PerLa

Focusing on motes

Riccardo Pinciroli 795582
Matteo Simoni 796384

PerLa

PerLa is a **declarative language** and **middleware** for pervasive systems. Its target is to allow users to query a wireless sensor network, using a full declarative SQL-like high level language hiding the complexity of handling different technologies through a middleware platform.

At a high level of abstraction, the architecture of the system is divided in three main components:

1. **Motes** with several sensors on board;
2. A **channel** that decouples communication between motes and server;
3. **Server** that makes requests to the WSN and that gathers responses.

Motes in PerLa

Motes are the **lowest level** in the architecture of PerLa.

They are one of the possible **sources of information** in PerLa; other ones can be web services and streams.

They have to sense data, transmit and receive packets via radio. One of them, moreover, will be the **sink** of the networks being physically linked to a PC (that act as channel of the infrastructure) via a serial port.

Besides **sensors**, they have **timers** used in order to manage periodical actions.

TinyOS

TinyOS is an open source **operating system** (more a **library**) designed for low-power wireless devices, such as those used in sensor networks.

It provides a software **abstraction of the underlaying hardware** by components that are used for sensing data, sending packet, managing the connection and more.

It is programmed in **nesC**, an extension of C language aimed at networked embedded systems.

File Produced

- **PerLaSensorAppC.nc**: the top-level configuration file. It wires together the component that are used in the application.
- **PerLaSensorC.nc**: the implementation of commands/events provided/used by the application.
- **PerLaMessage.h**: it specifies the structure that a message must assume in the PerLa system. It also contains some constants.
- **PerLaSetting.h**: This file contains some structures, used in the application, that producer can modify as he prefers. This will allow him not to touch the code of the application so deeply by accessing only this file.

Moreover, for testing purposes, we have implemented **two fake sensors** (temperature and humidity) that produce random values and the **Java classes** that abstract the channel that receives messages trought the serial communication.

PerLa Protocol (1)

We defined a **protocol** in order to set the rules in the communication between motes and the channel.

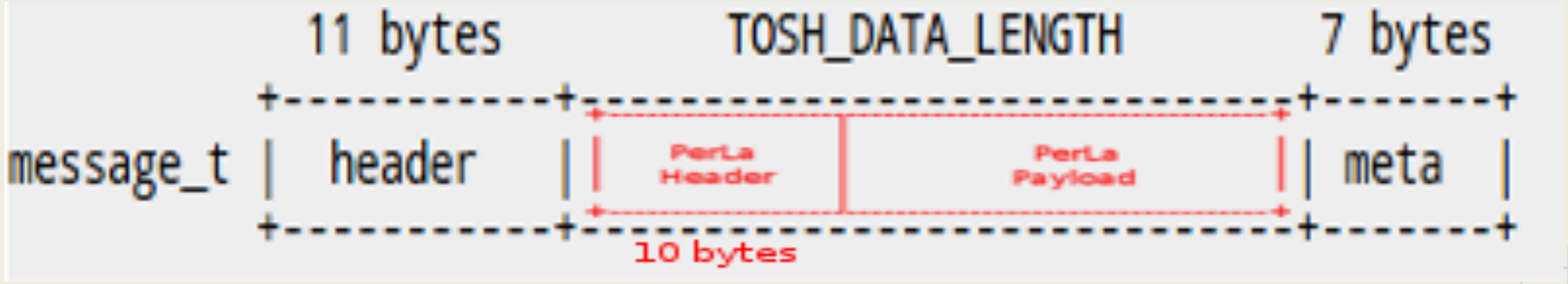
A **PerLa message** is composed by an **header** that contains general information such:

- id; (of sending motes)
- timestamp;
- type; (request/response and of which type)
- numPckt and numPcktToReceive. (used if the packet has to be splitted)

and by a **payload** that is a sequence of byte which structure is defined by an XML descriptor sent to the channel (that must interpretate the bytes) when nodes join the wireless sensor network.

PerLa Protocol (2)

Because it is not allowed to access directly the header of a TinyOS Message and modify it in order to add the above mentionated fields, we create our own PerLa Message that is then incapsulated in a TinyOS Message.



By doing this, we have a total control of our message (both the header and the payload part) and we lay on a standard.

Implemented Features

At the state of art, a mote with installed our TinyOS application is able to perform several operations. In particular it can:

- Sense **periodically temperature and/or humidity** with a specified time period (in seconds).
- Sense **temperature and/or humidity one-shot**.
- Sense **temperature and/or humidity by event**. (Until now the events supported are "temperature/humidity **greater than**")
- Send a packet with a sensed value or the XML descriptor **to the sink** (and split it in more packet if the information that has to be sent is too big)
- The sink is able to route the packet that receives from other motes **to the channel (e.g. a PC)** via serial communication.

Open Problems

PerLa is an **ongoing project**. The part about motes has also to be improved in order to make it more powerful.

Some **open problems** that we focused are:

- Extend the event-based sampling condition (maybe in a dynamic way);
- Routing Protocols;
- Battery Management;
- Packet Resending Management.

Q&A

QUESTIONS?

THANKS FOR YOUR ATTENTION