Wireless service for temporary traffic lights management

Principle

The aim of my project is to be able to manage red LEDs state on different M3 nodes which could be assimilated as temporary traffic lights (circulation in case of accident, public works). Thanks to the SSH front-end, which could be in practice a smart-phone or a tablet, we should be able to set the state of one or several lights on the network nodes. It could also be possible to set links between one node's LED state and another node's LED state (for example, making sure that one is always off while the other is on). As traffic lights usually change their state periodically, we should also be able to set a time interval on our front-end. Finally, the state of each node's LED should always be readable on our remote device.

Through this project, I have mainly thought about public workers which would appreciate having an easily installable system to manage circulation during temporary works on crossroads for example. This system could be a good solution to control the flow of cars according to the traffic on each street dynamically. A worker would also be able to have a global overview of each traffic light current state whenever he wants, and to set a light state manually through the front-end in case of emergency. As the workers cannot be mobilized all the time to manage traffic lights, this system provides automatic toggling of lights, as well as linking traffic lights states each other as I have

mentioned before. In other words, I wanted to experiment a fast and readily configurable system to manage circulation autonomously and dynamically, according to the traffic load.

Architecture of the system

For the aim of the project, the architecture has been simplified as much as possible to make the implementation less complex while highlighting its potential interest. It is composed of 4 elements :

- the SSH front-end
- 1 “controller” node
- 2 “light” nodes

I have chosen to work on two CoAP server nodes, which play the role of 2 traffic light actuators. The state of each traffic light is defined by the state of their red LED : we assume that we have a rough traffic light only composed by a red light. When turned on, the cars should stop. These two M3 nodes are supervised by a CoAP client node. The latter is the “controller”, responsible of automating the whole system : it will be used to set up links between our two lights state or rolling back the light states to the user.

The SSH front-end will be used for two main purposes. On the one hand, it will allow the user a manual control of the “lights” state, or to set a toggling period on one of them. On the other hand, it will communicate with the “controller” node to establish a link between the two lights or getting their current state.

What about the implementation ?

The 2 “traffic light” nodes

These M3-nodes can be seen as CoAP servers. To make the whole system work properly, some resources had to be implemented. First, we had to be be aware of the lights state at any time. Instead of sending it to the controller periodically, I have chosen to implement an “event” resource responsible of sending a message to the controller each time that the state of the red LED actually changes on the node for any reason (toggling, manual set). It avoids useless traffic load sending messages several times whereas the LED state has not changed. I have also used another resource to manage manual state modification requests on the red LED : it responds to CoAP “PUT” requests, sent with the “color=r” variable (the concerned LED is of course the red one), and a payload describing the aimed state (“mode=on/off”). Each time this resource is called, we also toggle the “event” resource to notify the controller.

Finally, another resource accessible with a CoAP “PUT” request has been implemented to set the toggling period of the light node. In practice, it receives the request parameter (period=x) and sets an integer variable to the requested value. The toggling timer is then set up to the new value.

The “controller” node

This M3-node is a CoAP client for the traffic lights, which also has some resources to provide services to the SSH front-end. Therefore, it can be seen as a CoAP server from this last point of view.

- On the **private network side** (traffic lights communication), the “controller” begins with subscribing to the “event” resource of the 2 traffic light nodes. Then, it keeps a size-2 array up to date, storing the current state of each traffic light according to the received events. The new state is readable in the payload of each “led state change” event.

  The controller also stores the link between the two lights, which can be set up either to the parameter “s” (synchronized, in other words the light nodes should always keep the same state), or “i” (inverted, always keeping a different state). This can be achieved thanks to a GET resource with the parameter “link=s/i”. When receiving a notification of any change of state on a traffic light, it makes sure that the link is set up correctly and sends the right requests if it is not the case. For example, let’s assume that the two lights are set up with a “s” link. If the 2nd node’s red LED turns ON for any reason, the controller will make sure that the 1st node LED is also ON. If not, it will turn it on manually. This is how the system becomes autonomous.

- On the **SSH front-end side**, the controller provides a “push” periodic resource, sending the “light” nodes state each 5 seconds to the user. It is also equipped with a CoAP “PUT” REST resource responsible of letting the user set up the link between the lights (either synchronized or inverted).

The user front-end

The SSH front-end communicates both with the controller and each “light” node independently. The system has a python command prompt which allows to to each command that the wireless network service provides.

- The **check** command accesses to the periodic resource of the controller and gets the last periodic notification of the light states as follows :

```
check
(2.05)
Node 1 : 1 ; Node 2 : 1
```

- The **link** command, either followed by ‘s’ or ‘i’, sets up the link between the “light” nodes. We also provide a “period” command, followed by the time in seconds, to set up the time interval that the 1st “light” node should respect. The other light automatically toggles when we would want it to, because of the link state that we have previously set.

- The **period** command should be followed by an integer value, and has the effect to set the toggle time interval (in seconds) into the 1st “light” node.

- Finally, the **set** command used with the concerned “light” node number (1 or 2) and the aimed state (on/off) can be used to set a LED state manually.

All of these commands can be seen on the downloadable video demonstration below. The left window is the SSH front-end command prompt, while on the right side we have, from top to bottom, the 1st light node, the 2nd light node and the controller serial ports output.

Nb : The nodes light state is “0” when light is turned ON, and “1” when turned OFF. This can be disturbing but this is the Contiki’s way to represent the current state of each LED.

Link : http://www.mediafire.com/file/wd97zysl9ezcy7g/coap_application.mp4

Practical details

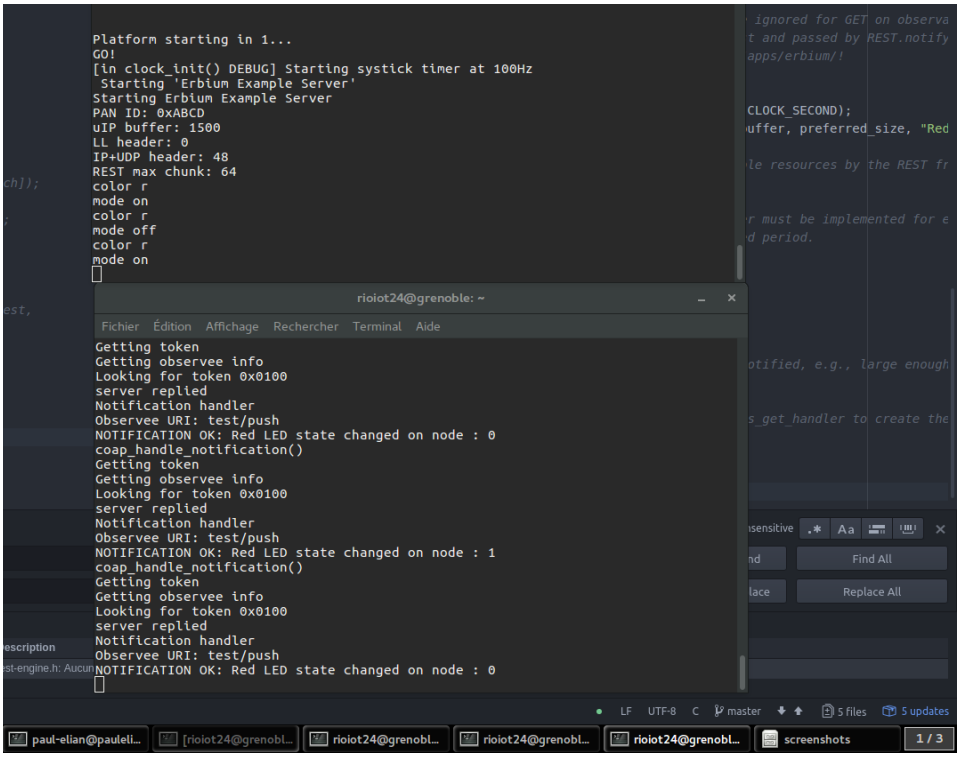
This part aims at giving more technical details about the implementation techniques used on each node. The source code is also available into the project archive.

Traffic lights CoAP server nodes

For the demonstration, I have implemented two kinds of server nodes : one initiates the toggling procedure (he toggles itself according to the *toggle_period* defined value) and sends events to the client when its LED toggles, whereas the other is just actuated from the client each time the first one toggles.

As a result, *er-server-1.c* contains the *toggle_period* variable on which the server node’s timer is based, and toggles the red LED when is expired. Beyond this subtlety, the two server nodes do basically the same job. They work with 3 main CoAP resources that I have mapped with the following URIs :

- **actuators/leds** : sets the red LED state if necessary, can be used both by the SSH front-end and the CoAP client
- **actuators/red_led_state** : observable resource which sends the new state of the red LED when it has been modified, used by the CoAP client
- **actuators/set_period** : receives a toggle time interval from the SSH front-end



Debugging the subscription resource
(top : server node. bottom : client node)

Controller CoAP client node

This node has two different roles : it provides services to the SSH front-end as a **server**, but communicates with the “traffic lights” CoAP servers as a **CoAP client**.

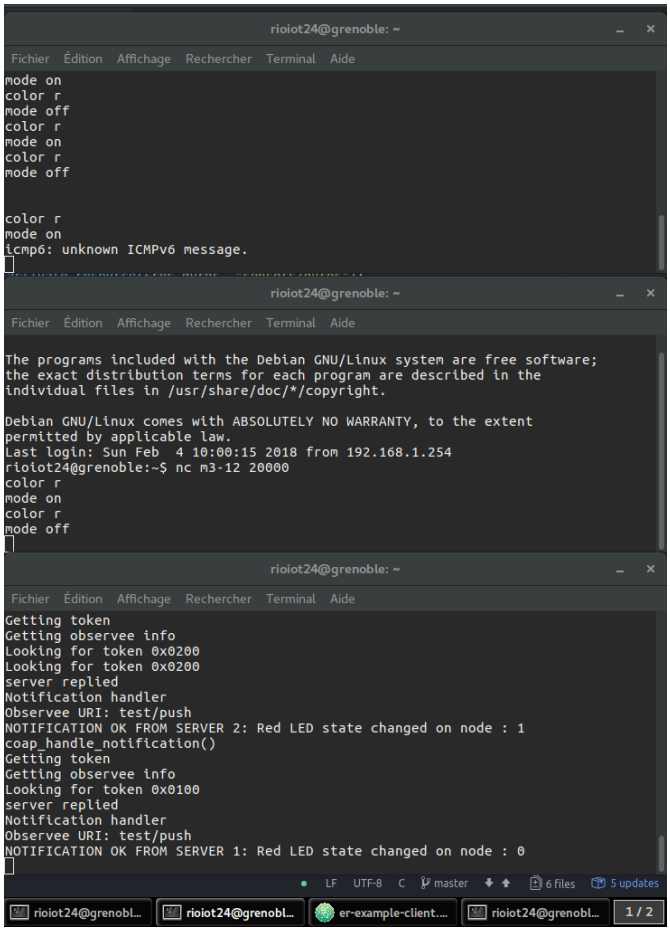
It is by far the most complex node of the network. It combines the ability of storing some of the system settings (link type between the two “light” nodes and their IP address) and of automating this sytem. It also stores the dynamic variables, that is to say the link type and the current state of the two LEDs.

- To always be aware of a node light state change, the client program starts with subscribing to the *actuators/red_led_state* resource on each server node. After that, its main process is yielded and can only be unlocked when it receives a notification from one node. The notification contains the new node state, which allows the controller to maintain constantly the right link between the two nodes. According to the source of the message and its state, we make sure that the other node’s red LED is actually into the right state using the *actuators/leds* resource to set the state if necessary.

```
source = get_notification_source();
destination = get_other_light(source); // If light is “light1”, will return “light2”
link_type = get_current_link_type() ;
if(link_type == “s”) {
	if(state_of(source) == ON)
		send_coap_request_to(destination, ON);
	else
		send_coap_request_to(destination, OFF);
}
else if (link_type == “i”) {
	if(state_of(source) == ON)
		send_coap_request_to(destination, OFF);
	else
		send_coap_request_to(destination, ON);
}
```

pseudo-algorithm to manage the “light” LEDs state

- On the side of the SSH front-end, the controller provides two REST resources, with the following URIs :
  - **test/set_link** : allows the front-end to set the link type which the controller will rely on to automate the system
  - **test/get_leds** : periodic resource which sends the two “light” nodes LED states each 5 seconds to the subscriber (in this case, the SSH front-end)



Controller ability to subscribe to several nodes (server nodes on top, client node below)

SSH front-end

This node is intended to be below the user interface (from a software level point of view) : it provides a simple command prompt so that the user is able to control the system easily as described above.

To achieve this, it first stores the static IP addresses of each node, assigned by the border router. Then, each possible command is translated to the corresponding CoAP request which may concern the **controller** (setting up the link, or subscribing to the periodic resources telling the state of each light) or a **specific light node** (setting up the toggle interval, or manually modifying a LED state).

Link : http://www.mediafire.com/file/wd97zysl9ezcy7g/coap_application.mp4

Conclusion

This project was a good way to overview the possibilities offered by the CoAP procotol for IoT services. The system of **publishers and subscribers** was the key feature for this basic traffic lights system. Even if my system only functions for 2 “light” nodes, we could reasonably imagine the same system implemented with more lights. It would be harder to implement though. Nevertheless, in this type of context (temporary traffic lights), there are rarely more than 5 traffic lights to manage in the most critical cases.

What I find interesting is the **flexibility** and the **ease of remote configuration** without lacking **reliability**. This also avoids a lot of cables going from one traffic light to another, to coordinate each other, which implies **better mobility** of lights as well.

I had not the time to work on an automatic configuration of each node (**dynamic IP discovery** by the controller and the SSH front-end) : it has been done statically into the code. The aim was more to give experimental demonstrations than providing a marketable solution. It would also have been interesting to study the **energy consumption**, especially if the traffic lights controller can’t be plugged on a local power supply. Nevertheless, as the lights are not likely to toggle billions of times a day, we can assume that maintaining the link between them would not be very greedy regarding to the consumed energy. **Security and integrity** studies would also be a way of improving the existing system.