Project

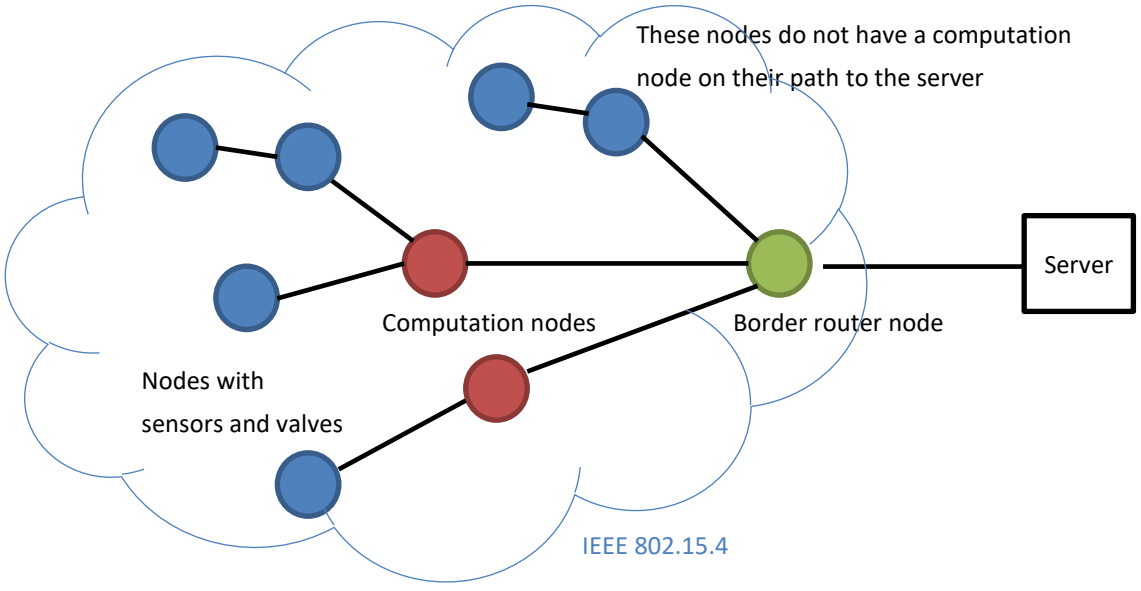
1. Scenario

We consider the (fictive) scenario of a building management system: There is a large number of wireless devices distributed in a building, where each device is directly attached (by cable) to an air quality sensor and to a motorized valve for the air ventilation. Based on the analysis of the sensor data, the device should open or close the valve. We assume:

- Sensor values are read once per minute.
- If the slope of the line obtained by a least-squares fit to the last thirty sensor values is above a certain threshold, the valve is opened for 10 minutes.
- Devices are not powerful enough to do the data analysis by themselves. Instead they send their latest sensor value to a server outside the wireless network which does the analysis and then sends a command to the device to open the valve if necessary.

We call these wireless devices “sensor nodes” in the following.

The sensor nodes communicate over a wireless IEEE 802.15.4 multi-hop network. However, a straight-forward implementation would result in too many communications if the number of sensor nodes is very large. To solve this problem, we introduce “computation nodes” in the wireless network. These are special devices that have more resources than the sensor nodes and that can do the data analysis for them. This should happen completely transparently: The sensor nodes still think that they send their data to the server. If there is a computation node on the path from a sensor node to the server, the computation node will catch the data message (i.e. not forward it to the server), store the data locally, do the data analysis, and then send the outcome of the analysis (“open valve) to the sensor device if necessary.



In order to not overload the computation nodes, a computation node should only do the data analysis for a limited number (e.g. 5) of sensor nodes. If there are more nodes, it will forward their messages to the server. Note that sensor nodes can appear and disappear, i.e. the system does not know in advance for which sensor nodes a computation node will be responsible. If a computation node has not received new data from a sensor node over a longer period, it assumes that the node does not exist anymore and that any stored data from it can be removed to make space for data from other nodes.

2. Implementation

You have to implement the system described in the previous section.

2.1 Sensor nodes and wireless network

Nodes communicate over IEEE 802.15.4. The nodes must organize themselves into a tree with the border router node as root of the tree. To select a parent node, a new node uses the signal strength[1]. New nodes can join the network at any time and nodes can also disappear. The routing must adapt to such changes. To simplify the design, unlike RPL[2], a node only chooses one parent at a given time, i.e., there is exactly one path from the root node to any other node.

Important: In your implementation, you are only allowed to use the Rime[3] modules for single-hop (reliable) unicast, and best effort local area broadcast. You are not allowed to use other protocol stacks nor the existing Rime modules for routing, multi-hop communication etc.

The network should be emulated in Cooja. For the sensor nodes, you can use the Z1 mote type. Since we don't have air quality sensors and valves in Cooja, the sensor nodes should produce fake sensor data. You can represent the status of the (fake) valve by an LED. For the computation nodes, you can use devices of the type "Cooja mote". These are powerful virtual devices that are directly executed on the CPU of the host computer without resource limitations.

2.2 Border router and server

The server is an application running on Linux. It receives and replies to messages from the nodes. You can write the server in C/C++, Java or python.

Some bridging functionality is needed between the border router (i.e. the node at the root of the wireless network) and the server. There are two ways to allow an external application to talk to a node in Cooja:

- Over a network connection to Cooja: To this end, you have to start a Serial Socket Server on the simulated bridge node in Cooja (see the exercises). The server application then can connect to the port of the Serial Socket Server.
- By installing the serial2pty plugin in Cooja: https://github.com/cmorty/cooja-serial2pty
  This plugin allows you to create a serial device for a simulated node in Cooja that you can access from outside. In that way, your simulated gateway node appears as a serial device on the PC like a real device connected over USB.

Both approaches have the same goal: Connect an external application (your server) to the serial interface of a node in Cooja. In that way, your Border router node can send messages to the server by writing on its serial interface and receive messages from the server by reading from its serial interface. This page describes how a node can access its serial interface:

https://github.com/contiki-os/contiki/wiki/Input-and-output

3. Deliverables

You have to deliver a short report in PDF format (~4 A4 pages) describing

- the message format in the sensor network,
- how routing in the sensor network works,
- how the computation nodes work.

Keep the report short and only give the essential information, i.e. your design decisions, not source code.

The report must contain a link to a github or bitbucket repository read-accessible to us with the commented source code of the code for your nodes (sensor nodes, computation nodes, border router) and the server.

Don't forget to put the names of all group members on the first page of the report.

[1] You can find several examples in the Internet how to get the signal quality of the last received packet in Contiki.
[2] Your protocol should be much simpler than RPL, but you are allowed to get "inspiration" from it.
[3] See http://contiki.sourceforge.net/docs/2.6/a01798.html and the exercises for more information on Rime.