RPL

Carlo Vallati
PostDoc Researcher@ University of Pisa
c.vallati@iet.unipi.it

RPL

- RPL stands for Routing Protocol for Low-power and Lossy Networks

- Layer 3 routing protocol.

- The root creates the RPL DAG.

Enable RPL

- In the program:
  - **#include "net/rpl/rpl.h"**
- In the Makefile
  - **CFLAGS+= -DUIP_CONF_IPV6_RPL**
- For debugging:
  - **CFLAGS+= -DRPL_CONF_STATS=1**

Initialize RPL

- Initialize a node as RPL ROOT node:

```
rpl_dag_t *dag;
 dag = rpl_set_root(RPL_DEFAULT_INSTANCE,
                          (uip_ip6addr_t *)&ipaddr);
 uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
 rpl_set_prefix(dag, &ipaddr, 64);
```

Right after network interface initialization

RPL global repair

- Root node can trigger RPL global repair:

```
rpl_repair_root(RPL_DEFAULT_INSTANCE);
```

Recall Trickle

- Each node maintains a counter c and a timer t in range [I/2, I] (at start, I = Imin)
- When a node receives metadata that is "consistent", it increments c
- At time t, the node broadcasts a summary of its metadata iff c < K (redundancy threshold)
- When the interval I expires
  - I is doubled (up to Imax)
  - c is reset to zero
  - t is reset to a new value in the range [I/2, I]
- When a node receives metadata that is "inconsistent" (and I > Imin), I is reset to Imin and c and t are reset

Contiki Trickle

- All the RPL configuration parameters are in:
  - core/net/rpl/rpl-conf.h
- K = RPL_CONF_DIO_REDUNDANCY
- Imin = RPL_CONF_DIO_INTERVAL_MIN
  - $2^{\text{RPL\_CONF\_DIO\_INTERVAL\_MIN}}$
- Imax = RPL_CONF_DIO_INTERVAL_DOUBLINGS
  - $2^{(\text{RPL\_CONF\_DIO\_INTERVAL\_MIN} + \text{RPL\_CONF\_DIO\_INTERVAL\_DOUBLINGS})}$

Change trickle parameter

- Modify the project-conf.h file to set trickle parameters (see core/net/rpl/rpl-conf.h):

```
#undef RPL_CONF_DIO_REDUNDANCY
#define RPL_CONF_DIO_REDUNDANCY 1

#undef RPL_CONF_DIO_INTERVAL_MIN
#define RPL_CONF_DIO_INTERVAL_MIN 3

#undef RPL_CONF_DIO_INTERVAL_DOUBLINGS
#define RPL_CONF_DIO_INTERVAL_DOUBLINGS 5
```

Display RPL output

- To perform some analysis you must modify source file inside core/net/rpl/
- Set DEBUG DEBUG_PRINT in rpl-timers.c to investigate Trickle.
- Can also set custom printf in order to detech custom events.

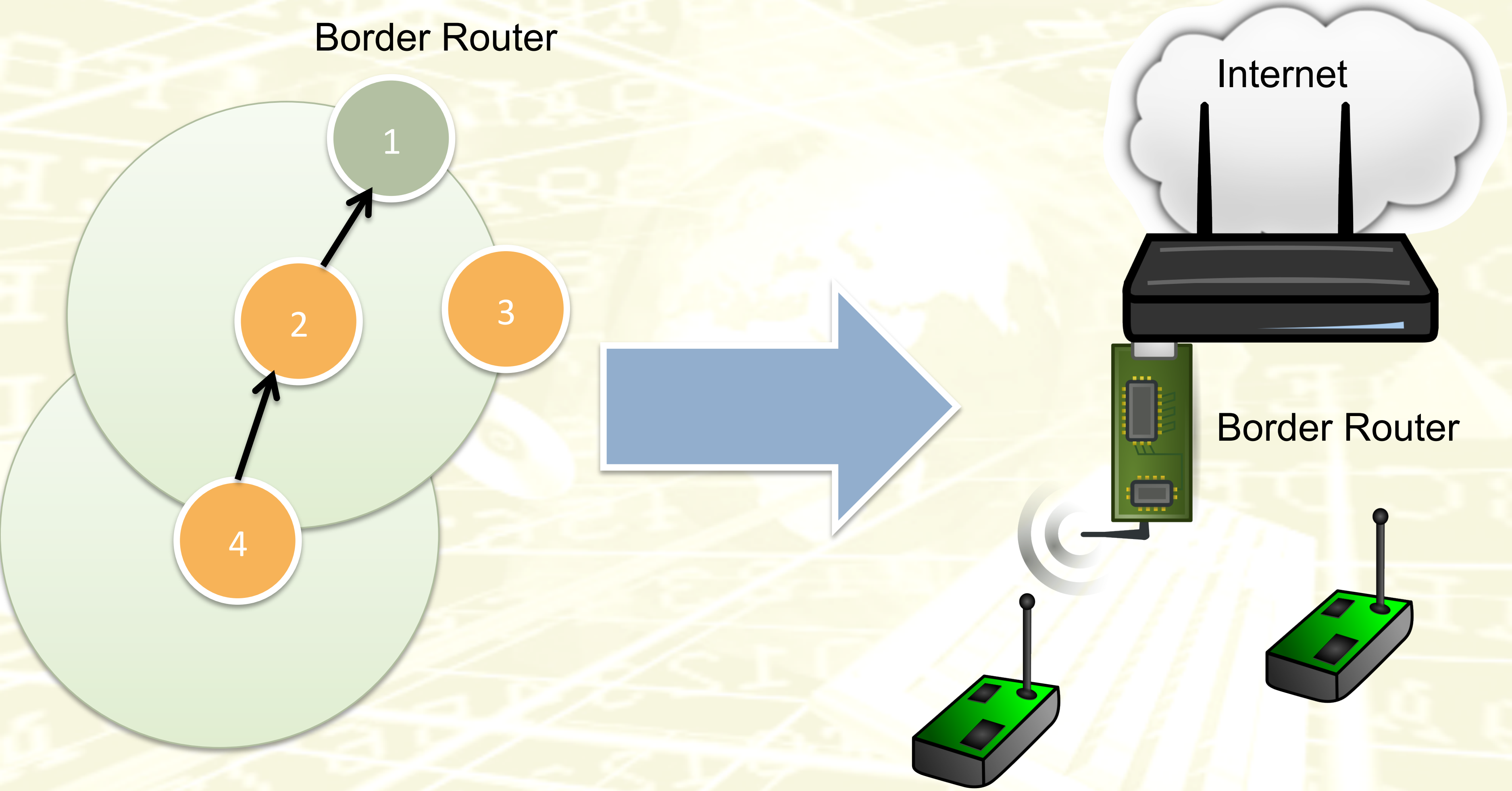
Do it!!

- Modify the examples *receiver.c* and *unicast-sender.c* from the previous lesson in order enable RPL (the receiver is the ROOT node).
- Change some RPL parameters and check the behaviour with wireshark or enabling the log in rpl-timers.c
- (opt) Modify the code of the root node in order to trigger the local repair procedure when the USR button of the mote is pressed.

RPL Border Router

- An RPL border router is used to:
  - – Set the IPv6 global scope address of all motes.
  - – Route messages from leafs to the root.
  - – Interconnect a WSN to the rest of Internet.

Typical scenario



tunslip6

- The tunslip6 will create a virtual interface (called tun0) which is bridged to the border router.
- The interface will have an IPv6 address (aaaa:: 1).
- The border router will use the prefix (aaaa) as the global IPv6 prefix. This will be forwarder and installed in the overall WSN.

Set up in cooja

Border Router has to be the first to be deployed!!

- Deploy a border router
  - examples/ipv6/rpl-border-router/border-router.c
- Add the socket on the border router
  - Tools -> Serial Socket (SERVER) -> Z1 1
- Deploy motes which will get the global IPv6 from the border router
- Use the tunslip6:
  - cd examples/ipv6/rpl-border-router/
  - make connect-router-cooja

Set up on real motes

- Deploy a border router
  - examples/ipv6/rpl-border-router/border-router.c
- Use the tunslip6:
  - cd examples/ipv6/rpl-border-router/
  - Connect the mote to USB
  - make TARGET=z1 border-router.upload
  - make connect-router

BORDER ROUTER MUST BE CONNECTED TO /dev/ttyUSB0

Do it!!

- Set up a WSN with a border-router (which is also the RPL root node), an UDP Receiver and an UDP Sender.

- Try to ping all the motes:
  - E.g. ping6 aaaa::c30c:0:0:1