Implementing the Internet of Things with Contiki

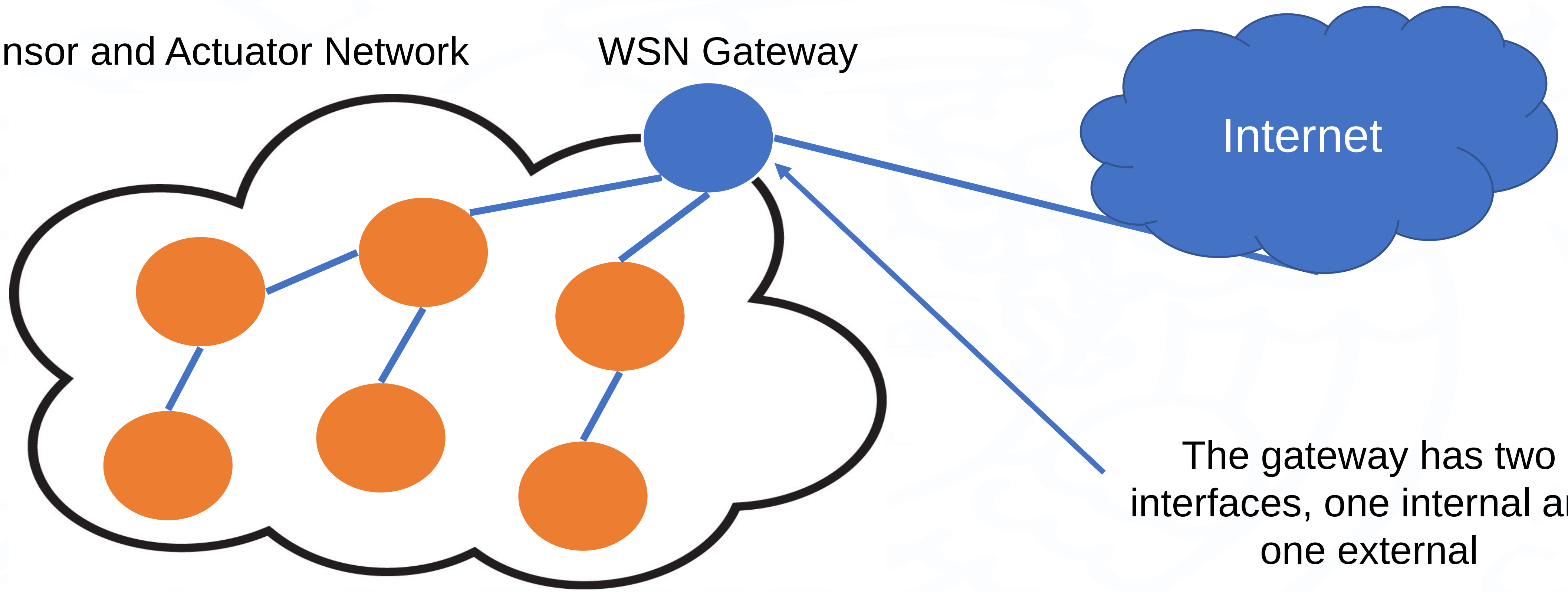
Laboratory within the postgraduate course “Topics in Distributed Computing” (MO809) delivered by the University of Campinas

Lecture 4

Outline of the course

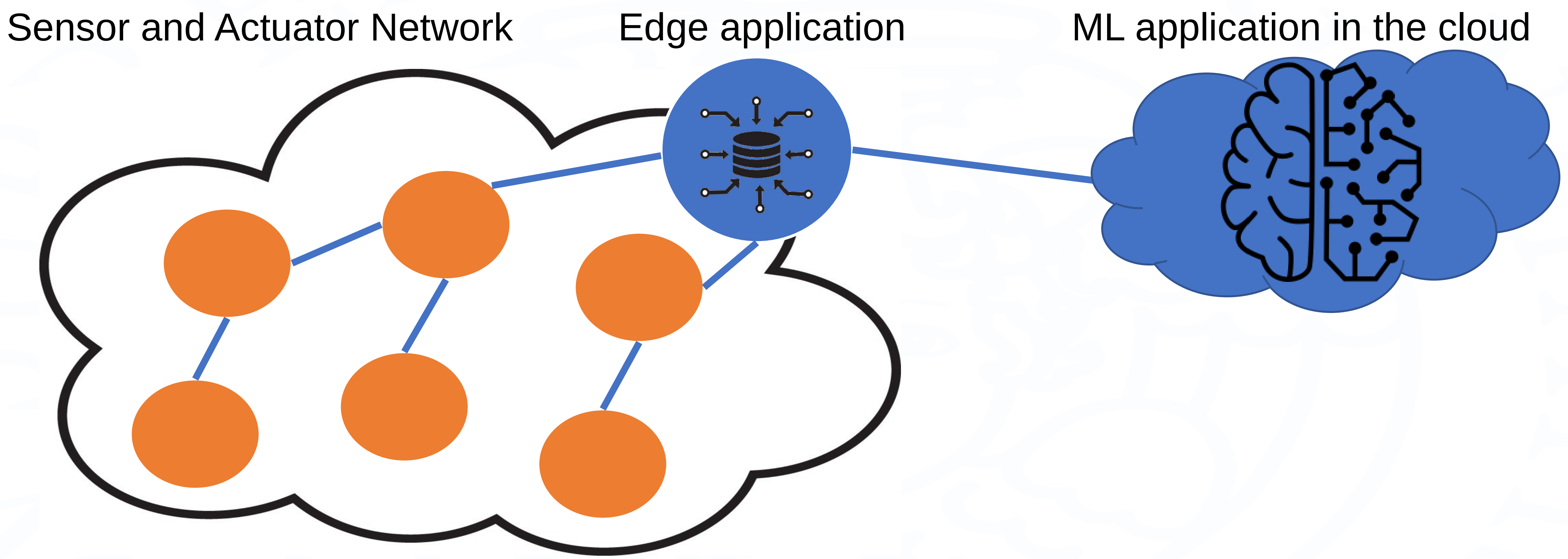
- WSANs, the Contiki OS, and the Cooja simulator
- The uIPv6 stack and first hands-on using UDP
- Working with RPL (IPv6 Routing Protocol for Low-power and Lossy Networks)
- **How to implement a WSAN Gateway**
- Implementing solutions based on CoAP (Constrained Application Protocol)

Internet of Things


Sensor and Actuator Network
WSN Gateway
Internet
The gateway has two interfaces, one internal and one external

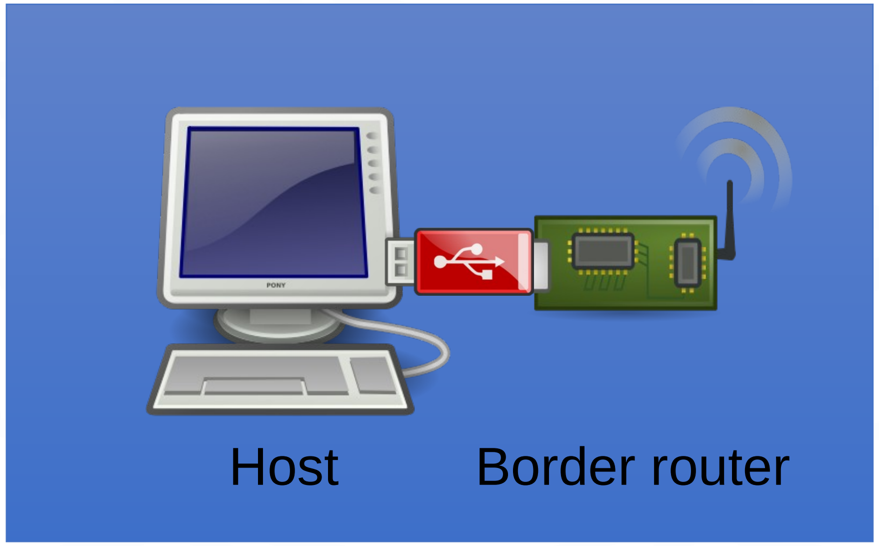
Sensors in the WSN must be able to communicate with applications on the Internet … they can do it through a **WSAN gateway**

Machine Learning (ML) as an example of a possible use case



- **Sensors** send their collected data to an edge application running on the gateway
- The **edge application** performs real-time data pre-processing and aggregation and sends part of the data to a ML application on the Internet
- The **ML application** performs long-term data analytics and makes further inference
- Eventually, **actuation** commands can be issued on the WSAN

How is the WSAN gateway composed?



Gateway

Data received by the border router (coming from the WSAN) will be forwarded to the host through the serial connection over USB. The host will then forward data to external networks (e.g., the Internet) to reach applications. In the opposite way, external applications can reach the motes.

How to implement a WSAN gateway?

In order to implement a WSAN gateway, we need tunslip6 and the code of the border router.

- **Tunslip6** is a tool that creates a connection between the mote (border router) and the host
- Tunslip6 creates a virtual interface (called ***tun0***), which is bridged to the mote
- The interface will have an IPv6 address (fd00::1, by default) and can be used by applications to send/receive data to/from the WSAN
- The border router will receive the **IPv6 prefix** (fd00, by default) from tunslip6.
- The border router also runs the **RPL root** code.
- The network prefix will be forwarded and installed in the overall WSAN in order to give motes an IPv6 address that can be used for external communications.

Code of the border router (only for Z1)

The code of the border router is

`/contiki/examples/ipv6/rpl-border-router/border-router.c`

- ➢ SkyMotes do not have enough memory to behave as border routers

This code, besides behaving as **border router**, also implements:

- The **RPL ROOT**
- A **Web Server** that provides information on the neighbours of the border router as well as on the routes to reach motes in the network.

Gateway setup in Cooja

- ➢ Deploy a mote behaving as border router

  (which must always be the first mote to be deployed)

- ➢ Set the serial socket on the border router
  - "Tools -> Serial Socket (SERVER) -> Z1 1 -> Start" (you can keep the default port)

- ➢ Deploy the other motes in the network

- ➢ Run tunslip6:
  - `cd /contiki/examples/ipv6/rpl-border-router/`
  - `make connect-router-cooja` (to leave the default prefix)

    OR

    `make connect-router-cooja PREFIX="abcd::1/64"`

Exercise 7

What to do: Deploy a network in Cooja with four motes in a line. The first mote must be the border router. The other motes must be the rpl-unicast-sender from Lecture 3. Run tunslip6 by specifying *abcd* as global prefix of the network.

Open your browser at the address
"*http://[global address of the border router]*" (keep the square brackets) and look for the global addresses of the other motes in the network

From terminal, try "*ping6 global_address*" to ping the *tun0* interface, the border router, and the other motes.

An application that interacts with motes

- Border router allows external applications to access motes by using global addresses
- Next question: how to write an application that accesses data generated by motes?
- A simple UDP socket can be used from applications to interact with motes

Exercise 8

What to do: Write a client-server application.
The client must be written in Java using this code (https://systembash.com/a-simple-java-udp-server-and-udp-client/) as reference. In a loop, the client asks to the user for the message to send to the server. If the message is “bye”, the client terminates. The client also prints out any response from the server. The client will run on your host.
Deploy a network in Cooja with two motes. The first mote must be the border router. The second mote, instead, must be an UDP server that waits for a request from the client and echoes the received data back to the client.

Solution: UDPClient.java
unicast-server.c