





# Real-Time Primitives for CoAP: Extending the Use of IoT for Time Constraint Applications for Social Good <sup>†</sup>

Gabriel M. Eggly <sup>1</sup> , Mariano Finochietto <sup>2,3</sup>, Emmanouil Dimogerontakis <sup>4</sup> ,  
Rodrigo M. Santos <sup>1</sup> , Javier Orozco <sup>1</sup> and Roc Meseguer <sup>4</sup> 

<sup>1</sup> Department of Electrical and Computers, Universidad Nacional del Sur, CONICET, Bahía Blanca 8000, Argentina; gmeggly@gmail.com (G.M.E.); ierms@uns.edu.ar (R.M.S.); jadorozco@gmail.com (J.O.)

<sup>2</sup> GIDI, Department of Information Technology, Universidad Nacional de Mar del Plata, Mar del Plata 7600, Argentina; mariano.fino@gmail.com

<sup>3</sup> SpinalCom, Orsay 91400, France

<sup>4</sup> Department of Computer Architecture, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain; edimoger@ac.upc.edu (E.D.); meseguer@ac.upc.edu (R.M.)

\* Correspondence: ierms@uns.edu.ar; Tel.: +54-291-4595101 (ext. 3304)

<sup>†</sup> Presented at the 12th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI 2018), Punta Cana, Dominican Republic, 4–7 December 2018.

Published: 24 October 2018

**Abstract:** Internet of Things (IoT) have become a hot topic since the official introduction of IPv6. Research on Wireless Sensors Networks (WSN) move towards IoT as the communication platform and support provided by the TCP/UDP/IP stack provides a wide variety of services. The communication protocols need to be designed in such a way that even simple microcontrollers with small amount of memory and processing speed can be interconnected in a network. For this different protocols have been proposed. The most extended ones, MQTT and CoAP, represent two different paradigms. In this paper, we present a CoAP extension to support soft real-time communications among sensors, actuators and users. The extension facilitates the instrumentation of applications oriented to improve the quality of life of vulnerable communities contributing to the social good.

**Keywords:** IoT; communication protocols; CoAP; real-time

## 1. Introduction

Internet of Things (IoT) has become a dominant subject in the last years as wireless connectivity has become common and with it, the possibility of interconnecting different sensors and actuators in a collaborative and intelligent environment irrespective of the communication means. Objects make themselves readable, locatable, addressable, recognizable and they obtain intelligence by making or enabling context related decisions thanks to the fact that they can communicate information about themselves. This information is obtained processing their own environment data and/or in collaboration with other objects [1].

Today, places and things have intelligence and provide users with different degrees of technological support in several areas from domestic issues to more complex ones like health care or security. Sensors and actuators constitute a digital ecosystem in which communication protocols are one of the main issues to solve.

In developing countries, there are strong difficulties that come from societies with an important degree of inequality. Even among developing countries there are major differences. We can mention the earthquakes in Haiti and Chile to show how a natural disaster of equivalent strength has severe consequences in one case while in the other not. In the same line, tsunami effects in Indonesia were

completely different to those that happen in Japan some years later. Bridging the digital gap in these situations may introduce new alternatives and provide more secure responses in extreme situations like the ones previously mentioned. However, the possibility of counting with digital support in daily life improves also the economic development in areas such as agriculture, health-care or transport among others. IoT has a huge potential in social good but to have a positive impact it needs to be able to operate with time constraints. To achieve this, protocols should differentiate traffic to provide real-time quality-of-service.

The IoT interaction among end-devices assumes a machine to machine (M2M) communication paradigm. For this, in the client-server model, two different approaches were adopted. The first one is the publish-subscribe protocol and the second the request-response. Both have been implemented in simple protocols executing between the transport and application layers within the Internet Layer Model. In the first case, in general, there is no direct connection between the producer and the consumer of information. In fact, both ends connect to a broker that acts as a relay between both ends. MQTT is the protocol that implements this approach [2]. In the second case, there is a direct connection between the producer and the consumer that works on demand. The Constrained Application Protocol (CoAP) [3] describes this interaction. In any case, there are no time constraints considerations and the communication is based on a best effort approach. While these protocols provide solutions to a wide variety of applications they are not able to satisfy the real-time quality of service requirement. In applications where time constraints are present, the behavior of the protocol without real-time handling is unpredictable because they work on a best-effort context.

In this paper, we present an extension of the CoAP protocol that can process messages with time restrictions. Based on the original implementation, the proposal introduces a set of primitives that allow both the consumer and the producer to determine if the information can be processed on time. An application scenario in the scope of the workshop is discussed in 5 to provide support for the ideas here introduced. In developing countries the public transport is usually chaotic and disordered. This introduces large travel times and with this a waste of resources as people spends a lot of time travelling in bad conditions reducing productivity at work and incrementing the number of traffic accidents.

The rest of the paper is organized in the following way. In Section 2 related papers are discussed and the main differences to our proposal are remark in each case. In Section 3 CoAP protocol is described with some details so the reader can understand the contribution we introduced in the following Section 4. Finally in Section 5 a discussion on the properties of the approach is presented and in Section 7 conclusions and future work are presented.

## 2. Previous Work

IP based communication is being adopted for IoT applications as it simplifies the network management. There are several approaches like the Device Profile for Web Services (DPWS) [4], but being based in HTTP makes it hard to implement in constrained devices. CoAP and MQTT [2,3] are being used in the case of M2M applications as they provide the possibility of implementing the protocols in low resolution processors with limited memory.

In the IoT world, machines exchange data with machines without the direct participation of people. In this intent, data producers and consumers exchange roles continuously. In the literature there is a large list of applications already thought that use IoT. Virtually everything can find support in the IoT paradigm. It is for this that in [5] the authors highlight the need to extend our knowledge of WSN and Cloud computing as instruments that support IoT software applications, and also understand the role played by big data, information sharing and collaboration for IoT-based service provision. Particularly, information sharing and collaboration on these infrastructures impose several challenges to the systems designers. Depending on the service to be provided, real-time interaction support can be or not required. From aided robotics [6] with time constraints in dedicated networks to the design of a low cost wireless marshalling module for industrial environments[7]. In any case, the model is extended to provide support to the data producers using the Internet to store the data in the cloud for

later processing and use [8] but in this case no considerations are provided for the timely behavior of the system. In [9], the same idea is exploited but in this case to retrieve sensitive health information. What differences these approaches from the one presented in this paper is that in all cases, data is transferred on an stable network.

After conducting an extensive survey on real-time data processing technologies and models for IoT applications, Yasumoto et al. [10] found no protocol or implementation capable of providing real-time QoS when working with open Internet. However, the literature reports several interesting works that can be used to support the definition or analysis of communication proposals in such a study domain. For instance, in [11,12] the authors introduce a temporal analysis of the CoAP [3], that allows to measure the latency or delay in the data transmission. This is useful to determine the performance of a particular protocol or link, however it is not enough to guarantee real-time constraints or QoS. In [12], the authors introduce an extension for the CoAP protocol. However, in that paper, the authors proposed the modification of the Ethernet protocol to transform it in a TDMA one with time synchronization based on a clock server. They are not working with LPWA network protocol so it is not really necessary to use a constrained protocol and the communication channel is private to the sensors and not public like in th case proposed here. In [12], the authors introduce a flexible binding between sensors and actuators to provide certain autonomy within CoAP implementations. They named the new entity as RESTlets. These count with inputs and certain functions that conditioned the output. In this way they can model different kind of scenario interactions. The proposal does not work with real-time deadlines. Another approach is proposed in RFC 7641 [13] in which using CoAP a registration primitive of the client within the server is allowed in such a way, that when there is an update in the value of the requested parameter, the server sends a message to the client with the new value. With this procedure it is not possible to know if the server is down or there is no modification in the sensor.

### 3. CoAP Overview

CoAP is described in the RFC 7252 [3]. It proposes a simple and light protocol to be used in lossy networks with constrained nodes usually based on low-power, 8 bit micro-controllers with little memory both in RAM and ROM. The protocol implements a request/response interaction model between application end-points, includes a built-in resource and devices discovery service and provides concepts associated to Internet such as URIs and Internet media types.

The Representational State Transfer (REST) architecture of the web is based on a set of principles that describe how networked resources are defined and addressed [14]. The architecture distributes functionality among resources and uses a reduced set of commands to address them. The architecture is layered and stateless and supports caching. In the scenario discussed here, in which networks operat with low throughput, low bandwidth like 6LoWPAN, [15] and nodes based on limited processors, we work with Constrained RESTful Environments (CoRE).

CoAP is a protocol that fulfills the M2M requirements in constrained environments. As transport protocol it uses UDP with optional support for unicast and multicast requests. The messages are transferred in asynchronous mode which provides an important flexibility while keeping a low overhead to reduce the complexity. It implements URIs ad content type support and has very simple proxy and caching capabilities. The protocol presents a stateless HTTP mapping, that allows proxies to be built providing access to CoAP resources via HTTP in a uniform way or to implement a simple HTTP interface over CoAP. The protocol relies security in the transport layer through DTLS [16].

The interaction model implemented in CoAP is like the HTTP Client/Server. However, as it is M2M protocol it usually finishes with both ends acting alternatively as client and server. The model is described more accurately as request/response. There are four types of messages: Confirmable, Non-confirmable, Acknowledgment and Reset. The protocol is located between the transport and application layers of the Internet Model.

Messages have a short fixed-length binary header composed of four bytes optionally followed with a set of compact binary options and payload. It has an identification (ID) that is used to detect duplications and to provide optional reliability. Messages marked as Confirmable (CON) should be Acknowledge (ACK). If they are not, the sender will retransmit them using timeout and exponential back-off algorithms until it receives the ACK. In the case the recipient is not able to process a CON message it should end the communication with a RST message. Messages non-confirmable (NON), do not require an ACK response. In this case, the message still has the ID field to detect duplicate messages.

Messages have a very simple frame format. The first two bits indicate the version of the protocol, right now these bits are always 01. The second field is the Type bits and these represent 0 for a Confirmable, 1 Non-confirmable, 2 Acknowledge and 3 for Reset. The third field in the header is TKL and indicates the length of the token which is variable between 0 and 8 bytes. The next field is the code and indicates in the classic HTTP way the kind of answer. There are three bits for the first part and 5 for the second which are used to identify a particular subclass. The next field is the Message Id and has 16 bits. Then the Token and the Option fields follow and finally, if exists, the Payload of the message.

The protocol defines several parameters that are used to determine if the network is operating within reasonable bounds and for repeating when necessary Confirmable messages. However, there is no time guards for messages with real-time requirements. In the next section, an extension to the protocol is proposed using the Options field to introduce this.

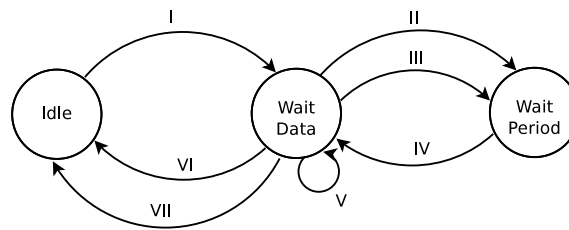
#### 4. CoAP Soft Real-Time Extension

Real-time systems are those that have to produce correct results from an arithmetic-logic point of view before a certain instant named deadline [17]. When missing a deadline has catastrophic consequences the system is said to be hard and it is named soft if some deadlines may be missed. The main characteristic of a real-time system is its predictability. Hard real-time systems are related to specific areas like military, avionics, space or controllers for power-engines, industrial processes and so on. There are many other areas in which a timely behavior is required but in which the eventual miss of deadlines is not critical. Among these we can mention environmental monitoring, weather forecasts or multimedia streaming. IoT applications sharing the network with all the other Internet traffic are not hard real-time since the network delay is unbounded in the open Internet.

It can be argue that Confirmable messages are preferred for real-time traffic as they should be acknowledge and with this the client is aware of the reception of the message by the server and its processing, while the response can be piggybacked in the ACK message. We think that while CON-ACK messages may be used, the non confirmable messages are useful too in the real-time scene. In fact, if the network is overloaded, the ACK messages even in the case of not complying the deadline of the request contributes to the congestion and provides no useful data. In any case, the client should wait for a valid on-time response and if this is not receive a new request should be issue, repeating it up to a limit. Periodic updates of data from sensors consumes bandwidth and in some cases it only repeats an old value. An update upon change would be a better option. However, if managed correctly they can be used to keep updated both data and sensor state without overloading the network traffic. The request for a periodic response from the server should be made in the terms of a registration like [13] but instead of updating upon changes, it should update upon expiration of the period. This procedure is necessary because CoAP uses UDP as transport protocol.

##### 4.1. Description of the Client

In Figure 1, a finite state machine (FSM) is presented for the client and in Table 1 the transitions are described. The FSM represents the actions associated to the verification of the time constraints.



**Figure 1.** Finite State Machine for the Client/Requester.

**Table 1.** Description of the Client Transitions.

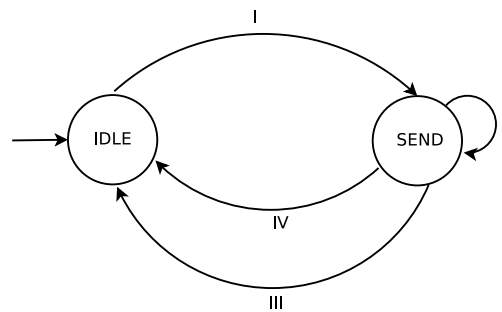
Transition	Conditions	Action
I	$App\_Data\_Req(Resource, D, P)$	$Fail = 0$ $A = TStampR$ $Snd\_Data\_Req(Resource, A, D, P)$
II	$(P > 0) \ \&\&$ $(act\_time \geq A + D)$	$Fail++$ $A = A + P$
III	$(P > 0) \ \&\&$ $(act\_time < A + D) \ \&\&$ $Receive\_Data(Resource, Data, TStampS)$	$Process\_Data(Resource, Data, TStampS)$ $A = A + P$
IV	$act\_time \geq A$	<i>null</i>
V	$(P == 0) \ \&\&$ $(act\_time \geq A + D)$	$Fail++$ $A = TStampR$ $Snd\_Data\_Req(Resource, A, D, P)$
VI	$(P == 0) \ \&\&$ $Receive\_Data(Resource, Data, TStampS)$	$Process\_Data(Resource, Data, TStampS)$
VII	$Fail \geq Limit$	$Error(errorcode)$

In the initial state, IDLE, it is waiting for a demand from the application. Once the application makes a request for a particular Resource,  $App\_Data\_Req(Resource, D, P)$ , the client changes its state to Wait\_Data and initializes the *Fail* variable with zero and the time accumulator *A* with the time at which the request is made to keep record of the elapsed time, then sends a  $Snd\_Data\_Req(Resource, A, D, P)$  message to the Server. The message has four fields. The first one indicates the kind of data that is requested (Temperature, Pressure, Traffic state, etc); the next two parameters are the actual time of the request and the relative deadline respectively and the last parameter is the period. In the case that the request is not periodic, we define  $P = 0$ . In transition II, there is no message from the server and the deadline has expired, so *Fail* is incremented and the client passes to state Wait\_Period. In transition III, the client moves to this state too, but in this case it sends to the application the data received as it is within time,  $Process\_Data(Resource, Data, TStampS)$ . In transition IV, the period is reached thus a new reception from the server is expected so the client moves to Wait\_Data state. Transition V is followed in the case of a non-periodic request and when the deadline is expired without reception of data from the server. In this case the event is registered by incrementing *Fail*. Transition VI represents the case of a non-periodic message that receives on time the data from the server and passes it to the application like in transition III. In the case,  $Fail \geq Limit$ , where *Limit* is defined by the client, it informs the application of the presence of an error with a certain code, transition VII.

#### 4.2. Description of the Server

In Figure 2 the FSM for the server is shown and in Table 2 the transitions are described. Like in the case of the client, only the transitions associated to the real-time behavior are presented. In this case, the server is IDLE while there is no Request from the client. When a valid request is received,  $Rec\_Data\_Req(Resource, TStampR, D, P)$ , it goes to state SEND. The first thing is to inform the server

application that a request for a particular *Resource* has been received, generated at  $TStampR$  with a certain deadline  $D$  and period  $P$ . Like in the client case, the server uses variable  $A$  to keep record of the accumulated time to check the period and keep updated the deadline. The server remains in the SEND state if the request is periodic. Transition II shows how the server updates the variables and sends to the client the resource requested,  $Snd\_Data(Resource, TStampS)$ . Transition III describes the conditions and actions when the request is not periodic,  $P = 0$ . Data is sent to the client if it is within the deadline and it returns to the IDLE state. If the reception of the request is beyond its deadline or the resource is not available, the request is discarded as shown in transition IV and it informs the client with an error code.



**Figure 2.** Finite State Machine for the Server/Responder.

**Table 2.** Description of the Server Transitions.

Transition	Conditions	Action
I	$Rec\_Data\_Req(Resource, TStampR, D, P)$	$Req\_Proc(Resource, TStampR, D, P)$ $A = TStampR$
II	$(P > 0) \ \&\&$ $(A < act\_time \leq A + D)$	$A = A + P$ $TStampS = act\_time$ $Snd\_Data(Resource, TStampS)$
III	$(P == 0) \ \&\&$ $(act\_time \leq A + D)$	$TStampS = act\_time$ $Snd\_Data(Resource, TStampS)$
IV	$Resource\_unavailable \parallel (act\_time > A + D)$	$Error(errorcode)$

## 5. Discussion

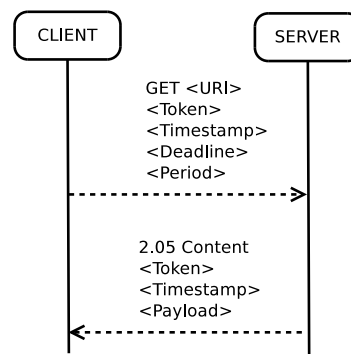
The extension presented in the previous section is proposed to deal with real-time messages. As the network is not controlled by the end-devices (clients and servers), the protocol restricts the traffic when there is no possibility of meeting the deadlines. The protocol can be easily implemented using the Options field in the header of the messages. Both Confirmable and Non-Confirmable messages may be used to do this. However, the first ones provide more reliability for the application at the cost of incrementing the traffic in the network.

It is important to remark that the modification we introduce, does not affect the CoAP functioning. By incorporating time stamps to the messages, both ends are aware of the actual delay in the network so they can determine the quality of the link and the feasibility of sending and receiving messages with time constraints. As the supported systems are soft real-time deadline misses are tolerated. There is an advantage in using the real-time control extension as messages with no possibility of arriving to destination before their deadlines are not sent. This reduces the bandwidth demand reducing the traffic and allowing other messages to reach destination on time. Besides an energy reduction is achieved at the same time because messages are not sent. A common problem in using the open Internet network is that in some situations the links and routers may be overflowed by messages. As the messages are not acknowledged, there is a new request that contributes to increment the demand on the collapsed



link. By introducing the time-stamps and deadline control at the *server*, the traffic is eventually lessen and the network can recover its real-time throughput.

In Figure 3 the exchange of messages between the client and the server for a Non Confirmable message and not periodic request of a specific resource is shown. The exchange is very simple and with a low footprint can be implemented in any low cost microcontroller like the ATmega328P or similar.



**Figure 3.** Send\_Data\_Req() and Send\_Data() primitives as CoAP Request/Response messages.

IoT is gaining momentum every day and new challenges arise continuously. One of the main issues is related to the traffic quality of service as the limited bandwidth will act as bottleneck for the development. In this line, the possibility of providing real-time quality of service control becomes important. In the actual context, an IoT end-device can only modulate the message transmission based on the worst-case expected transmission delay avoiding the transmission in the cases that deadlines are not met. This is the main point of the extension presented in this paper. We argue that reducing the traffic of messages that would with a high probability arrived outdated will improve the general throughput of the open network.

Developing countries with vulnerable communities have difficulties in accessing reliable wide band Internet. In these countries however, the possibilities of using IoT systems to improve the quality of life is remarkable as there is a poor digitalization of services. The introduction of simple protocols to handle IoT traffic with time restrictions constitutes an interesting option that requires a low investment. In the next section, we discuss with details an scenario in which information provided with real-time constraints may improve considerably the quality-of-life in cities with informal public transport, organizing not only vehicular traffic but allowing passengers to save both time and money.

## 6. Application Scenario

Public transport within vulnerable communities or development countries represent an important handicap as people waste too many hours moving in the cities. While in more advanced countries buses, trains, underground trains or even taxis are regulated, follow strict circuits and have a predictable timetable, in vulnerable communities small vans named “combis” circled the cities in search of clients and adapt their route dynamically. Besides there are “colectivos” that are taxis taken by several clients in a similar way to “combis” but with less passengers. Cities like Arequipa in Peru organize their public transport around this kind of “cooperative” systems and an important amount of small single client taxis that arrange the price of the trip at the moment of taking it in the street. Arequipa is a very important city in Peru, so there are also regular buses following strict circuits. Anyway, “combis”, “colectivos”, and taxis seem to dominate the market as they are cheap.

The “combis” go through announcing the general direction of the trip by shouting the successive stops. However, this destination is not fixed but adaptive and changes according to the passengers aboard. So a client may pick up one or the other according to how close or how much it may deviate the route to satisfy its own demand. However, the client is not aware that one block away in the next

corner, there may be another “combi” with a closer route to his/her destination. So people may waste time and money by picking not the best transport.

Figure 4 shows in a picture the presence of a regular bus in the background, a combi and a colectivo behind.



Figure 4. Bus, combi, colectivo.

The scenario we propose in this paper is used to identify both “combis” and “colectivos” with a particular identification within an IoT scenario. Each one, should have a GPS to provide its exact location, the number of available places, the planned route with intermediary stops and even the price of the trip per distance for example. This kind of equipment may be deployed also in more traditional taxis. The transport vehicle is identified with a resource name in a particular URI. Table 3 represents possible naming according to the categories of the public transport. In the URI, the name identifies the kind of transport (bus, combi, colectivo or taxi). To distinguish among the different ones, buses incorporate the Line Identification (Line\_ID), that is probably a number combined with letters like 519A. After that, the vehicle as such is identified with a particular plaque that has a unique set of letters and numbers. As combis, colectivos and taxis do not have a fixed circuit they are not identified with a Line\_ID but they have a particular identification number (ID) that is also unique.

Table 3. URIs for the Public Transport.

Bus	<a href="http://www.bus_LineID_bus_ID.com/">http://www.bus_LineID_bus_ID.com/</a>	position empty seats next stops min price
Combi	<a href="http://www.combi_combi_ID.com/">http://www.combi_combi_ID.com/</a>	position empty seats next stops min price
Colectivo	<a href="http://www.colectivo_colectivo_ID.com/">http://www.colectivo_colectivo_ID.com/</a>	position empty seats next stops min price
Taxi	<a href="http://www.taxi_taxi_ID.com/">http://www.taxi_taxi_ID.com/</a>	available min price

As the system proposed is oriented to work with vulnerable communities or in developing countries an important issue is to provide a low cost technological solution. If this solution is expensive, nobody will adopt it.

As explained before, each vehicle counts with a computational equipment. These need to be connected to the Internet and addressable from any point to request the necessary information. At this point we adopt the LoRA communication standard that provides transmission ranges over several



hundred meters with low power, low bandwidth, operates in the non licensed spectrum and a radio system can be obtained in the market for less than 40 dollars. The whole set will be around 100 dollars per vehicle and this is affordable even in very challenged communities.

The system should be completed with special *totems* deployed along the city that request information periodically to vehicles in their radio. These totems provide potential clients with a list of the available options in their proximity. Clients may poll the vehicles directly too. But in this case, they should know beforehand the URIs they are looking for. Both the totems and the vehicles operate with the extended CoAP protocol. In the case of the vehicles, they always act like servers that respond to the demand of totems or end-users. The totems have a double function as they request data to vehicles (client) and respond to users (servers). A totem is identified with URIs according to Table 4. The totem collects data from vehicles in the system and save it ordered.

**Table 4.** URIs for Totem.

bus	<a href="http://www.totem_totemID.com/buses/list.bus">http://www.totem_totemID.com/buses/list.bus</a>
combi	<a href="http://www.totem_totemID.com/combis/list.combi">http://www.totem_totemID.com/combis/list.combi</a>
colectivo	<a href="http://www.totem_totemID.com/colectivos/list.colectivo">http://www.totem_totemID.com/colectivos/list.colectivo</a>
taxi	<a href="http://www.totem_totemID.com/taxis/list.taxis">http://www.totem_totemID.com/taxis/list.taxis</a>

As CoAP is a RESTful protocol, it can be polled by an HTTP application through a proxy as stated in Section 3. This may be implemented in low cost smartphones that can be acquired by less than 100 dollars. Even for developing countries and vulnerable communities, these are technological solutions that are already being used worldwide. In this way, a potential client checks with the smartphone the availability of transport towards a particular destination. The data should be updated with real-time constraints upon demand.

## 7. Conclusions

In this paper we have presented an extension to CoAP. The extension covers some primitives to be used in the case that messages should count with certainty on the moment at which the information was requested/produced by the client and the server. In this case, what the extension provides is a way in which messages that will not verified the deadlines are not send and with this the traffic in the network is reduced. The modifications proposed can be easily implemented in the option field of the CoAP header message and requires a simple verification of the time stamps associated with each request and possible response. As the application scenario, we propose the implementation of a transport public aware system in which the *informal* transport vehicles get an ID and can promote upon request their routes and fees. This application impacts on the quality of life of thousands of people that have to move in big cities within development countries. The improvement contributes to what is commonly known as social good.

**Funding:** This research was funded by the EU Horizon 2020 Framework Program project netCommons (H2020-688768), by the EMJD-DC program, by the Spanish Government under contract TIN2016-77836-C2-2-R, and by the Generalitat de Catalunya as Consolidated Research Group 2017-SGR-990.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Patel, K. Internet of Things-IOT: Definition, characteristics, architecture, enabling technologies, application & future challenges. *Int. J. Eng. Sci. Comput.* **2016**, *6*, 6122–6131, doi:10.4010/2016.1482.
2. Banks, A.G.R. MQTT Version 3.1.1. 2014. Available online: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.pdf> (accessed on 1 June 2018).
3. Shelby, Z.; Hartke, K.; Bormann, C. The Constrained Application Protocol (CoAP). RFC 7252. 2014. Available online: <http://www.rfc-editor.org/info/rfc7252> (accessed on 1 June 2018).

4. Nixon, T.; Mensch, A. Devices Profile for Web Services Version 1.1. 2009. Available online: <http://ws4d.org/2009/public-review-of-ws-dd-specifications/> (accessed on 1 June 2018).
5. Lee, I.; Lee, K. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Bus. Horiz.* **2015**, *58*, 431–440.
6. Grieco, L.; Rizzo, A.; Colucci, S.; Sicari, S.; Piro, G.; Paola, D.D.; Boggia, G. IoT-aided robotics applications: Technological implications, target domains and open issues. *Comput. Commun.* **2014**, *54*, 32–47.
7. Han, S.; Lin, T.; Chen, D.; Nixon, M. WirelessCHARM: An open system low cost wireless marshalling module for industrial environments. In Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT), Seoul, Korea, 6–8 March 2014; pp. 502–505, doi:10.1109/WF-IoT.2014.6803218.
8. Distefano, S.; Merlino, G.; Puliafito, A. A utility paradigm for IoT: The sensing Cloud. *Pervasive Mob. Comput.* **2015**, *20*, 127–144, doi:10.1016/j.pmcj.2014.09.006.
9. Xu, B.; Xu, L.D.; Cai, H.; Xie, C.; Hu, J.; Bu, F. Ubiquitous Data Accessing Method in IoT-Based Information System for Emergency Medical Services. *IEEE Trans. Ind. Inf.* **2014**, *10*, 1578–1586, doi:10.1109/TII.2014.2306382.
10. Yasumoto, K.; Yamaguchi, H.; Shigeno, H. Survey of real-time processing technologies of IoT data streams. *J. Inf. Process.* **2016**, *24*, 195–202, doi:10.2197/ipsjip.24.195.
11. Konieczek, B.; Rethfeldt, M.; Golasowski, F.; Timmermann, D. Real-Time Communication for the Internet of Things Using jCoAP. In Proceedings of the 2015 IEEE 18th International Symposium on Real-Time Distributed Computing, Auckland, New Zealand, 13–17 April 2015; pp. 134–141, doi:10.1109/ISORC.2015.35.
12. Konieczek, B.; Rethfeldt, M.; Golasowski, F.; Timmermann, D. A Distributed Time Server for the Real-Time Extension of CoAP. In Proceedings of the 2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC), York, UK, 17–20 May 2016; pp. 84–91, doi:10.1109/ISORC.2016.21.
13. Hartke, K. Observing Resources in the Constrained Application Protocol (CoAP). RFC 7641. Available online: <http://www.rfc-editor.org/info/rfc7641> (accessed on 1 June 2018).
14. Fielding, R.T. REST: Architectural Styles and the Design of Network-Based Software Architectures. Ph.D. Thesis, University of California, Irvine, CA, USA, 2000.
15. Montenegro, G.; Kushalnagar, N.; Hui, J.; Culler, D. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944. Available online: <http://www.rfc-editor.org/info/rfc4944> (accessed on 1 June 2018).
16. Rescorla, E.; N., M. Datagram Transport Layer Security Version 1.2. RFC 6347. Available online: <http://www.rfc-editor.org/info/rfc6347> (accessed on 1 June 2018).
17. Stankovic, J.A. Misconceptions about Real-Time Computing. *IEEE Comput.* **1988**, *21*, 10–19.



© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).