A Proposal for Bridging the Message Queuing Telemetry Transport Protocol to HTTP on IoT Solutions

Mauro A. A. da Cruz[1,2], Joel J. P. C. Rodrigues[1,3,4], Ellen S. Paradello[1],
Pascal Lorenz[5], Petar Solic[6], Victor Hugo C. Albuquerque[4]

[1] National Institute of Telecommunications (INATEL), Santa Rita do Sapucaí-MG, Brazil
[2] Instituto Superior de Tecnologias de Informação e Comunicação (ISUTIC), Angola
[3] Instituto de Telecomunicações, Portugal
[4] University of Fortaleza (UNIFOR), Fortaleza-CE, Brazil
[5] University of Haute Alsace, Colmar, France
[6] University of Split, Split, Croatia

maurocruzter@gmail.com; joeljr@ieee.org; ellensilveira@gec.inatel.br;
lorenz@ieee.org; psolic@fesb.hr; victor.albuquerque@unifor.br

***Abstract*— Internet of Things (IoT) is a promising market and data gathered by IoT devices is highly valuable. The software that handles and stores these data is known as IoT middleware. The devices transfer data to middleware through an application protocol, which can be different from those supported by middleware. This paper proposes an application layer gateway that converts Message Queuing Telemetry Transport Protocol (MQTT) messages into HTTP. This solution can be deployed on a computer or Raspberry Pi, allowing devices to seamlessly send data to any REST endpoint. Instead of sending data directly to a middleware, IoT devices can send a smaller message to this bridge, which reconstructs it and forwards to a middleware, reducing the stress on the IoT device. The graphical user interface allows users to configure aspects related to messages conversion and forwarding in runtime. The paper demonstrates the efficiency of this approach by evaluating three scenarios where data is sent to Orion context broker (a Fiware project), which reveals that packet size that is sent by an IoT device through this proposed approach is 10 times smaller than other bridges and 17 times smaller than sending an HTTP request straight to the server.**

Keywords*— *Application layer; Bridge; Gateway; Protocol; Internet of things; IoT; Message Queuing Telemetry Transport Protocol

I. INTRODUCTION

Internet of Things (IoT) is a paradigm that intends to connect every object ("thing") to the Internet [1][2]. These objects can belong to the real world (living organisms and inanimate objects) or a virtual world (simulating or representing a real environment) [3]. These objects are connected with the purpose of exchanging and collecting data to make the surrounding environment smarter. Humanity is far from connecting everything (every object). However, the milestone of 30 billion Internet-connected objects is expected to be breached by 2020 [4]. Most IoT devices present reduced size and their resources are constrained (disk space, memory, processing power). Naturally, such a high number of devices rises various challenges, most of them related to their physical size. These challenges include connectivity, communication, security, standardization, user trust, scalability, and much more.

The few resources available for IoT devices contrast with the current Internet, where clients may be desktops, laptops, and smartphones. In the current Internet, efficacy (producing the expected results) is prioritized over efficiency (how the results were achieved). Users barely notice this tradeoff because they possess potent devices, such as laptops and smartphones. IoT favors efficiency because fewer resources are available. In practice, if a developer writes an application for a smartphone with poor code, at most, there will be performance issues that can later be solved. IoT developers cannot take the same approach because a poor code might not even run on the device. Nevertheless, an IoT device is also connected to the Internet. However, the main difference between an IoT device and a Smartphone lies on the fact that primary concern of an IoT device is to consume the least possible resources (i.e., energy and memory) without compromising its objectives. Despite being constrained in terms of resources (by current standards), IoT devices are smaller in terms of size but also more powerful than most early computers.

IoT environments are diverse, with various types of objects, produced by multiple brands. In the current state of IoT evolution, it is common for only partner brands to be compatible with each other (Apple Homekit is an excellent example of this practice). This limitation causes inconvenience for the users because it limits the brands that can be purchased without losing functionalities. IoT consumers should not be concerned about compatibility with other devices when purchasing household items, such as a washing machine or refrigerator. Another problem comes from the fact that it is impossible for a brand to

manufacture every type of IoT devices, which means that sooner or later consumers must purchase incompatible equipment and not use it with fullest potential if the current situation is not reversed.

Ideally, the compatibility/interoperability problem among IoT devices should be solved through a global standard [5]. However, several standards have been proposed, each one with its limitations and advantages. Enforcing a global standard is not a trivial task, and power plugs prove it, since there are different plugs around the world. Another alternative for the interoperability problem is the communication through an IoT middleware. An IoT middleware is a software that not only allows incompatible devices and applications to communicate, but also stores the data collected by such devices, offering support for data interpretation and analysis [5][6]. A middleware is usually installed on a robust server, meaning that data must be transmitted from devices to the server using an application layer protocol. Due to the limitations in IoT, alternative application layer protocols were developed for communications. These alternatives were proposed because the Hypertext Transfer Protocol (HTTP) consumes too many resources [7].

Unfortunately, most IoT middleware merely supports two application layer protocols (HTTP and another one). It means that if one of the devices is compatible with an application protocol that is not supported by the middleware, the user once again is not able to use such device to its fullest potential. Instead of forwarding data directly to the middleware, there is an intermediary device, like a gateway, that translates the message into one of the application protocols supported by the middleware. This intermediary step should be seamless for both sender and receiver, and the message would still be delivered to the middleware. In order to overlap this problem, this paper proposes an application layer gateway for IoT protocols that can "translate" MQTT messages into HTTP reducing the packet size sent by an IoT device and it is fully configurable through a graphical user interface in runtime. The utility of such solution in IoT scenarios is highlighted, showcasing its functionalities, and demonstrating the efficiency of the solution by evaluating three scenarios where data is sent to Orion context broker (from the EU Fiware project).

The remainder of the paper is organized as follows. Section II provides valuable background on the topic. Section III provides information about the software functionalities and demonstrates the efficiency of the proposed approach by sending data to Orion context broker (which is part of the Fiware project). Section IV concludes the paper, suggesting future challenges and open issues.

II. Background

The IoT vision of connecting everything to the Internet is fascinating and ambitious, with an economic impact estimated to range from 2.7 to 6.2 trillion USD (United States dollars) by 2025 [8]. However, IoT is known for its scarce resources and presents many challenges. The computational capacity present in IoT scenarios is a significant challenge, and almost all the aspects of the current Internet has a lightweight alternative that was developed for IoT at each layer of the TCP/IP architecture [9]. At the application layer, several alternative protocols to the HyperText Transfer Protocol (HTTP) are the following: Message Queueing Transfer Protocol (MQTT), Constrained Application Protocol (CoAP), Advanced Message Queueing Protocol (AMQP), and Data Distribution Service (DDS). Moreover, much more were proposed for IoT. Despite its inefficiency, HTTP is widely used in IoT applications through Representational State Transfer (REST) interfaces, because REST provides a simple way of exposing functionalities.

Connected objects do not add much value by themselves and remotely accessing objects functionalities does not entirely justify the expected trillion USD impact. However, collecting and analyzing data of surrounding environments reveals user habits and trends, which is extremely valuable and represents a significant source of income for Facebook and Google, for instance [10]. The data collected in IoT environments is stored in an IoT middleware, which is a software that stores the collected data and also allows incompatible devices and applications to communicate [11][12]. An IoT middleware is deployed in a robust server and is also referred to as an IoT platform, IoT middleware platform, or simply middleware. These terms can be used interchangeably in the remainder of this paper and in the related literature. Devices communicate with a middleware through an application layer protocol. Gadgets that are not compatible with the protocols supported by the middleware cannot be used to its full potential. A simple solution for this problem is based on forwarding data to an intermediary device that converts the message for a protocol that is understood by the middleware. The software responsible for such conversion is called application layer bridge (ALB), application layer gateway (ALG), or simply bridge; the three terms can be used interchangeably in the literature. An ALB operation is similar to a sentence translation, where a person asks a translator to transmit a message in a foreign language. A bridge operation is illustrated in Figure 1.

Fig. 1. Illustration of a bridge operation where a gateway messages through a protocol A and translates them to a protocol B.

An application layer gateway is a software that intermediates the connection between the application server and the Internet. ALGs operation is straightforward since they receive a message through a protocol (protocol A) and forward the message in another protocol (protocol B). They are useful in IoT scenarios because most devices support a single application protocol (i.e., MQTT), and the server that stores the data might support a different protocol (i.e., HTTP). The ALG merely forwards the received message to the server and the communication is seamless for the user (the sender and receiver do not notice the intermediary). These Bridges allow incompatible devices to

communicate with the middleware without changing code neither in the middleware nor at the devices. In theory, ALGs can even be used to bypass restrictions imposed by a firewall or a network administrator where a specific protocol traffic is blocked inside a network (i.e., BitTorrent).

All ALGs follow a similar principle and users can opt for every device to have a dedicated bridge (direct bridge) or share a single bridge for every device (indirect bridge) [13]. Regarding performance, a direct bridge is advantageous because only one device makes use of the gateway and there is no concurrency for its resources. However, dedicating a gateway for every device is costly and increases the solution complexity. The concept of ALBs is not new in information and communication technologies, neither in IoT. A few ALGs have been proposed for IoT scenarios such as Ponte [14] and Gothings [15], both supporting MQTT, CoAP, and HTTP. The main difference between the Ponte and Gothings is that besides translating messages, Ponte also allows data to be persisted on the device in which it is installed. Ponte is an Eclipse project and an evolution of the QEST broker [16]. The most significant disadvantage of Ponte and Gothings is the absence of a graphical user interface (GUI) which turns it difficult to deploy in IoT solutions. Also, they do not provide any functionalities beyond the "message translation."

III. Bridging from MQTT to HTTP

The application layer gateway presented in this paper translates MQTT requests into HTTP. Its primary goal is to allow constrained devices to send fewer data when communicating with IoT middleware. The graphical user interface is simple and enables users to configure aspects related to the conversion and forwarding of messages during runtime. The solution can be deployed on any device with Java installed, which means that it can be deployed on a local Raspberry Pi or even on the server that hosts the Middleware. Besides Java, an MQTT broker must be available (Mosquitto was used). To use the solution, users must specify the MQTT broker details (IP, port, topic) and the middleware server details (IP, port, path). The software then subscribes to the specified topic and forwards the message to the HTTP server (seamless to the sender and receiver). Fig. 2 showcases the GUI that was configured to send data to Orion context broker.

Most bridges follow a similar architecture because their operation method is simple and straightforward. The proposed software does not deviate from that basic architecture, and the only difference is the graphical user interface. The architecture is composed by the following three elements: *i*) message translator, *ii*) protocol plugins, and *iii*) graphical user interface. **Message translator** is the entity responsible for receiving, modifying, and forwarding the message to the desired protocol. **Protocol plugins** are the implementation of specific application protocols, communications between the different protocols must go through the message translator (only MQTT and HTTP are currently supported). The **graphical user interface** allows users to configure aspects related to the conversion and forwarding of messages during runtime. On the device-side, if it supports MQTT, a minor modification is necessary because instead of sending data to the default server, the gadget sends it to the proposed bridge. To operate the program and apply the appropriate modifications to the gadget, users should have an intermediate understanding of MQTT and HTTP requests. If the gadget vendor does not provide means to make such modifications, the bridging operation is not possible.

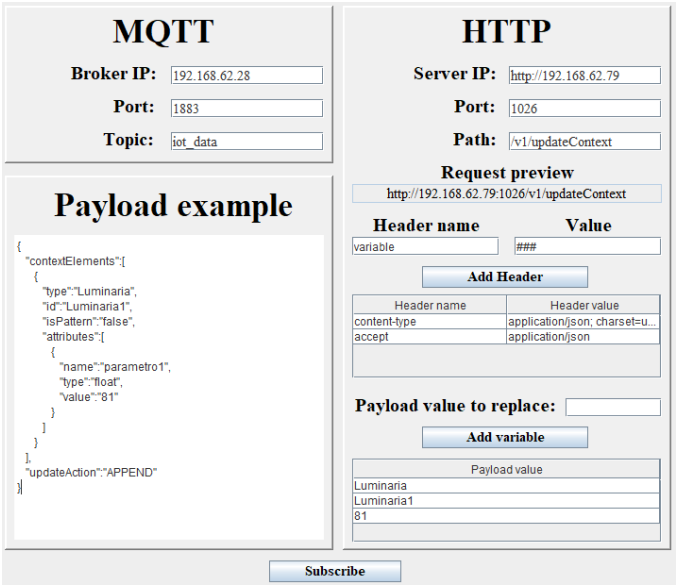

Fig. 2. Softwares' graphical user interface configured to send data to the Orion context broker.

To reduce packet size that is sent by the IoT device, users specify the variables that will be sent (through the GUI). Therefore, instead of sending "Temperature:X", the device sends "X", a similar concept is applied to the headers, where the name and value of the header are specified. If the value of a certain header is specific to each sender, users can specify the header name and leave the value as blank. The software reconstructs the message before forwarding it to the middleware platform. The reconstruction process is illustrated in Fig. 3 and it is seamless to both sender and receiver. Note that for each additional character on the payload, the message size is increased by 1 Byte (blank spaces also count [11]). In the temperature example, the device sends 12 Bytes less on the payload. For security reasons, it is recommended that software should be installed in the same machine as the MQTT broker because it can restrict the broker from accepting subscriptions other than the ones from localhost.

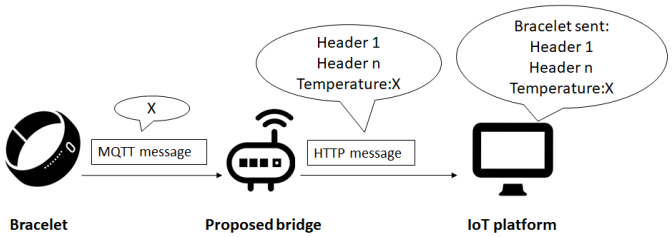

Fig. 3. Illustration of a bracelet sending a small message to the proposed bridge that is reconstructed and sent to the IoT platform.

A. The Orion context broker use case

Orion context broker [17] is a middleware platform developed by Fiware and is a publish/subscribe implementation

of the NGSI-9 and NGSI-10 Open RESTful API specifications. The communication with the server is made through REST (Representational State Transfer), making Orion an excellent candidate to test and evaluate the proposed software, namely because devices that only support MQTT cannot send data to it. The performance of the solution regarding packet size is evaluated in this sub-section. The packet size is crucial because most of the energy on a device is wasted on the communication and, in theory, a smaller message implies a shorter transmission.

The experiments were performed in a real wired LAN, and the packets were generated through Apache Jmeter. Three scenarios were evaluated: *i*) Users define the device specific variables (that change according to each device), as well as the headers (proposed bridge scenario). *ii*) Users do not define the device specific variables, and the headers are sent within the payload (other MQTT-HTTP bridges). *iii*) An HTTP request is directly sent to the server.

When the solution is used, the user specifies the headers and corresponding values. After that, he specifies the request body. Then, the device specific values (that change according to each device) are specified. So, instead of sending a payload identical to the one presented in Fig. 2, the device sends an MQTT message with the payload "Luminaria,Luminaria1,99", the software reconstructs the rest of the message before sending the HTTP request, eliminating blank spaces beforehand. In the scenario where other bridges are used, the MQTT message contains the same payload as the HTTP, and the headers must be specified along with the message. The headers and body that are sent in the scenario where an HTTP request is made is displayed in Fig. 2. Note that most application layer protocols send an ACK (acknowledgment) or NACK (negative acknowledgment) message to the sender informing the success or failure of the communication. The MQTT ACK or NACK is smaller than the HTTP. Fig. 4 presents the packet size analysis for the three proposed scenarios.

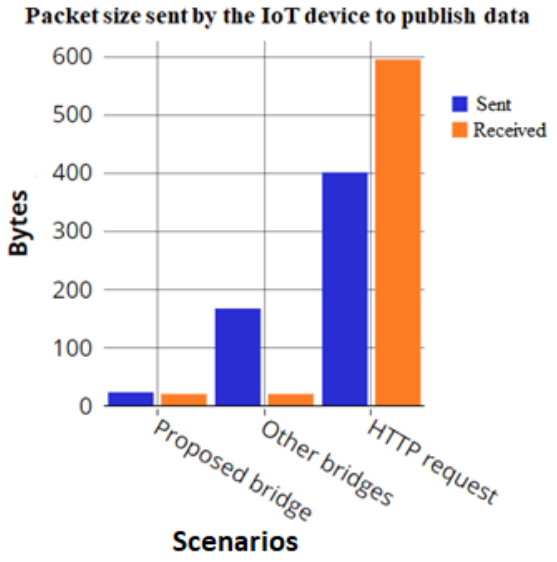

Fig. 4. Analysis of the packet size of a single request in the scenarios where it is used *i*) Proposed bridge, *ii*) Other bridges, and *iii*) HTTP request.

It is observed that an HTTP request is significantly less efficient than MQTT, with 401 Bytes sent and 595 received. Other bridges send 234 and receive 20, while the proposed approach sends 23 and receives 20. The packet size that is sent is 10 times smaller than other bridges and 17 times smaller than a regular HTTP request. Also, both bridges produce an ACK message is 29 times smaller than a regular HTTP request.

IV. Conclusion and Future Work

The IoT market is very promising, and a significant part of its value lies in the data gathered by IoT devices, which is stored in a software called IoT Middleware. Most IoT gadgets can only transmit data using one application protocol due to their constraints, which can be different from those protocols supported by the Middleware. The paper proposed an application layer gateway that converts MQTT messages into HTTP, which can be deployed on any computer with Java installed allowing devices to seamlessly send data to Middleware. The proposed software provides an easily configurable graphical user interface that allows users to configure aspects related to the conversion and forwarding of messages in runtime. The biggest advantage of the solution comes from the fact that it considerably reduces the size of the packet sent by the IoT device because a shorter message is sent to the software, which then reconstructs it and forwards to the middleware. The paper assessed and demonstrated the efficiency of such technique by evaluating three scenarios where data is sent to Orion context broker, which revealed that the packet size that is sent to the software is 10 times smaller than other bridges and 17 times smaller than a regular HTTP request.

As future work, it would be exciting to modify the proposed solution to support bi-directional conversion among multiple protocols such as CoAP, DDS, and other IoT protocols that might be proposed in the future, expanding the utility of the solution. The approach in this paper can even be used to optimize the packet size of requests made in the same protocol especially HTTP to HTTP. Also, a broader performance analysis should be made, verifying the behavior of the software in high load scenarios when deployed on an intermediary device such as the Raspberry Pi, or directly on the server in which the middleware is located. A broader performance evaluation would confirm the low packet loss and response times, further proving the reliability of the solution.

Acknowledgements

This work has been supported by FADCOM - *Fundo de Apoio ao Desenvolvimento das Comunicações*, presidential decree nº 264/10, November 26, 2010, Republic of Angola; National Funding from the FCT - *Fundação para a Ciência e a Tecnologia* through the UID/EEA/50008/2013 Project; by Finep, with resources from Funttel, Grant No. 01.14.0231.00, under the *Centro de Referência em Radiocomunicações* - CRR project of the *Instituto Nacional de Telecomunicações* (Inatel), Brazil; by Finatel through the Inatel Smart Campus project; and by Brazilian National Council for Research and Development (CNPq) via Grant No. 309335/2017-5.

REFERENCES

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 4, pp. 2347–2376, 4th Quart., 2015, DOI: 10.1109/COMST.2015.2444095.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Futur. Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013, DOI: 10.1016/j.future.2013.01.010.

[3] International Telecommunication Union, "Recommendation ITU-T Y.2060: Overview of the Internet of things," pp. 1–22, Jun. 2012.

[4] D. Minoli, K. Sohraby, and B. Occhiogrosso, "IoT Considerations, Requirements, and Architectures for Smart Buildings-Energy Optimization and Next-Generation Building Management Systems," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 269–283, Feb. 2017, DOI: 10.1109/JIOT.2017.2647881.

[5] M. A. A. da Cruz, J. J. P. C. Rodrigues, J. Al-Muhtadi, V. Korotaev, and V. H. C. Albuquerque, "A Reference Model for Internet of Things Middleware," *IEEE Internet Things J.*, vol. 5, no 2, pp. 871-883, Apr. 2018, DOI: 10.1109/JIOT.2018.2796561.

[6] A. Farahzadi, P. Shams, J. Rezazadeh, and R. Farahbakhsh, "Middleware technologies for cloud of things-a survey," *Digit. Commun. Networks*, Apr. 2017, DOI: 10.1016/j.dcan.2017.04.005.

[7] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi, "Toward better horizontal integration among IoT services," *IEEE Commun. Mag.*, vol. 53, no. 9, pp. 72–79, Sep. 2015, DOI: 10.1109/MCOM.2015.7263375.

[8] J. Manyika *et al*., "Disruptive technologies: Advances that will transform life, business, and the global economy," *McKinsey Glob. Insitute*, May, p. 163, 2013.

[9] J. Zhou, Z. Cao, X. Dong, and A. V Vasilakos, "Security and Privacy for Cloud-Based IoT: Challenges," *IEEE Commun. Mag.*, vol. 55, no. 1, pp. 26–33, Jan. 2017, DOI: 10.1109/MCOM.2017.1600363CM.

[10] E. Ahmed *et al.*, "The role of big data analytics in Internet of Things," *Comput. Networks*, vol. 129, part 2, pp. 459–471, Dec. 2017, DOI: 10.1016/j.comnet.2017.06.013.

[11] M. A. A. da Cruz, J. J. P. C. Rodrigues, A. K. Sangaiah, J. Al-Muhtadi, and V. Korotaev, "Performance evaluation of IoT middleware," *J. Netw. Comput. Appl.*, vol. 109, pp. 53–65, May 2018, DOI: 10.1016/j.jnca.2018.02.013.

[12] A. H. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and M. Z. Sheng, "IoT Middleware: A Survey on Issues and Enabling technologies," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 1–20, Feb. 2017, DOI: 10.1109/JIOT.2016.2615180.

[13] V. Issarny, A. Bennaceur, and Y.-D. Bromberg, "Middleware-Layer Connector Synthesis: Beyond State of the Art in Middleware Interoperability," in *Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 217–255, DOI: 10.1007/978-3-642-21455-4_7.

[14] Eclipse Foundation, "Ponte - Bringing Things to REST developers." [Online]. Available: https://www.eclipse.org/ponte/. [Accessed: 16-Mar-2018].

[15] W. L. D. a M. Macêdo, T. Rocha, and E. D. Moreno, "GoThings An Application-layer Gateway Architecture for the Internet of Things," *W* in *11th International Conference on Web Information Systems and Technologies (Webist 2015)*, Lisbon, Portugal, May 20-22, 2015, pp 135-140, DOI: 10.5220/0005493701350140.

[16] M. Collina, G. E. Corazza, and A. Vanelli-Coralli, "Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST," in *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*, Sydney, NSW, Australia, Sep. 9-12, 2012, pp. 36–41, DOI: 10.1109/PIMRC.2012.6362813.

[17] T. Zahariadis *et al.*, "FIWARE lab: Managing resources and services in a cloud federation supporting future internet applications," *Proc. - 2014 IEEE/ACM 7th Int. Conf. Util. Cloud Comput. UCC 2014*, London, UK, Dec. 8-11, 2014, pp. 792–799, DOI: 10.1109/UCC.2014.129.