Implementing the Internet of Things with Contiki

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

Lecture 3

Outline of the course

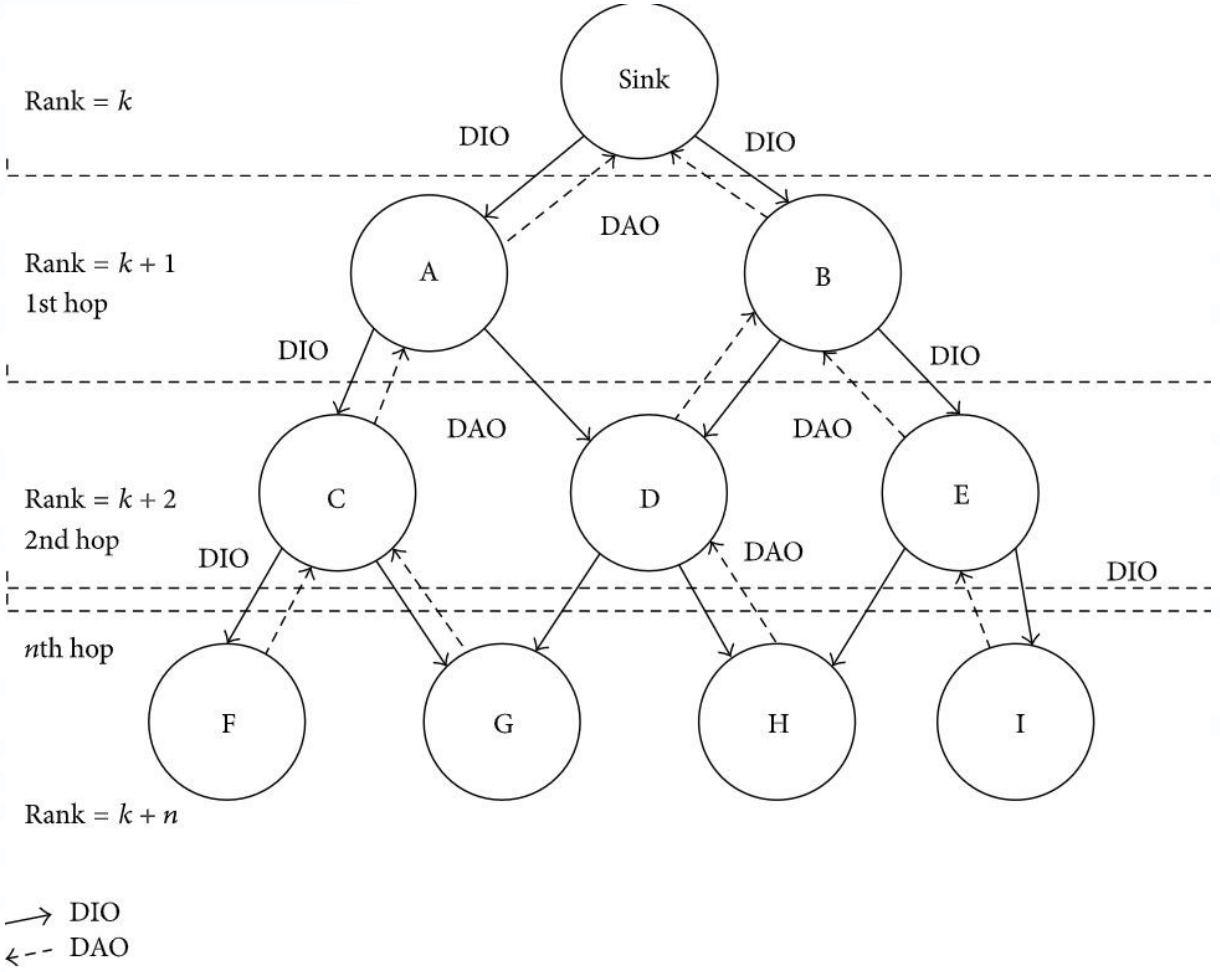
- WSANs, the Contiki OS, and the Cooja simulator
- The uIPv6 stack and first hands-on using UDP
- **Working with RPL (IPv6 Routing Protocol for Low-power and Lossy Networks)**
- How to implement a WSAN Gateway
- Implementing solutions based on CoAP (Constrained Application Protocol)

RPL (Ipv6 Routing Protocol for Low-power and lossy networks)

- So far, we have seen only single-hop communications. What if we want **multi-hop communications**? A routing protocol is required.
- **RPL** is the default routing protocol in Contiki.
- RPL builds Directed Acyclic Graphs (**DAG**s).
- **Nodes** in the DAG are motes; **arches** are routing paths among them.
- Every network must have a **DAG ROOT**, which originates the RPL DAG.
- Nodes have a **rank** – the higher the rank, the farther is the node from the root.
- Each node selects a set of neighbors as **parents**, i.e., candidates for upstream data delivery.
- A **preferred parent** is selected for the actual data forwarding.

RPL messages

- DODAG Information Object (**DIO**) messages - broadcast by every node to build and maintain upward routes of the DODAG.
- DODAG Information Solicitation (**DIS**) messages – a node can request DODAG information by sending DIS messages, soliciting DIO messages from its neighbors.
- DODAG Advertisement Object (**DAO**) messages - sent from nodes to the DAG ROOT to form downward routes.



Preliminary exercise to work with RPL: Configure a global IPv6 address on a mote

```
uip_ipaddr_t ipaddr; //declare the address

//first step to set up the address
uip_ip6addr(&ipaddr, 0xabcd, 0, 0, 0, 0, 0, 0, 0);

//set the 64 least significant bits of the global address as the 64
least //significant bits of the link-local address (which are based on the
MAC //address of the device)
uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);

//assign the created IPv6 global address to the wireless interface
//0 means infinite lifetime
//ADDR_AUTOCONF is because the global address is based on the MAC address
uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
```

Get all the IPv6 addresses of the interface of a mote

```
int i;

printf("IPv6 addresses: ");

//cycles for the max number of addresses that the interface can have
for(i = 0; i < UIP_DS6_ADDR_NB; i++){

  if(uip_ds6_if.addr_list[i].isused){

    //print the address
    uip_debug_ipaddr_print(&uip_ds6_if.addr_list[i].ipaddr);

    printf("\n");

  }

}
```

Exercise 5

What to do: Write a program that sets a global IPv6 address and retrieves all the IPv6 addresses assigned to the interface of the node.

Solution: get-address.c

Enable RPL in Contiki

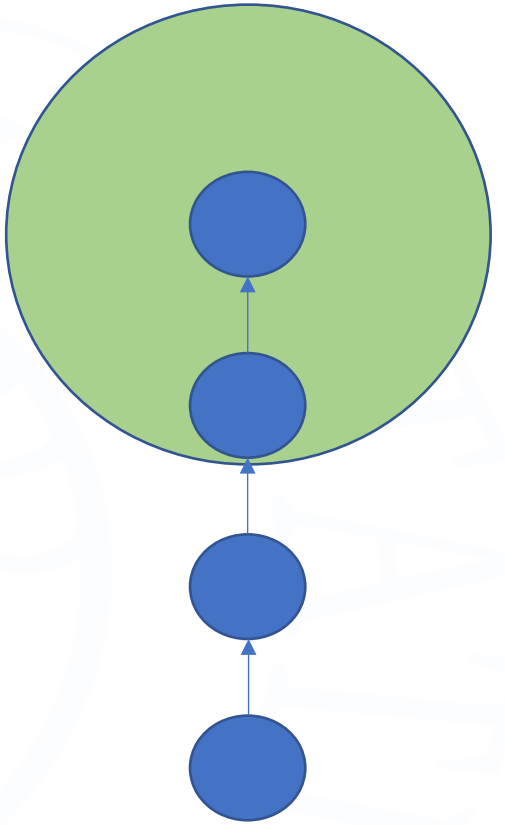
- **RPL is enabled by default** on every Contiki node.
- No extra configuration needs to be done for **NON-ROOT** nodes.
- An extra configuration must be done on the **ROOT** node to enable the RPL root capabilities:
  - **`#include “net/rpl/rpl.h”`**.
  - Implement the network interface initialization (i.e., **`set_global_address()`** from the **`get-address.c`** exercise).
  - Locate the following code right after the network interface initialization, to set the node as root and set the network prefix for later advertisement.

```
rpl_dag_t *dag;
  //elect as root of the DAG
  dag = rpl_set_root(RPL_DEFAULT_INSTANCE,(uip_ip6addr_t *)&ipaddr);
  uip_ip6addr(&ipaddr, 0xabcd, 0, 0, 0, 0, 0, 0, 0);
  //set the global prefix of the dag to be the first 64 bits of ipaddr
  rpl_set_prefix(dag, &ipaddr, 64);
```

Display RPL output

- To obtain an insight on RPL operations, the advanced debug can be enabled inside the single files within **`core/net/rpl/.`**
- To print out messages exchanged by RPL, go to **`rpl-icmp6.c`** and change **`DEBUG_NONE`** to **`DEBUG_PRINT`**. This activates all the **PRINTF** in that source file.

Exercise 6



What to do: Modify the unicast-example project from the previous lecture as follows.

- Extend the unicast-receiver to behave as **DAG ROOT.**
- Deploy several (e.g., 3) **unicast-senders in a line**, so that they are neighbors only with the adjacent ones. Unicast-senders send their messages to the DAG ROOT, using its global address.
- The motes **print all their addresses** in a loop.
- Enable **DEBUG_PRINT** to see RPL outputs.

➢ Does the DAG ROOT receive packets from any mote?

➢ Can you manage to deploy this exercise on **both Z1 and Sky motes?**

Solution: - rpl-root-receiver.c
- rpl-unicast-sender.c