Client / Server connection using CoAP

TP#2 using FIT/IoT-Lab

Lecture slides for COMASIC
27-10-2020

Why CoAP?

- **CoAP vs HTTP**

Feature	CoAP	HTTP
Protocol	It uses UDP.	It uses TCP.
Network layer	It uses IPv6 along with 6LoWPAN.	It uses IP layer.
Multicast support	It supports.	It does not support.
Architecture model	CoAP uses both client-Server & Publish-Subscribe models.	HTTP uses client and server architecture.
Synchronous communication	CoAP does not need this.	HTTP needs this.
Overhead	Less overhead and it is simple.	More overhead compare to CoAP and it is complex.
Application	Designed for resource constrained networking devices such as WSN/IoT/M2M.	Designed for internet devices where there is no issue of any resources.

In general,

- CoAP is suited for lightweight IoT devices
- Less overhead, but how can we really see this?

CoAP tutorial

- https://www.iot-lab.info/legacy/tutorials/contiki-coap-m3/

- **For now, only work on 3 sensors**
  - 1 Border router
  - 2 CoAP servers
  - Try to see if you can make a two-hop network

Site	Number of subnets	from	to
Grenoble	128	2001:660:5307:3100::/64	2001:660:5307:317f::/64
Lille	128	2001:660:4403:0480::/64	2001:660:4403:04ff::/64
Saclay	64	2001:660:3207:04c0::/64	2001:660:3207:04ff::/64
Strasbourg	32	2001:660:4701:f0a0::/64	2001:660:4701:f0bf::/64

CoAP tutorial

- **To know you have succeeded,**
  - On the bash command, type

A program to collect data

- **Now we know that both HTTP and CoAP servers are public**
  - We can do everything with them
  - Let’s make a program in python to get data from the CoAP server

Example in python for HTTP

```
  GNU nano 2.2.6                Fichier : test.py

###################### Simple program for receiving HTTP data from Python
#"""
import subprocess

command = "lynx -dump "
http_server = "http://[2001:660:5307:3102::a881]"

string = command + http_server

result = subprocess.check_output(string, shell=True)

print(result)
#"""
```

Example in python for CoAP

```
###################### Simple program for receiving CoAP data from Python
"""
import subprocess

command = "coap get "
coap_server = "coap://[2001:660:5307:3102::a881]"
port = ":5683"
output1 = "/sensors/accel"

string = command + coap_server + port + output1

result = subprocess.check_output(string, shell=True)

print(result)
"""
```


TELECOM
ParisTech
INSTITUT
Mines-Télécom

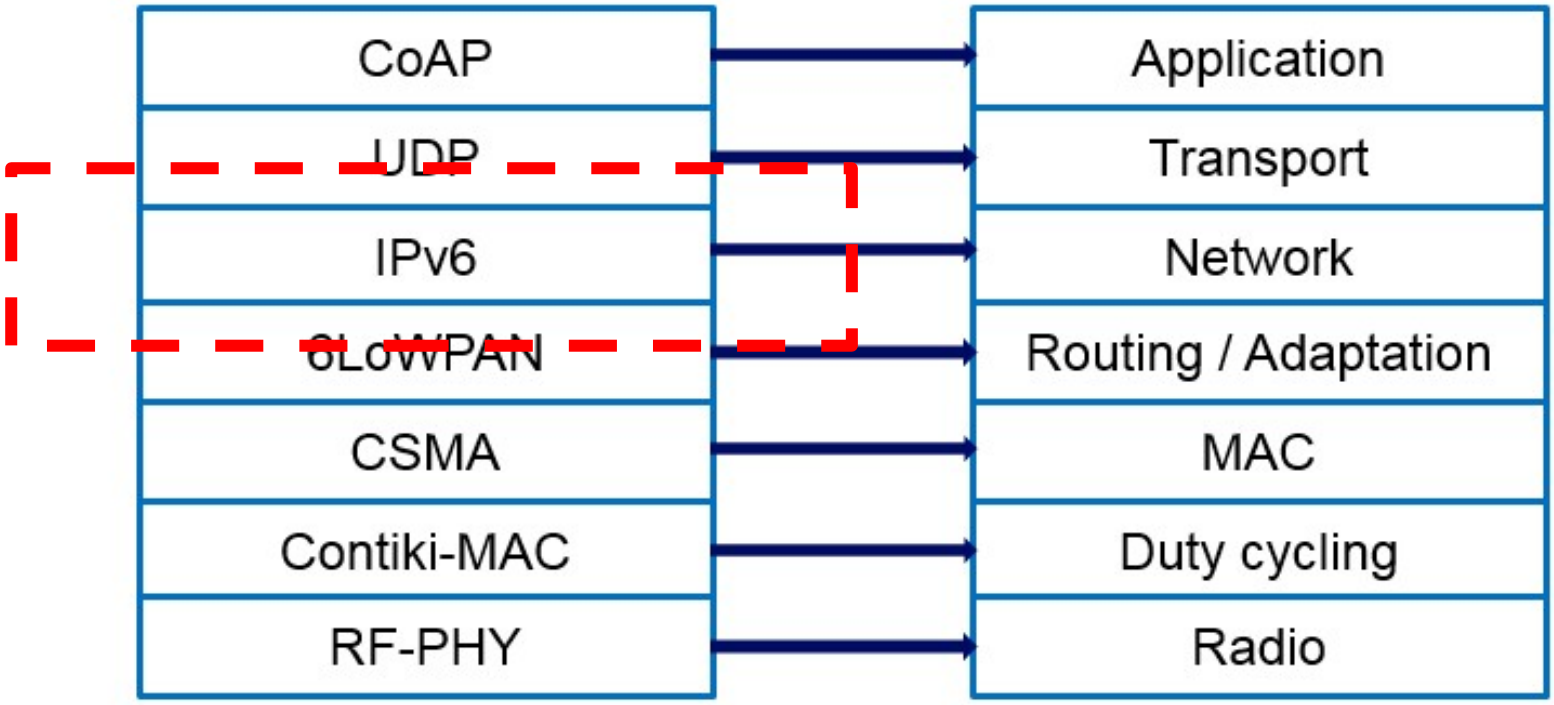
CoAP vs HTTP


université
PARIS-SACLAY

On why it is efficient to use COAP

- **We have been talking a lot about CoAP being better than HTTP**
  - But it would be better to see with our eyes
- **How to see this?**
  - Let’s create a very simple packet sniffer
  - Pity we can’t use TCPdump in FIT/IoT-Lab

Where to sniff the packets?



Very simple sniffing code

- Goto

Part receiving TCP/UDP packets

Save and compile

- **Now the IPv6 will print data packets of CoAP (UDP) and HTTP (TCP) when they are received**

- **Go to compile coap-server**
  - Cd ~/iot-lab/parts/contiki/examples/iotlab/04(tab)
  - Make TARGET=iotlab-m3
  - Node-cli .....
  - Nc m3-XXX 20000

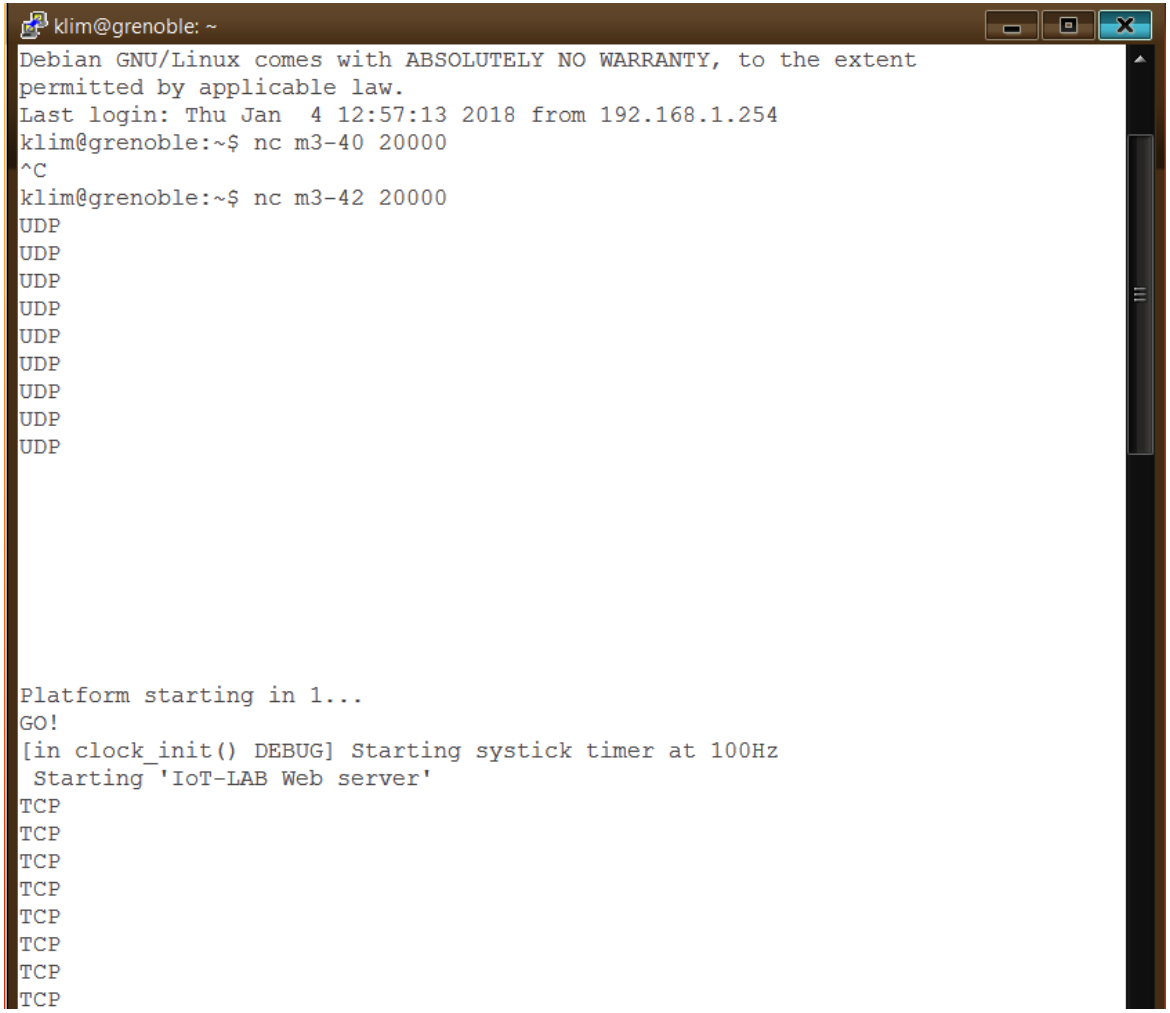
- **Send data in another shell**
  - Send a data through your python interface
  - Check output of UDP packets received

Save and compile HTTP

- **Go to compile http-server**
  - Cd ~/iot-lab/parts/contiki/examples/ipv6/http-server
  - Make TARGET=iotlab-m3
  - Node-cli .....
  - Nc m3-XXX 20000

- **Send data in another shell**
  - Send a data through your python interface
  - Check output of TCP received packets

What you should see



Balance HTTP and CoAP

- **Right now, CoAP only sends its data**
- **HTTP sends other information as well**

- **Try to balance the packet transmission**
  - For HTTP, remove all the parts of other transmission, and send only light sensor information
  - For CoAP send only light sensor information

- **What difference does this bring?**

TELECOM ParisTech

INSTITUT Mines-Télécom

Client on the sensor

université PARIS-SACLAY

Up to now...

- **We have the CoAP clients only at the user's point of control**
  - Front-end
  - Local computers

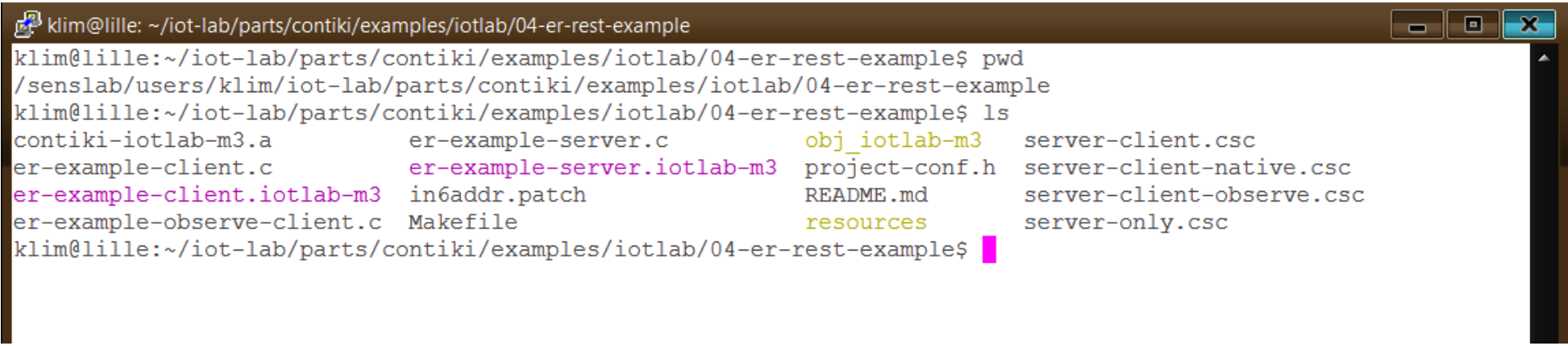
- **But what if we want the sensors to talk with each other directly?**
  - Or rather, sensor □□ actuator?
  - Actuator = CoAP client?
  - Sensor = CoAP server?

Why?

- **Do you remember the Smart Home environment?**

CoAP client for sensors

- **Goto**
  - ~/iot-lab/parts/contiki/examples/iotlab/04-er-rest-example
  - Nano er-example-client.c



CoAP client code

- https://github.com/contiki-os/contiki/blob/master/examples/er-rest-example/er-example-client.c

- **Focus on:**
  - Its basic operation (time period)
  - What is it doing? (toggle through post)
  - Sending and receiving

Let’s change the code a bit!

- **Debug code 0 1**

```
#include "dev/button-sensor.h"

#define DEBUG 1
#if DEBUG
#include <stdio.h>
```

```
/* Example URIs that can be queried. */
#define NUMBER_OF_URLS 4
/* leading and ending slashes only for demo purposes, get cropped automatically$
char *service_urls[NUMBER_OF_URLS] =
{ ".well-known/core", "/actuators/toggle", "battery/", "sensors/light" };
#if PLATFORM_HAS_BUTTON
```

Let’s change the code a bit!

- **Type of request made**

```
/* prepare request, TID is set by COAP_BLOCKING_REQUEST() */
coap_init_message(request, COAP_TYPE_CON, COAP_GET, 0);
coap_set_header_uri_path(request, service_urls[3]);

//const char msg[] = " ";
PRINTF("%.*s\n",request);
```

- COAP_POST COAP_GET
- Service_urls[1] [3]

Don’t forget the address!

- Change the address to your CoAP server’s

```
 GNU nano 2.2.6                    Fichier : er-example-client.c

#define PRINT6ADDR(addr)
#define PRINTLLADDR(addr)
#endif

/* FIXME: This server address is hard-coded for Cooja and link-local for unconnected-border-router. */
#define SERVER_NODE(ipaddr)   uip_ip6addr(ipaddr, 0xfe80, 0, 0, 0, 0x0212, 0x7402, 0x0002, 0x0202)      /* coo
/* #define SERVER_NODE(ipaddr)   uip_ip6addr(ipaddr, 0xbbbb, 0, 0, 0, 0, 0, 0, 0x1) */

#define LOCAL_PORT      UIP_HTONS(COAP_DEFAULT_PORT + 1)
#define REMOTE_PORT     UIP_HTONS(COAP_DEFAULT_PORT)

#define TOGGLE_INTERVAL 10

PROCESS(er_example_client, "Erbium Example Client");
AUTOSTART_PROCESSES(&er_example_client);

uip_ipaddr_t server_ipaddr;
static struct etimer et;
```

Debug the CoAP engine

- **Let’s try to print some information on the screens**
  - ~/iot-lab/parts/contiki/apps/er-coap
  - nano er-coap-engine.c

- **Change debug code from 0 to 1**

Recompile

- **Go back to the 04-er-rest-example**
  - Make TARGET=iotlab-m3
  - iotlab-node –update .....
  - Nc m3-XXX 20000 (do it for both client and server)

Exercise

- **Change the interval of the actuator sending requests to the CoAP server**
- **Also, make several requests for different information (light, accel) to gather more information**
- **Finally, when receiving light information, if the light value changes, printf an alarm on the screen.**

Ultimate challenge

- **If a CoAP client actuator generates an alarm,**
  - Right now we can only observe it through the nc command

- **DELIVER the alarm from the CoAP client to the user front-end**
  - If you are using python, to the python code
  - But how?

To retrack..

- **So far, it was really only about utilizing CoAP**
- **Now let's take one step more**
  - Change the codes
  - Create our own resource
  - Design an alarm system
    - On-demand
    - Proactive
- **Next week!**