LEM: a Tool for Large-Scale Workflow Control in Edge-based Industry 5.0 Applications

Rui Reis[*], Pedro M. Santos[†§], Mário J. Sousa[*†], Nuno Martins[‡], Joana Sousa[‡], Luis Almeida[*†]
[*] *Universidade do Porto – Faculdade de Engenharia, Portugal*
[†] *Instituto Superior de Engenharia do Porto (ISEP), Portugal*
[§] *CISTER Research Center in Real-Time & Embedded Computing Systems, Portugal*
[‡] *NOS Inovação, Portugal*
up201909552@edu.fe.up.pt, pss@isep.ipp.pt, msousa@fe.up.pt, {nuno.mmartins, joana.sousa}@nos.pt, lda@fe.up.pt

***Abstract*—Edge computing is poised to greatly boost Industry 5.0 by providing a computing substrate that can run highly specialized applications with minimal latency, reducing the requirements on computational resources at the end-user and allowing the sharing of resources between multiple end-users. The intended services and workflows need to be designed, configured and deployed at target edge devices, and during operation their status needs to be monitored. There are several industrial frameworks that provide such functionalities, but often there is little support to execute them simultaneously over many target devices. In this paper we present the LEM tool, a tool that enables the management (monitor, configure, deploy) of workflows in a large number of edge devices. The tool connects to existing industrial frameworks to provide these functionalities, abstracting the low level configuration aspects of each framework and enabling simultaneous configuration of multiple devices, a feature that many frameworks do not support. The tool offers a web-based GUI that allows the infrastructure manager to conveniently check the status of all devices and flows running in each device, deploy new flows on selected devices, and start and stop flows. Experimental measurements show residual response time overhead as the number of devices increases.**

***Index Terms*—Industry5.0, edge, workflow, Node-RED**

I. Introduction

IoT and edge devices enable new applications and have been experiencing widespread deployment. Edge devices, being deployed close to the end-user, can host services that end-user devices do not have the computational power to support, with a latency inferior to that of the cloud [1]. In certain scenarios IoT devices can be deployed in great numbers to provide one or more services, e.g., smart water meters or routers for Internet provisioning at customer premises. The management of such large sets of similar devices involves monitoring the status of said devices and hosted service(s), and ultimately deploy, update or stop the services in the edge devices. For example, an Internet Service Provider (ISP) wishing to provide a new service to its customers, e.g. cyber-attack detection, needs to update the Customer-Premises Equipment (CPE) at all customers. It entails that conveniently managing such large set of devices poses challenges of its own.

Many recent industrial processes are data-driven and as such workflows to collect and process data and extract actionable information, often hosted at edge devices, become commonplace. Several industrial frameworks have been proposed to design and deploy workflows. We analyzed four reference frameworks, and identify that there is little support for conveniently configuring and deploying workflows over many target devices. In this paper, we describe a tool for **Large-scale Edge node Management (LEM)**, a control software that allows to monitor, configure and deploy similar workflows over a large number of target edge devices. The **LEM tool** is meant to be integrated with an existing industrial framework to provide large-scale management, a functionality often not provided by said frameworks. In this work, we showcase the integration of the LEM tool with Node-RED [2], but the tool could be integrated with other frameworks. The LEM tool offers: (i) instantiate new workflows and manage active ones (add/remove/edit); (ii) list and manage existing devices, and monitor their usage; (iii) manage library of available modules used in the workflows. The system provides an intuitive GUI for performing all the above actions[1]. An evaluation of the response time of the LEM tool in obtaining the status of multiple flows deployed over various devices is reported.

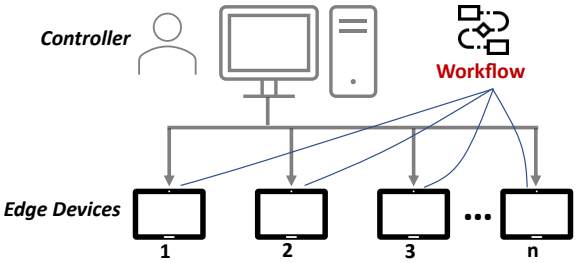


Fig. 1: Workflow management in many edge nodes.

This paper is organized in the following manner. A review of existing industrial frameworks is presented in Section II. Section III describes the LEM tool, and Section IV describes the integration with the NodeRED framework. Evaluation is described in Section V. Section VI presents the conclusions.

II. Overview of Industrial Frameworks

We review selected well-established industrial frameworks concerning their support for large-scale device management. An in-depth comparison can be found in [3].

[1]The GUI of LEM tool can be seen in: https://youtu.be/nK6KzKOGWrw

Arrowhead [4] framework is based on a Service-Oriented Architecture (SOA). Arrowhead proposes the concept of a *local cloud*, a set of connected devices in which at least some core Arrowhead systems are provided. The Arrowhead core systems include a dynamic service association/dissociation system (*service registry*), an access control system (*authorization system*), and dynamic service connection system (*orchestration system*). Arrowhead has been successfully applied to electrical vehicle chargers [5], smart lighting [6], and systems that perform predictive maintenance [7]. The authors of [8] discuss the role of Arrowhead in enabling edge automation services. There is not, to the extent of our knowledge, a tool for large-scale node management.

BaSys 4.0 [9] is a Basic System for production plants that realizes the efficient changeability of a manufacturing process. BaSys design was founded on three central pillars: creation of a structured run-time environment (RTE), process planning and the use of digital twins [3]. The open source software development kit (SDK) platform Eclipse BaSyx [10] (provided in Java, C++ and C#) is an implementation of the BaSys, and offers one of the first implementations of digital twins using Asset Administration Shell (AAS).

FIWARE [11] is an open source platform to create IoT applications, developed in great part to foster a data economy. The key component in any FIWARE deployment is the Context Broker Generic Enabler (CBGE), to which additional components for storing, processing, transmitting, or visualizing data can be connected. Components in the FIWARE framework exchange their data according to the ETSI Next Generation Service Interface (NGSI) information model. FIWARE's focus is on promoting data sharing across systems, and thus it provides no tool for large scale deployment.

Kubernetes [12] is an orchestrator that oversees containerized applications, a form of virtualization. Kubernetes enables the configuration and deployment of applications while automatically managing their life-cycles, storage and service discovery [13]. Its clusters consist of sets of nodes *Master* and *Worker*; the Master contains the API and it is responsible for authorization at an operational level. There are multiple tools for cluster management, such as e.g. Kubernetes Dashboard, that match the scope of the LEM tool but for containers.

Node-RED [2] is an open-source low code programming framework for event-driven applications. It uses flow-based programming, in which nodes – functional blocks – are connected to each other to create workflows. Nodes are written in Javascript, and there is an ample library of nodes supported by a growing community. Nodes communicate via messages that are Javascript objects in binary format. Node-RED is provided by a Node.js runtime that is installed in the target device (e.g., edge device) and provides two main functionalities: (i) to design Node-RED flows through a browser-based interface (by wiring nodes); and (ii) to deploy/run those flows in the host device. Flows can be conveniently exported as JSON files to be executed in a different device that is also running a Node-RED runtime. The runtime also provides an HTTP API for remote management (discussed in detail in Section IV). There is not, to the extent of our knowledge, a Node-RED tool for large-scale node management.

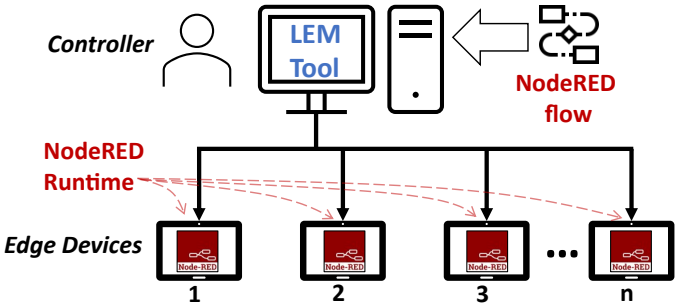


Fig. 2: Intended usage of the LEM tool.

A. Use of Node-RED in this Work

Node-RED was selected as a representative industrial framework to showcase our large-scale management tool according to the following criteria: (1) ease of setup and use; (2) suitability to workflow design and development; and (3) platform requirements. Past experience with Arrowhead indicated us the existence of a steep learning curve to install and use it. While simpler to use, FIWARE is focused on data collection, processing and APIs for sharing data, and thus untailored to the deployment of processing workflows on target edge devices. A similar argument can be made for Kubernetes: although simple to use, it focus on the deployment and management of containers and not strictly workflows. Furthermore, it imposes steep computational requirements for operation. BaSys 4.0 (and its SDK BaSyx) is focused on enabling digital twins and facilitating reconfiguration of manufacturing infrastructure; but little is said about their support for edge devices. All things considered, Node-RED offers simplicity of installation and use, it is tailored to describe and deploy workflows, and presents modest requirements for operation.

As mentioned earler, flows created in a device running the runtime can be exported as a JSON file. One of the main goals of the LEM tool, to deploy the same flow over many devices, can be accomplished by designing said flow in an initial device and exporting it to the target devices. The LEM tool takes charge of all aspects regarding the large-scale deployment of said flow. Figure 2 depicts the vision.

III. Large-scale Edge Management (LEM) Tool

The LEM tool operates as an overlay to an existing industrial framework to enable the management of a large number of edge devices and services/workflows hosted at those devices. The following functionalities are offered by the LEM tool:

- **Monitoring**
- **[Deploy | Edit | Delete] Devices**
- **[Deploy | Edit | Delete] Flows**
- **Device Control [Start | Stop]**

Being an overlay, an interface component connecting the LEM tool and the industrial framework must exist.

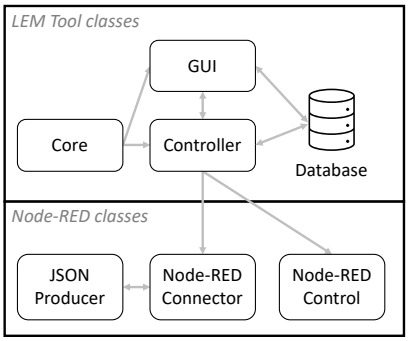


Fig. 3: Simplified software architecture diagram of LEM Tool

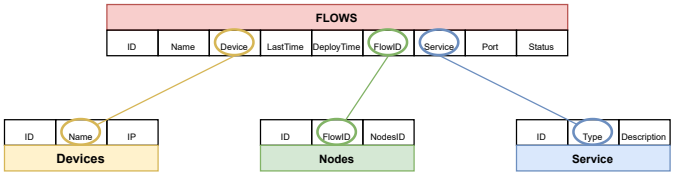
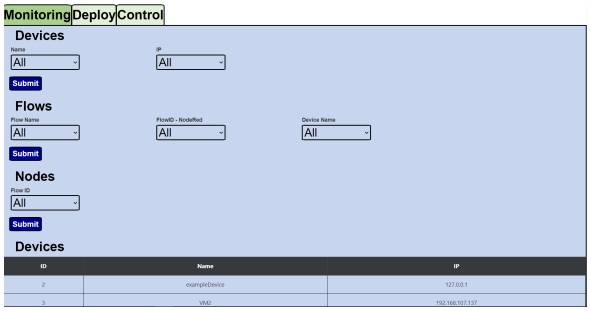
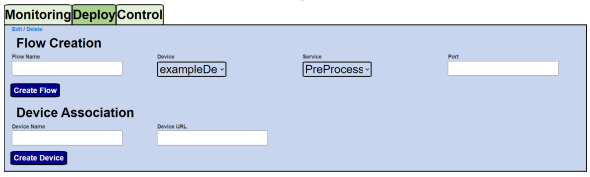


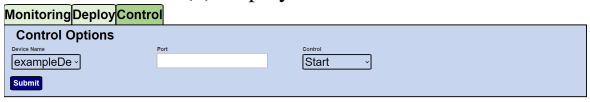
Fig. 4: Database Table Relations



(a) Monitoring tab in GUI



(b) Deploy tab in GUI



(c) Control tab in GUI

Fig. 5: Web-based Graphical User Interface

A. Architecture & System components

The LEM tool has the following main software components: the `Core`, `Database`, `Controller`, `GUI`, `Node-RED connector` and `Node-RED control`. The architecture is shown in Figure 3, with a distinction between the LEM core classes (top part) and those in charge of interfacing the underlying framework (bottom part). We describe the role of the most relevant classes.

`Core`: in charge of periodically updating the DB tables and initiating the Java Spring Boot application.

`Controller`: this component fetches data from the database to update the information displayed in the GUI. It can also command the deployment of a service or its deletion.

`Database`: keeps multiple tables to store all information regarding flows, devices and nodes. The information model is described in the next subsection.

`Graphical User Interface (GUI)`: presents status information of all devices and flows, and allows the user to deploy new Node-RED flows at each device, to edit information of flows or devices, and to delete created flows or associated devices.

`Node-RED Connector & JSON Producer`: enables the connection between to edge devices using the Node-RED API (Section IV). The `Node-RED Connector` class performs POST, PUT and DELETE requests to a target device, while the `JSON Producer` class generates new flows with different node IDs for deployment over multiple devices.

`Node-RED Control`: starts or stops the Node-RED service in edge devices through a remote command (SSH-based).

B. Information Model

The information is stored in three key tables: `Flows`, `Devices`, and `Nodes`; the `Services` table provides support. Their structure is the following.

- **Flows** fields: *flow ID*, flow *Name*, *device name* where the flow is running, *deploy time*, *Last Update Time*, *ID of NodeRED*, *port*, *status* and *service*.
- **Devices** fields: *device ID*, *Name* and *IP*.
- **Nodes** fields: *node ID*, *flow ID* and *ID* of all nodes belonging to the flow.
- **Services**: *ID*, *service type* and *description*.

These tables are tightly related in order to keep a consistent representation of the existing devices and deployed workflows. Figure 4 depicts table relationships. When a flow is created, its name and ID, hosting device, deploy time, service and port are instantly saved in the `flows` table. The `nodes` table keeps a record of all individual nodes and associates each node to the flow it belongs to. The `devices` table stores the *Name* and *IP* attributed to a machine.

C. User Interface

The interface, coded in Java Spring Boot, has three tabs: `Monitoring`, `Deploy` and `Control`.

Monitoring tab (Fig. 5a): provides status of devices and flows upon selection by user.

Deploy tab (Fig. 5b): allows the user to create a new flow on an edge device, or register a new device. Under the *flow creation* section, the user inputs the flow name, target device, service type, and port to use. The *device association* section enables the user to assign a name and an IP to a new device.

Control tab (Fig. 5c): it is possible to start or stop any instance of Node-RED at will on any associated device. In this tab, the user can select the device, the port where the instance should be run (or is currently running), and the command to execute (over a SSH connection).

D. Functionalities

The functionalities of the LEM tool, listed at the start of this section, are now revisited in more detail.

1) **Monitoring***:* To collect the status of devices and flows, **the LEM tool creates an thread per device in order to parallelize collection of monitoring data**. Each thread fetches information from a single device, checking the status of the Node-RED service in that device and the last instant the service was run. This allows for scalability, as it enables that additional devices impact marginally the interval it takes for the LEM Tool to collect the status information from all edge devices. Once this information is obtained, the threads update the `Database`. This process is set to occur every ten seconds.

2) **[Deploy | Edit | Delete] Flow***:* The user submits the desired flow name, the service, the device where he wants to deploy, and the port of the device. Once this information is submitted, the `Controller` checks if this information already exists in the `Database`. If not, the `Controller` checks if the device's runtime environment is operational and, if so, it deploys the flow in the device. In the case of Node-RED, this means that the JSON description of the flow is sent using the HTTP API of the device's Node-RED runtime. If successful, the runtime replies with the ID of the created flow and the `Controller` saves the information in the `Database`. The `GUI` is updated to confirm the flow was correctly deployed. If the connection with the device fails, the `Controller` sends a message informing that the Node-RED service in the device is down.

To delete a flow, the user selects a flow and the `Controller` fetches the information associated to the chosen flow. As before, it is checked if the Node-RED service in the device is working properly. If so, a delete command is sent to the device, that in turn sends a reply confirming the deletion. Upon reception by the LEM tool, the entry in the `Database` is deleted and a confirmation is presented in the `GUI`, to provide visual feedback to the user.

3) **[Deploy | Edit | Delete] Device***:* In device deployment, the user submits a name and IP address in the `GUI`. This information is sent to the `Controller`, that in turn confers with the `Database` to check if a device with the same name or same URL exists. If not, it saves the information and sends a deployment confirmation to the `GUI`. If it does, it provides a message informing that a device with the same name or same URL already exists.

4) **Control***:* The *Control* functionality allows the user to start or stop the Node-RED runtime in the target device. The user selects the *device*, *port* and control option (*Start* or *Stop*). Once the request is submitted, the `GUI` communicates with the `Controller` to carry out the request. If the service is running (or inactive) and the user requests terminating (or starting) it, the `Controller` sends a `SSH` (Secure Shell) command to the device, killing (or starting) the Node-RED runtime process *ID* of that device. In case the service is already running or inactive, an error message is printed in the GUI warning the user.

Endpoint		Description
GET	/auth/login	Get the active authentication scheme
POST	/auth/token	Exchange credentials for access token
POST	/auth/revoke	Revoke an access token
GET	/settings	Get the runtime settings
GET	/flows	Get the active flow configuration
POST	/flows	Set the active flow configuration
POST	/flow	Add a flow to the active configuration
GET	/flow/:id	Get an individual flow configuration
PUT	/flow/:id	Update an individual flow configuration
DELETE	/flow/:id	Delete an individual flow configuration
GET	/nodes	Get a list of the installed nodes
POST	/nodes	Install a new node module
GET	/nodes/:module	Get a node module's information
PUT	/nodes/:module	Enable/Disable a node module
DELETE	/nodes/:module	Remove a node module
GET	/nodes/:module/:set	Get a node module set information
PUT	/nodes/:module/:set	Enable/Disable a node set

Fig. 6: Node-RED API Methods (adapted from [14])

IV. Integration with Node-RED

The communication concerning monitoring and management of Node-RED workflows (by the LEM tool at a central command device) and the target (edge) devices running a Node-RED runtime happens through two mechanisms: the Node-RED HTTP API, and dedicated Node-Red nodes for monitoring. We describe both in the following sections respectively.

A. Node-RED HTTP API

The Node-RED API [14] provides a variety of HTTP methods to monitor/control the runtime's operation, as listed in Figure 6. The LEM tool's `Controller` component generates the necessary code to be posted to the endpoint.

We focus first on the methods concerning the retrieval of the list of flows and of the current flow, and the deployment of flows. The user can request the JSON code of all flows running in the edge node through a HTTP GET method; an example is shown (with port `1880` being default for Node-RED runtime):

```
GET <IP target device>:1880/flows
```

The following GET request provides the information of only one flow; e.g., for flow with ID 51638b50621f7dd3:

```
GET <IP target device>:1880/flow/51638b50621f7dd3
```

To create a flow, we access the endpoint */flow* with a *POST* method:

```
POST <IP target device>:1880/flow
```

In its message payload, the JSON describing the intended flow should be inserted. If the *POST* is successful, the Node-RED runtime returns the ID of the created *flow*:

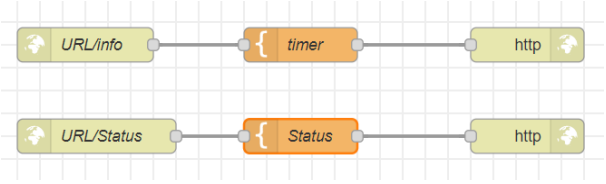


Fig. 7: Additional nodes to monitor workflow status

```
{
  "id": "51638b50621f7dd3"
}
```

However, for the purposes of the LEM tool, this procedure is not directly applicable. **A major limitation of Node-RED is that it does not allow to force a specific ID on a flow.** What happens is that the Node-RED runtime at the target device assigns an arbitrary ID to each newly created flow (when the above *POST* command is executed). This is an issue as it difficults the association, in the database, of a flow to the nodes that compose it. To circumvent this issue, we first perform the *POST* command with a void flow, to trigger the assignment of a flow ID. Upon reception of the flow ID from the target device, we generate random IDs for all nodes in the flow; this is done by adding five random characters in front of the received flow ID. An example of a resulting node ID is:

```
51638b50621f7dd3.PX1CI
```

Once this is done, we use the *PUT* method to inject the complete JSON code, updated with the new node IDs, in the Node-RED runtime of the edge device. The LEM tool accesses the following endpoint to deploy the flow properly.

```
PUT <IP target device>:1880/flow/51638b50621f7dd3
```

We observed some more limitations of the Node-RED HTTP API: (i) it is not possible to import a new flow code if there is one already with the same ID on a node; (ii) if the PUT method is used incorrectly, the runtime completely bugs and does not allow the user to delete the bugged flow manually.

B. Monitoring Flow Status & Last Instant of Execution

The Node-RED API does not offer in-detail information on the status of flows, e.g., the flow status or when was the workflow last executed. To get access to the last instant of execution and flow status, we prepared two sub-flows that can be added to existing Node-RED flows (Figure 7). These two sub-flows include HTTP servers, and the LEM tool can fetch the relevant data using a HTTP *GET* method. The URLs to be accessed would be (respectively):

```
<IP target device>:1880/flow/51638b50621f7dd3/info
```

```
<IP target device>:1880/flow/51638b50621f7dd3/status
```

To retrieve the value of any variable in Node-RED, variables can be made global to be accessible to any flow and *POST* this variable in a new URL.

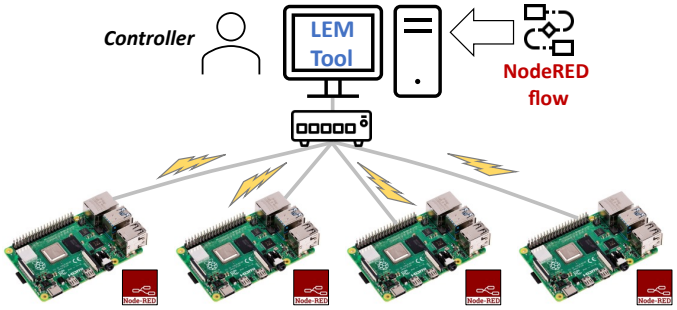


Fig. 8: Experimental setup

V. Evaluation

In this section we report the response time of the LEM tool while collecting the relevant monitoring data. Being a tool tailored to large-scale management, it is important to prove its scalability by showing that response times remain relatively stable as the number of devices increases. We used four physical devices (Raspberry Pi) with the Node-RED runtime installed, managed by an instance of the LEM tool installed in a laptop, to measure latency samples over a local network using two media (Ethernet and WiFi).

A. Workflow for Anomaly Detection in Network Traffic

A representative workflow was deployed and replicated at each device to induce load on the target devices and on the LEM tool's mechanisms. This workflow captures network traffic of the edge device and classifies it as anomalous or not using machine learning models. The workflow uses Python and third-party tools, as explained in detail in [16], and its NodeRED description is shown in Figure 9, following the mapping process described in [15]. The workflow is initiated when the left-most block in the flow identified by 1 in Figure 9 is triggered manually. Doing so initiates the third-party tool `Tstat`, that collects network traffic and outputs a set of characteristics for each observed packet flow in that traffic into an output file. A support flow monitors that output file for changes (2), and when a new entry is detected, it is read by a Python script that performs the ML prediction (3). The sub-flows for flow monitoring (Section IV-B) were added.

B. Experimental Results

We use a testbed composed of one laptop hosting the LEM tool and four Raspberry Pis with NodeRED installed. All devices are connected in a local network via Ethernet and WiFi. We carried out two sets of tests, by having communication between laptop and Raspberries at the local network take place over: (1) Ethernet; (2) WiFi. The setup can be seen in Figure 8.

We evaluate the time the LEM tool takes to collect the status of all flows deployed at all devices, for 1, 5, 10 and 20 flows. Figure 10 present the results for the wired (Ethernet) setup. We observe that more flows lead to higher response times, but as we increase the number of devices, delay remains consistent. This is due to the multiple-thread approach of the *LEM tool* to probe the flow status at different devices.

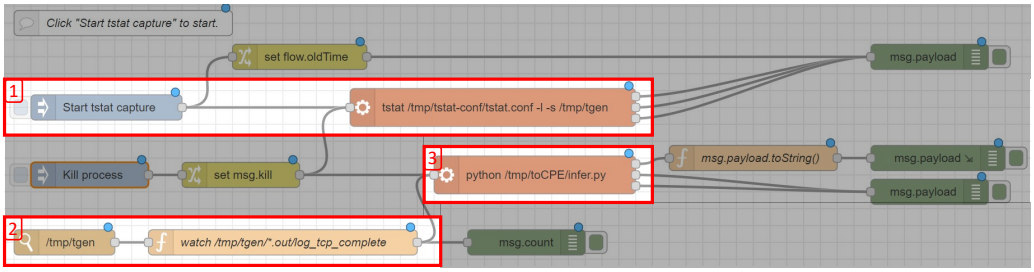
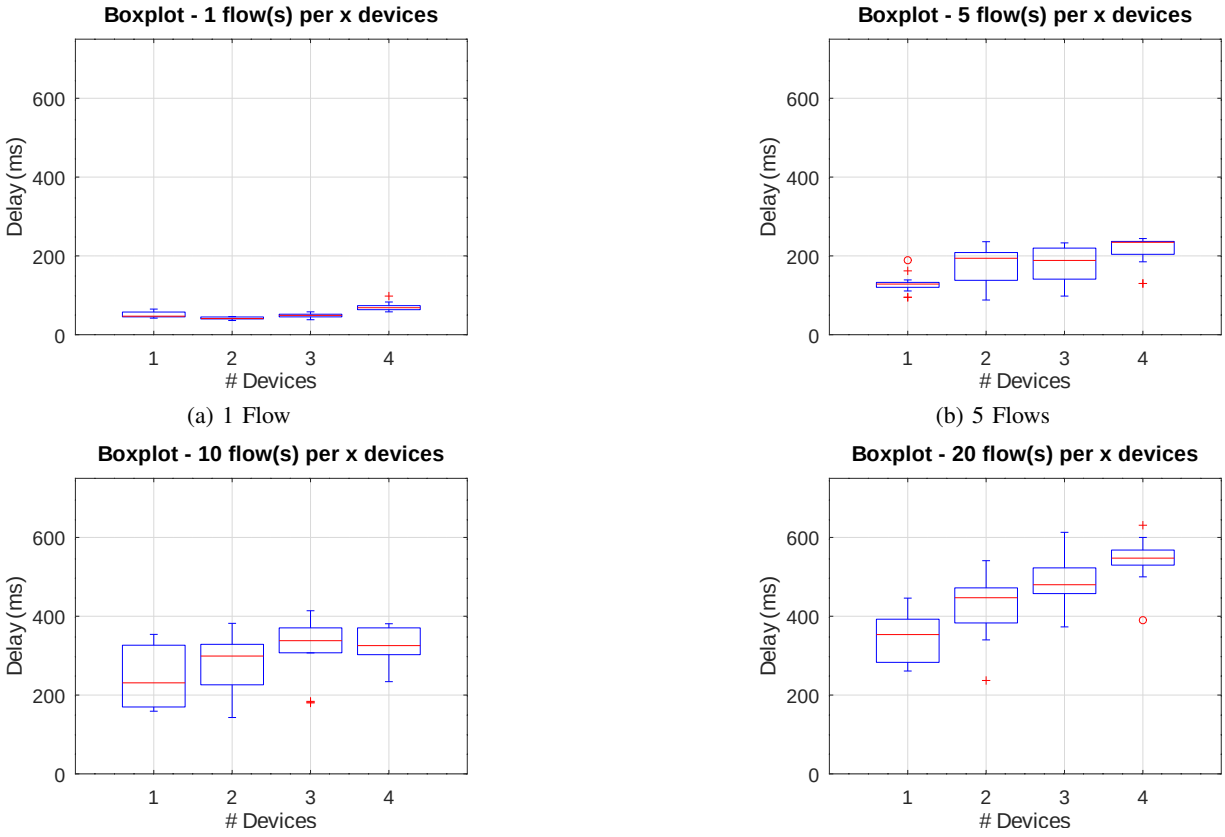


Fig. 9: Node-RED Representation of Workflow for Anomaly Detection in Network Traffic (as in [15])



(a) 1 Flow (b) 5 Flows

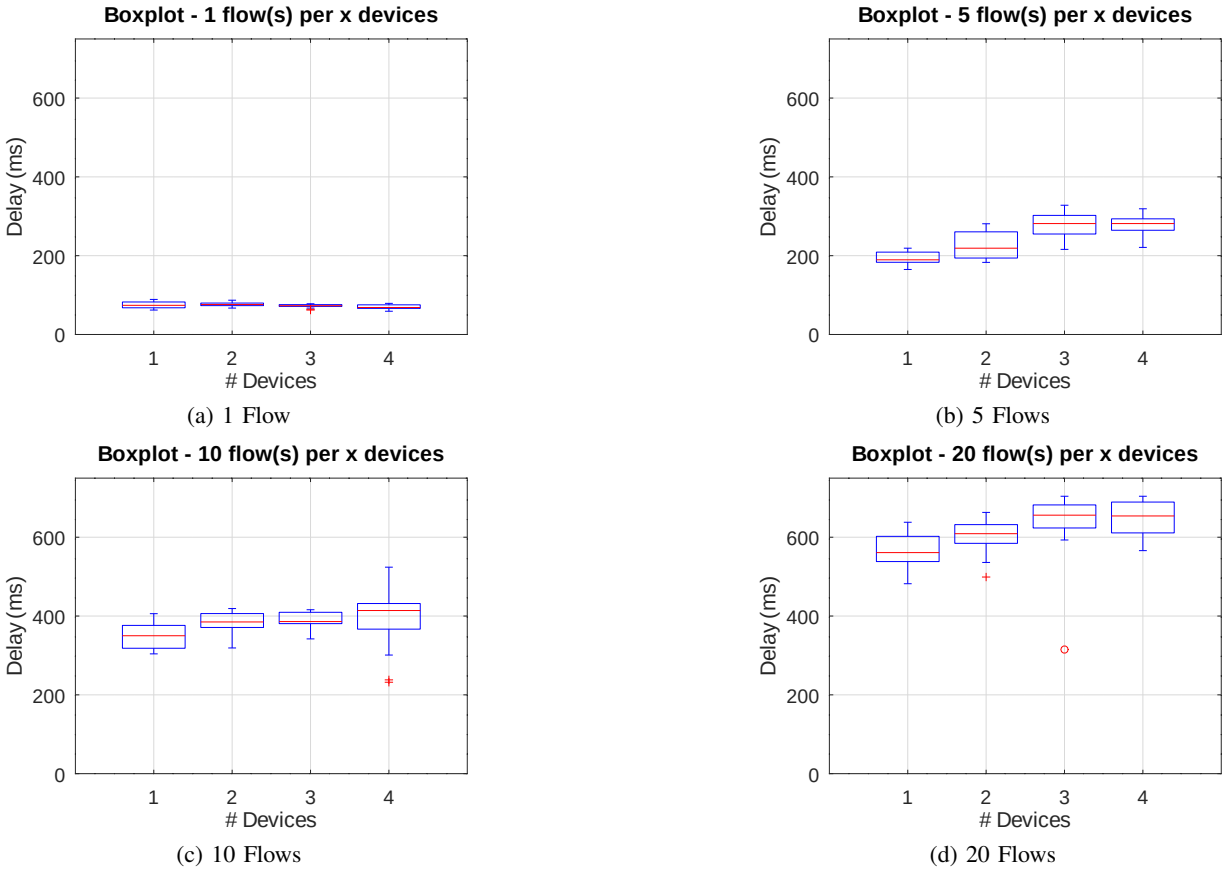
(c) 10 Flows (d) 20 Flows

Fig. 10: Response time for collecting monitoring data - Ethernet LAN connection

Figure 11 presents the results for the wireless setup. We observe higher latency values than in the wired case (higher medians), but an overall higher consistency of the collected samples (range between 2nd and 3rd quartiles is inferior in most cases). The higher latency can be due to the larger susceptibility of the wireless medium to congestion. As with the wired setup, we observe that more flows lead to higher latency, but increased number of devices results in a residual increase. Thus, similar properties of the LEM tool are observed over different media.

VI. Conclusion

We present the LEM tool, a tool for managing (monitor, configure, deploy) workflows over a large number of edge devices. The tool connects to existing industrial frameworks to provide these functionalities, abstracting the low level configuration aspects of each framework and enabling simultaneous configuration of multiple devices, a feature that many frameworks do not support. The tool offers a web-based GUI that allows the infrastructure manager to conveniently check the status of all devices and flows running in each device, deploy new flows on selected devices, and start and stop flows. Experimental measurements show residual response time overhead as the number of devices increases, which is attained by design as inquiries to each device are done in parallel (distinct threads). In turn, an increase in response time is observed as each device is running many flows, which can be attributed to the sequential nature of inquiries about each flow per node. Next steps involve supporting more frameworks and improving the information display at the GUI.



(a) 1 Flow (b) 5 Flows

(c) 10 Flows (d) 20 Flows

Fig. 11: Response time for collecting monitoring data - WLAN connection

ACKNOWLEDGEMENTS

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDB/04234/2020), and by the Portuguese National Innovation Agency (ANI) through the Operational Competitiveness Programme and Internationalization (COMPETE 2020) under the PT2020 Partnership Agreement, through the European Regional Development Fund (ERDF), within project(s) grant POCI-01-0247-FEDER-069522 (MIRAI). The authors also thank Nuno Schumacher, Yimin Zhang and Barikisu Asulba for the example workflow and its representation in Node-RED.

REFERENCES

[1] A. Hristoskova, N. González-Deleito, S. Klein, J. Sousa, N. Martins, J. Tagaio, J. Serra, C. Silva, J. Ferreira, P. M. Santos, R. Morla, L. Almeida, B. Bulut, and S. Sultanoğlu, "An Initial Analysis of the Shortcomings of Conventional AI and the Benefits of Distributed AI Approaches in Industrial Use Cases," in *Artificial Intelligence Applications and Innovations. AIAI 2021 IFIP WG 12.5 International Workshops*. Cham: Springer International Publishing, 2021, vol. 628, pp. 281–292.

[2] "Node-RED." https://nodered.org/, accessed: 2023-04-24.

[3] C. Paniagua and J. Delsing, "Industrial Frameworks for Internet of Things: A Survey," *IEEE Systems Journal*, vol. 15, no. 1, pp. 1149–1159, Mar. 2021.

[4] J. Delsing, Ed., *IoT Automation: Arrowhead Framework*. Boca Raton: CRC Press, Feb. 2017.

[5] B. Peceli, G. Singler, Z. Theisz, C. Hegedus, P. Vargaz, and Z. Szepessy, "Integrating an Electric Vehicle Supply Equipment with the Arrowhead framework," in *42nd Conf. of IEEE Industrial Electronics Society (IECON)*, Florence, Italy, Oct. 2016, pp. 5271–5276.

[6] J. Jokinen, T. Latvala, and J. L. Martinez Lastra, "Integrating smart city services using Arrowhead framework," in *42nd Conf. of IEEE Industrial Electronics Society (IECON)*, Florence, Oct. 2016, pp. 5568–5573.

[7] B. Bulut, H. Burak Ketmen, A. S. Atalay, O. Herkiloglu, and R. Salokangas, "An Arrowhead and Mimosa Based IoT Framework with an Industrial Predictive Maintenance Application," in *2021 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*. Kocaeli, Turkey: IEEE, Aug. 2021, pp. 1–5.

[8] H. Derhamy, M. Andersson, J. Eliasson, and J. Delsing, "Workflow management for edge driven manufacturing systems," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. St. Petersburg: IEEE, May 2018, pp. 774–779.

[9] "BaSys 4.0," https://www.basys40.de/, accessed: 2023-04-24.

[10] "BaSyx," https://wiki.eclipse.org/BaSyx, accessed: 2023-04-24.

[11] "FIWARE," https://www.fiware.org/, accessed: 2023-04-24.

[12] "Kubernetes," https://kubernetes.io/, accessed: 2023-04-24.

[13] L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, and M. Kihl, "Impact of etcd deployment on Kubernetes, Istio, and application performance," *Softw Pract Exper*, vol. 50, no. 10, pp. 1986–2007, Oct. 2020.

[14] "Node-RED: Api admin methods." https://nodered.org/docs/api/admin/methods/, accessed: 2023-04-24.

[15] Y. Zhang, B. Asulba, N. Schumacher, M. Sousa, P. Souto, L. Almeida, P. M. Santos, N. Martins, and J. Sousa, "Implementing and Deploying an ML Pipeline for IoT Intrusion Detection with Node-RED," in *Real-Time And intelliGent Edge Computing (RAGE) Workshop*. San Antonio, TX, USA: IEEE, May 2023.

[16] N. Schumacher, P. M. Santos, P. F. Souto, N. Martins, J. Sousa, J. M. Ferreira, and L. Almeida, "One-Class Models for Intrusion Detection at ISP Customer Networks," in *IFIP Advances and Innovations in Artificial Intelligence*, León, Spain, Jun. 2023.