Towards a Scalable IIoT Solution for Factory Monitoring

Igor S. de Brito[1], Antonio Gabriel da S. Fernandes[1], Guilherme G. de F. Salvo,[1], Rafael S. R. Alves[1], Thomas E. Maia[1], Fernanda E. C. Chaves[1], Marcelo S. Previti[1], Flávia Pisani[1], Juliana F. Borin[1]

[1]Universidade Estadual de Campinas (UNICAMP) – Campinas, São Paulo, Brazil

igor.brito@students.ic.unicamp.br, {fpisani,juliana}@ic.unicamp.br

Abstract. *With the declining cost of computer chip production, they have been increasingly integrated into everyday appliances and devices, giving rise to the Internet of Things. In an industrial context, these devices improve control and productivity in a production line. This paper evaluates the scalability of connections of those devices to an edge gateway, as well as the scalability of connections of edge gateways to a server platform in a simulated industrial setting.*

1. Introduction

Initially used in 1999 [Rose et al. 2015], the term Internet of Things (IoT) refers to devices connected to each other in different environments, such as residential, commercial, and industrial. Furthermore, there is currently a great process of automation through intelligent monitoring, especially of the machinery present in factories, giving rise to what is called the fourth industrial revolution (Industry 4.0) [Boyes et al. 2018]. This term is considered by some authors as a synonym for Industrial Internet of Things (IIoT) [Hermann et al. 2016]. More clearly, Hermann et al. define Industry 4.0 as a collective term for technologies and concepts of organization of the production chain, with the monitoring of physical processes and decentralized decision-making to increase industrial efficiency, productivity, security, and transparency. The term IoT similarly refers to modules that communicate and cooperate with each other and with humans in real-time.

Therefore, aiming to improve the productivity of a factory, it is possible to establish a reference architecture of IIoT scenarios commonly organized in three abstraction layers: edge, platform, and enterprise. The edge layer includes sensors, actuators, and controllers connected via the local network to an edge gateway. This edge gateway connects to the platform layer, which is responsible for receiving the data collected in the edge layer, transforming, and processing it. Finally, the enterprise layer implements the applications, business logic, and user interfaces [Salhaoui et al. 2019].

We can implement a solution following this architecture in already-existing factories. For instance, the edge layer can be created by equipping production lines with small Wi-Fi-capable sensors that collect information about each machine (e.g., electric current, temperature, vibration). In a factory with many production lines, each line can have its own dedicated edge gateway, responsible for performing lightweight operations (e.g., parsing, filtering, and aggregation) and then sending the clean data to the platform layer. The platform layer can be implemented with approaches varying from an on-site

This paper's results were obtained through the project "An intelligent IIoT platform for factory monitoring," funded by Samsung Eletrônica da Amazônia Ltda., in the scope of the Informatics Law #8.248/91.

server to an external cloud, depending on the solution's processing and security requirements. By centralizing the data from all production lines, this layer can perform more robust data analysis operations considering the whole context of the factory.

This paper analyzes the scalability of a solution that implements the edge layer with a Raspberry Pi acting as a gateway dedicated to a production line and implements the platform layer with an on-site server.

2. Edge Layer Scalability

In this section, we assess the performance of a Raspberry Pi gateway receiving messages at increasing rates. For each value of messages per second, starting at 100 up to 1600, we measured the CPU and RAM usage, the temperature of the Raspberry Pi's System on a Chip (SoC), and the overall message throughput at the gateway. To do so, a Python script running on a separate computer sent a specified number of messages per second for one minute. The messages were sent via Wi-Fi using MQTT, a lightweight, publish-subscribe network protocol, to a broker running on the gateway. They were then passed on to a client, also running on the gateway, which pre-processed them, checking if the current message received for each topic differed from the previous one, and logged them in a CSV file. We executed the tests five times for each value of messages sent per second. The results presented are the average and standard deviation of these executions.

2.1. Experimental Setup

In this experiment, we used a Samsung X55 laptop with 16 GB of RAM and an Intel i7 10th Gen processor to run the publisher client. The gateway used to run the subscriber client was a Raspberry Pi 4 Model B with 8 GB of RAM and a 32 GB SD Card. Both devices were connected to an AX3000 Dual Band Network Gigabit router in the same 2.4 GHz Wi-Fi network. We monitored CPU usage, RAM usage, and SoC temperature using the RPi-Monitor[1] executing in the gateway.

We used the Paho MQTT module[2] to implement the publisher and subscriber clients. This module implements a receiving buffer to store incoming messages until they are processed, allowing us to run tests with a higher count of messages per second than the maximum throughput of the Raspberry Pi. We employed the Eclipse Mosquitto MQTT broker[3] and opted to execute it on the gateway instead of a separate device to reduce the communication overhead between the subscriber and the broker.

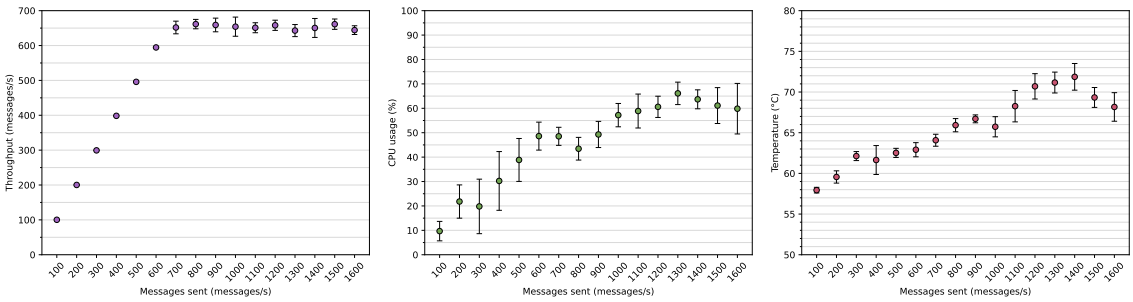
2.2. Experimental Evaluation

In Figure 1a, we can see the throughput of the messages received by the gateway. Initially, the gateway can keep up with the influx, but it reaches its limit at $\sim$650 messages/second, when it starts to delay processing all the messages. Still, no information is lost due to the Paho MQTT client buffer that stores all the messages received by the broker.

In Figure 1b, we can see the CPU load increasing as the rate of messages goes up. However, even at maximum throughput, the Raspberry Pi's CPU usage was never close to 100%, meaning that the hardware's processing power was never used at its total capacity.

[1] https://github.com/XavierBerger/RPi-Monitor
[2] https://www.eclipse.org/paho
[3] https://mosquitto.org

Thus, it might be possible to improve the gateway's performance using multi-threading techniques. The RAM usage remained stable during the tests, consistently averaging around 325 kB, which is almost negligible to the Raspberry Pi's 8 GB of RAM. As seen in Figure 1c, the temperature rises as the rate of messages increases but never reaches a dangerous level for the system. We note that no cooler was used during the tests.



(a) Gateway throughput. **(b) CPU usage.** **(c) SoC temperature.**

Figure 1. Edge layer experimental results.

3. Platform Layer Scalability

In this section, we aim to test the performance of a platform layer server and evaluate the use of the HTTP and MQTT communication protocols for connecting it to an edge gateway. First, we systematically increased the size of a transmitted payload, measuring the time spent during the dispatch and verifying the receipt of data on the server platform. Then, we simulated the accumulation of 65 000-byte payloads on the device for burst transmissions, sending a new request only after the previous request concluded. The results presented in this section are the average and standard deviation of the measurements made using 100 messages for each payload size.

3.1. Experimental Setup

We employed two laptops to carry out the experiments: one with 4 GB of RAM and an Intel i3 3rd Gen processor running the ThingsBoard[4] application acting as the server, and the other with 8 GB of RAM and an Intel i5 8th Gen processor acting as the gateway.

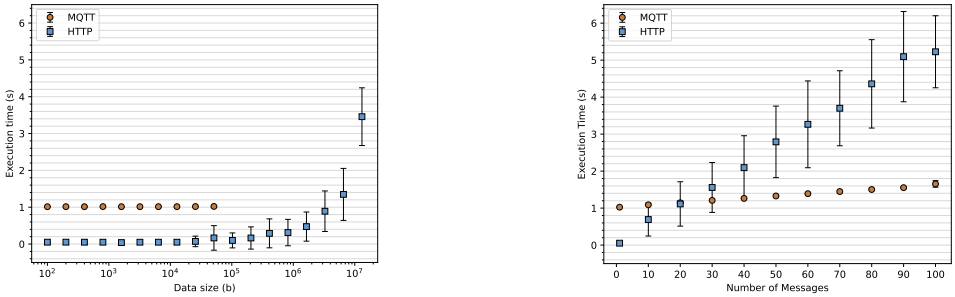
To emulate the gateway sending messages to a server, we developed a Python script to send fixed-sized payloads in sequence, using either HTTP or MQTT, then check if the request succeeded. The server has a PostgreSQL database, and we implemented a memory queue service for the messages.

3.2. Experimental Evaluation

In our experiments, we observed that MQTT has a smaller upper boundary for its payload size than HTTP due to the maximum allowed size of a topic string. Thus, as shown in Figure 2a, the only option for sending payloads above 65 507 bytes is HTTP, which was able to transmit up to 13 107 200 bytes in a single message, albeit at the cost of a longer transmission time. In Figure 2b, we noticed that HTTP's transmission time increases more rapidly than MQTT's as the number of messages grows. Despite having a higher initial cost, MQTT scales better when messages are accumulated to be transmitted in bursts, as

[4]https://thingsboard.io

it can reuse a single connection to send multiple messages. Therefore, HTTP performs better with up to 20 messages in a burst scenario, but above that, MQTT outperforms it.



(a) Time vs. payload size. **(b) Time vs. payload accumulation.**

Figure 2. Platform layer experimental results.

4. Conclusions and Future Work

Based on our experiments, we conclude that our Raspberry Pi edge gateway solution remains stable when receiving large amounts of data, with a maximum throughput of approximately 650 messages/second. This is suitable for an IIoT scenario where a gateway receives the data collected in a production line. In the future, we intend to investigate further the impact that the Paho MQTT message buffer, the network throughput, and the gateway processing power may have on this bottleneck. Furthermore, considering that the CPU usage never reached 75% and that the hardware temperature remained at acceptable levels in all tests, we plan to explore multi-threading in the gateway. Lastly, given that the memory usage was low, it could be interesting to explore the use of cheaper Raspberry Pi models with less RAM than the one we employed in these experiments.

Moreover, we observed that the server platform presented good performance, with MQTT being suitable for an IIoT scenario where a gateway sends bursts of small messages to the server. We emphasize that we used the basic default installation configurations for ThingsBoard, and the experiments were conducted on a personal laptop instead of a more powerful server machine. Therefore, possible next steps are installing the platform on a server, adding robustness to the system by using a Cassandra database and Kafka messaging service, and simulating a larger number of gateways. This way, we can analyze the platform's performance in a more suitable setup for real industrial environments and consider the impact of factors other than the data transmission time.

References

Boyes, H., Hallaq, B., Cunningham, J., and Watson, T. (2018). The industrial internet of things (IIoT): An analysis framework. *Comput. Ind.*, 101:1–12.

Hermann, M., Pentek, T., and Otto, B. (2016). Design Principles for Industrie 4.0 Scenarios. In *Proc. 49th HICSS*, pages 3928–3937.

Rose, K., Eldridge, S., and Chapin, L. (2015). The Internet of Things (IoT): An Overview – Understanding the Issues and Challenges of a More Connected World.

Salhaoui, M., Guerrero-González, A., Arioua, M., Ortiz, F. J., El Oualkadi, A., and Torregrosa, C. L. (2019). Smart Industrial IoT Monitoring and Control System Based on UAV and Cloud Computing Applied to a Concrete Plant. *Sensors*, 19(15).