Lab 3

Rafik Zitouni

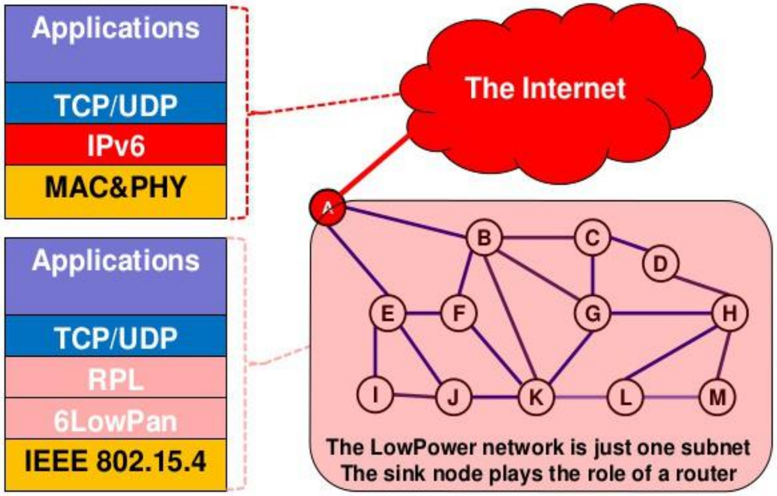
ECE Paris

rafik.zitouni@ece.fr

6LowPAN

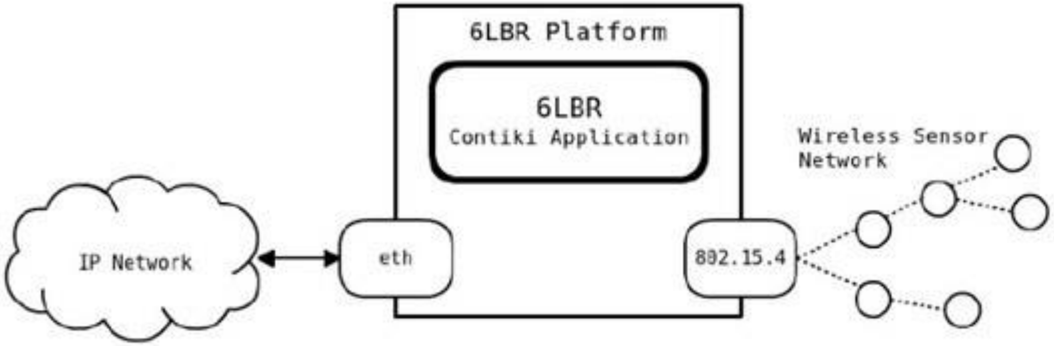
Main proprieties of 6LowPan

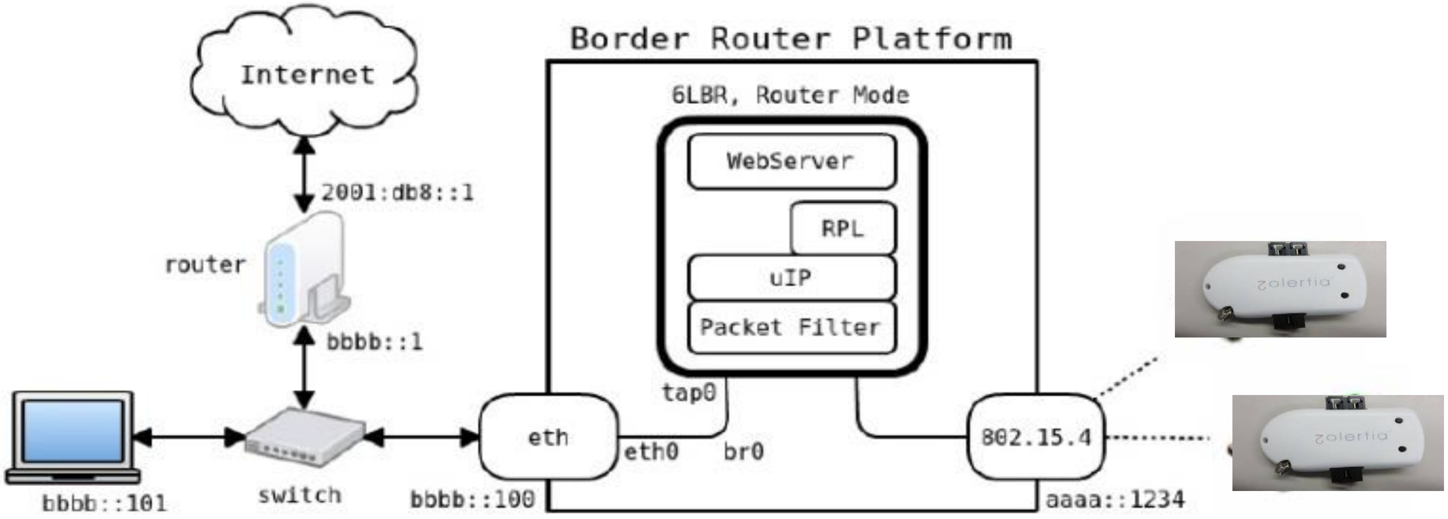
- The maximum size available for transmitting IP packets over an IEEE 802.15.4 frame is **81 bytes**
- The minimum MTU **(Maximum Transmission Unit**) that link layer should offer to IPv6 layer is **1280 bytes**
- Mesh Routing Protocol [RFC 6550]: **RPL** (Routing protocol for Low Power and Lossy Networks)
- IEEE 802.15.4 defines four types of frames:
    - Beacon frames
    - MAC command frames
    - Acknowledgement frames
    - Data frames



6LowPAN

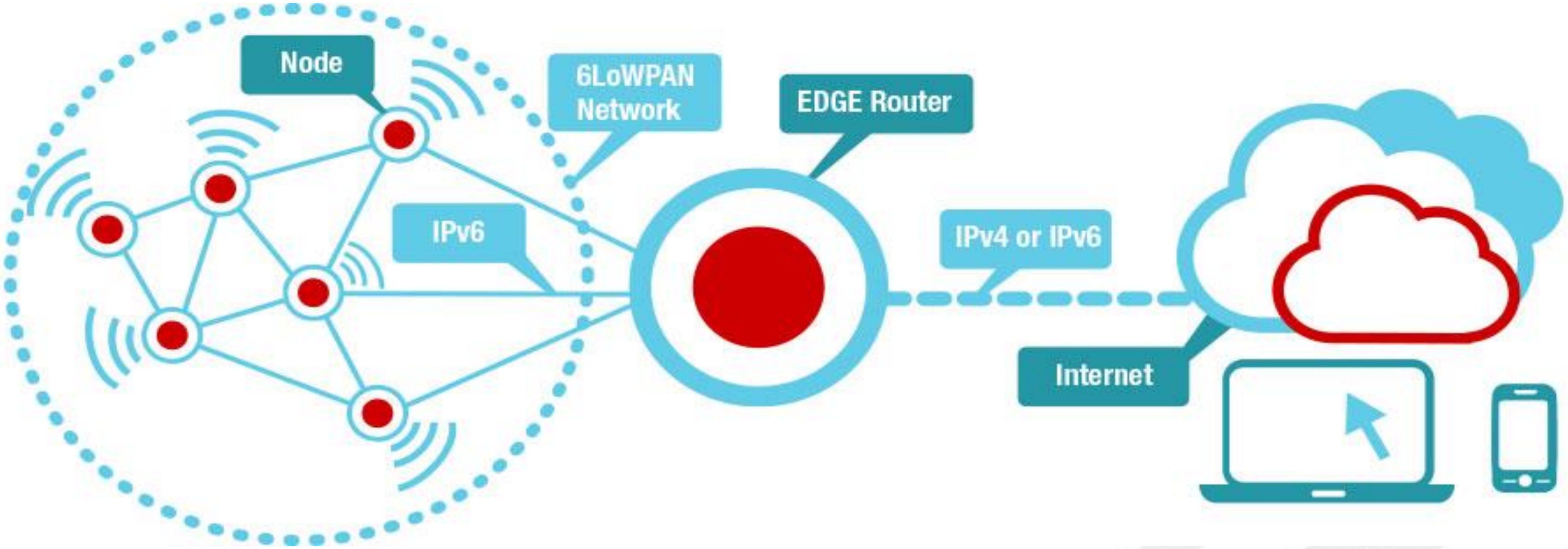
The edge router (6 LowPAN Border Router – 6LBR)





6LowPAN

Installing router border



In Contiki :

- The border router implement **serial based interface SLIP (see $tools/tunslip6)**
- Over serial port, contiki creates tunneled network interface
- The border-router applications includes a built-in web server
- Example of application is located at **examples/ipv6/rpl-border-router**

6LowPAN

UIP/IPv6 and RPL

Open source TCP/IP stack for 8 and 16 bits micro controllers.

To enable IPv6 in contiki, you should change **project-conf.h** by adding:

```
#define UIP_CONF_IPV6 1
```

Activate RPL routing protocol

```
#define UIP_CONF_IPV6_RPL 1
```

```
/*Routing*/
#define UIP_CONF_ND6_SEND_RA 0
#define UIP_CONF_IP_FORWARD 0
#define RPL_CONF_STATS 0
```

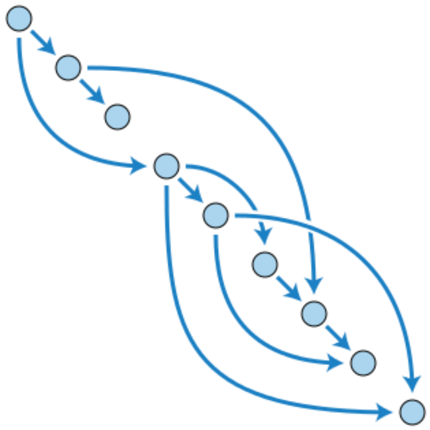
6LowPAN

Application: rpl-border-router

The border router is set as the root of the DAG after which it sets the prefix of the rest of the nodes in the network.

To understand the exemple in **examples/ipv6/rpl-border-router,** open your program **border-router.c**

Border Router BR

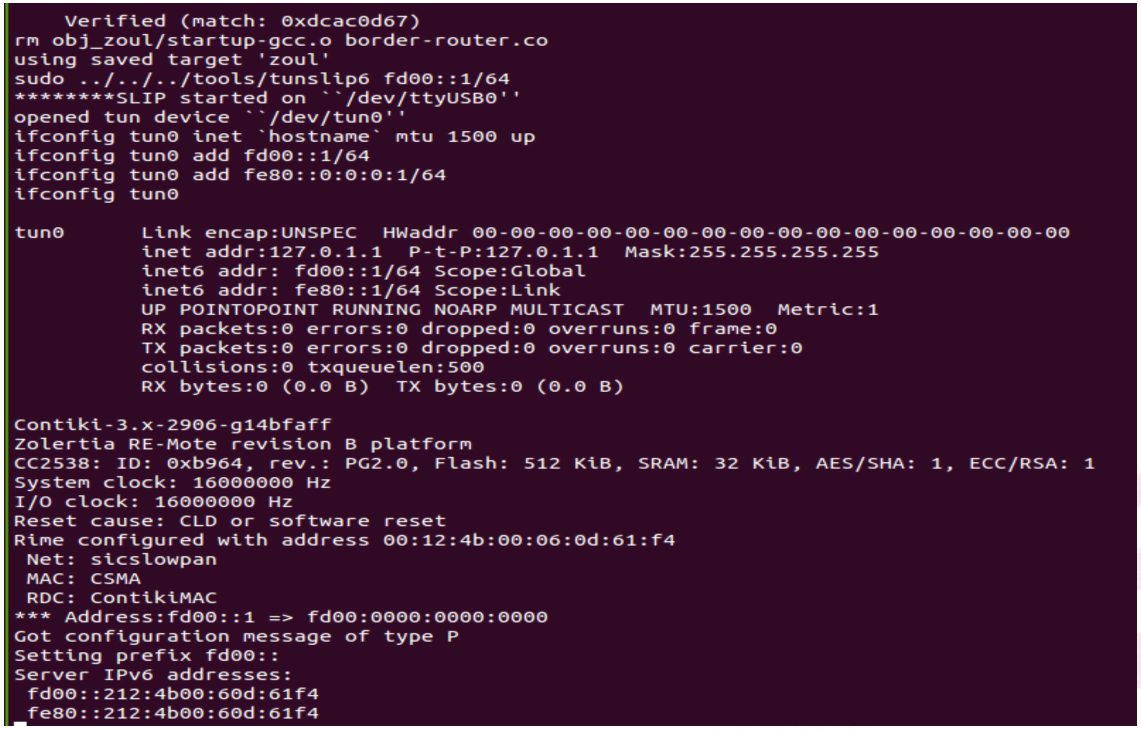


6LowPAN

The edge router (6 LowPAN Border Router – 6LBR)

See the example in **examples/ipv6/rpl-border-router**

$ sudo make border-router.upload && make connect-router



```
    Verified (match: 0xdcac0d67)
rm obj_zoul/startup-gcc.o border-router.co
using saved target 'zoul'
sudo ../../../tools/tunslip6 fd00::1/64
********SLIP started on ``/dev/ttyUSB0''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` mtu 1500 up
ifconfig tun0 add fd00::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
          inet6 addr: fd00::1/64 Scope:Global
          inet6 addr: fe80::1/64 Scope:Link
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

Contiki-3.x-2906-g14bfaff
Zolertia RE-Mote revision B platform
CC2538: ID: 0xb964, rev.: PG2.0, Flash: 512 KiB, SRAM: 32 KiB, AES/SHA: 1, ECC/RSA: 1
System clock: 16000000 Hz
I/O clock: 16000000 Hz
Reset cause: CLD or software reset
Rime configured with address 00:12:4b:00:06:0d:61:f4
 Net: sicslowpan
 MAC: CSMA
 RDC: ContikiMAC
*** Address:fd00::1 => fd00:0000:0000:0000
Got configuration message of type P
Setting prefix fd00::
Server IPv6 addresses:
 fd00::212:4b00:60d:61f4
 fe80::212:4b00:60d:61f4
```

If you want to change to change the IP address of your router :

$ make connect-router PREFIX=2001:abcd:dead:beef::1/64

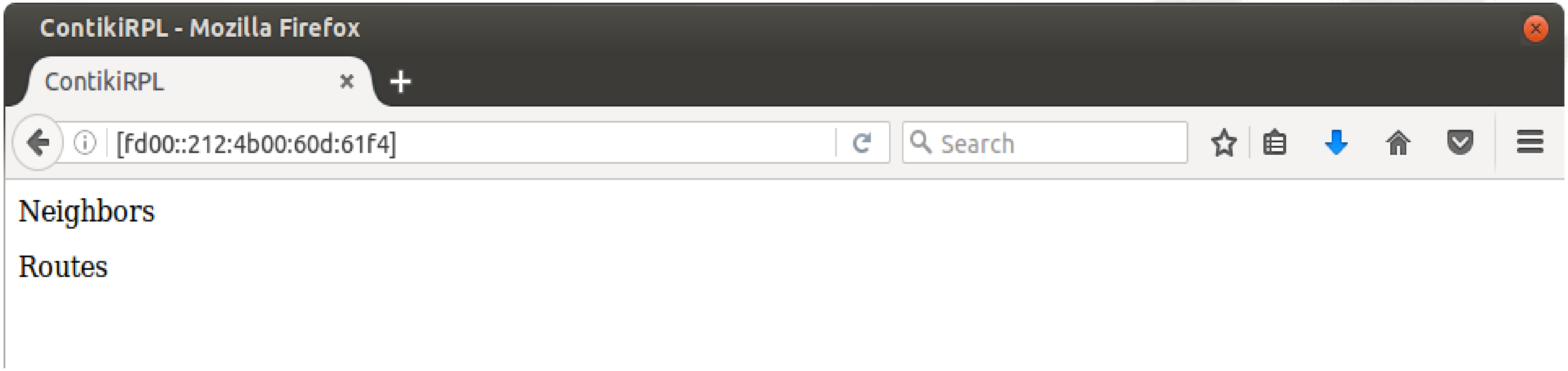
6LowPAN

The edge router (6 LowPAN Border Router – 6LBR)

The address of your 6LBR is :
fd00::212:4b00:60d:61f4

You can check if your router can be pinged by your host:
$ ping6 **fd00::212:4b00:60d:61f4**

In contiki, 6LBR plays a web server. You can check if your router has neighbors with following URL :
http://[fd00::212:4b00:60d:61f4]

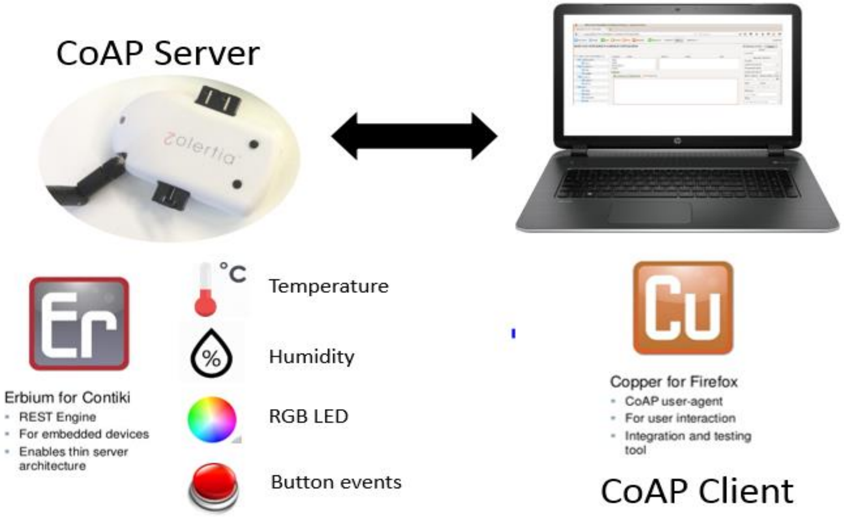


Neighbors

Routes

CoAP API – Tutorial and Exercise

CoAP application:



CoAP Server

Erbium for Contiki
- REST Engine
- For embedded devices
- Enables thin server architecture

°C Temperature

% Humidity

RGB LED

Button events

Copper for Firefox
- CoAP user-agent
- For user interaction
- Integration and testing tool

CoAP Client

➢For this example you need **2 motes**:
1) Border Router Mote
2) CoAP server Mote

➢**Copper** is an Internet of Things browser on the CoAP. It allows a designer to debug existing CoAP devices. It is available on **Mozilla Firefox** navigator.

Get the Copper (Cu) CoAP user-agent or client
https://addons.mozilla.org/en-US/firefox/addon/copper-270430/

The mandatory **CoAP** port is **5683**

CoAP API – Tutorial and Exercise

CoAP application:

Before compiling the project, check **project-conf.h** if you have the following code:

```
#undef NETSTACK_CONF_RDC
#define NETSTACK_CONF_RDC nullrdc_driver
```

It allows motes to manage efficiently dutty cycles and battery usage

CoAP Server Mote

Save your TARGET and compile the server application

```
$ cd examples/er-rest-example/
$ sudo make savetarget TARGET=zoul
$ sudo make er-example-server.upload && sudo make login
```

Exercise 1:

1) What is the result of the previous commands?
2) Do your server has an IPv6 address ? Give this address ?
3) Open the source code of the application and give the sub program which set this IP address.

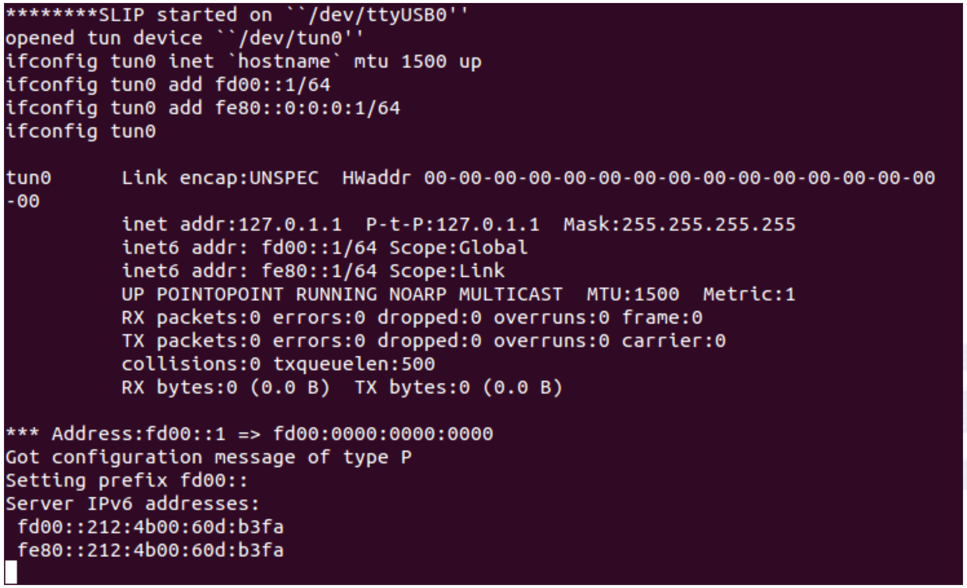
CoAP API – Tutorial and Exercise

Border Router Mote

Save your TARGET and compile the server application

```
$ cd examples/ipv6/rpl-border-router/
$ sudo make savetarget TARGET=zoul
$ sudo make border-router.upload && sudo make connect-router
```

Leave the window opened and you will get something like:



```
********SLIP started on ``/dev/ttyUSB0''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` mtu 1500 up
ifconfig tun0 add fd00::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
-00
          inet addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
          inet6 addr: fd00::1/64 Scope:Global
          inet6 addr: fe80::1/64 Scope:Link
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

*** Address:fd00::1 => fd00:0000:0000:0000
Got configuration message of type P
Setting prefix fd00::
Server IPv6 addresses:
 fd00::212:4b00:60d:b3fa
 fe80::212:4b00:60d:b3fa
```

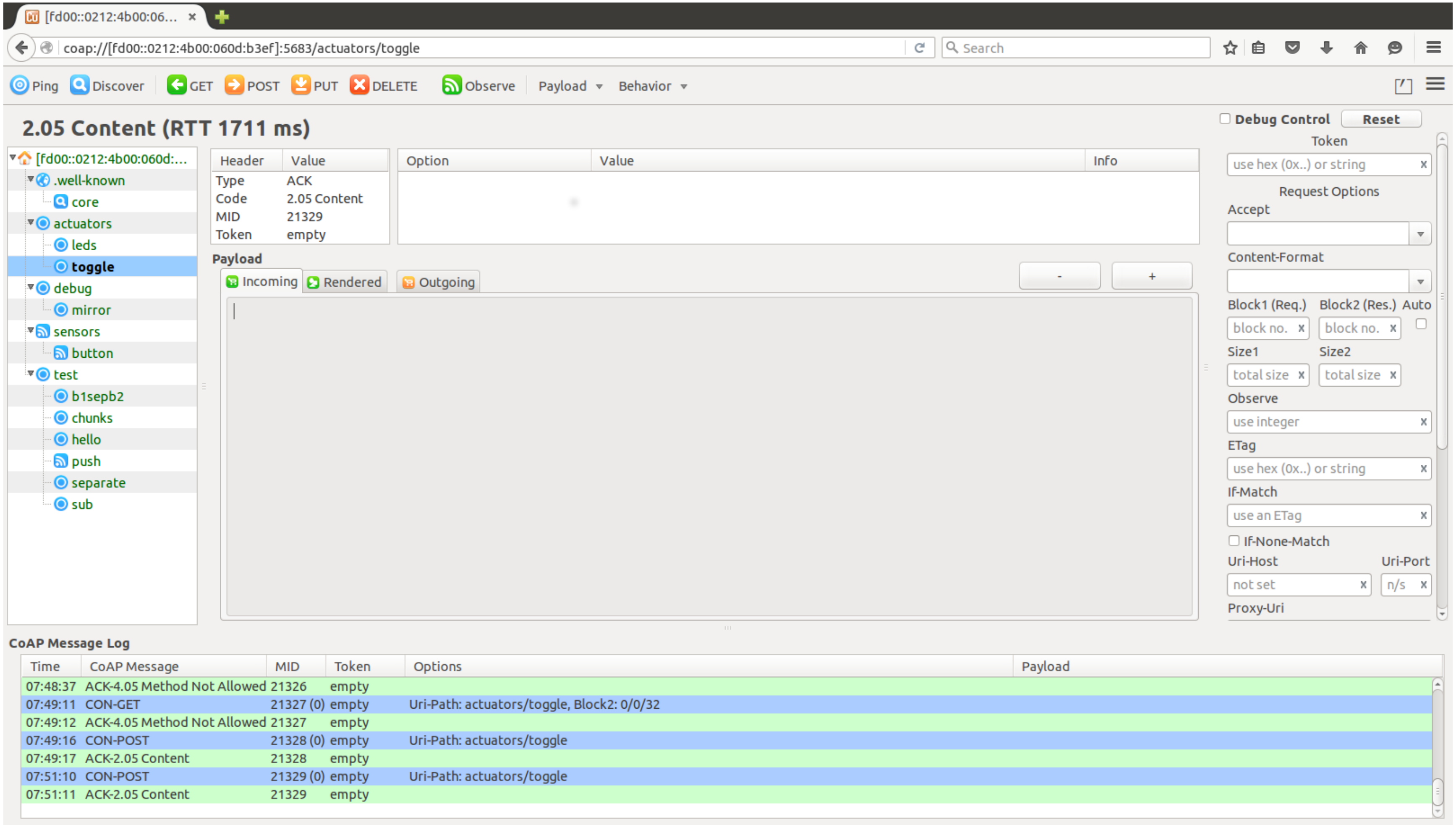
CoAP API – Tutorial and Exercise

CoAP application:

- From a terminal, you should be able to ping your border-router and you CoAP server using **ping6.**
- On your **Mozilla Firefox** Navigator you use your server address **coap://[fd00::0212:4b00:060d:b3ef]:5683**
- Discover your available resources using **Discover** button
  - Choose your resource such as **toggle** and use **POST,** you can see how the red LED of the server mote will toggle.
  - You should get hello resources by **POST** and **GET**
- If you enable the other resources in your server source code, you should have sensors. You should select button, and use observe to catch each push button.
- Try to understand how to work your CoAP server..

CoAP API – Tutorial and Exercise

CoAP application:



coap://[fd00::0212:4b00:060d:b3ef]:5683/actuators/toggle

2.05 Content (RTT 1711 ms)

Header	Value
Type	ACK
Code	2.05 Content
MID	21329
Token	empty

Option	Value	Info

CoAP Message Log

Time	CoAP Message	MID	Token	Options	Payload
07:48:37	ACK-4.05 Method Not Allowed	21326	empty		
07:49:11	CON-GET	21327 (0)	empty	Uri-Path: actuators/toggle, Block2: 0/0/32	
07:49:12	ACK-4.05 Method Not Allowed	21327	empty		
07:49:16	CON-POST	21328 (0)	empty	Uri-Path: actuators/toggle	
07:49:17	ACK-2.05 Content	21328	empty		
07:51:10	CON-POST	21329 (0)	empty	Uri-Path: actuators/toggle	
07:51:11	ACK-2.05 Content	21329	empty		

CoAP API – Tutorial and Exercise

Exercise 2:

Include in your COAP server a light sensor as resource. Your application should be able to GET the values of light sensor and print it through COAP client.

CoAP API – Tutorial and Exercise

Representational State Transfer (REST)

- It relies on a stateless, client-server, cacheable communications protocol
- **RESTful** applications use HTTP-like requests to post data, read data, and delete data
- **REST** uses HTTP for **CRUD** (**C**reate/**R**ead/**U**pdate/**D**elete) actions
- **CoAP** software layer translate to **\\:http** for simplified integration of WSN data with the web.
- **CoAP** can run on most devices that support UDP.

CoAP API – Tutorial and Exercise

Organization of the CoAP API on Contiki

apps/er-coap → The location of CoAP.

apps/rest-engine →The location of Erbium REST engine

er-coap-engine.c → Implementation of CoAP-18 version

To invoke the CoAP engine we use **REST.get_query_variable();**

Web services are viewed as ressources, and can be uniquely identified by their URLs. The basic REST design uses HTTP or CoAP protocol methods for **CRUD** operations.

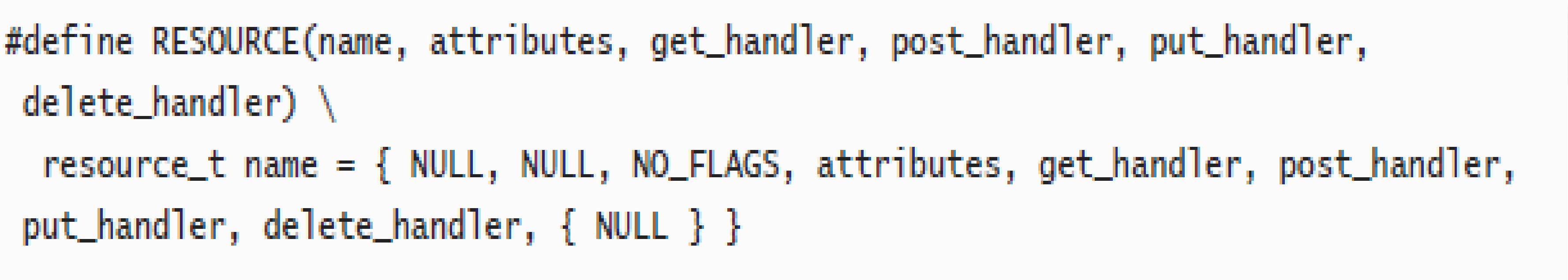
POST: Create a resource
GET: **R**etrieve a ressource
PUT: **U**pdate a ressource
DELETE: **D**elete a resource

CoAP API – Tutorial and Exercise

Server:
The server offers a number of ressources
-The REST is an intermediate layer between client requests and server
-The REST answer by the response back containing the ressource requested

apps/rest-engine contains a corresponding macros.

Normal ressource : It is a static function that is associated with a ressource handler. It is defined in **rest-engine.h**

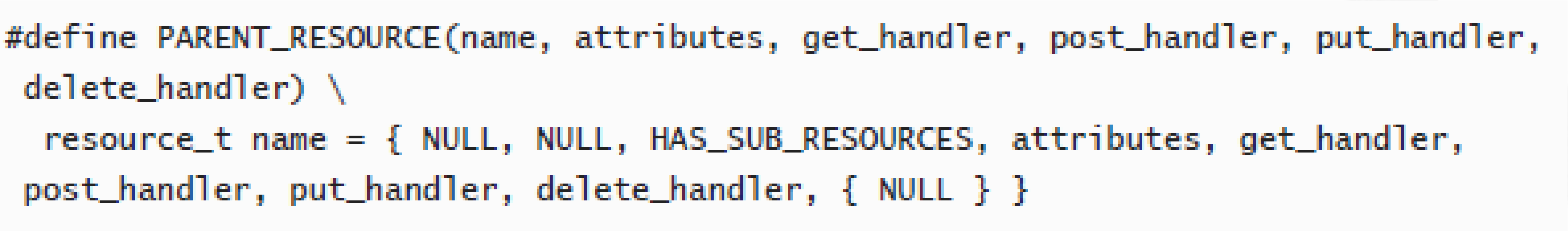


```
#define RESOURCE(name, attributes, get_handler, post_handler, put_handler,
 delete_handler) \
  resource_t name = { NULL, NULL, NO_FLAGS, attributes, get_handler, post_handler,
 put_handler, delete_handler, { NULL } }
```

CoAP API – Tutorial and Exercise

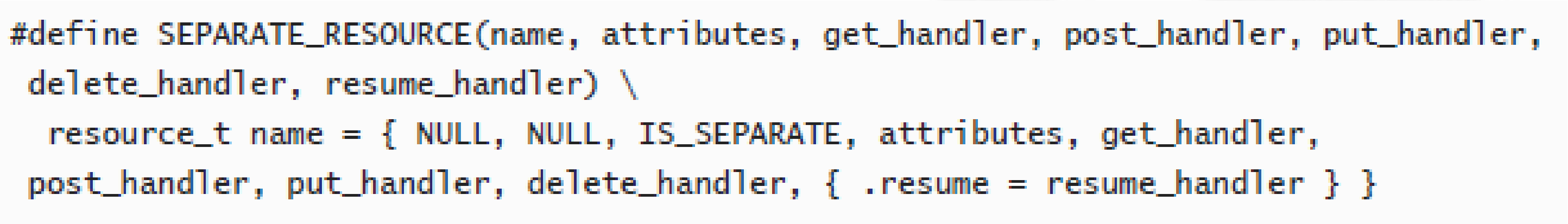
Server:

Parent ressource : It manages several sub-ressourcesis defined in URI path.



```
#define PARENT_RESOURCE(name, attributes, get_handler, post_handler, put_handler,
 delete_handler) \
  resource_t name = { NULL, NULL, HAS_SUB_RESOURCES, attributes, get_handler,
 post_handler, put_handler, delete_handler, { NULL } }
```

If server is not able to respond to a CON request, it simply responds with an empty ACK and client stop re-transmitting the request. After a period of time when the server is ready with a response, it send CON message.

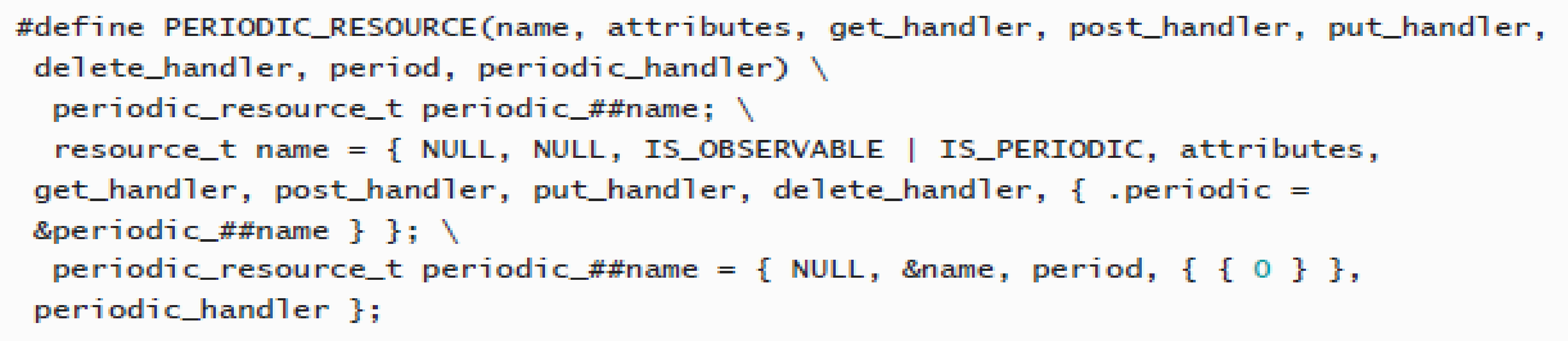


```
#define SEPARATE_RESOURCE(name, attributes, get_handler, post_handler, put_handler,
 delete_handler, resume_handler) \
  resource_t name = { NULL, NULL, IS_SEPARATE, attributes, get_handler,
 post_handler, put_handler, delete_handler, { .resume = resume_handler } }
```

CoAP API – Tutorial and Exercise

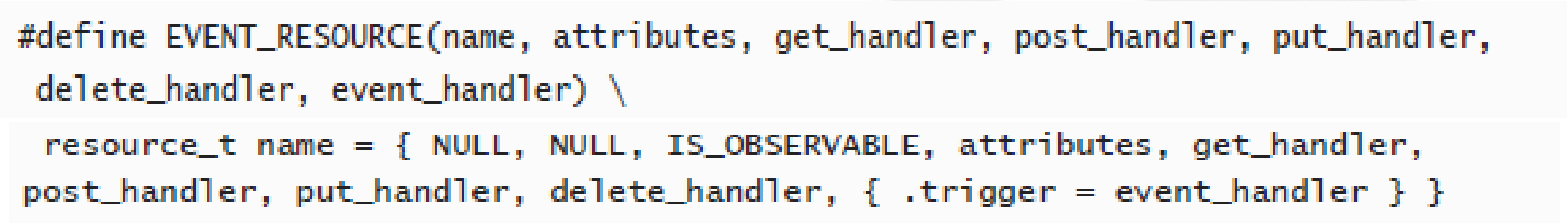
Server:

Periodic ressource : It allows server to poll a sensor and publish a changed values to subscribed clients.



```
#define PERIODIC_RESOURCE(name, attributes, get_handler, post_handler, put_handler,
 delete_handler, period, periodic_handler) \
  periodic_resource_t periodic_##name; \
  resource_t name = { NULL, NULL, IS_OBSERVABLE | IS_PERIODIC, attributes,
 get_handler, post_handler, put_handler, delete_handler, { .periodic =
 &periodic_##name } }; \
  periodic_resource_t periodic_##name = { NULL, &name, period, { { 0 } },
 periodic_handler };
```

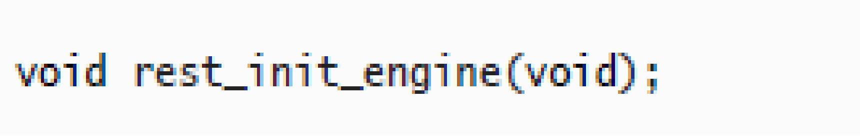
Event ressource : It is simiar to a periodic ressource, but the second handler is called by a non periodic event such as press button.



```
#define EVENT_RESOURCE(name, attributes, get_handler, post_handler, put_handler,
 delete_handler, event_handler) \
  resource_t name = { NULL, NULL, IS_OBSERVABLE, attributes, get_handler,
 post_handler, put_handler, delete_handler, { .trigger = event_handler } }
```

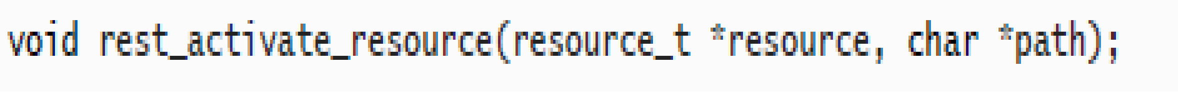
CoAP API – Tutorial and Exercise

How to initiaise the REST framework and start the HTTP and CoAP process?



```
void rest_init_engine(void);
```

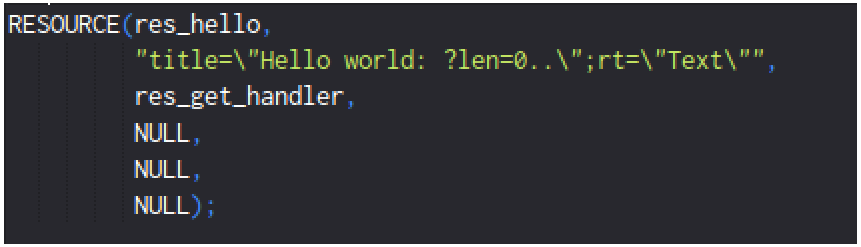
For each accessible declared resource, we need to call:



```
void rest_activate_resource(resource_t *resource, char *path);
```

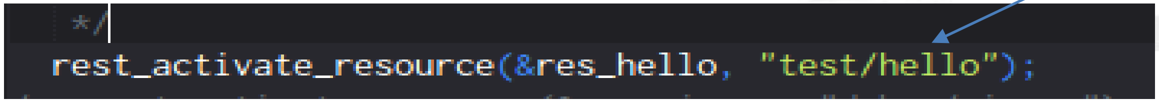
Example of CoAP application:

We can see the example in **/examples/er-rest-example/ressources/res-hello.c**



```
RESOURCE(res_hello,
         "title=\"Hello world: ?len=0..\";rt=\"Text\"",
         res_get_handler,
         NULL,
         NULL,
         NULL);
```

In server side, we enable the resource by



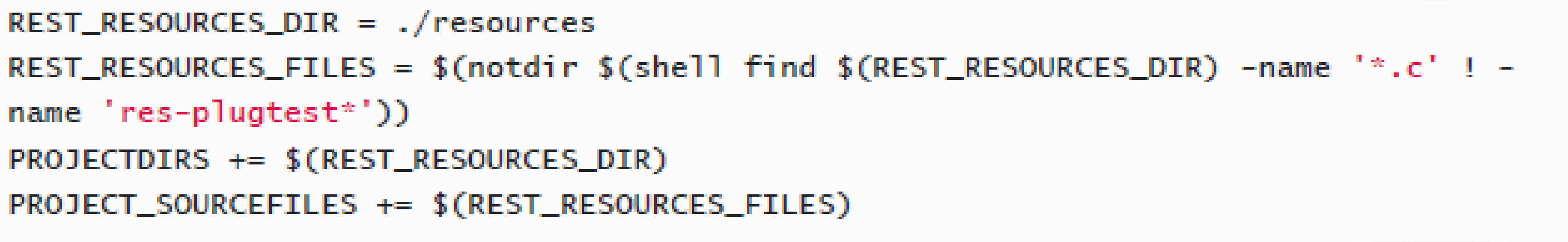
```
  */
rest_activate_resource(&res_hello, "test/hello");
```

The URI path of the resource is **test/hello**

CoAP API – Tutorial and Exercise

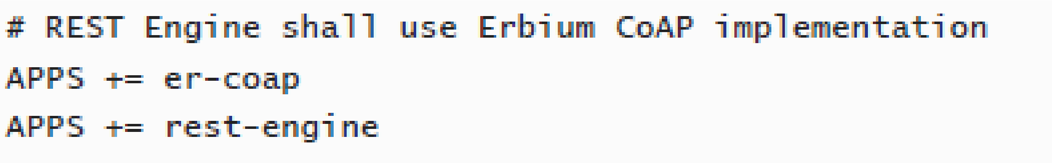
How it works the application example of CoAP

Makefile : → It Includes resources folder and files as a project directory for compilation



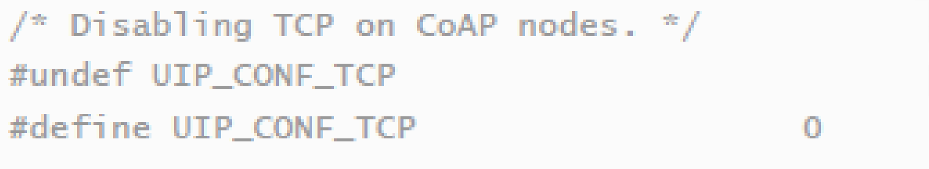
```
REST_RESOURCES_DIR = ./resources
REST_RESOURCES_FILES = $(notdir $(shell find $(REST_RESOURCES_DIR) -name '*.c' ! -
name 'res-plugtest*'))
PROJECTDIRS += $(REST_RESOURCES_DIR)
PROJECT_SOURCEFILES += $(REST_RESOURCES_FILES)
```

Makefile : → It includes the **er-coap** and **rest-engine** application



```
# REST Engine shall use Erbium CoAP implementation
APPS += er-coap
APPS += rest-engine
```

project-conf.h: → Disable TCP

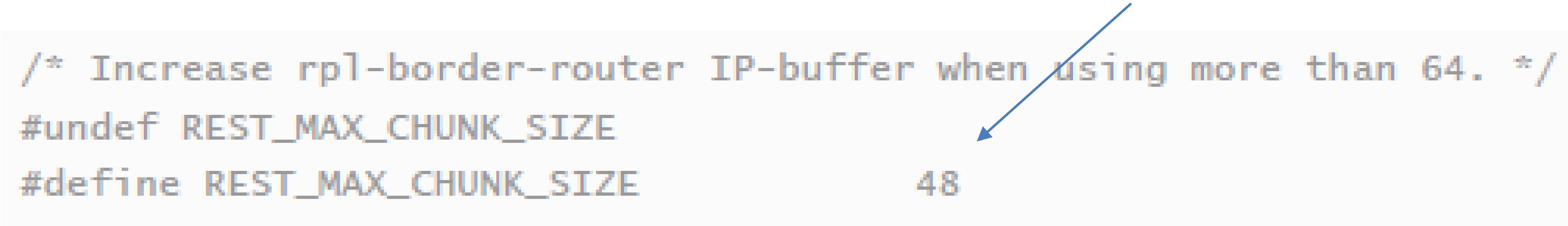


```
/* Disabling TCP on CoAP nodes. */
#undef UIP_CONF_TCP
#define UIP_CONF_TCP                    0
```

CoAP API – Tutorial and Exercise

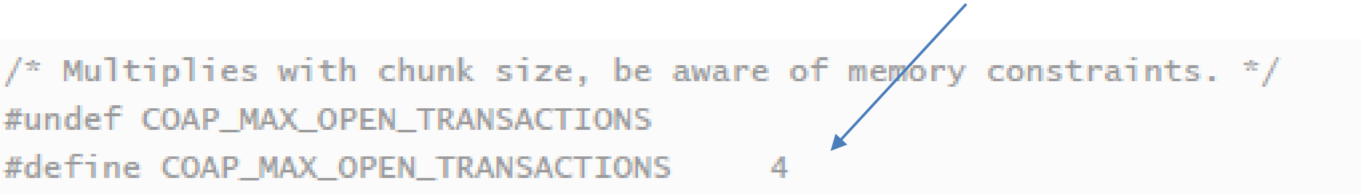
How it works the application example of CoAP

The maximum buffer size



```
/* Increase rpl-border-router IP-buffer when using more than 64. */
#undef REST_MAX_CHUNK_SIZE
#define REST_MAX_CHUNK_SIZE            48
```

The maximum number of transactions that the node can handle



```
/* Multiplies with chunk size, be aware of memory constraints. */
#undef COAP_MAX_OPEN_TRANSACTIONS
#define COAP_MAX_OPEN_TRANSACTIONS     4
```

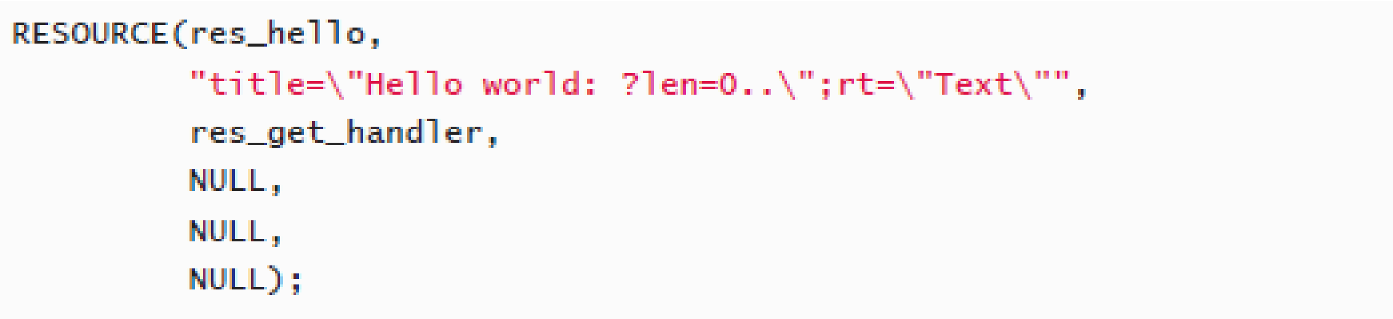
/resources → The folder contains the resources

er-example-server.c → includes these ressources using **extern resource_t res_light;**

CoAP API – Tutorial and Exercise

How it works the application example of CoAP

res-hello.c → It defines the Hello resource



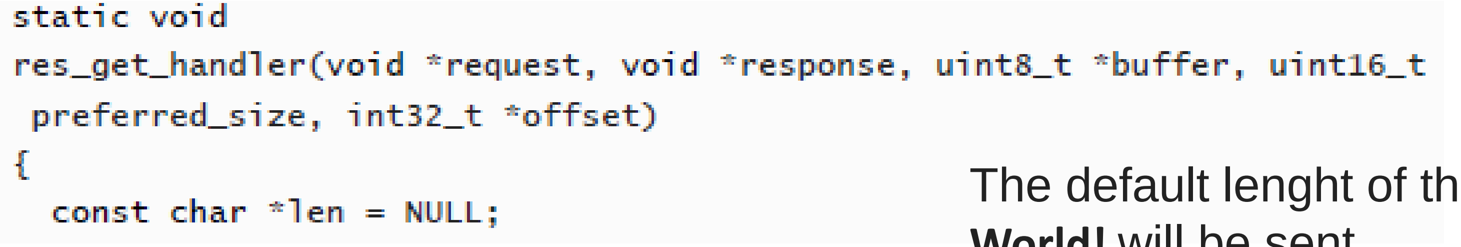
```
RESOURCE(res_hello,
         "title=\"Hello world: ?len=0..\";rt=\"Text\"",
         res_get_handler,
         NULL,
         NULL,
         NULL);
```

POST, **PUT** and **DELETE** parameters take **NULL** because they are not supported by this ressource

CoAP API – Tutorial and Exercise

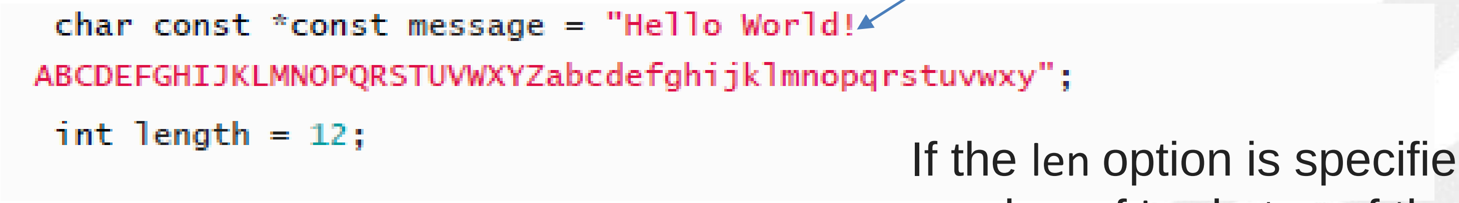
How it works the application example of CoAP

The **res_get_handler** is the event callback for **GET** requests:



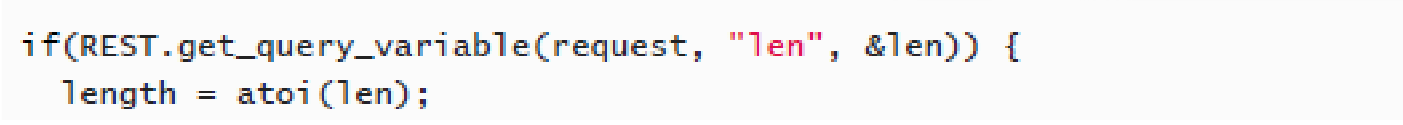
```
static void
res_get_handler(void *request, void *response, uint8_t *buffer, uint16_t
 preferred_size, int32_t *offset)
{
  const char *len = NULL;
```

The default lenght of the reply, only **Hello World!** will be sent



```
   char const *const message = "Hello World!
  ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxy";
   int length = 12;
```

If the len option is specified, then a number of len bytes of the message string will be sent

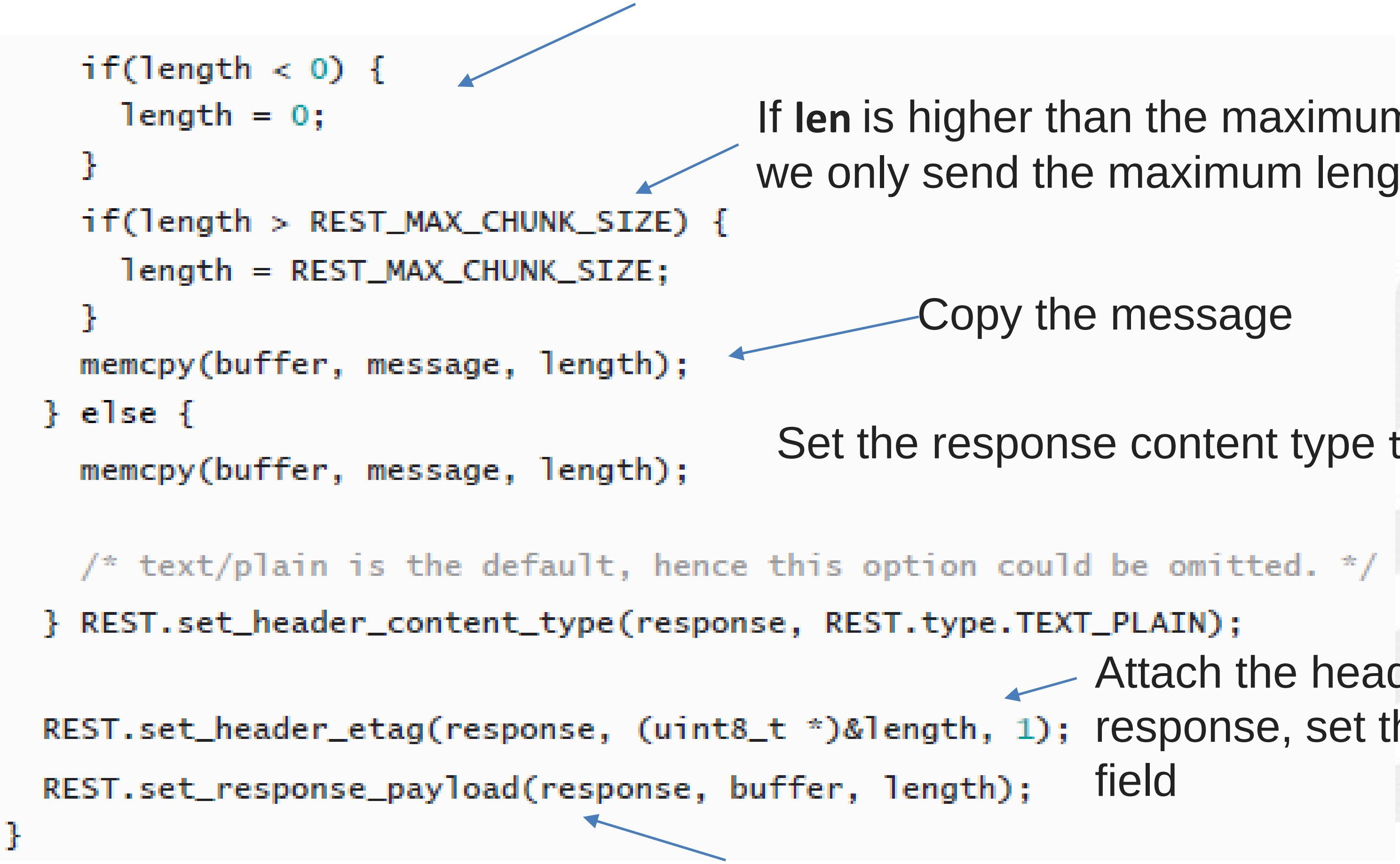


```
 if(REST.get_query_variable(request, "len", &len)) {
   length = atoi(len);
```

CoAP API – Tutorial and Exercise

How it works the application example of CoAP

If the value is a negative one, send an empty string



```
    if(length < 0) {
      length = 0;
    }
    if(length > REST_MAX_CHUNK_SIZE) {
      length = REST_MAX_CHUNK_SIZE;
    }
    memcpy(buffer, message, length);
  } else {
    memcpy(buffer, message, length);

    /* text/plain is the default, hence this option could be omitted. */
  } REST.set_header_content_type(response, REST.type.TEXT_PLAIN);

  REST.set_header_etag(response, (uint8_t *)&length, 1);
  REST.set_response_payload(response, buffer, length);
}
```

If **len** is higher than the maximum allowed, then we only send the maximum lenght value

Copy the message

Set the response content type text/plain

Attach the header to the response, set the payload lenght field

Attach the payload to the response