Power Aware Wireless Sensor Networks

Module Library

Introduction

The PAWiS Project at the Institute of Computer Technology, University of Technology, Vienna, targets on optimized wireless sensor networks (WSN). These require nodes with exceptionally low power consumption, yet have to drive sensors and radio communication.

Common optimization approaches concentrate on modules to make every module as good as possible. Unfortunately this reveals only local optima. To find the best possible solution within certain constraints, the global optimum must be sought. The PAWiS Framework comes in at this point enabling cross layer and cross module optimization.

The PAWiS Simulation Framework is supplemented by the Module Library. This provides numerous implementations of a set of module types. The module types and the interfaces between the modules are defined by the Interface Specification.

Module Types

The main approach of the module library is to offer modules according to defined module types. This enables the user to easily exchange module implementations without touching other modules. The module types which belong to the network stack are:

- **Application Layer:** Typically implements tasks like measurement, create a packet and the reception packet.
- **Transport:** This layer is optional and used to ensure a reliable end-to-end communication.
- The **Network** layer handles the routing of packets. This includes to find a route for delivering a packet as well as forwarding of packets.
- Coordination between nodes to access the RF channel is performed by the **MAC** module.
- The **Physical** module performs the basic communication between the nodes via the Air object.

Management Planes

- The **Cross Layer Management Plane** (CLAMP) stores global variables for all other modules.
- **Energy Management Plane**: This plane is responsible for an estimation of remaining battery capacity and energy aware control of the other modules.
- Cross-layer security services and key management are provided by the **Security Management Plane**.

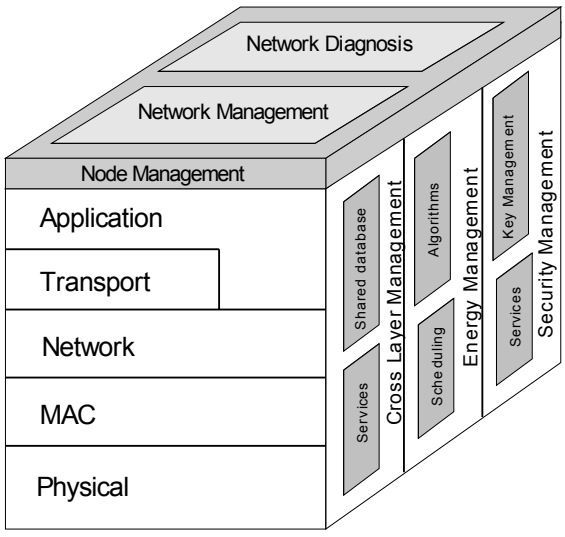
For any common functionality like the node start-up the **Node Management** is responsible.
Additionally to these rather software related modules there are also hardware modules.

- The **CPU** module implements the timing and power consumption of a real microcontroller CPU.
- Other modules like a **Timer**, **ADC**, **GPIO**, **SPI**, ... are usually provided by a microcontroller and implemented accordingly.

For every module of the module library a detailed **Module Datasheet** is enclosed. This documents its function, interfaces parameters, initialization, interrupts and other specialties.

Node Management

Common node functionality as well as the integration of the modules to a whole node are implemented in the Node Management.



- **Initialization:** The starting point of CPU execution of a node is put to the node management. It calls initialization functions of the other modules as specified in their datasheet.
- **Timer:** To enable firmware modules to be independent of the actually used microcontroller the timer has to be abstracted.

Therefore the Node Management implements a driver for the timer. The abstraction implements delays with constant duration and with stopping conditions as well as other frequently used functions.

- **Periodic Scheduling** implements the execution of periodic tasks, e.g., for measurement and listen.
- **Interrupts:** For platform independence the interrupt routing, including interrupt sources, vectors and service routines have to be adapted for the combination of the microcontroller and the utilized modules.
- **Main Loop:** After initialization the main operation of the WSN node is started. The parts which are not captured by interrupt service routines are implemented here.

Most of the above jobs are done implicitly in the real firmware. Only for the simulation and the desired platform independence the adaption between modules and the microcontroller (CPU, Timer) have to be implemented.

Cross Layer Management Plane

The main idea of the **Cross LAyer Management Plane (CLAMP)** is to provide a rich set of performance aware and energy aware network parameters to different layers to dynamically adapt according to application requirements. This helps to overcome the limitation, that interfaces are only between adjacent layers of the network stack.

Interfaces

The interfaces for communication between the modules are grouped into three types.

- **Mandatory** interfaces are strictly defined and must be implemented by the module. As use you can rely on its existence.
- **Optional** interfaces are also strictly defined but it is not required to be implemented. If it is implemented then it must conform to the specification. The module datasheet mentions the implemented optional interfaces.
- **User-defined** interfaces are not defined by the Interface Specification and can be freely specified and implemented by the module programmer. It must be documented in the module datasheet.

Common interfaces between the network layers are for sending and receiving.

- **Send:** Originates from the application layer and propagates the packet through the network layers top-down to the physical layer.
- The **Receive** interface is only implemented from the routing layer upwards and invoked asynchronously from bottom-up. The communication between the MAC and Phy use a dedicated listen interface which is invoked by the MAC layer according to the implemented protocol.

Common interfaces to the management planes include

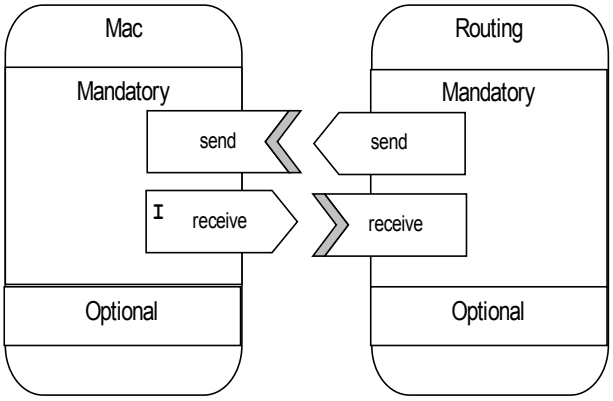
- to CLAMP: subscribe, query, publish, update
- from CLAMP: onChange (=notify)
- from Nm: init

Available Modules

CPU: MSP430
Routing: EADV
MAC: CSMA-MPS
Phy: CC2400

Availability

The PAWiS Module Library is published under the terms of the GNU LGPL (Lesser General Public License). That means that you can download, use and redistribute it free of charge while your source code does not need to be licensed under the LGPL.



Resources

PAWiS Homepage:

http://pawis.sourceforge.net/

PAWiS Wiki:

http://wiki.pawis.sourceforge.net/

System Requirements: The PAWiS Framework is implemented platform independently and supports Unix systems (e.g. Linux) as well as the Windows operating system.

The PAWiS Framework is based on the OMNeT++ 3.3 Discrete Event Simulation System.

Build tools:
Unix: GNU GCC 4.0+; Autoconf 2.60a, Automake 1.9.6
Windows: Microsoft Visual C++ Ver. 7
LUA 5.2, Doxygen 1.4.7

Recommended system parameters: 100 MB free HDD space, 256 MB RAM, 1024x768 screen resolution, CPU at 1200 MHz