A Framework for Configurable Scalability Evaluations of IoT Platforms

Fabio Muratori, Yuening Yang, Zeineb Rejiba and Andrei Marian Dan

Abstract Testing Internet of Things (IoT) platforms is essential for assessing their reliability, scalability and potential performance issues. Such tests are particularly challenging to set up when these IoT platforms are managing large fleets of devices. A promising approach to address this is to use test frameworks that employ simulated IoT devices to interact with the target IoT platform. This work introduces Fleet sImulator for Scalability Tests (FIST), an extendable test framework that simulates a variety of realistic IoT scenarios, including large fleets of devices and realistic network conditions. The framework's flexibility is demonstrated through the evaluation of the open-source IoT platform ThingsBoard. We evaluate the capabilities of this platform in dealing with multiple device connections and firmware update tasks. Overall, this study contributes to the development of robust and reliable IoT platforms by providing a comprehensive test framework for performance evaluations.

1 Introduction

The advent of the Internet of Things (IoT) has resulted in an increased demand for IoT platforms that manage large fleets of devices [7, 22, 2, 18]. Selecting a suitable IoT platform requires a prior evaluation of its functionality and performance. Robust testing frameworks should evaluate the reliability, scalability, and performance of IoT platforms [15, 17, 9, 21]. A testing framework serves as a structured approach to evaluate the quality of an IoT system, encompassing a suite of tools, scripts, scenarios, rules, and templates designed to facilitate the assessment of modern IoT platforms.

The complexities inherent to testing and benchmarking IoT platforms and device fleet management systems result in different challenges that need to be addressed. An effective evaluation method is testing using a fleet of simulated devices, which

Hitachi Energy Research, Segelhofstrasse 1A, Baden-Daettwil, Switzerland, e-mail: firstname.lastname@hitachienergy.com

typically requires careful management of the platform setup and the interaction with virtual devices. As the number of configurations grows, it becomes increasingly difficult to manage and analyze the results, particularly when attempting to replicate the behavior of numerous devices. Network issues such as message loss, corruption, and limited bandwidth also add another layer of complexity. Moreover, the variety of device implementations and the absence of a standardized method for simulating realistic device behavior adds to these challenges. Finally, the lack of a ready-made solution for simulating a large fleet of devices presents a further hurdle in testing the scalability and reliability of IoT platforms under various network conditions. Existing works in this area do not address these challenges as they focus on feature comparison at a conceptual level [15, 17, 9] or focus on comparing communication protocols instead of IoT platforms [21].

In contrast, our work addresses these challenges by designing, developing, and evaluating Fleet sImulator for Scalability Tests (FIST), a configurable framework to test IoT fleet management platforms that ensures the accuracy and reliability of the evaluation process.

2 Fleet sImulator for Scalability Tests (FIST)

To support comprehensive evaluations for IoT platforms, we propose the Fleet sImulator for Scalability Tests framework. In the following, we provide a detailed description of the FIST architecture.

Main components. The FIST framework comprises four essential components, as depicted in Figure 1.

The Simulation Manager manages the simulation configuration and its lifecycle. It monitors hardware resource utilization and ensures the graceful termination of simulations, results aggregation, and environment clean-up.

The Fleet Simulator component simulates multiple virtual devices and executes the core simulation tasks provided by the IoT platform (e.g., firmware updates). It can be customized to accommodate a large number of simulated devices. The simulated devices can run either separately in distinct containers, achieving the *share nothing* mode, or several simulated devices in each container.

The Network Environment Simulator manages various network-related behaviors. Configurations include ingress bandwidth limitations, outgoing message drops, packet duplication, corruption, and delays (Section 2.2). The Network Environment Simulator relies on a customized version of the open-source project named *docker-tc* [12].

The YAML configuration is a file that specifies how to set up the framework components and other scenario execution parameters.

In addition to these components, FIST interacts with the target IoT platform to be evaluated. More specifically, FIST interacts with the backend that provides the IoT services used by the simulated devices.

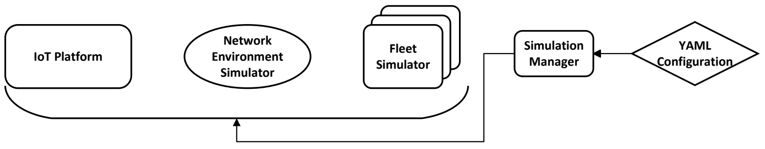

Fig. 1 FIST framework main components and interactions.

Framework Layers. The FIST functionality is separated into the control and simulation layers.

Control Layer. The control layer orchestrates the various components and manages the entire simulation lifecycle, including results aggregation, termination, and clean-up. The overall control process consists of the following steps:

Step 1: Definition of a simulation scenario via the YAML configuration file. Leveraging this configuration, the Simulation Manager starts the backend of the IoT platform and periodically sends requests to check the readiness of the backend components.

Step 2: The Simulation Manager launches the Fleet Simulator by deploying multiple Docker containers, as specified in the initial YAML file. Each container is responsible for a subset of the simulated devices. The manager also sets up the necessary files for each container and ensures that all Fleet Simulator instances and devices successfully connect to the IoT platform before starting the main simulation task (described in Section 2.1). This guarantees that all devices are online and ready, minimizing any warm-up period that might impact test results.

Step 3: The Network Simulator is instantiated and configured to automatically apply the network simulation rules defined in the YAML file to all Fleet Manager containers.

Step 4: The Simulation Manager initiates the execution of the actual simulation task by invoking a special trigger that interacts with the IoT platform.

Step 5: During the task execution, the Simulation Manager monitors the task status by invoking an HTTP endpoint provided by the Fleet Simulator components. Once all Fleet Simulator containers have completed their tasks, i.e., all devices within each container have finished their assigned work, the Simulation Manager aggregates and stores relevant data and statistics. Finally, all resources are cleaned up to ensure a stable environment for subsequent simulations.

Simulation Layer. The simulator layer resides at the core of the FIST framework, simulating all devices within one or multiple Docker containers. These containers establish the communication with the IoT platform backend and execute tasks such as firmware updates, which constitute an important feature of IoT platforms. A Fleet Simulator container can represent one or multiple devices, depending on the configuration. Figure 2 illustrates the internal structure of a single Fleet Simulator container.

Within the Fleet Simulator, a **Fleet Manager** is responsible for managing the lifecycle of the simulated devices. It ensures their proper initialization, task exe-

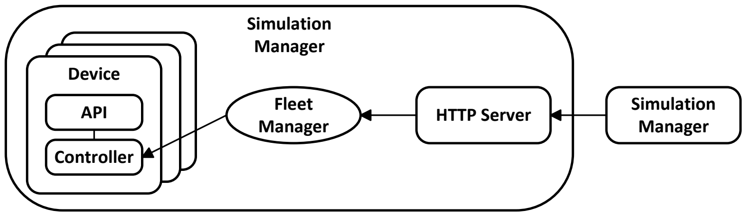

Fig. 2 Fleet Simulator internal structure overview.

cution, and termination. Additionally, the Fleet Manager provides an interface for monitoring the simulation process and retrieving real-time statistics. The **HTTP Server** allows external entities to observe the ongoing simulation and retrieve the corresponding data.

Each simulated device has two components: the **Application Programming Interface (API)**, which facilitates communication between the device and the IoT platform and defines the task execution behavior for each virtual device, and a **Controller**, that is assigned by the Fleet Manager. The Controller holds essential information required for device-side tasks, including adding realistic behaviors that can affect a task execution, such as device workloads and abrupt crashes. Furthermore, the Controller reports the device status to the Fleet Manager for control and statistical analysis.

2.1 Device simulation

Each simulated device is encapsulated and managed by one Fleet Simulator per container. The lifecycle of a simulation, from the perspective of an individual device, consists of two phases: the connection to the IoT platform and the execution of the tasks.

Connection to the IoT platform. The connection phase starts immediately after the creation of the device. The Simulation Manager controls when and how devices are allowed to connect to the IoT platform backend services. This is done through a REST API call to each of the Fleet Simulator HTTP servers. FIST offers a variety of methods to manage and initiate the connection of multiple devices to the IoT platform, even when these devices are distributed across multiple containers. This is implemented to facilitate different connection workloads between the simulated devices and the IoT platform. This configuration comprises two adjustable parameters provided directly within the YAML configuration: the policy for creating and starting containers and the policy for creating and registering devices. Both parameters permit sequential and parallel initialization, resulting in a total of four potential connection workloads.

Tasks execution. The second phase of a device's lifecycle is the execution of the user-defined tasks, while supporting simulated device workloads and crashes. Each device uses an internal scheduler to manage the execution of generic tasks. This approach ensures that only one task is executed at any given moment for a device, with subsequent tasks being queued for future execution. An additional configuration is the ability to simulate device crash behaviors.

2.2 Simulation of the Network Environment

To mimic realistic network conditions, the Network Environment Simulator component uses `docker-tc`, which is designed for Docker containers and is a modification of the Linux Traffic Control utility [13]. More specifically, any packet transiting through the container's `veth` interface may be affected by `docker-tc` policies based on the following configurable rules:

- Limitation of incoming network bandwidth for a container, ensuring that the download speed of a container does not exceed a specified value.
- Delay, duplication, drop, or corruption of a percentage of outgoing packets from a container.

During our experiments, we identified some issues that occurred under high traffic scenarios simulated by the FIST framework. To address this, we implemented a patch to the `docker-tc` API that enables more configurable network limiting parameters. This patch is available as part of the FIST GitHub repository.

To ensure a proper use of `docker-tc` within FIST, both simulated devices and target IoT platform must be within the same Docker Bridge network. Optionally, it is possible to apply the network simulation environment to individual devices by running each of these simulated devices in a separate container.

2.3 IoT platform integration

The FIST framework is designed to allow an easy integration of different target IoT platforms. To this end, we provide minimal implementations on how to interact with and manage IoT platforms, as well as how a simulation should be conducted. The following integration steps (illustrated in blue in Figure 3) are required to support a new IoT platform :

- Declare how the FIST framework should handle different phases of the IoT platform lifecycle, primarily its creation, task triggering, termination, and cleanup (IoT Platform drivers).
- Specify the device-specific configurations (e.g., authentication certificates) that are primarily involved in the creation and setup of Fleet Simulator containers (Device drivers).

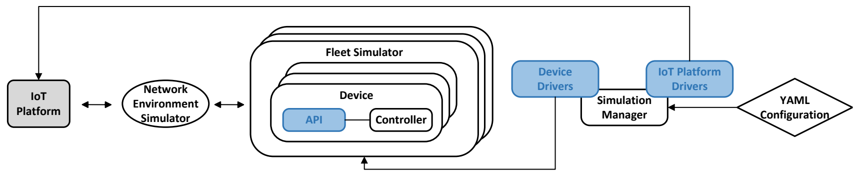

Fig. 3 User-defined components (blue) for integrating new IoT platforms with FIST.

- Implement the simulated device behavior, such as handling communication with the IoT platform and implementing task execution details.

3 Experimental setup

The FIST implementation is based on Python `v3.10.12` for the Simulation Manager component and Go `1.21.2` for the Fleet Simulator. The repository with the complete implementation of FIST framework and the code used to generate all the experimental results is available online[1].

Experiment tasks description. In the following subsections, we present the tasks that are executed in each of the simulated scenarios. The corresponding results are detailed in Section 4.

Connection Capacity. This experiment evaluates the capacity of each IoT platform to handle a large number of simultaneous incoming connection requests. The simulation setup consists of $20,000$ devices that simultaneously attempt to connect to the target IoT platform. Finally, we evaluate how many devices successfully establish a communication channel with the IoT platform. We measure the time required for the scenario to complete, which includes the connection attempts of all the simulated devices. Each scenario is executed five times and we report the average and standard deviation for these executions.

Firmware Update Distribution. This scenario evaluates the efficiency and reliability of firmware update distribution for different firmware and fleet sizes. Our experimental setup consists of 10 Fleet Simulator containers, each containing an equal number of simulated devices. We fix the firmware size to 32MB and vary the total number of simulated devices from $1,000$ to $5,000$.

We use the docker-tc network simulator to define the following parameters for the Fleet Simulator containers: 2% of messages are lost during transmission, 2% of messages are duplicated, 1% of messages experience corruption, and messages consistently encounter a 50ms delay before transmission[2,3].

[1] https://github.com/hitachienergy/fleet-simulator-for-scalability-tests

[2] https://www.pingman.com/kb/42

[3] https://obkio.com/blog/acceptable-packet-loss/

Table 1 Successful connections for Thingsboard, when $20,000$ devices concurrently attempt to connect.

Configuration	Succ. connections		Devices task duration [mm:ss]	
	Mean	SD	Mean	SD
Memory	19,794	451	01:07	00:44
Memory+Net.	19,976	51	01:50	00:53
Kafka	19,889	177	01:43	00:50
Kafka+Net.	19,960	88	03:04	01:17

Hardware platforms. We ran the experiments using a workstation with a 62 GB RAM (DDR4 2133Mhz) and a 10-core Intel Xeon CPU (E5-2660 v3, 2.60 GHz).

To evaluate the FIST framework, we focus on the Thingsboard IoT platform. Additional results (i.e., including Eclipse hawkBit) are detailed in the extended report (the link will be available in the camera ready version).

Thingsboard. ThingsBoard[22] is an open-source IoT platform that supports multiple features, including various industry-standard communication protocols (e.g., MQTT, CoAP, and HTTP). We explore two distinct configurations of the ThingsBoard platform. The first configuration uses in-memory file storage, while the second uses Apache Kafka[23, 1], an open-source distributed message broker. Furthermore, our implementation considers the use of an HTTP communication channel between the virtual devices and the IoT platform. Other supported protocols can be enabled as well.

4 Experimental results

In this section, we present the outcomes of our experiments evaluating Thingsboard using the FIST framework. Additional experiments (including Eclipse hawkBit) are available in the extended report.

4.1 Thingsboard results

Connectivity capabilities. In Table 1, we investigate the connection capabilities of the Thingsboard IoT platform across various Thingsboard configurations and network conditions. Using the network simulation increases the successful connections, albeit associated with a minor increase in task duration. This result is likely due to the impact of network delays, which extend the overall connection session duration. The increase in task duration in both Thingsboard configurations during realistic network emulation further supports this hypothesis. In certain simulation runs, all $20,000$ devices were able to successfully connect to the platform.

Number of devices. Table 2 summarizes the results of having different numbers of devices for firmware transfer simulations. The duration of the task is primarily influenced by the number of devices under consideration. This observation is consistent with the ThingsBoard documentation[4], which states that a fixed number of devices (e.g., 50) are actively processed at any given moment, while the remaining devices are in a waiting state. While this approach ensures a more balanced load, it comes at the cost of a slower task execution. It is possible that increasing the pool size could increase the task speed. However, more investigation is needed to validate this hypothesis. Furthermore, both configurations of the ThingsBoard platform exhibit a linear relationship between the number of devices and the task duration, with a noticeable impact of the realistic network environment in both configurations.

4.2 *Discussion*

We identify several factors that influence the experimental results above:

Device tasks. The accurate simulation of devices relies on the implementation of the device's primary task execution. It is therefore the developer's responsibility to use a primary task that is close the realistic behavior of the real device.

Additional platform parameters. Our experiments explore the main subset of configurations of the ThingBoard IoT platform. Additional tuning of the remaining parameters could have an impact on the obtained results. To support further experiments, the FIST YAML configuration file allows specifying parameter values to be used by the target IoT platform.

Network interface. For the FIST framework implementation, the simulated devices within a single Fleet Manager Docker container communicate via a single network interface with the IoT platform. This single interface could become a bottleneck in high traffic load scenarios. To avoid this, FIST enables creating a custom number of Docker containers for a simulation, each with a configurable number of simulated devices. This reduces the single interface bottleneck, as it distributes the total number of simulated devices across multiple containers.

Table 2 Firmware update duration for Thingsboard, using a 32MB firmware binary.

# of devices	Duration [hh:mm:ss]			
	Memory	Memory+Net.	Kafka	Kafka+Net.
1,000	**00:10:02**	00:11:56	00:10:03	00:13:08
2,000	00:20:09	00:29:41	**00:20:04**	00:26:24
3,000	00:30:14	00:42:29	**00:30:05**	00:39:57
4,000	**00:40:06**	00:54:29	**00:40:06**	00:50:30
5,000	**00:50:07**	01:04:07	00:50:09	01:04:55

[4] https://thingsboard.io/docs/user-guide/install/config/

5 Related work

Multiple studies evaluate the technical features of IoT platforms. A framework for evaluating the high-level features and design of an IoT platform was proposed by Mazhelis et al. [15]. Mineraud et al. [17] conducted a comprehensive gap analysis of 39 platforms, identifying gaps and providing recommendations for enhancing platform performance. Guth et al. [9] compared the architectures of various IoT platforms and introduced a reference architecture. Thangavel et al. [21] developed a middleware that uses MQTT and CoAP and conducted a corresponding performance evaluation.

A cloud-based performance evaluation was conducted by Happ et al. [10] to evaluate publish/subscribe protocols within the context of a smart city. They used data from social weather services, smart car sharing, and traffic monitoring to test the performance of the protocols in each case. Talaminos et al. [20] introduced a software framework called Distributed Computing Framework (DFC) for benchmarking Machine-to-Machine (M2M) protocols in healthcare applications.

Vandikas et al. [24] presented the first performance evaluation of a platform named IoT-Framework. In their experiments, they employed a load generator tool, *Tsung*, to apply varying loads to their platform. Alexey et al. [16] evaluated the performance of the OpenIoT platform in terms of its data ingestion and storage capability. Recent studies [14], [8] have been conducted to evaluate the performance of Fiware. In [8], two Fiware protocols, MQTT and LWM2M, were evaluated. Cardoso et al. [3] conducted a performance evaluation between Fiware and ETSI M2M. Cruz et al. [4] provided a performance evaluation of five open-source IoT platforms using *Jmeter* [11], evaluating the REST API and measuring the error percentage and average response time under different load rates and packet sizes.

Our work introduces a framework for standardizing IoT platform simulations. We configure it to be used with open-source IoT platforms, including realistic task execution and device behavior emulation. We then use our framework to evaluate the performance of these IoT platforms.

6 Conclusion

In this work, we introduced a new framework (FIST) designed and built to facilitate the testing of IoT platforms under different configurations and realistic network conditions. By configuring FIST to test two popular IoT platforms, we have validated its capabilities via realistic use cases. Leveraging FIST's extensibility, future work will target further performance and scalability analysis of different IoT platforms, as well as modelling heterogeneous IoT devices.

References

1. Apache Software Foundation: Apache Kafka. https://kafka.apache.org. Accessed: February 9, 2024
2. Balena, IoT Fleet Management Platform. https://www.balena.io/. Accessed: March 20, 2024
3. Cardoso, J., Pereira, C., Aguiar, A., Morla, R.: Benchmarking iot middleware platforms. In: A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2017 IEEE 18th International Symposium on, pp. 1–7. IEEE (2017)
4. Da Cruz, M.A., Rodrigues, J.J., Sangaiah, A.K., Al-Muhtadi, J., Korotaev, V.: Performance evaluation of iot middleware. Journal of Network and Computer Applications **109**, 53–65 (2018)
5. Direct device integration API, Eclipse Hawkbit. https://eclipse.dev/hawkbit/apis/ddi_api/. Accessed: February 9, 2024
6. Device management federation API, Eclipse Hawkbit. https://eclipse.dev/hawkbit/apis/dmf_api/. Accessed: February 9, 2024
7. Eclipse Hawkbit. https://eclipse.dev/hawkbit/. Accessed: February 9, 2024
8. Estuardo, V.: Performance evaluation of scalable and distributed iot platforms for smart regions. Master's thesis, Lulea University of Technology, Sweden (2017)
9. Guth, J., Breitenbucher, U., Falkenthal, M., Leymann, F., Reinfurt, L.: Comparison of IoT platform architectures: A field study based on a reference architecture. In: Cloudification of the Internet of Things (CIoT), pp. 1–6. IEEE (2016)
10. Happ, D., Karowski, N., Menzel, T., Handziski, V., Wolisz, A.: Meeting iot platform requirements with open pub/sub solutions. Annals of Telecommunications **72**(1-2), 41–52 (2017)
11. Jmeter load generator. https://jmeter.apache.org/ (2018). Online; accessed 23-March-2018
12. Łukasz Lach: docker-tc: Docker traffic control. https://github.com/lukaszlach/docker-tc (2024)
13. Linux man-pages project: tc(8) - Linux man page (2024). URL http://man7.org/linux/man-pages/man8/tc.8.html. Accessed: Feb 12, 2024
14. Martínez, R., Pastor, J.A., Álvarez, B., Iborra, A.: A testbed to evaluate the fiware-based iot platform in the domain of precision agriculture. Sensors **16**(11), 1979 (2016)
15. Mazhelis, O., Tyrvainen, P.: A framework for evaluating internet-of-things platforms: Application provider viewpoint. In: Internet of Things (WF-IoT), 2014 IEEE World Forum on, pp. 147–152. IEEE (2014)
16. Medvedev, A., Hassani, A., Zaslavsky, A., Jayaraman, P.P., Indrawan-Santiago, M., Haghighi, P.D., Ling, S.: Data ingestion and storage performance of iot platforms: Study of openiot. In: International Workshop on Interoperability and Open-Source Solutions, pp. 141–157. Springer (2016)
17. Mineraud, J., Mazhelis, O., Su, X., Tarkoma, S.: A gap analysis of internet-of-things platforms. Computer Communications **89**, 5–16 (2016)
18. OpenRemote, Open Source IoT Platform. https://openremote.io/. Accessed: March 20, 2024
19. Rabbitmq. https://www.rabbitmq.com/. Accessed: February 9, 2024
20. Talaminos-Barroso, A., Estudillo-Valderrama, M.A., Roa, L.M., Reina-Tosina, J., Ortega-Ruiz, F.: A machine-to-machine protocol benchmark for ehealth applications–use case: Respiratory rehabilitation. Computer methods and programs in biomedicine **129**, 1–11 (2016)
21. Thangavel, D., Ma, X., Valera, A., Tan, H.X., Tan, C.K.Y.: Performance evaluation of mqtt and coap via a common middleware. In: Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on, pp. 1–6. IEEE (2014)
22. Thingsboard, open-source IoT platform. https://thingsboard.io/. Accessed: February 9, 2024
23. Thingsboard Kafka integration guide. https://thingsboard.io/docs/user-guide/integrations/kafka/. Accessed: February 9, 2024
24. Vandikas, K., Tsiatsis, V.: Performance evaluation of an iot platform. In: Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on, pp. 141–146. IEEE (2014)