Implementing the Internet of Things with Contiki

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

Lecture 2

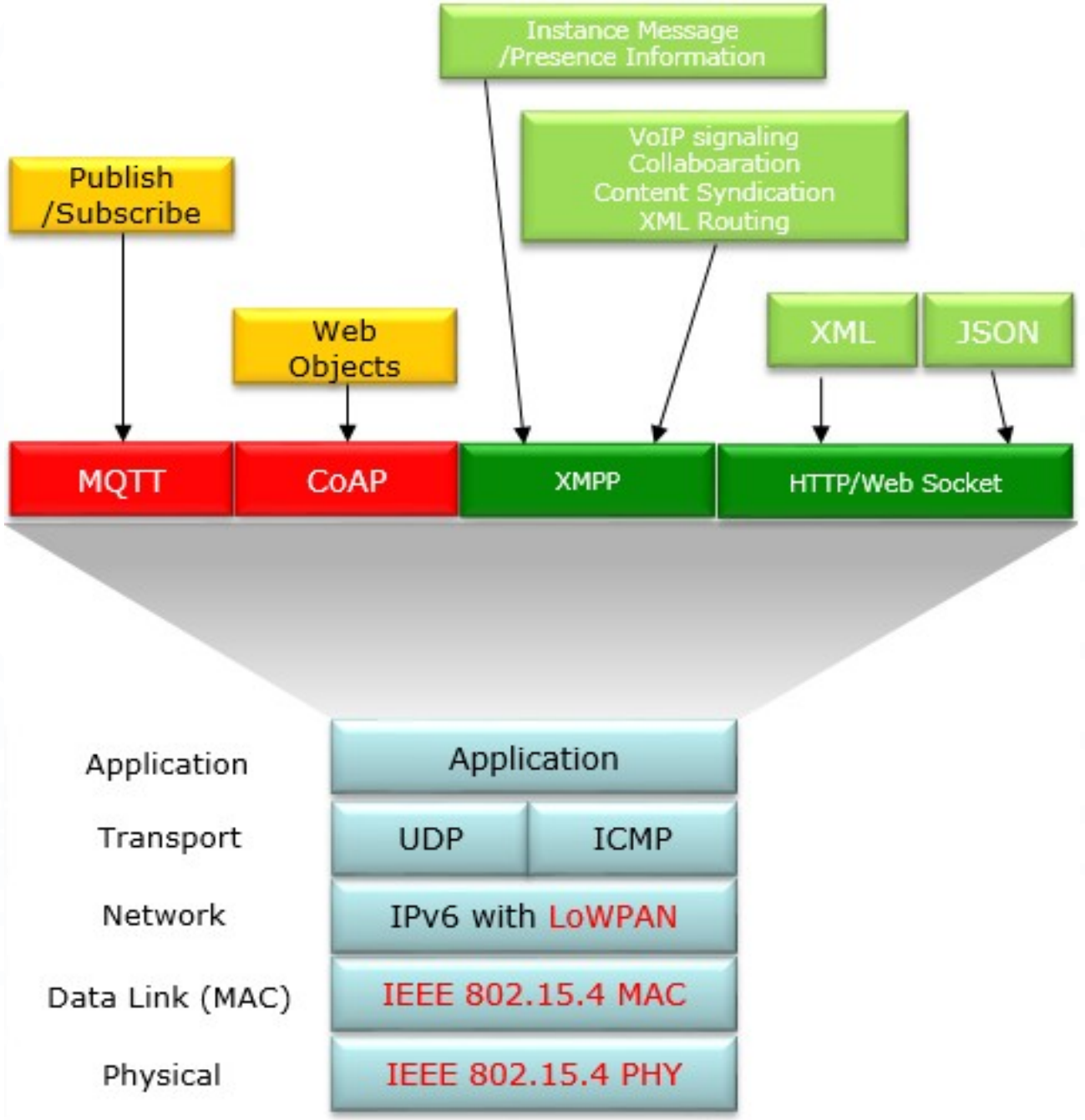
Outline of the course

- 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)

uIPv6 in Contiki

Contiki implements a TCP/IP-compliant network stack, targeted at resource-limited devices.

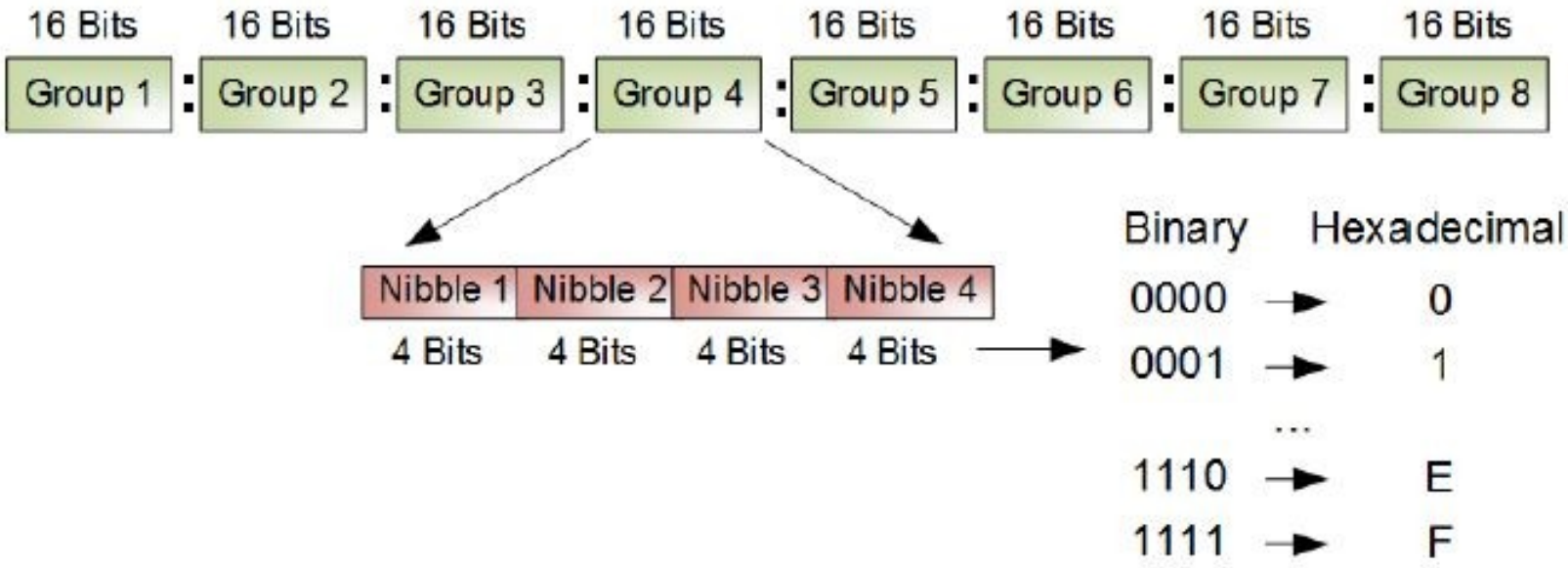
Some of the protocols are: IEEE 802.15.4, IPv6, 6LoWPAN, RPL, TCP/UDP, CoAP/HTTP



IPv6 addresses

IPv6 addresses are **128 bits** (16 bytes) long. This allows many more addresses than with IPv4!

IPv6 addresses are represented in hexadecimal format as 8 groups of 4 nibbles each.



The most common types of unicast IPv6 addresses

- Each mote has one network interface.
- Each interface can have **up to UIP_DS6_ADDR_NB unicast IPv6 addresses**.
- One of these addresses must always be the **link-local address** (i.e., FE80::/10), which can be used to communicate with neighbor hosts attached to the same link (like 169.254.0.0/16 addresses in IPv4).
- Another address can be a **global address**, which is unique in the whole Internet and used to communicate over it (like public addresses in IPv4).
- Another address can be the **ULA**, (i.e., FC00::/8 and FD00::/8), intended for local communications inside a single site (like private addresses in IPv4).

Enable IPv6 - Makefile

Modify the Makefile as follows:

```
CONTIKI_PROJECT = source-name

all: $(CONTIKI_PROJECT)

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

CONTIKI = /home/user/contiki

WITH_UIP6=1

UIP_CONF_IPV6=1

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

Enable IPv6 – Source code

Include in the program:

```
#include "net/ip/uip.h"

#include "net/ipv6/uip-ds6.h"

#include "net/ip/uip-debug.h”
```

Setup for single-hop communications

By default, the routing algorithm is enabled. In this lecture, we want to test **single-hop communications**. Routing can be disabled, in the project-conf.h, as follows:

```
#undef UIP_CONF_IPV6_RPL

#define UIP_CONF_IPV6_RPL 0
```

Neighbor Discovery has to be fully enabled as follows (ND is for IPv6 as ARP is for IPv4):

```
#undef UIP_CONF_ND6_SEND_NA

#define UIP_CONF_ND6_SEND_NA 1
```

Manipulate IPv6 (link-local) addresses

```
//declare the address
uip_ipaddr_t ipaddr;

//set the address to a unicast link-local address
uip_ip6addr(&ipaddr, 0xfe80, 0, 0, 0, 0x0212, 0x7401, 0x0001, 0x0101);

//set the address to the broadcast link-local address
uip_create_linklocal_allnodes_mcast(&ipaddr);

//print an address
uip_debug_ipaddr_print(&ipaddr);
```

Simple UDP – Create a connection

Let’s work with UDP on top of IPv6. How to create a UDP connection?

```
//include the header file
#include "simple-udp.h”

//declare the UDP connection
static struct simple_udp_connection example_connection;

//configure the connection.
//the third parameter is a pointer to the remote IPv6 address.
//NULL means that packets from any address should be accepted.
//receiver is the callback function that must be called for
//incoming packets.
simple_udp_register(&example_connection, LOCAL_UDP_PORT, NULL,
REMOTE_UDP_PORT, receiver);
```

Simple UDP – Receive

```
static void receiver
(struct simple_udp_connection *c,
 const uip_ipaddr_t *sender_addr,
 uint16_t sender_port,
 const uip_ipaddr_t *receiver_addr,
 uint16_t receiver_port,
 const uint8_t *data,
 uint16_t datalen)
{
DO_SOMETHING;
}
```

Simple UDP – Send

```
// ... Examples ...
//Send a string message with content “Test”
#include <string.h>
char* msg = “Test”;
simple_udp_sendto(&example_connection, msg, strlen(msg), &remote_addr);

//Send sensed temperature
int temp = (sht11_sensor.value(SHT11_SENSOR_TEMP)/10-396)/10;
simple_udp_sendto(&example_connection, &temp, sizeof(temp), &remote_addr);

//Send sensed temperature to the IP address specified in simple_udp_register()
int temp = (sht11_sensor.value(SHT11_SENSOR_TEMP)/10-396)/10;
simple_udp_send(&example_connection, &temp, sizeof(temp));
```

Exercise 4

What to do: Write a program that, in a loop, sets a timer to a random number of seconds that is below *X*. Every time the timer expires, the program senses the temperature and sends it in broadcast. Anytime data are received, the program prints out: the sender address and port, the receiver port, the data, and the data length.
Test it in Cooja using three motes.

Solution: broadcast-example.c

Exercise 5

What to do: Modify broadcast-example to produce two separate codes. One code only processes received packets. The other code periodically sends unicast messages to the receiver node. The message content is a counter that is incremented each time.

Tips

- Sky motes in Cooja have the following link-local unicast addresses: 0xfe80, 0, 0, 0, 0x0212, 0x7401, 0x0001, 0x0101 (where red numbers are the node ID in Cooja)
- Z1 motes in Cooja have the following link-local unicast addresses: 0xfe80, 0, 0, 0, 0xc30c, 0, 0, 0x0001

Solution: unicast-sender.c / unicast-receiver.c