An open source research framework for IoT-capable smart traffic lights

Gianluca Brilli, Paolo Burgio
University of Modena and Reggio Emilia, Italy
{gianluca.brilli, paolo.burgio}@unimore.it

***Abstract*—Recent technological advances are completely reshaping the way we build our cities, and the way we enjoy them. Future smart cities will employ a number of smart sensors, which cooperatively work to deliver advanced services that improve security and quality of life. The capability of deploying and testing such technologies directly on-the-field is paramount to research, however comes with a significant effort in terms of time and price. For this reason, we introduce an open-source design framework for highly-connected smart sensors, and we implemented it in an advanced controller for traffic light, providing a single component to support researchers and engineers from the earliest stages of development in laboratories till on-the-field research and testing.**

I. Introduction

Fig. 1. Connected traffic light to enable safe automatical stop

In the last ten years, advances in the field of embedded systems enabled a new generation of power efficient, high-performance computing platforms, that paved the way to future automotive systems and smart transports. In future cities, vehicles and environmental sensors such as traffic lights and cameras will be connected with low-latency reliable media such as 5G, short-range DSRC/802.11p WiFi [8], [13], Low-energy Bluetooth (BLE) and Zigbee [12], and cooperatively exchange information to build a common vision of the city environment, and deliver services to increase safety and reduce pollution in our roads.

For instance, figure 1 shows a possible scenario where a vehicle automatically adjusts its speed depending on the light phase and time-to-change of a smart traffic light. This both increases safety (no "last-second braking"), and reduces CO2 emissions by removing unnecessary brakes and acceleration, since now the vehicle potentially drives at constant speed. We foresee [2], [6] that the Global Connected Car Market will value 220 million of dollars by 2025, with North America having the highest number of connected vehicles on their roads. Also, European countries are moving in this direction: the European Union is promoting research in the field, and allocated a significant budget for their H2020 program.

At the local level, several countries already created special urban areas to test and stress vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication, and to research technologies for the smart cities of tomorrow and autonomous driving. Many projects involving smart city applications have been proposed in the last decades with an increasing effort of providing urban environments with pervasive and ubiquitous computing capabilities thanks to digitally instrumented devices [1]. This includes wireless/wired communication networks; remotely controlled utility services; smarter and better synchronized public transport; traffic, air and pollution sensors; security camera networks; smart public lighting; mobile applications for citizens and tourists, etc. One of the most notable European examples is given by SmartSantander [1], an app providing information about multiple places in the city of Santander, along with real time access to traffic and beaches cameras, public buses information and bike-rental service. For the city of London, an online city dashboard [2] provides similar live feeds. The Italian city of Modena, recently developed the Automotive Smart Area (MASA [3] – see Figure 2), an ambitious program to set real urban laboratory of one square kilometer with 5G connectivity, enabling multiple (IoT) devices (e.g., smart cameras, traffic scanner and counter, smart traffic lights...) to exchange massive amounts of information to jointly cooperate for an efficient traffic management, creating a so-called city distributed awareness of what happens in

[1] http://www.smartsantander.eu/
[2] http://citydashboard.org/london/
[3] https://www.automotivesmartarea.it/

the area. The existence of places such as MASA enables research in the field, however, accessing them is cumbersome, because they are *public* city areas, i.e., with limited access. For this reason, typically researchers employ simulators (Such as MatSim [3]) at early stages of their work, to efficiently test their solutions, before accessing the "real" physical area. The main issue with simulators, is that sensors, such as cameras and traffic lights cannot be accurately replicated, especially with respect to communication latencies between them and vehicles. Since communication between city sensors and vehicles is the key aspect of such systems, researchers struggle to model the timing behavior of such components in laboratories, at early development stages, without having to wait to go "on the field" to stress these functionalities. Unfortunately, this is not always possible, and especially sensors behave quite differently in simulators and in "real" installations.

Fig. 2. The Modena Automotive Smart Area

We want to tackle this problem, and in this this work we provide an open-source, low-cost and easy-to-use software infrastructure for smart city edge sensors, based on state-of-the-art, realistic low-latency communication protocol. We instantiated it in a small controller for traffic light, which runs on low-end computing platform, a Raspberry Pi 3 [10], yet providing advanced functionalities such as remote controller via a web-based frontend, and supporting several communication protocols such as low-overhead UDP (used in MASA), Modbus, and Https.

Section II describes both the communication protocol, and the modular design of the controller. In section III we depict more in details the hardware and software architecture implementation depicted more in details This software is released open-source with GPL license [4]

II. System design

Our goal is to build a traffic controller system that:

- is open-source, and easily instantiable and usable;
- supports a range of state-of-the-art communication protocols;
- can easily be extended by adding modules;
- can be used at different stages of development, i.e., from simulation in laboratory to "real" on-field deployment.

For this reason, the controller is implemented to run as a standalone component on a Raspberry Pi 3, but we also provide instructions on how to create the required electrical connectivity to drive a "real" traffic light.

With respect to communication protocols, we want to be able to remotely control the system, hence we implemented a control console in a web page, which sends commands and receives the status of the traffic light *via* a web-server. This is also released as open-source. More in details, we employ three different lights controllers, described in the next section, namely i) an internal timer with pre-defined time phases, ii) an external webpage, which enables complete control of the light phases and status; and iii) a server/slave which accepts commands from an external *master* using the Modbus protocol. Figure 3 depicts the hardware/software structure of the semaphore. Please, note that this structure applies to any sensor type (e.g., cameras, traffic sensors..).

Fig. 3. System design

[4]https://github.com/HiPeRT/IoT-Semaphore

Fig. 4. The semaphore controller

III. System Architecture

This section describes first the required electronic connectivity to drive the phases of a real traffic lights, then shows how the software controller is implemented.

A. Hardware Architecture

The heart of the system is based on the Raspberry Pi 3 Single Board Computer [10], a board that embeds a Broadcom BCM2837 SoC [4] consisting of a 1.2 GHz 64-bit quad-core ARM Cortex-A53 CPU and a 1 GB memory (shared with the GPU) LPDDR2. Furthermore, the Raspberry Pi 3 is equipped with all the necessary connectivity (GPIOs, Ethernet, WIFI, etc.) for the realization of an IoT device connected to the external world and an adequate computing capability with a power consumption of about 2 Watt [9], which makes it particularly suitable for implementing battery-powered systems.

The traffic light power supply system is handled entirely by a single 12 V car battery, to which are connected the required voltage regulators that allow to serve all the voltages necessary for powering the system in its entirety, including the lamps of the traffic lights. In this way, in theory, the system can keep to work for several weeks without human intervention. The switching of the traffic light's lamps connected to the control unit is managed by relay boards (one for each traffic light) that are directly driven by the GPIOs of the Raspberry Pi 3 itself. In figure 5 we can see the schematic adopted for building the electronic of the system.

Fig. 5. Electric scheme to connect the system with a real traffic light

The main objective of the developed circuit is to amplify the output voltage of the GPIOs of the Raspberry Pi 3, which is fixed to 3.3 V, through an external 5 V power supply and a BJT transistor for each GPIO. By doing so, it is possible to supply the right voltage to the relays.

B. Software Architecture

In this section, we describe the three main communication protocols supported in our package. Thanks to its modularity, it can be easily extended to support many more, however, these already cover, and are representative of, state-of-the-art communication *media* for next-generation smart cities.

Modbus. From the point of view of system connectivity to the outside world, it was decided to exploit a wired connection (via ethernet) for managing the Modbus protocol [11]. Modbus is a serial connection that over the years has become the *de facto* standard for PLC communication in an industrial environment. As part of the project, it was used through the

Fig. 6. Electric scheme to connect the system with a real traffic light

`Libmodbus` library [5], a version based on the TCP / IP stack. In this way a generic *Modbus master* is able to control the traffic light crossroad. It was also decided to interconnect the developed system with a server in the cloud by means of a 4G LTE connection. In this way, through any device connected to the Internet, it is possible to visualize and change the status of the smart crossroad and obtain some relevant information, including the *time-to-green* and the *time-to-red* , respectively the time that elapses when the state changes from red to green and from yellow to red. This information exchange happens via the HTTPs protocol.

HTTPS server. We deployed an HTTPs server with a GUI that enables i) real-time monitoring of the semaphore status and ii) sending command to the semaphore controller. It is shown in figure 7. The GUI is implemented as a simple static webpage, and it fetches information through REST calls to a Web API, exchanging JSON payload

```
{
  "id" : "SEM1",
  "modbus_init" : true,
  "status" : "Red",
  "ctrl_type" : 3, // internal
  "time_to_change" : 4000 // in ms
}
```

Low-latency Real-Time UDP (MASA). In addition to the two aforementioned protocols, we decided to integrate a socket-based client-server communication mechanism, based on the UDP protocol, called *MASA Protocol*. UDP is an *unacknowledged* protocol, which means that there is no mechanism to retransmit data that has been lost during the communication. This characteristic makes UDP superior for real-time communications, infact there are fewer overheads that take up useful space, that could be used for sending relevant data, making transmission *quicker* and potentially *more efficient*. Also in this case we used an aggregator server that collects all the data received from the traffic light crossroad and from the other connected road users. The format of the messages used by the MASA Protocol includes the following fields:

- *Traffic Light Position*: geographical position of each traffic light, including latitude, longitude and orientation;
- *Traffic Light Status*: current state of lamps of each traffic light;
- *Time-To-Change*: time left for each traffic light to change its state.

The position parameters of each traffic light are specified by the following fields inside the configuration file:

```
"semaphores": [
  {
    "position": {
      "latitude"   : 0.1,
      "longitude"  : 0.2,
      "orientation": 2
    }
  }
]
```

If we need to use both Modbus and HTTPs connections at the same time, we can configure the Raspberry Pi 3 connectivity as follows:

```
auto eth0
allow-hotplug eth0
iface eth0 inet static
  address 169.254.53.170
```

```
auto wlan0
iface wlan0 inet dhcp
wpa-ssid Semaphore1
wpa-psk
```

In this case we have the *eth0* interface for handling the Modbus protocol (with static parameters) and the *wlan0* interface for connecting the system to a LTE router.

Internals, and configuration. From the software point of view, a parallel control program was developed, in C++ language and based on the POSIX Threads standard, specifically the following parallel Tasks were provided:

Fig. 8. caption

- *Thread A*: (*internalCtrl4WayCrossroadThrd*) responsible for managing the parameters *time-to-red* and *time-to-green*. Essentially it receives these two parameters from the calling thread and takes care of decrementing and re-initializing them when needed.
- *Thread B*: (*tlPinsThrd*) responsible for managing the pins of the Raspberry Pi 3 in charge of handling the traffic light lamps. Specifically it decodes the configuration parameters received from the HTTP server and acts accordingly.
- *Thread C*: (*serverHandlerThrd*) manages the client-server interaction of the developed system. In particular it receives the configuration parameters from the HTTP server and sends the status of the traffic light lamps to the server itself.

Fig. 7. Web GUI for the traffic light

- *Thread D*: (*modbusHandlerThrd*) takes care of receiving incoming parameters from a control device configured as Modbus Master.

From the implementation viewpoint, we implemented them using POSIX Threads (PThreads [7]).

The control program can be configured through a JSON configuration file. The main parameters that we can configure are the GPIOs that are connected to a single traffic light lamp:

```
"semaphores": [
  {
    "//": "Traffic Light TLA1",
    "r-pin": 18,
    "g-pin": 24,
    "y-pin": 25
  }
]
```

some generic configuration parameters as follows. In particular we have to configure the control type of the crossroad, a parameter that specifies from which source the Raspberry Pi

3 take the control.

The options that we have are: 1) the traffic light's lamps are controlled through the HTTP server. 2) The system that controls the smart crossroad is a Modbus Master. 3) The phases of the traffic lights are internally-timed, in this way we have access to the aforementioned *time-to-red* and *time-to-green* parameters.

Finally the parameters (*http_server.enable* and *http_server.url*) are used to enable and specify the HTTP server address.

```
"ctrl_type" : 3,
"http_server.enable" : true,

"//": "URL or IP address of the IoT Server",
"http_server.url": "http://your-server.com"
```

If the Modbus connections is required, we can specify some standard parameters, the most important ones are the IP / Port configuration.

```
"modbus-master": {
  "ip-addr": "169.254.53.170",
  "port"   : 1502,
  "debug"  : 1
}
```

We can also specify the delay-times, expressed in seconds, for every semaphore phase. This parameters are used only when the control type is set to internal.

```
"delays": {
  "//": "Time for each traffic light phase expressed
     in seconds",
  "R": 10,
  "G": 8,
  "Y": 2
}
```

IV. Conclusions

We proposed an open-source framework for testing IoT-capable devices and sensors for smart cities. We implemented it in a smart traffic light, which is connected via state-of-the-art communication protocols, namely a "standard" 4/5G data connection, a short-range DSRC/WiFi communication and industrial Modbus standard. It provides a single software component for all development phases, from early stages in the laboratory, to on-field deployment of a traffic light, provided a very simple electric connectivity via relays is implemented. Also, we provide instruction for these. We also released an HTTPs server to remotely control the traffic light, via a web-page.

Acknowledgment

This work was supported by the Prystine Project, funded by Electronic Components and Systems for European Leadership Joint Undertaking (ECSEL JU) in collaboration with the European Union's H2020 Framework Programme and National Authorities, under grant agreement n° 783190, and by the Class project, funded by European Union's Horizon 2020 research and innovation programme under the grant agreement No 780622.

References

[1] A. Greenfield. Everyware: The Dawning Age of Ubiquitous Computing. New Riders, Boston, 2006.
[2] Allied Market Research. Connected Car Market by Technology (2G, 3G, and 4G/LTE), Connectivity Solutions (Integrated, Embedded, and Tethered), Service (Driver Assistance, Safety, Entertainment, Well-being, Vehicle Management, and Mobility Management), and End Market (OEM and Aftermarket): Global Opportunity Analysis and Industry Forecast, 2018 - 2025, 2018.
[3] M. Balmer, M. Rieser, K. Meister, D. Charypar, N. Lefebvre, and K. Nagel. Matsim-t: Architecture and simulation times.
[4] Broadcom Corporation. Broadcom BCM2837.
[5] Libmodbus. A Modbus library for Linux, Mac OS X, FreeBSD, QNX and Win32.
[6] Market Watch. Global Connected Vehicle and Telematics Market with Blooming CAGR in Forecast Period 2019 to 2026, 2019.
[7] F. Mueller. Pthreads library interface. Technical report, 1999.
[8] Y. Y. Nasrallah, I. Al-Anbagi, and H. T. Mouftah. A quality of service model for ieee 802.11p communication protocol in a smart city. In *2014 Global Information Infrastructure and Networking Symposium (GIIS)*, pages 1–3, Sep. 2014.
[9] pidramble.com. Power Consumption Benchmarks.
[10] Raspberry Pi Foundation. Raspberry Pi 3 Model B, 2016.
[11] The Modbus Organization. Modbus.
[12] The Zigbee Alliance. The Zigbee Portocol, 2002.
[13] Wikipedia. Dedicated Short-Range Communication, 2003.