano, C., Urbieta, A., and Zurutuza, U. (2023). Gotham testbed: A reproducible iot testbed for security experiments and dataset generation. *IEEE Transactions on Dependable and Secure Computing*, 21(1):186–203.

Sah, D. K., Vahabi, M., and Fotouhi, H. (2025). A comprehensive review on 5G IIoT test-beds. *IEEE transactions on consumer electronics*.

Syed, N. F., Baig, Z., Ibrahim, A., and Valli, C. (2020). Denial of service attack detection through machine learning for the IoT. *Journal of Information and Telecommunication*, 4(4):482–503.

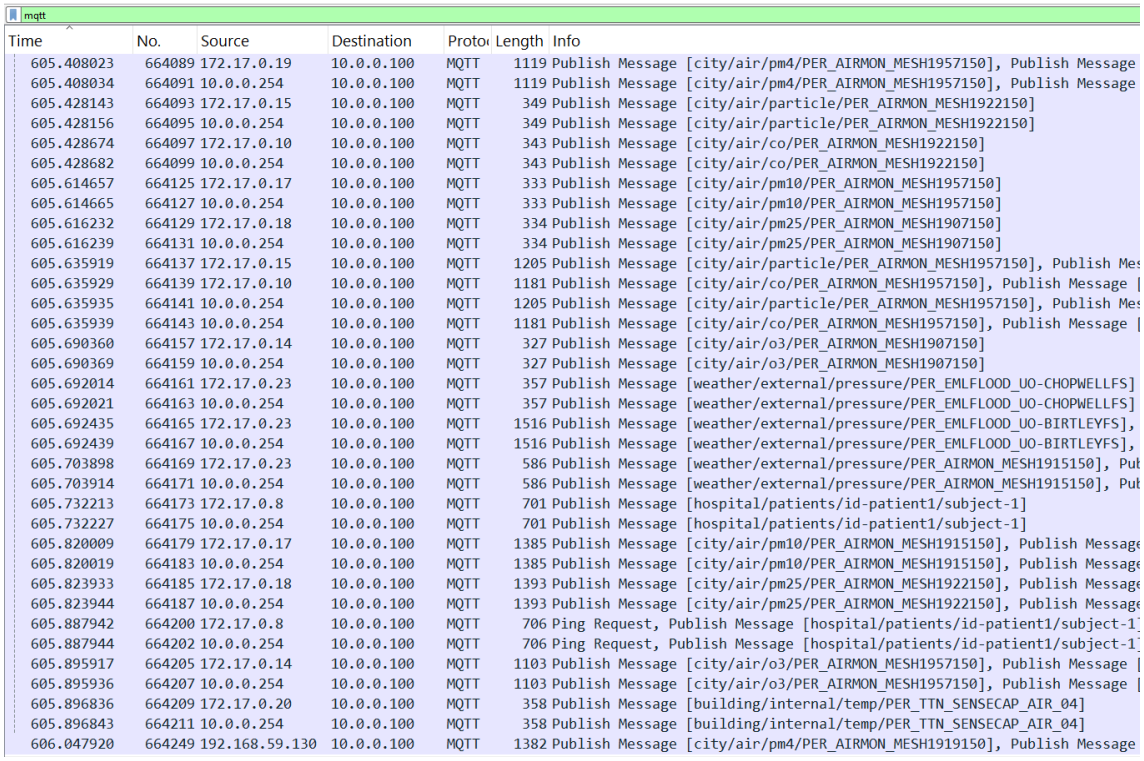
Waraga, O. A., Bettayeb, M., Nasir, Q., and Talib, M. A. (2020). Design and implementation of automated IoT security testbed. *Computers & security*, 88:101648.

Zachos, G., Mantas, G., Porfyrakis, K., de Bastos, J. M. C. S., and Rodriguez, J. (2025). Anomaly-based intrusion detection for IoMT networks: Design, implementation, dataset generation, and ML algorithms evaluation. *IEEE Access*, 13:41994–42028.

A. Example of Captured Traffic Output

Figure 3 shows an example of network traffic generated by IoT-Zoo and automatically captured during execution. The screenshot displays MQTT `PUBLISH` messages in Wireshark, highlighting heterogeneous topic namespaces and payload flows across multiple IoT domains, including Smart City telemetry (e.g., air quality), environmental monitoring (e.g., wind and pressure), e-Health (patient data streams), and industrial sensing (e.g., vibration). This output demonstrates the ability of IoT-Zoo to generate reproducible, multi-domain traffic traces in PCAP format, suitable for downstream analysis and benchmarking.

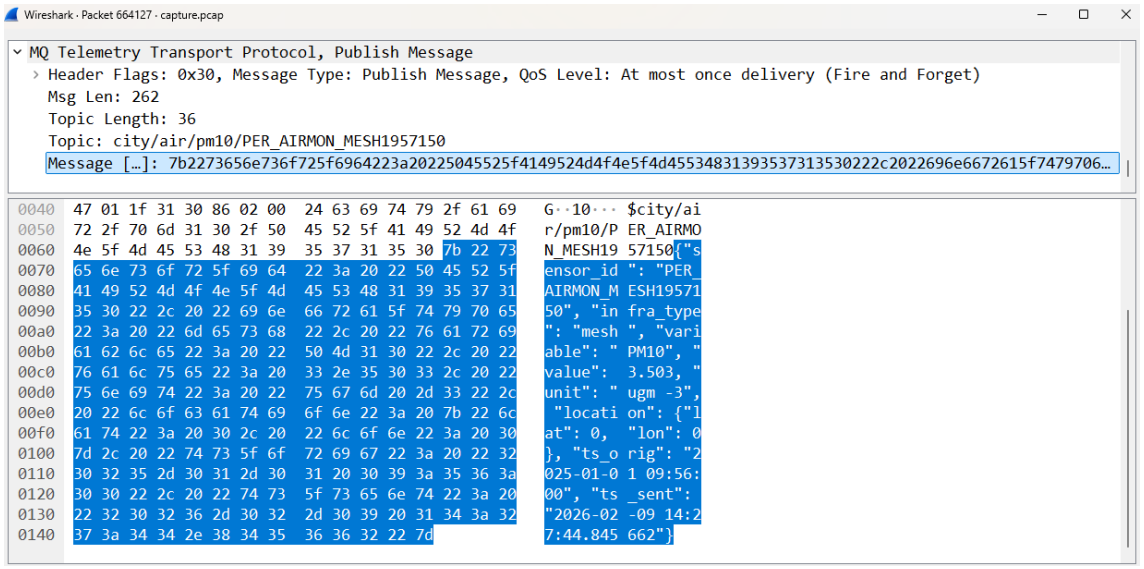
Figure 4 complements this view by presenting a decoded MQTT `PUBLISH` packet in Wireshark, illustrating an example of a structured application payload replayed from the original dataset. This demonstrates that IoT-Zoo preserves dataset-driven semantics at the application layer, rather than reproducing only protocol-level traffic patterns.



mqtt

Time	No.	Source	Destination	Proto	Length	Info
605.408023	664089	172.17.0.19	10.0.0.100	MQTT	1119	Publish Message [city/air/pm4/PER_AIRMON_MESH1957150], Publish Message
605.408034	664091	10.0.0.254	10.0.0.100	MQTT	1119	Publish Message [city/air/pm4/PER_AIRMON_MESH1957150], Publish Message
605.428143	664093	172.17.0.15	10.0.0.100	MQTT	349	Publish Message [city/air/particle/PER_AIRMON_MESH1922150]
605.428156	664095	10.0.0.254	10.0.0.100	MQTT	349	Publish Message [city/air/particle/PER_AIRMON_MESH1922150]
605.428674	664097	172.17.0.10	10.0.0.100	MQTT	343	Publish Message [city/air/co/PER_AIRMON_MESH1922150]
605.428682	664099	10.0.0.254	10.0.0.100	MQTT	343	Publish Message [city/air/co/PER_AIRMON_MESH1922150]
605.614657	664125	172.17.0.17	10.0.0.100	MQTT	333	Publish Message [city/air/pm10/PER_AIRMON_MESH1957150]
605.614665	664127	10.0.0.254	10.0.0.100	MQTT	333	Publish Message [city/air/pm10/PER_AIRMON_MESH1957150]
605.616232	664129	172.17.0.18	10.0.0.100	MQTT	334	Publish Message [city/air/pm25/PER_AIRMON_MESH1907150]
605.616239	664131	10.0.0.254	10.0.0.100	MQTT	334	Publish Message [city/air/pm25/PER_AIRMON_MESH1907150]
605.635919	664137	172.17.0.15	10.0.0.100	MQTT	1205	Publish Message [city/air/particle/PER_AIRMON_MESH1957150], Publish Mes
605.635929	664139	172.17.0.10	10.0.0.100	MQTT	1181	Publish Message [city/air/co/PER_AIRMON_MESH1957150], Publish Message [
605.635935	664141	10.0.0.254	10.0.0.100	MQTT	1205	Publish Message [city/air/particle/PER_AIRMON_MESH1957150], Publish Mes
605.635939	664143	10.0.0.254	10.0.0.100	MQTT	1181	Publish Message [city/air/co/PER_AIRMON_MESH1957150], Publish Message [
605.690360	664157	172.17.0.14	10.0.0.100	MQTT	327	Publish Message [city/air/o3/PER_AIRMON_MESH1907150]
605.690369	664159	10.0.0.254	10.0.0.100	MQTT	327	Publish Message [city/air/o3/PER_AIRMON_MESH1907150]
605.692014	664161	172.17.0.23	10.0.0.100	MQTT	357	Publish Message [weather/external/pressure/PER_EMLFLOOD_UO-CHOPWELLFS]
605.692021	664163	10.0.0.254	10.0.0.100	MQTT	357	Publish Message [weather/external/pressure/PER_EMLFLOOD_UO-CHOPWELLFS]
605.692435	664165	172.17.0.23	10.0.0.100	MQTT	1516	Publish Message [weather/external/pressure/PER_EMLFLOOD_UO-BIRTLEYFS],
605.692439	664167	10.0.0.254	10.0.0.100	MQTT	1516	Publish Message [weather/external/pressure/PER_EMLFLOOD_UO-BIRTLEYFS],
605.703898	664169	172.17.0.23	10.0.0.100	MQTT	586	Publish Message [weather/external/pressure/PER_AIRMON_MESH1915150], Pub
605.703914	664171	10.0.0.254	10.0.0.100	MQTT	586	Publish Message [weather/external/pressure/PER_AIRMON_MESH1915150], Pub
605.732213	664173	172.17.0.8	10.0.0.100	MQTT	701	Publish Message [hospital/patients/id-patient1/subject-1]
605.732227	664175	10.0.0.254	10.0.0.100	MQTT	701	Publish Message [hospital/patients/id-patient1/subject-1]
605.820009	664179	172.17.0.17	10.0.0.100	MQTT	1385	Publish Message [city/air/pm10/PER_AIRMON_MESH1915150], Publish Message
605.820019	664183	10.0.0.254	10.0.0.100	MQTT	1385	Publish Message [city/air/pm10/PER_AIRMON_MESH1915150], Publish Message
605.823933	664185	172.17.0.18	10.0.0.100	MQTT	1393	Publish Message [city/air/pm25/PER_AIRMON_MESH1922150], Publish Message
605.823944	664187	10.0.0.254	10.0.0.100	MQTT	1393	Publish Message [city/air/pm25/PER_AIRMON_MESH1922150], Publish Message
605.887942	664200	172.17.0.8	10.0.0.100	MQTT	706	Ping Request, Publish Message [hospital/patients/id-patient1/subject-1
605.887944	664202	10.0.0.254	10.0.0.100	MQTT	706	Ping Request, Publish Message [hospital/patients/id-patient1/subject-1
605.895917	664205	172.17.0.14	10.0.0.100	MQTT	1103	Publish Message [city/air/o3/PER_AIRMON_MESH1957150], Publish Message [
605.895936	664207	10.0.0.254	10.0.0.100	MQTT	1103	Publish Message [city/air/o3/PER_AIRMON_MESH1957150], Publish Message [
605.896836	664209	172.17.0.20	10.0.0.100	MQTT	358	Publish Message [building/internal/temp/PER_TTN_SENSECAP_AIR_04]
605.896843	664211	10.0.0.254	10.0.0.100	MQTT	358	Publish Message [building/internal/temp/PER_TTN_SENSECAP_AIR_04]
606.047920	664249	192.168.59.130	10.0.0.100	MQTT	1382	Publish Message [city/air/pm4/PER_AIRMON_MESH1919150], Publish Message

Figure 3. Example of MQTT traffic captured from an IoT-Zoo execution (PCAP inspection in Wireshark).



Wireshark · Packet 664127 · capture.pcap

```
MQ Telemetry Transport Protocol, Publish Message
  > Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
    Msg Len: 262
    Topic Length: 36
    Topic: city/air/pm10/PER_AIRMON_MESH1957150
    Message […]: 7b2273656e736f725f6964223a20225045525f4149524d4f4e5f4d4553483139353731353022c2022696e6672615f7479706…
```

```
0040  47 01 1f 31 30 86 02 00  24 63 69 74 79 2f 61 69   G··10··· $city/ai
0050  72 2f 70 6d 31 30 2f 50  45 52 5f 41 49 52 4d 4f   r/pm10/P ER_AIRMO
0060  4e 5f 4d 45 53 48 31 39  35 37 31 35 30 7b 22 73   N_MESH19 57150{"s
0070  65 6e 73 6f 72 5f 69 64  22 3a 20 22 50 45 52 5f   ensor_id ": "PER_
0080  41 49 52 4d 4f 4e 5f 4d  45 53 48 31 39 35 37 31   AIRMON_M ESH19571
0090  35 30 22 2c 20 22 69 6e  66 72 61 5f 74 79 70 65   50", "in fra_type
00a0  22 3a 20 22 6d 65 73 68  22 2c 20 22 76 61 72 69   ": "mesh ", "vari
00b0  61 62 6c 65 22 3a 20 22  50 4d 31 30 22 2c 20 22   able": " PM10", "
00c0  76 61 6c 75 65 22 3a 20  33 2e 35 30 33 2c 20 22   value":  3.503, "
00d0  75 6e 69 74 22 3a 20 22  75 67 6d 20 2d 33 22 2c   unit": " ugm -3",
00e0  20 22 6c 6f 63 61 74 69  6f 6e 22 3a 20 7b 22 6c    "locati on": {"l
00f0  61 74 22 3a 20 30 2c 20  22 6c 6f 6e 22 3a 20 30   at": 0,  "lon": 0
0100  7d 2c 20 22 74 73 5f 6f  72 69 67 22 3a 20 22 32   }, "ts_o rig": "2
0110  30 32 35 2d 30 31 2d 30  31 20 30 39 3a 35 36 3a   025-01-0 1 09:56:
0120  30 30 22 2c 20 22 74 73  5f 73 65 6e 74 22 3a 20   00", "ts _sent":
0130  22 32 30 32 36 2d 30 32  2d 30 39 20 31 34 3a 32   "2026-02 -09 14:2
0140  37 3a 34 34 2e 38 34 35  36 36 32 22 7d            7:44.845 662"}
```

Figure 4. Decoded MQTT `PUBLISH` packet showing an example of the application payload replayed from the dataset.