# Implementing the Internet of Things with Contiki

## Laboratory within the postgraduate course “Topics in Distributed Computing” (MO809) delivered by the University of Campinas

*Lecture 5*

# Outline of the course

- WSANs, the Contiki OS, and the Cooja simulator
- The uIPv6 stack and first hands-on using UDP
- Working with RPL (IPv6 Routing Protocol for Low-power and Lossy Networks)
- How to implement a WSAN Gateway
- **Implementing solutions based on CoAP (Constrained Application Protocol)**

# Limitations of a Simple-UDP solution (or similar)

- How to discover functionalities exposed by each mote?
- How to interact with a mote? For example, how to ask for a specific information, such as temperature?
- How to encode the information?

With a Simple-UDP solution (or similar), the implementation of these aspects is left to the programmer

- Different solutions
- No interoperability

# CoAP (Constrained Application Protocol)

- **CoAP** is an application protocol similar to HTTP.
- Specifically designed **for constrained devices** (e.g., Contiki motes).
- It works **over UDP** by default.

- CoAP is based on the **REST** paradigm, according to which servers expose <u>resources</u>.
- **Anything can be a resource** (e.g., a variable, a sensor, an actuator).
- Resources can be accessed through the following methods:
  - **GET** – to read the value of a resource.
  - **POST** and **PUT** – to create or update a resource.
  - **DELETE** – to delete a resource.

# Erbium

- Erbium is the REST engine and CoAP implementation for Contiki.
- Erbium implements both server and client functionalities.
- It is in **`apps/er-coap`**
- The Erbium REST engine is in **`apps/rest-engine`**
- To use Erbium, in your source code:

```
#include “contiki-net.h”

#include “rest-engine.h”
```

# Makefile to work with CoAP

```
CONTIKI_PROJECT = source-name

all: $(CONTIKI_PROJECT)

CFLAGS += -DUIP_CONF_IPV6=1 -DWITH_UIP6=1 -DUIP_CONF_TCP=0 -
DPROJECT_CONF_H=\"project-conf.h\"

CONTIKI = /home/user/contiki

APPS += er-coap

APPS += rest-engine

WITH_UIP6=1

UIP_CONF_IPV6=1

include $(CONTIKI)/Makefile.include
```

# Example of project-conf.h to work with CoAP

Example of how to **reduce RAM consumption** when using CoAP:

```
#ifndef PROJECT_CONF_H_
#define PROJECT_CONF_H_

#undef REST_MAX_CHUNK_SIZE //Set the max response payload before fragmentation
#define REST_MAX_CHUNK_SIZE 64

#undef COAP_MAX_OPEN_TRANSACTIONS //Set the max number of concurrent transactions that the node can
handle
#define COAP_MAX_OPEN_TRANSACTIONS 4

#undef NBR_TABLE_CONF_MAX_NEIGHBORS //Set the max number of entries in neighbors table
#define NBR_TABLE_CONF_MAX_NEIGHBORS 10

#undef UIP_CONF_MAX_ROUTES //Set the max number of routes handled by the node
#define UIP_CONF_MAX_ROUTES 10

#undef UIP_CONF_BUFFER_SIZE // Set the amount of memory reserved to the uIP packet buffer
#define UIP_CONF_BUFFER_SIZE 280

#endif
```

# Definition of a resource and its get_handler

Define a resource. At least one handler (i.e., a callback function) needs to be defined for each resource.

```
//preferred_size and offset needed if response is longer than REST_MAX_CHUNK_SIZE (not our case)
void get_handler(void* request, void* response, uint8_t *buffer, uint16_t
preferred_size, int32_t *offset)
{
      //Set the content type to plain text
      REST.set_header_content_type(response, REST.type.TEXT_PLAIN);
      //Prepare the response message and store it in buffer, (e.g., using sprintf() or memcpy())
      sprintf((char *) buffer, "Value: %d", value);
      //Set the response payload to the content of buffer, specifying the length of buffer
      REST.set_response_payload(response, buffer, strlen((char *) buffer));
}

//Title is a human-readable description; rt is the resource type (e.g., temperature)
//Specify the handlers associated to the resource
//Use NULL to specify that those handlers are not allowed (order is GET, POST, PUT, DELETE).
RESOURCE(resource_name, "title=\“Temperature in room A118\"; rt=\“temperature\"",
```

# Definition of a resource and its put_handler

```
void put_handler(void* request, void* response, uint8_t *buffer, uint16_t 
preferred_size, int32_t *offset)
{
        //code that updates the resource (it will be clearer with the introduction of body parameters)
       …
        //To communicate that PUT request terminated successfully
        REST.set_response_status(response, REST.status.OK);
        //To communicate that something went wrong
       REST.set_response_status(response, REST.status.BAD_REQUEST);
}

//Title is a human-readable description; rt is the resource type (e.g., temperature)
//Specify the handlers associated to the resource
//Use NULL to specify that those handlers are not allowed (order is GET, POST, PUT, DELETE).
RESOURCE(resource_name, "title=\"LEDs in room A118\"; rt=\"LEDs\"", NULL, NULL, 
put_handler, NULL);
```

# Handle request parameters: header fields (e.g., *Accept*)

The *Accept* header field allows the client to communicate the media type that the response should take. The server then responds accordingly.

```
void get_handler(void* request, void* response, uint8_t *buffer, uint16_t preferred_size, int32_t
*offset)
{
     unsigned int accept = -1;
     //Get the format accepted by the client
     REST.get_header_accept(request, &accept);

     if(accept == -1 || accept == REST.type.TEXT_PLAIN){
       REST.set_header_content_type(response, REST.type.TEXT_PLAIN);
       …
     }else if(accept == REST.type.APPLICATION_JSON){
       REST.set_header_content_type(response, REST.type.APPLICATION_JSON);
       …
     }else{
       REST.set_response_status(response, REST.status.NOT_ACCEPTABLE);
       …
     }
}
```

# Handle request parameters: Query and body parameters

- Both respect the following key-value syntax, e.g.,: *color=red&mode=on*

- Query parameters are those appended to the URL (e.g., URL?color=red).

  To get a query parameter in the code of a handler:

  ```
  int len;
  const char *color = NULL;
  len = REST.get_query_variable(request, "color", &color)
  ```

- Body parameters are those passed in the body of a request. To get a body parameter in the code of a handler:

  ```
  int len;
  const char *mode = NULL;
  len = REST.get_post_variable(request, "mode", &mode)
  ```

# Example of PROCESS_THREAD of a CoAP server

```
PROCESS_THREAD(server, ev, data)
{
      PROCESS_BEGIN();

      rest_init_engine();
      // res_name is the variable name used in RESOURCE
      // ”test/myresource” is the URL of the resource
            rest_activate_resource(&res_name, “test/myresource”);

      while(1)
      {
         PROCESS_WAIT_EVENT();
      }

      PROCESS_END();
}
```

# Which CoAP client can you use?

In the following exercises you can use whichever **CoAP client** you prefer.

Possible examples are:

- A **Java** CoAP client based on libraries such as Californium
- A **Python** CoAP client based on libraries such as CoAPthon, CoAPthon3, aiocoap
- A **Node.js** CoAP client based on modules such as node-coap
- **Command-line interface** tools such as coap-client (in Ubuntu) or CoAP-CLI for Node.js
- **Copper**, i.e., a browser add-on that implements a CoAP client

# How to downgrade Firefox and enable Copper

Copper is no more supported from Firefox v56. To **downgrade Firefox**, do the following:

- Uninstall the current version of Firefox: `sudo apt-get remove firefox -y`
- Download Firefox v55:
  - `cd Downloads/`
  - `wget https://ftp.mozilla.org/pub/firefox/releases/55.0/linux-i686/en-US/firefox-55.0.tar.bz2`
- Extract the archive: `tar xvjf firefox-55.0.tar.bz2`
- Install/configure, check version, and start Firefox v55:
  - `ln -s /home/user/Downloads/firefox/firefox /usr/bin/firefox`
  - `firefox -v`
  - `firefox`

To **enable Copper**, open the Firefox menu at the top-right and click on “Add-ons”. Enable Copper

# Exercise 9

**What to do:** Deploy in Cooja two motes -- a border router and a CoAP server. In the CoAP server, define a resource that accepts the GET method and returns the temperature value measured by the mote.
Allow *plain_text* and *JSON* formats or return NOT_ACCEPTABLE with any other format specified by the client.

**Solution:** coap-temperature.c

# Exercise 10

**What to do:** Deploy in Cooja two motes -- a border router and a CoAP server. Define a resource that allows to remotely change the status of LEDs depending on query and body parameters, as follows:

- Query parameter
  color=r|g|b
- Body parameter:
  mode=on|off

The server returns OK if the request is correct. If instead a parameter is missing or is not as expected, then the server returns BAD_REQUEST.

**Solution:** coap-led.c