Modeling, Prototyping and Evaluating Internet of Things Solution for Urban Traffic-light Control

Jérémy Petit[1], Aghiles Djoudi[3], Rafik Zitouni[23], Laurent George[3]
[1] VEDECOM Institute, [2] ECE Paris, 37 Quai de Grenelle, 75015 Paris
[3] LIGM/ESIEE Paris, 5 boulevard Descartes, Cité Descartes, Champs-sur-Marne
jeremy.petit@ece.fr, aghilesdjoudi@gmail.com, rafik.zitouni@ece.fr, laurent.george@esiee.fr

Abstract—Major traffic light's control systems of actual cities are wired and have a static behavior. They are only time-based or with a pre-configured pattern. Even if the connectivity between vehicles and lights is now possible, it remains insufficient to ensure adaptability, scalability and interoperability among wireless and wired networks. One of the main criterion of such systems is Quality of Service (QoS) or delay aware. In this paper, we propose an Urban Traffic Light Control based on an IoT network architecture (IoT-UTLC). The objective is to interconnect both vehicles and roads' infrastructure to traffic lights through an IoT Cloud platform. We designed our IoT-UTLC by selecting sensors, actuators, wireless motes and protocols. Message Queuing Telemetry Transport (MQTT) protocol has been integrated to manage QoS. It enables lights to adapt remotely and smoothly interrupt traffic light's classic cycles. After verification and validation using a UPPAAL model checker, our system has been prototyped. Motes functions have been implemented on Contiki OS and connected through a 6LoWPAN wireless network. Time-stamping messages have been performed throughout the system to evaluate the MQTT protocol with different reliability levels and data rates. Our experimental results show the MQTT protocol decreases the packets delays when the packets number exceeds 35% of all the traffic.

Index Terms—Internet of Things, Smart Cities, Wireless Sensor Networks, IoT Cloud Platform, 6LoWPAN, Contiki OS, QoS, MQTT, UPPAAL

I. Introduction

The exponential growth of 5G networks and the development of IoT that will greatly come with it, would considerably raise the number of Smart Cities applications. The aim of such technology is mainly to improve the comfort and the safety of users through wireless IoT networks. Wireless Sensor Networks (WSNs) are the source of sensed data of cities things, i.e. roads, cars, pedestrians, houses, parking, etc. The cloud is the entity that collects the sensed data and allows users and machines to do data analysis and improve services. For Smart Cities, one objective is improving the welfare of citizens as well as its safety getting a real-time information about the city infrastructure. One application would be the transportation systems, and traffic lights control having as an objective avoids congestion and dangerous situations. A static cycle of traffic lights has a direct impact on traffic jams. The long period at red or green light could impact the fluidity of the city traffic.

The Internet of Things (IoT) would give an answer to the required interoperability between heterogeneous wireless networks. Our objective is to model, prototype and evaluate a traffic control system. Indeed, different infrastructures have different purposes and technologies, this means that it is not possible to state communication between two infrastructures following a Device-to-Device approach. However, thinking of an indirect or Device-to-Cloud communication between infrastructures seems useful when every connected system has its own technologies, e.g. Zigbee, LoRa, SigFox, ITS-G5. Consequently, IP stack would be the suitable mediator for interconnecting these networks. It removes the barriers of rigid standard specifications of the hardware despite the overhead of the extra network configuration. Furthermore, we want to have a scalable solution not limited only on traffic light management system. We can deploy sensors and actuators to measure noise or air pollution via panels or roads and offer new services, e.g. where and when a jogging is better.

To implement our Urban Traffic Light Control based on an IoT network architecture (IoT-UTLC), we setting a real IEEE 802.15.4 WSN devices that would act as actuators and sensors. All this small traffic light devices are driven by a Border Router (BR) which is a gateway to the Internet. This BR would be connected to a host computer (or sink) connected as well to an IoT Cloud platform. WSN devices forward their data to the IoT cloud through the sink. The collected data could be transmitted to different devices such as sink, BR or wireless sensor/actuator devices. This platform would collect data and exchange with clients, in our case the sink. When WSN devices detect the arrival of priority vehicles, sensed data are routed to the IoT cloud. After that, the sink takes a decision to change the light's state and forward generated messages to actuator devices through BR. Thanks to the IPv6 over Low power Wireless Personal Area Network (6LoWPAN) chalappuram_development_2016 our WSN would be energy-efficient and IPv6 accessible. This paper is a proof of concept that we are developing at ECE Paris, to simulate how traffic lights could be more adaptive in certain situations such as prioritizing specific vehicles or reducing pollution.

This paper is organized as follows. Section III overviews the design of our prototyping by defining the conception

and the goal of this solution. Particularly, this section describes the use case we defined with the design model to see how this project has been modelized. Section V will define how we managed to prototype it by listing and explaining our specifications and choice of technologies for this project. Finally, Section V presents the obtained results that show evidence of the best practice for using MQTT and its QoS functionality by pushing the limits of our system.

II. Related work

Petri nets was widely used for traffic light modeling and control in the literature. For example, Febbraro and Giglio difebbraro_trafficresponsive_2006 applied deterministic-timed Petri nets, the authors build a model consisting of two signalized intersections. Undesirable deadlock states appear when the nets were tested in some case studies. In an other paper, they modified their model and applied stochastic-timed Petri nets [1] to model one single signalized intersection. Dotoli and Fanti [2] built a colored timed Petri net with a deterministic modular framework, Examples using modularity are given in Soares and Vrancken [3], in which a p-timed Petri net is used for the control of a traffic signal in both main road and side streets. However, formal characteristics of PNs (e.g., deadlock and liveliness) haven't been discussed.

In Web0 cameras and on-street wired sensors detect vehicles and pedestrians in order to adapt the cycle of traffic light control systems. However, such a solution has to be implemented for crossroads with huge road work to install expensive extra infrastructure. Moreover, the system uses only its local view of the environment. Other solutions use recent technologies such as wireless sensors devices to limit the cost and reduce the time-work deploying extra hardware. In [4] and rose_internet_2015 the authors propose an adaptive system based on local wireless communication between lights and vehicles. An intelligent solution needs a global interconnection between all road's users and infrastructure even if their communication technologies are different.

The focuses of all previous cited papers are on the structural analysis of their models and the transitions between signal indications (3 lights) However, the implementation of their models as a service in smart cities is not discussed in their approach. Moreover, their methodology is not tested with any real traffic data.

III. Use Case and Model Design

For our system, we want our implementation to be robust and able to guarantee that it behaves like in real environment. In [5], a crossroad traffic light design has been proposed based on Petri Nets. Authors demonstrated the necessity of monitoring checkpoints like transitions from red to green and define critical control points to be sure that the model is correct. Authors have also shown its limits, they let us see vulnerable points of control we miss and to what kind of solution could be used to avoid bad behaviour in case of packet loss which is acknowledgments. Indeed, this model is theoretical and static, and would not model entirely our project. Thus, we decided to go further in modelling our project.

In this section, we explain our model, we detail our use cases by showing where we are headed. Next, we present our model using UPPAAL.

A. Use case

We choose for our first use case, vehicles with high priority like ambulances, fire-fighters or public transportation. Time is important in such use cases, being able to cross a city from A to B without experiencing traffic jams could take users much more time than expected. So, in our vision of the use case, a vehicle will start by approaching a crossing with the way signals at red. The next step would be notifying yourself to the network that you would like to pass your way at the green light. In that case, it could be possible to interrupt the usual cycle of that crossing, get your way at green light to continue your itinerary in order to gain time. Thus, we naturally chose, a crossing with traffic lights. It will be composed of two roads and two traffic lights by roads for a total of four traffic lights. To simplify our use case, we planned on having a single priority vehicle coming to a crossing with no other vehicles on the road. Situations like multiple priority vehicles or even crowded roads will not be exposed in detail for this project.

In order to match as close as possible real urban traffic, we made the closest representation of a crossroad in Paris. We took an actual crossing with its dimension and timers. We modeled our solution with a scale of 1:68. It works as a simple traffic light system, where signals on each road will have opposite colors (or state). Every 30 seconds, the traffic light will change their states, starting with green lights switching to yellow for 3 seconds then go to red and after that red lights will go to green after a 5 seconds extra delay. This extra delay is made to avoid any synchronous problem that could occur.

Figure 1 shows that both traffic signal and vehicle are connected wirelessly. These tow elements could use different technologies as presented with different colors. To access the Internet, all messages sent by those elements will pass through a router. This router would be connected to the Internet and will forward all packets. The Cloud would be in charge of processing those messages and send responses to vehicles and traffic lights by using the same "channel" of communication.

B. Design Model

To model our rigorous design, we use a model checker software which is UPPAAL. UPPAAL is a timed-based modeling software with a graphical user interface, it is created in collaboration between the Department of

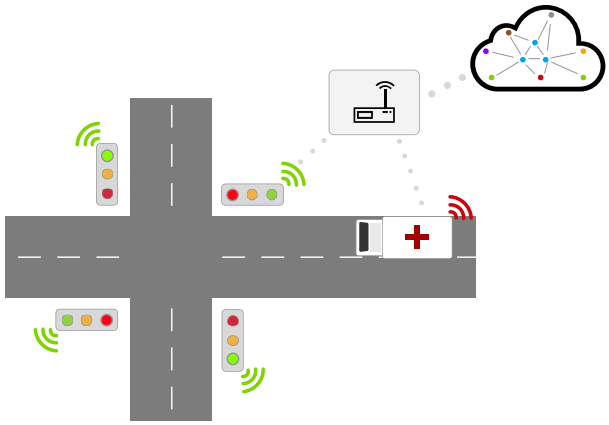

Fig. 1: Use Case Illustration.

Information Technology at Uppsala University in Sweden (UPP) and the Department of Computer Science at Aalborg University in Denmark (AAL). It allows us to model how our system works and simulate every possibility. It can use networks of automata agreement with clocks and data variables. In our work, we proposed a UPPAAL [6] model to verify formally the change of green, yellow and red states of our IoT-UTLC. We simulated our system through the automata (Figures 2. 3. and 4.). We proved that our model worked without deadlock, this means that in our system, there is always a transition to get to the next state. Obtaining it, proves that the system will not stop functioning during up time.

Designing this system is important to avoid failures. Traffic lights are a safety infrastructure, some behavior like four signals at green should not occur, we have to set rules and model our system to list the dangerous states and avoid them. The goal of avoiding deadlock was reached getting a functional system to work with.

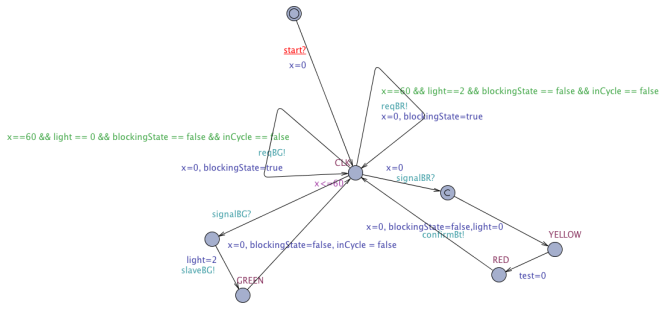

Fig. 2: Model of our Traffic Lights in UPPAAL.

Figure 2 shows the model of the traffic light. It shows traffic light behavior explained previously by sending a request every minute to change its state. When it gets an answer, a cycle is started to pass the signal to the desired state. The algorithm of that mechanism is presented in 1.

Algorithm 1: Traffic light

```
1  init_60s_timer(); while true do
2  |  if end_timer() then
3  |  |  send_request_new_state(); reset_timer();
4  |  end
5  |  if msg_received_red() and my_state is green
   |   then
6  |  |  change_state(yellow); wait();
   |  |   change_state(red);
   |  |   send_confirmation_to_middleware();
7  |  end
8  |  if msg_received_green() and my_state is red
   |   then
9  |  |  change_state(green);
   |  |   send_confirmation_to_middleware();
10 |  end
11 end
```

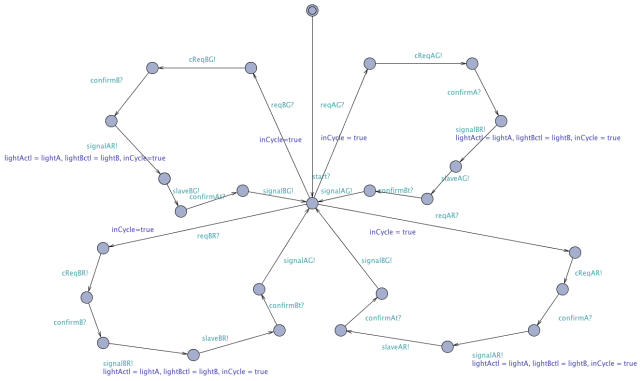

Fig. 3: Model of our middleware in UPPAAL.

Every traffic lights, masters and slaves, has to be handled by the middleware. The model in Figure 3, let us see the different possibilities in term of intern cycles depending on the request made by a traffic light. It mainly sends the message to the cloud and wait for its response. Next, {it sends messages to traffic lights master and slaves to change their state using the order, every signal go to red light before setting green signals.} The algorithm of that mechanism is presented in 2.

At the end of that communication, we have the IoT Cloud Platform showed in Figure 4. We simulate the subscription mechanism, according to the message sent by the middleware to update its own data.

IV. Prototyping

After defining what we are trying to achieve in this paper, we focus on this section on technologies used to implement our project. One evident choice was to use a wireless solution. Indeed, we want to avoid excessive costs and work on an existing crossroad for instance. We were looking for something portable to be placed on existing

Algorithm 2: Middleware

```
initialization();
while true do
    if received_red_from_roadA() then
        send_msg_to_Cloud_Platform();
        confirmation_msg(); send_red_to_roadA()
        get_confirmation_from_roadA();
        send_green_to_roadB()
        get_confirmation_from_roadB();
    end
end
```

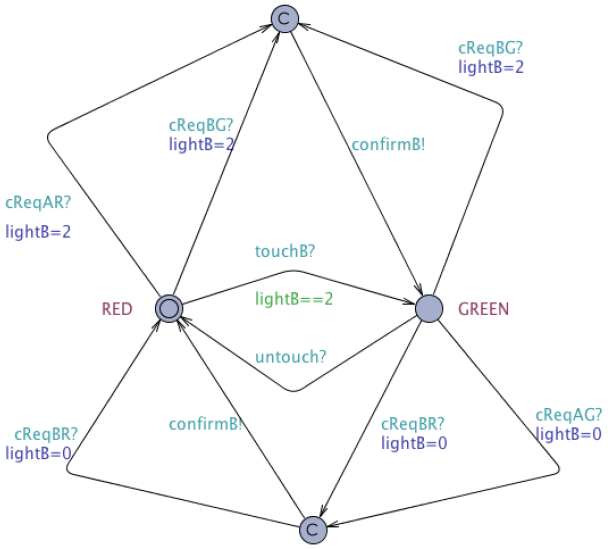

Fig. 4: Model of our Cloud variables in UPPAAL.

infrastructures and efficient depending on the situation. In addition, we wanted something that will not require a lot of power all the time, this is why we were looking for an energy-efficient device.

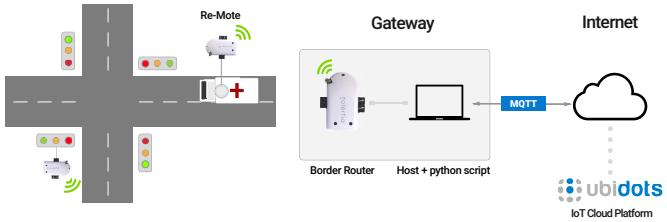

Fig. 5: Architecture of our Iot-UTLC.

Figure 5 shows the architecture of our IoT-UTLC with three parts. From left to right, we have the WSN part with connected traffic lights' actuators, sensors and transceivers. The second part is the Gateway of the WSN ensured by the Border Router. The last part is the Ubidots IoT Cloud Platform, it allows us to collect WSN information and trigger actions. Those different parts will be exposed on the following subsections.

A. 6LoWPAN, Contiki OS, Re-Mote and Border Router

A part of our WSN, we use an IPv6 LowPower Wireless Personal Area Network (6LoWPAN). It is well adapted to embedded developments with interesting features in addition to its own low power wireless network. One of them is the possibility for a mesh network to interconnect the devices inside the network. This network relies on two standards = IPv6 and IEEE 802.15.4 [Quick explanation of encapsulation, mesh network]

To implement those IoT devices, choosing Contiki [1] was a normal choice for us because of its low-power consumption capabilities. It is an operating system that has all the tools and protocols to develop wireless solutions with low-power consumption in mind. It is a great choice for embedded systems and fully supports Zolertia's Re-motes which we will be using. It has its own IP network stack that is compatible with the IEEE 802.15.4 standard and recent protocols like 6LoWPAN, RPL, CoAP or MQTT. It has also a huge community and a fine list of examples for different platforms to use in order to help develop solution.

To set up this WSN, we use Zolertia's Re-motes [2] which are fully compatible with this type of network. They are wireless devices with power consumption utilities and an Ultra-low power operation mode. This choice was motivated by long radio range, e.g. two frequency bands 868-915 MHz and ISM 2.4 GHz. The Re-motes contain IEEE 802.15.4 transceivers with the 6LoWPAN stack. It has analog and digital ports, which allows us to bring several sensors and actuators if needed. A Re-mote could be driven by a computer and become a sink or a border router by being the gateway between the 6LoWPAN network and the computer.

why one better than the other wifi frequency => possible perturbations-] To implement this model, we used 6 Re-motes. One for a border router, four to simulate the traffic lights and at least two Re-motes to simulate the arrival of a priority vehicle near the crossing.

expliquer pourquoi utiliser ceci à la place de RFID pour l'instant et leur modélisation dans notre système] In our prototyping, we chose to represent our priority vehicles by using a Re-mote connected with a touch sensor. It allows us to detect when the vehicle is near the crossing and act accordingly.

We have four types of programs running on Re-motes for this prototype: Traffic lights, Sensors, Priority vehicles and a Border Router. Sensors will send periodically information to the IoT Cloud Platform with temperature,

[1] http://www.contiki-os.org/
[2] https://github.com/Zolertia/Resources/wiki/RE-Mote

pressure or any relevant information that could be sensed. Traffic lights are sub-divided into two modes or roles: slaves and masters. It is necessary when all devices are working together to avoid a collision from devices on the same road, for instance. Masters would be the only one to request the middleware to change its light's color (or state) and slaves would simply change its state depending on the received packet. The working process could be described as: Master Traffic lights are sending periodically packets to request a change of state to the middleware which forwards them to Ubidots. By subscription, the middleware gets new states for the traffic lights and sends them accordingly to the master and slave traffic lights. It sends red states first and then green states to avoid unwanted behaviors. It also uses acknowledgments from the traffic lights to ensure that the new state has been set. In order to do those two features, we used a system to retain messages if the IoT Cloud Platform send green states before red states. Algorithm 3 shows how it works.

Algorithm 3: Confirmation and packet order

```
1  if message_received() then
2  |  if is_green() then
3  |  |  retain_msg(); //green then red
4  |  end
5  |  else
6  |  |  if retained_msg_exist() then
7  |  |  |  update_to_red(); //green then red
8  |  |  end
9  |  |  else
10 |  |  |  update_to_red(); //red then green
11 |  |  end
12 |  end
13 end
14 while true do
15 |  confirmation_red_lights(); update_to_green();
   |  confirmation_green_lights();
16 end
```

The border router is at first a Re-mote, but it has very different behavior and function. Indeed it has the task of re-routing every packet it receives from its cohorts to the serial port of the host machine. This machine will create a connection to the IoT Cloud platform and send the Re-motes messages to it and get responses for the Re-motes. The border router is a central node because it knows all to Re-motes in the 6LoWPAN which packets have transited by it. It acts as a Router and has a routing table of those Re-motes that pass through it. A web server page can be accessed to retrieve that information. We also tried a different approach of it.

what is good about it

As we have seen before, this approach requires a Border Router linked the computer itself to the internet, we tried to see if we can remove one part. Thus, we added a component to the Re-mote in order to connect it directly to the internet via an Ethernet cable. It acts as a full router and needs some adjustments on different Re-motes used previously. Indeed, if the border router becomes an Ethernet router (Figure 6), there will no longer be any connection between the machine and the IoT Cloud platform. Every Re-mote has been able to connect to the IoT Platform. This approach has quite some advantages, such as the autonomy of the devices, but it means also more connections, more packets and more difficulties to sync Re-motes with each other. Therefore, the first approach with a host machine was chosen because we can have more control over it and could be more flexible and resilient for our project.

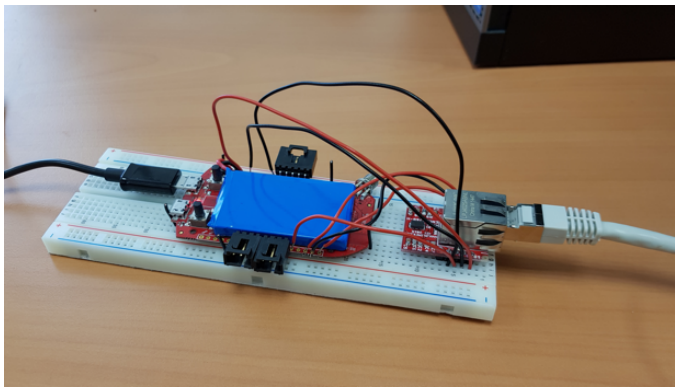

Fig. 6: Our Ethernet router.

B. MQTT and UBIDOTS

We have seen in previous section what we used for our local sensors network, let's see now how we communicate with the IoT Cloud Platform. We will use MQTT which is a light-weight transportation protocol. In our solution, a python script will run an MQTT client to connect our WSN to Ubidots. MQTT ensures the QoS and publishes/subscribes mechanisms with a noteworthy topic organization. The topics are strings used to filter messages and define the hierarchy of our data structure. They allow us to organize how to receive multiple data from sensors such as temperature, up time, battery status and how to display them and obtain a real-time glance of our system. In addition, the broker behaves as a server by filtering messages and organize them in topics. It gets its messages from publishers and will send any modifications to entities which would have subscribed to the updated topics. We used this mechanism with the middleware in order to publish messages to the broker and get from the main topic the new values using the subscriber functionality.

The MQTT protocol uses Inserv mechanisms to increase the quality of services of the network, it tags incoming packets in the edge routers with different levels of priority. Core routers read incoming packets headers and queue them according to their priority, packets with high priority

will be sent most frequently than packets in low level priority queue.

The QoS is a feature in the MQTT protocol to manage network resources by handling re-transmissions and to guarantee the delivery of messages. It allows more control on messages by defining the level of guarantee we want. The QoS is defined by three levels. The first one, level 0, is 'At most one'. It is a non-connected way like UDP where no confirmation would be made. The level 1 is 'At least one' where there is an acknowledgment to let the sender know its packet has been received. Finally, level 2 'Exactly once' is the highest verification with a request/response flows to ensure that only one message will be delivered and processed by the receiver.

As for the MQTT broker, we wanted an easy and powerful IoT Cloud Platform. It will be compatible with all technologies we chose for our prototyping, so with an integration of MQTT and QoS levels. an IoT Cloud platform is a central point of the system as it keeps all the information about our WSN. Using the Cloud allows the system being accessible from anywhere on the internet. It is also a tool to filter and display relevant information in real-time from our system in the centralized dashboard. It can be possible to trigger some actions on the system via the dashboard.

Building this virtual infrastructure for this project has been challenging. we used MQTT topics mechanism to get the most of Ubidots to structure every data sent. We have a main topic which contains the states of the traffic light (RoadA and RoadB) in real time, we created individual topics for every Re-mote acting as traffic lights or sensors. With this architecture, we can have deep information on every device (such as its battery, sensors data, etc...). Moreover, it could be scaled to match future needs.

V. Results

In this section, we report the results of applying the MQTT protocol in our system, we report also the limits of technologies used in this project. To test the efficiency of our MQTT protocol, we made 2 scenarios, in the first scenario, the frequency of sending packets is 1 packet per 10s, in the second scenario, the frequency of sending packets is 1 packet per 1s. Those delays are taken between the gateway to the cloud platform (Ubidots) through a wired connection. In each scenario, more than 100 values have been taken, tests have been reproduced couple times.

To study the behavior of packet delays in each scenario, we select the best distribution that fit the round trip time (RTT) of packets, we choose 3 common distributions normal, gamma and logistic distribution shown in figure 7. Table I shows the correlation matrix between distributions and scenarios used in our study. Results show that packet delays in both scenarios with the level of QoS match better the logistic distribution than the tow other distributions.

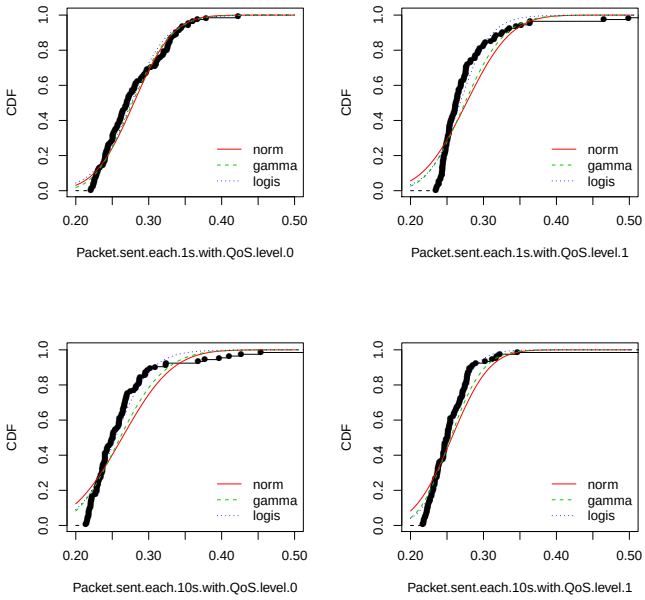

Fig. 7: Normal,Gamma and Logistic distribution.

The standard logistic law is the logistic law of parameters 0 and 1, its distribution function is the sigmoid:

$$F(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

	norm	gamma	logis
1s with QoS level 0	172.12074	175.2950	185.4433
1s with QoS level 1	159.59630	172.8193	189.7002
10s with QoS level 0	146.85668	161.2369	175.3682
10s with QoS level 1	176.28502	192.6108	204.3235

TABLE I: Correlation matrix.

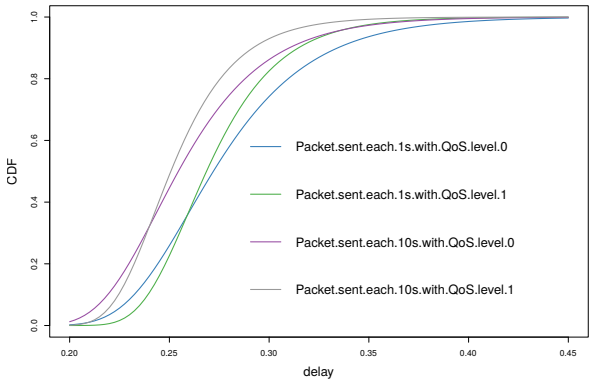

Fig. 8: Cumulative distribution function of RTT delay.

Figure 8 shows that the MQTT protocol is more efficient when the number of packets is greater than 35% of the

total number of packets sent, this can be explained by the fact that priority queues are more useful when all levels of queues are filled, so the packets with a high priority level will achieve their destination faster. Moreover, when we tried to increase the number of packets sent to Ubidots at one per second, the MQTT still offers the same efficiency with suitable delays.

VI. Conclusion

This paper has gone through the architecture's elements and tools used to build our prototype. We see that IoT could platforms are useful to interconnect different network technologies by supporting more dynamic use cases in UTLC systems. The MQTT protocol used in our system to descriminate between traffic priority when exchanging messages is efficient since the delay wasn't compromised by increasing the number of packets sent. Our experiment investigates the relationship between using the MQTT protocol and the traffic flow congestions, results show that the MQTT is efficient when the number of packet sent exceeds 35% of the total number. While we are still developing our project, we plane to investigate further use cases such as smart buildings and industrial IoT and measure and compare the required QoS of each cases. As a near future work, we plane to extend our experimentation to measure both delay and packet delivery ratio (PDR) in order to get more information about the behavior of our system.

Acknowledgement

We would like to thank our colleague Sebti Mouelhi, the associate professor from ECE Paris who provided insight and expertise on UPPALL.

References

[1] A. D. Febbraro, N. Sacco, and D. Giglio, " On Using Petri Nets for Representing and Controlling Signalized Urban Areas: New Model and Results," in 2009 12th International IEEE Conference on Intelligent Transportation Systems, 00018, Oct. 2009, pp. 1–8 (p. 2).

[2] M. Dotoli and M. P. Fanti, " An Urban Traffic Network Model via Coloured Timed Petri Nets," IFAC Proceedings Volumes, 7th International Workshop on Discrete Event Systems (WODES'04), Reims, France, September 22-24, 2004, vol. 37, no. 18, pp. 207–212, Sep. 1, 2004, 00101 (p. 2).

[3] M. dos Santos Soares and J. Vrancken, " A Modular Petri Net to Modeling and Scenario Analysis of a Network of Road Traffic Signals," Control Engineering Practice, Special Section: Wiener-Hammerstein System Identification Benchmark, vol. 20, no. 11, pp. 1183–1194, Nov. 1, 2012, 00017 (p. 2).

[4] M. Tlig, O. Buffet, and O. Simonin, " Decentralized Traffic Management: A Synchronization-Based Intersection Control," in 2014 International Conference on Advanced Logistics and Transport (ICALT), 00013, May 2014, pp. 109–114 (p. 2).

[5] Y. Huang, Y. Weng, and M. Zhou, " Modular Design of Urban Traffic-Light Control Systems Based on Synchronized Timed Petri Nets," IEEE Transactions on Intelligent Transportation Systems, vol. 15, no. 2, pp. 530–539, Apr. 2014, 00050 (p. 2).

[6] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen, " Uppaal SMC Tutorial," International Journal on Software Tools for Technology Transfer, vol. 17, no. 4, pp. 397–415, Aug. 2015, 00176 (p. 3).