Multi-device Notifications: A Comparison Between MQTT and CoAP

Luís Augusto Silva*, Gabriel de Mello*, Bruno A. da Silva*, Gabriel Villarrubia González‡, Juan De Paz Santana‡, Paula Prata†, Valderi R. Q. Leithardt*†
*Laboratory of Embedded and Distributed Systems, University of Vale do Itajai (UNIVALI), Brazil, 88302-901
†Instituto de Telecomunicações, Delegação da Covilhã Departamento de Informática
Universidade da Beira Interior-PT, Portugal 6201-001
‡ Expert Systems and Applications Lab, Faculty of Science, University of Salamanca
Plaza de los Caídos s/n, 37008 Salamanca, Spain
Email: {luis.silva, gabrieldemello, silvabruno}@edu.univali.br,
{fcofds, gvg}@usal.es, {valderi.leithardt, pprata}@ubi.pt

Abstract—**New devices generate, send, and display messages about their status, data retrieval, and device information. An increase in the number of notifications received, tends to reduce their tolerance. This article sets out a notification management system focused on user profiles and environments. The solution involves transferring notifications in a multi-device scenario using MQTT and CoAP technologies, while also adopting privacy criteria. It consists of three modules, the first of which was a prototype and evaluated using real devices, the second is a decision module and the third which was a dispatcher module for processing the messages.**

Keywords–Notifications management; data privacy; internet of things.

I. INTRODUCTION

Ubiquitous computing is currently an everyday phenomenon in which we are surrounded by mobile devices, such as smartphones, tablets, clocks, televisions, and other smart devices. According to [1], the computer has been imperceptibly shipped into the environment for the user. These devices also meet the demands of the users and, in turn, collect data from them [2]. In light of this, there has been a comparable growth in mobile device networks, and as [3] and [4] state, devices can be used to process an extensive collection of data from the most wide-ranging events and points of interest.

The devices have become ubiquitous, and as a result, there has been a rise in the number of notifications delivered. This delivery requires a management system that can deal with multiple devices since the number of interruptions caused by so many notifications can distract the user's attention. According to [5], there is a need to bring together communication, the user, and the devices being employed, to ensure data privacy is protected. This task entails providing accurate information to the right recipient, establishing rules for timely decision-making, as well as choosing the location.

This work requires using the control and management notifications with a suitable network for transferring messages through devices using CoAP and MQTT protocols. The CoAP was defined by the IETF [6] in 2014 and is the most recent protocol of the two. The method of using CoAP is similar to HTTP, since it follows the client/server model, and makes use of REST. Its mode of operation includes HTTP methods such as POST, PUT, UPDATE and DELETE. However, the UDP-based communication of this protocol is different from that of HTTP. The MQTT is an application layer protocol that is already designed for devices with low computational power and it uses the publish/subscribe architecture. This means that when a client posts a **M** message on a particular topic, each client enrolled in the **T** topic will receive the M message. In the same way as the HTTP, the MQTT depends on the TCP protocol and IP that are involved. However, compared with HTTP, MQTT is designed to have a lower cost in the protocol stack.

The main difference between CoAP and MQTT is that the former runs on UDP, while the latter runs on top of TCP. Since UDP is not reliable in its acknowledgment of a receipt, CoAP provides a reliability mechanism. This is carried out by using both confirmable and non-confirmable messages. Confirmable messages are entirely dependent on a commit message, while non-confirmable messages do not need acknowledgment. Another difference between CoAP and MQTT is the availability of different levels of QoS. The MQTT defines three levels of QoS while the CoAP does not offer different levels.

This paper is structured into six sections: Section II presents the related works; Section III is dedicated to the description of the proposed solution and comparison between MQTT and CoAP are presented in Section IV; Section V presents the prototype and the experimental results and, finally; in Section VI, we present the conclusions and contributions obtained.

II. RELATED WORK

The papers selected in this chapter emerged from a systematic review of the literature. In the search for the best time for the delivery of notifications and breakpoint discovery methods [7], the authors sought to execute the task directly on the mobile device. The application can detect breakpoints and then deliver notifications.

In [8], a study was conducted to predict the most opportune time for delivering the notification to the user employing a dataset obtained from the Android system which displays a pop-up notification format. The process ends after a pairing device and browser. In the study by [9], it is stated that before notifications can be delivered effectively to a user with IoT devices, it is first necessary to understand the user's real need. The main challenge is determining the user´s interest, which may vary according to his/her status.

The work of [10] is a provisional study and only attempts to predict which device should be selected to deliver the notification. Although the dataset of notification is used to train the algorithms, the solution is based on different machine learning algorithms and allows the system to be evaluated. The result is to some extent synthetic and assumes that the data available for the notification are precise. As early as [9], the author examines an approach to access user notifications, (usually for registration purposes), by establishing an open-source framework for searching notifications on mobile devices.

III. PRISER

The solution that was found makes use of UbiPri's privacy and control management middleware [11], which differs from other existing solutions by providing a generic privacy management and control model. One of the components refers to services; this is called PRISER [12] and is responsible for managing notifications. The application involves assigning a lifetime value to the notification. In the case of a device without Internet access, in a given environment, it should be possible to send medium and high priority notifications in another way, such as by sending an SMS.

In PRISER, a mobile device application was developed to gather, and record notifications executed in the second plan. All the requirements and running jobs in the second plan are reliable and can be run in almost all android versions. After granting users rights to manage services, this service is executed in the second plan on a permanent basis and receives a callback when a notification is added to or removed from the system. The Notifications are stored in the device memory and can be navigated by the device administrator. A JSON object comprises all the notification information. The infrastructure is based on open standards and used on the Internet and IoT devices, such as the CoAP protocol and the MQTT message queuing protocol for low-capacity devices. The proposed architecture is composed of three modules: the collector mentioned above, the decision and dispatcher. These modules are highlighted in blue in Figure 1.

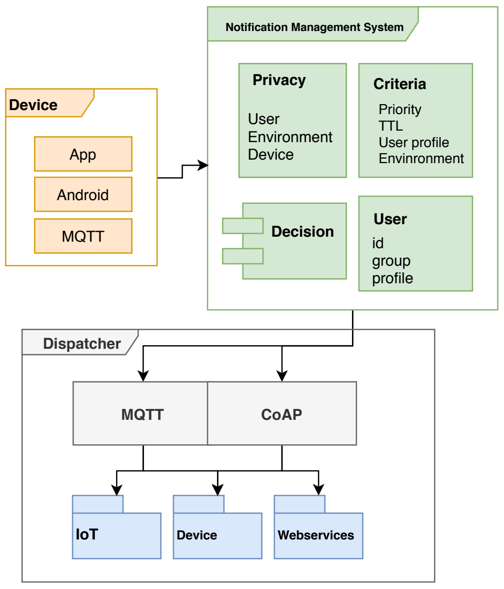


Figure 1: PRISER: Notification Management System [13]

A. Notification Collector

The notifications of the device are stored in the device memory and can be navigated by the device administrator. An Android app was used for this test, and its installation was based in [5]. A JSON object comprising all the notification information is obtained, in accordance with the items mentioned above.

B. Decision Maker

The main purpose of the decision module in the notification management system is decision-making, and receiving information concerning the privacy of the environment, the device or even the user. The module seeks to answer the following questions: *(i)* what is the best location to receive the message?; *(ii)* when is the best time to show the notification to the chosen user? *(iii)* in which device will the chosen users receive the notification? and *(vi)* what is the best way to send the notification?. The most important feature of this module is the way it is used to make decisions.

According to [10], a decision tree algorithm creates a flowchart pattern, in which each internal node represents a test attribute, each branch represents the test result, and each node in the sheet represents a label. The root- to-leaf paths represent the classification rules for issuing the notification This module also aggregates criteria information about the user context (e.g., location, status, current activity), as well as information related to the notification for a lifetime. The criteria serve important purposes in the NMS, such as the way information is used to choose the device and to alert the user (e.g., vibration, sound or light) or to display notifications received that are based on the user's location.

C. Dispatcher

Finally, the dispatcher adapts the notifications to the chosen target devices and then sends them. When handling notifications that are only intended for one device, this causes certain problems. The first point is that the user must always be charging the device, or remain close to it. The second point refers to connectivity, during which the device that the user uses may become disconnected, or even be without a battery. The dispatcher module is based on an architecture that uses multiple devices as shown above in Figure reffig:diagrama1. Both MQTT and CoAP can be used in the dispatcher. Routing messages for IoT include smart-things and devices and provide a web service for third-party applications. The MQTT and CoAP applications are described below.

IV. MQTT AND CoAP

This section is dedicated to experiments performed on multiple devices, including proposals for message transmission and notifications between devices using the MQTT and CoAP message protocols. Packet transmission times and tests from notification quantities were used.

A. Collect and Share using MQTT

The notifications collected in an Android device are shared with the decision module through the publish/subscribe that implements an MQTT protocol. The WebSocket unit provides the communication layer between a combination of the client-side and the server-side for MQTT. The reaper unit receives heartbeat events from operational devices and the component for connecting the devices is called triproxy because it deals with three endpoints instead of the usual two. They can have more than one instance of running simultaneously, and then give a comma-separated list to the provider.

The unit below uses a provider in the dispatcher module unit for making a connection to the Android Debug Bridge (ADB) and starting worker processes for each device. It then sends and receives commands from the processor. Its purpose is to send and receive requests from the app units and distribute them across the processor among the units.

B. *Collect and Share using CoAP*

In the case of CoAP, the notification system employs a request/response system for transferring messages to other devices. A gateway called COSGP-IoT was implemented that relied on the methods and other resources provided by the libcoap2 library, which is the default system for limited capacity devices. This method makes the server responsible for assigning the appropriate working logic of the GET, PUT and DELETE methods defined by the CoAP protocol specifications. It follows the Constrained Restful Environment (CORE) architecture, that includes the Write / Read and Full / Partial of the OSGP model. Also, it supports Pending Events Descriptors (PEDs), which act as the Pending Activator Tables, that are essential for maintenance and general scalability. It should be noted here that there is an absence of a POST function, which can be explained by the lack of an analog method from the OSGP model; thus, it is not necessary to implement the type of request in question.

The following are invoked for its use: a) the gateway function, b) an individual call for each request and response, and c) a corresponding translation method and d) , mapping the data when extracting certain attributes, such as a message identifier, request/response, packet size and token. The data repository of the CoAP server is one of the resources that can be added in the context of the application, and it is through this that the methods must be called. When used for the CoAP requests, this work carried out the implementation of a client in Python, owing to the practicality of the language and ease of use of the libraries available for the platform.

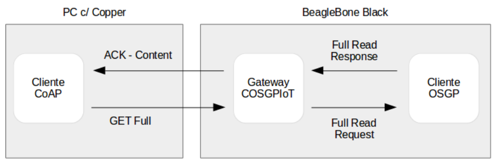
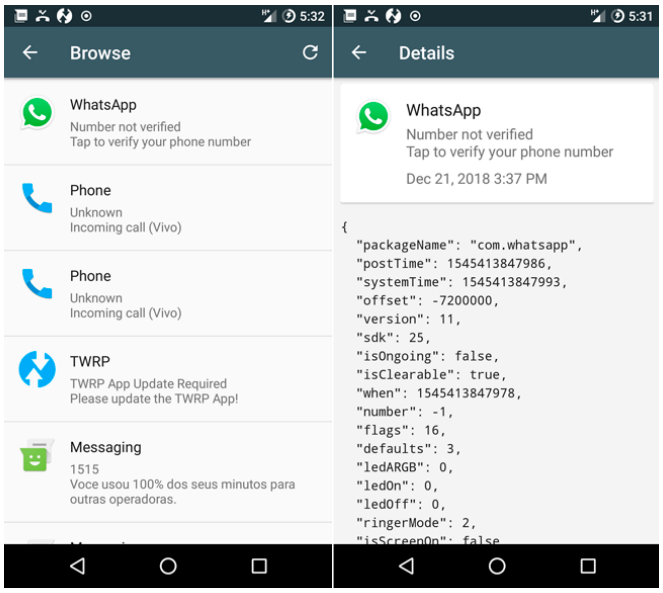


Figure 2: GET Full method for received message.

The testing environment consisted of the hardware implementation of the code developed. The chosen development platform, that was focused on integrating implementations planned in the work, were divided between the client and server. In the case of the CoAP client, it was decided to use the BeagleBone Black microcontroller, which has 512MB of RAM and an ARM Cortex A8 AM3358 CPU with a core operating at 1GHz, while running the Debian 9.4OS and Raspberry Pi model 3 which has 1GB of RAM and an ARM Cortex-A53, 1.2GHz.

V. Prototype and Experimental Results

The initial tests of the collector module proved that, depending on the number of notifications a user receives, the collection process can alternate between every 10 and 60 seconds. A large accumulation of notifications of just the operating system was noticed when using notification by application. The application used for the tests were initially developed by Weber et al. [5] and described in PRISER [13]. This application is shown in Figure 3. Only the message transmission control part and the protocols were used for this work. This resulted in a comparison System for Notifications x Notifications of an application shown in Figure 4.



WhatsApp
Number not verified
Tap to verify your phone number

Phone
Unknown
Incoming call (Vivo)

Phone
Unknown
Incoming call (Vivo)

TWRP
TWRP App Update Required
Please update the TWRP App!

Messaging
1515
Voce usou 100% dos seus minutos para outras operadoras.

WhatsApp
Number not verified
Tap to verify your phone number
Dec 21, 2018 3:37 PM

```
{
  "packageName": "com.whatsapp",
  "postTime": 1545413847986,
  "systemTime": 1545413847993,
  "offset": -7200000,
  "version": 11,
  "sdk": 25,
  "isOngoing": false,
  "isClearable": true,
  "when": 1545413847978,
  "number": -1,
  "flags": 16,
  "defaults": 3,
  "ledARGB": 0,
  "ledOn": 0,
  "ledOff": 0,
  "ringerMode": 2,
  "isScreenOn": false
```

Figure 3: NMS Notification collector. [13]

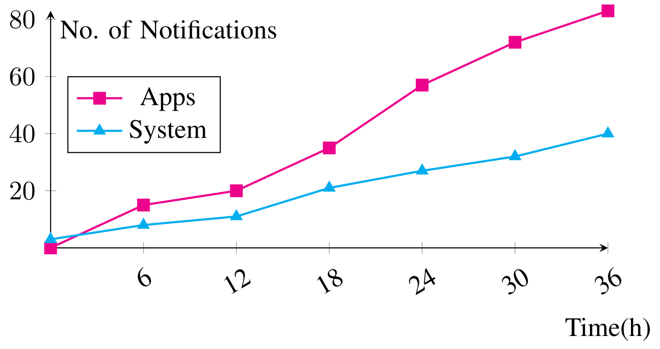


Figure 4: Notification by System x Notification by App per hour.

A. *MQTT Results*

The purpose of the second experiment is to compare the MQTT transfer and latency to forward a notification. First, a transmission was carried out by dividing the packets into the MQTT. 64, 128, 256 and 512 bytes were used for a total load of 1024, 2048, 4096 and 8192. In every case, a JSON file was simulated. The results are shown in Figure 5.

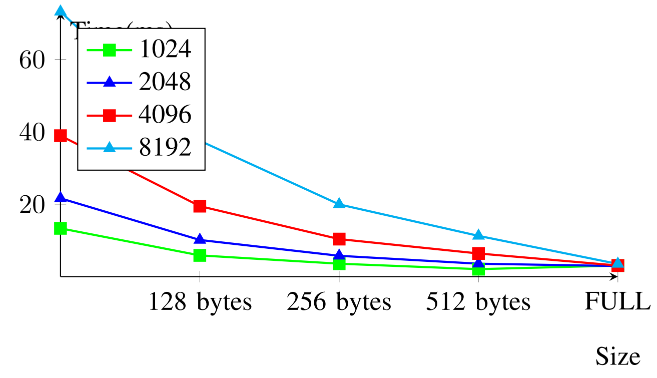


Figure 5: MQTT Publish transmission splited in packets on Beaglebone Black microcontroller.

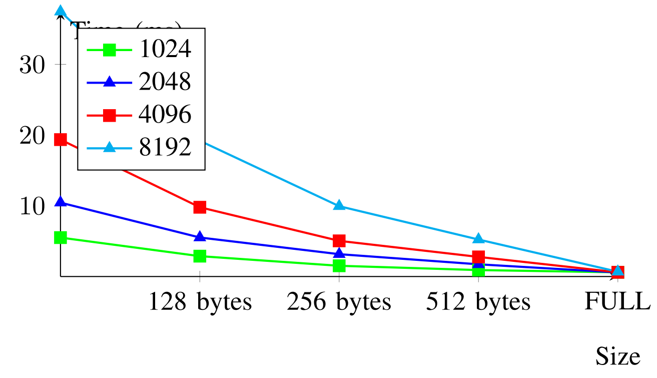


Figure 6: MQTT publishing with split packages on Raspberry Pi device.

The results obtained through the comparison showed that the CoAP is efficient for a low volume of messages but when the volume increases the MQTT is more efficient; further tests must be carried out to measure the degree of efficiency during the work. This result is shown in Figure 7.

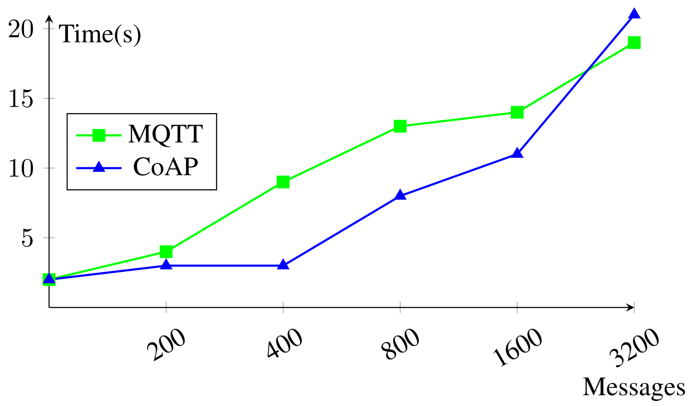


Figure 7: MQTT vs CoAP.

The requirements imposed on NMS were based on the taxonomy and continued as rules and regulations in accordance with research in the literature. The evidence obtained from this article is as follows.

B. CoAP Results

The performance results of CoAP and the translation functions, together with the resources coming from the standard C language libraries, were obtained directly from the source code of the gateway. The results are arranged in milliseconds in the graphs below. The communication latency between the devices was discounted . Only the PUT and GET Fulls methods were tested, in view of the complexity involved in describing the results and the nature of the article. The payloads of the CoAP packages had JSON files of sizes varying between 1024, 2048, 4096 and 8192 bytes. These were divided into packets smaller than 64 bytes, which allowed a larger sample and greater control. Each method was tested 4 times, making a total of 960 packets for each of the two methods tested. The first test can be seen in Figure 8.

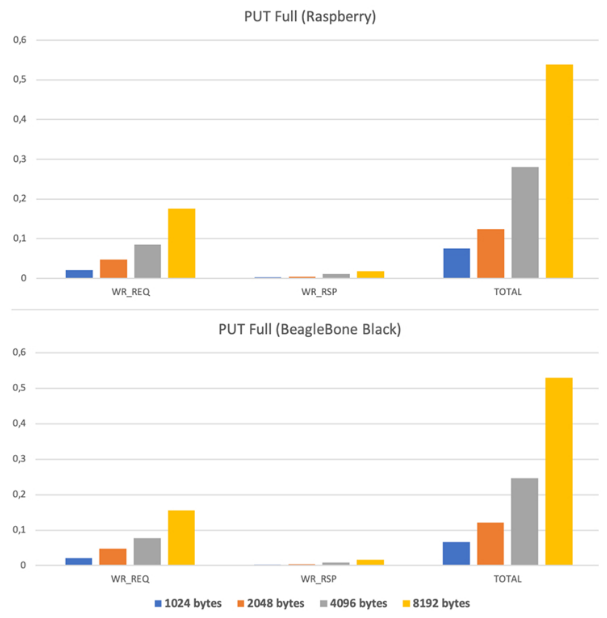


Figure 8: Comparative graphs between the Raspberry and BeagleBone microcontroller for the PUT Full method.

There is a difference between the processing time of the PUT Full requests made by the two microcontrollers. The requests for translation from the CoAP script to OSGP took up more time (between 0.021 ms and 0.175 ms) than the response translations (0.002ms to 0.018 ms), owing to the number of fields and amount of information used by the OSGP packages. The total time for the method ranged from 0.076 ms to 0.538 ms. However, in Figure 9, the processing time between both types of hardware is technically the same (except for the fluctuation rates that may occur) for the GET Full method.

The read response translations from OSGP to CoAP were predictably the most detachable, with times varying between 0.41ms and 0.318ms. The requests were given in the order of 0.003ms to 0.027ms and the total amount of time ranged from 0.083ms to 0.608ms.

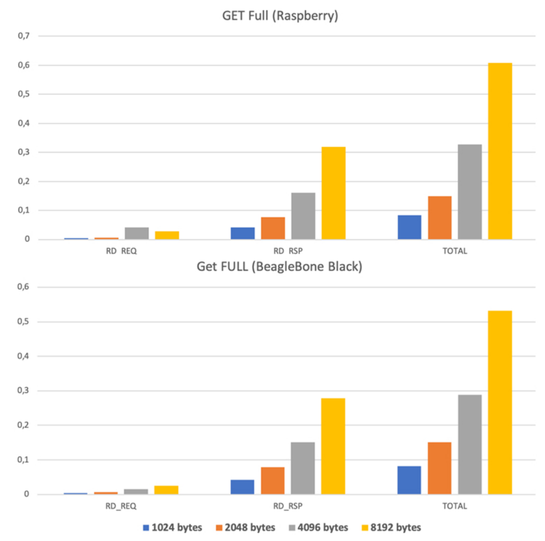


Figure 9: Comparative graphs between the Raspberry and BeagleBone microcontroller for the GET Full method.

VI. Conclusion and Future Work

Throughout this work, stress was laid on the importance of using the privacy criteria with regard to the environment and the hierarchy assigned to the user, and the taxonomy discussed earlier was highlighted. In this way, we were able to contribute to applications of different types of environments and deal with different types of notifications. In addition, it was possible to ensure that relevant notifications were sent and received in compliance with the defined rules. Our architecture is divided into three key modules to manage the notifications received.

A simplified version of the architecture was prototyped, and a preliminary validation was made of the collection module. Besides, the transfer of messages between devices was tested through CoAP and MQTT. New tests must be conducted to determine the variables and make comparisons. Also, a careful evaluation of the decision algorithms had to be implemented, so that different algorithms could be employed and compared. Finally, the prototype must be improved by including an assessment of a large numbers of users.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brasil (CAPES)—Finance Code 001. Supported by project PLATAFORMA DE VEHÍCULOS DE TRANSPORTE DE MATERIALES Y SEGUIMIENTO AUTÓNOMO — TARGET. 463AC03.

Project co-financed with Junta Castilla y León, Consejería de Educación and FEDER funds. This work was partially funded by FCT- Fundação para a Ciência e a Tecnologia through national funds and when applicable co-funded by FEDER – PT2020 partnership agreement under the project UID/EEA/50008/2019. Operação Centro-01-0145-FEDER-000019 – C4 – Centro de Competências em Cloud Computing, co-financed by the Programa Operacional Regional do Centro (CENTRO 2020), through the Sistema de Apoio à Investigação Científica e Tecnológica – Programas Integrados de IC&DT. Including a cooperation with the project international cooperation project Control and History Management Based on the Privacy of Ubiquitous Environments—Brazil/Portugal.

References

[1] M. Satyanarayanan, "Pervasive computing: Vision and challenges," IEEE Personal Communications, vol. 8, no. 4, 2001, pp. 10–17, DOI:10.1109/98.943998.

[2] F. Viel, L. A. Silva, R. Q. Valderi Leithardt, and C. A. Zeferino, "Internet of things: Concepts, architectures and technologies," in 2018 13th IEEE International Conference on Industry Applications (INDUSCON), Nov 2018, pp. 909–916, dOI:10.1109/INDUSCON.2018.8627298.

[3] M. Stolpe, "The internet of things: Opportunities and challenges for distributed data analysis," ACM SIGKDD Explorations Newsletter, vol. 18, no. 1, 2016, pp. 15–34, DOI:10.1145/2980765.2980768.

[4] S. K. Goudos, P. I. Dallas, S. Chatziefthymiou, and S. Kyriazakos, "A survey of iot key enabling and future technologies: 5g, mobile iot, sematic web and applications," Wirel. Pers. Commun., vol. 97, no. 2, Nov. 2017, pp. 1645–1675, DOI:10.1007/s11277-017-4647-8.

[5] D. Weber, A. Voit, and N. Henze, "Notification log: An open-source framework for notification research on mobile devices," in Proceedings..., ser. UbiComp '18, International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers. New York, NY, USA: ACM, 2018, pp. 1271–1278, dOI:10.1145/3267305.3274118.

[6] Z. Shelby, K. Hartke, and C. Bormann, "Constrained application protocol (coap) - draft-ietf-core-coap-18," ago 2019, https://datatracker.ietf.org/doc/draft-ietf-core-coap/.

[7] T. Okoshi, J. Nakazawa, and H. Tokuda, "Attelia: Sensing user's attention status on smart phones," in Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication, ser. UbiComp '14 Adjunct. New York, NY, USA: ACM, 2014, pp. 139–142, DOI:10.1145/2638728.2638802.

[8] A. Sahami Shirazi, N. Henze, T. Dingler, M. Pielot, D. Weber, and A. Schmidt, "Large-scale assessment of mobile notifications," in Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems, ser. CHI '14. New York, NY, USA: ACM, 2014, pp. 3055–3064, DOI:10.1145/2556288.2557189.

[9] Z. Pan, X. Liang, Y. C. Zhou, Y. Ge, and G. T. Zhao, "Intelligent push notification for converged mobile computing and internet of things," in 2015 IEEE International Conference on Web Services, June 2015, pp. 655–662, DOI:10.1109/ICWS.2015.92.

[10] F. Corno, L. D. Russis, and T. Montanaro, "A context and user aware smart notification system," in 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), Dec 2015, pp. 645–651, DOI:10.1109/WF-IoT.2015.7389130.

[11] V. R. Leithardt, L. H. A. Correia, G. A. Borges, A. G. Rossetto, C. O. Rolim, C. F. Geyer, and J. M. S. Silva, "Mechanism for privacy management based on data history (ubipri-his)," Journal of Ubiquitous Systems and Pervasive Networks, vol. 10, no. 1, 2018, pp. 11–19.

[12] L. A. Silva, D. dos Santos, R. Dazzi, J. S. Silva, and V. Leithardt, "PRISER - Utilização de BLE para localização e notificação com base na privacidade de dados," Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação, vol. 2, no. 1, 2018, DOI:10.5281/zenodo.1336806.

[13] L. A. Silva, V. R. Q. Leithardt, C. O. Rolim, G. V. González, C. F. R. Geyer, and J. S. Silva, "Priser: Managing notification in multiples devices with data privacy support," vol. 19, no. 14, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/14/3098