Hive - Wireless Sensor Networks simulator

Catalina Macalet, Sorin Dumitru
Automatic Control and Computers Faculty
University POLITEHNICA of Bucharest
Splaiul Independenței nr. 313, Bucharest, Romania
{*catalina.macalet,sorin.dumitru*}*@cs.pub.ro*

June 4, 2012

Abstract

This paper describes Hive, a Wireless Sensor Nodes network simulator. Hive is component based and highly scalable allowing simulations of thousands of nodes.

It provides a standard library based on events for developing applications for Wireless Sensor Nodes. It abstracts architecture, network protocols and components in order to provide portability.

1 Introduction

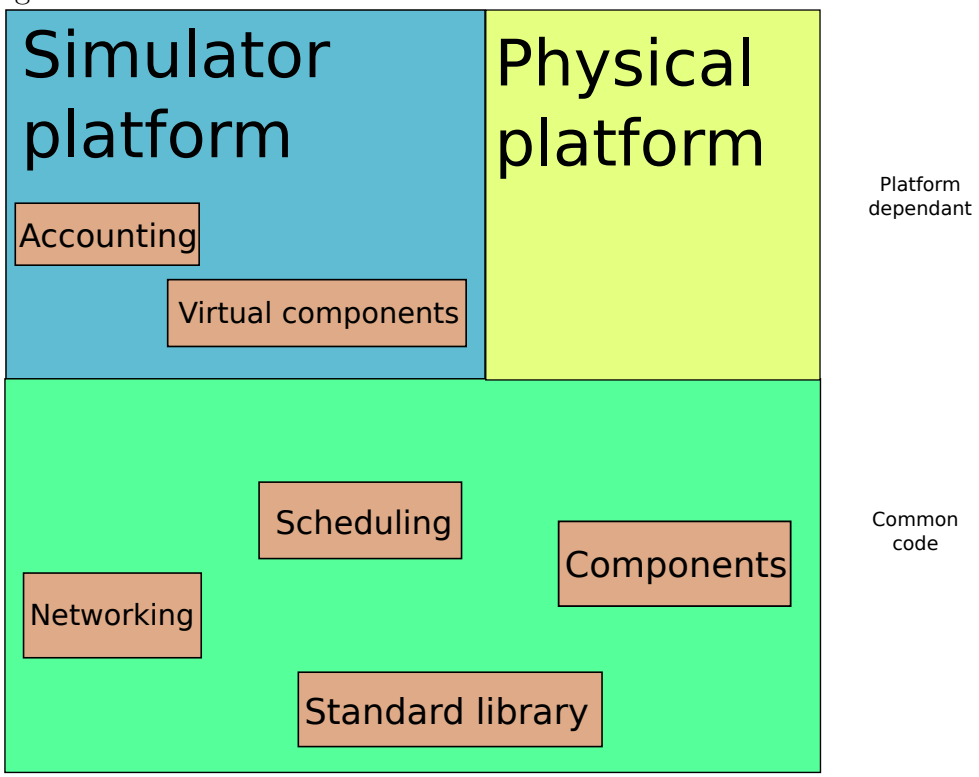
Hive is a Wireless Sensor Network Simulator that is intended to be used for simulating routing protocols for Wireless Sensors Networks in order to determine the best suited protocol for a given topology. There exist a number of simulators used for simulating Wireless Sensor Networks such as NS-2, J-sim[1] and TOSSIM[2]. We will not go into details regarding these simulator since this was covered in our previous work. What is to be noted though is that, after the survey we did on these simulators, we were able to define some key features that we wanted the *Hive* simulator to have, such as, but not limited to: to be component based in order to make it easy to extend and to make future development easier to be done, to be able to simulate a wide variety of nodes in order to not be limited to only some sensors types, to make it easy to write code that is to be run on nodes in order to make the implementation of the routing protocols easier to be done and, after looking at the trends in the Wireless Sensor Networks research, we decided that the simulator should, in the begining support at least the 6LoWPAN and ZigBee protocol stacks. Also, unlike some of the simulators enumerated above, *Hive* targets topologies of Wireless Sensor Networks and so, during the desing and implementation phases decisions were made so that *Hive* could simulate as accurate as possible a real sensor.

The structure of the paper is at follows: in section 2 we present the *Architecture* of the *Hive* simulator, discussing the reasons for which such an architecture was chosen and present the communication mechanisms between the components. Then, the *Implementation* section follows where we briefly present the libraries and tools used for implementation and the reasons for which we chose to use these and, also, go into some details concerning the implementation. In section *Testing* we present the tests we want to do in order to test the way *Hive* behaves and to be able to provide some numbers for

the scalability matter. In the end some conclusions are drawn, and the next steps are highlighted in the *Conclusions and Further Work* section.

2 Architecture

Design and architectural details.



3 Implementation

3.1 Standard library

Hive provides for users a standard library for programing wireless sensor nodes. It provides an API similar to the POSIX library.

3.2 Networking

3.3 Scheduler

Most of work on a wireless sensor node is triggered by an event. For example we read the temperature sensor every 5 minutes, which is a timer event, or an actual event happens on the node, for example a packet arrives.

The job of the scheduler is to provide a way to register for and notify of events. It's split into two parts, a platform agnostic one and a platform specific one. The platform agnostic part contains the API for users to use the scheduler. It allows specifing callback for events or timers:

```
1 typedef void (*callback_t)(void *arg);
2 void schedule_timer(struct scheduler *scheduler,
```

```
3                       /* Timeout in miliseconds *?
4                       int timeout,
5                       /* Callback function */
6                       callback_t *cb,
7                       /* Argument to be passed to the callback */
8                       void *arg);
```

The platform specific part contains the platform dependent implementation of the scheduler. Each platform has a different implementation for timers so each platform will have to implement its own version of this. For example linux has timerfds, but an Atmega microcontroller has hardware timers. The scheduler implementation for that platform will use what the platform provides.

The implementation of the scheduler on the simulation platform will be done using libevent. The libevent API provides a mechanism to execute a callback function when a specific event occurs on a file descriptor or after a timeout has been reached. Furthermore, libevent also support callbacks due to signals or regular timeouts. Libevent uses the event loop mechanisms provided by the system on which it runs. This can be /dev/poll, kqueue, select, event ports, poll, epoll.

3.4 Accounting

Everything that runs on a node consumes some power and the simulator will have to reflect that in the battery level of the node. For example reading a sensor will consume a small part of the battery while sending messages to a node will consume much more power, depending on the distance of the target node from the source node.

Every event on a simulated node will pass one or more times through the scheduler. This will happen either a normal timer event or by the simulator part of the driver for the component. If a timer expires, we call the callback function which means the node is running some code and we will be able to estimate how much battery it consumes based on the time the callback takes. For a network packet we will be able to do the accouting when the packet reaches the physical layer of the simulator where it will be able to compute the power consumption to communicate with the target node.

4 Testing

5 Conclusion and Further Work

The desing of the *Hive* simulator provides an easy-to-use interface for adding support for other protocol stacks and also for adding new types of nodes by defining values for sensor properties such as power, CPU, resources consumed when sending/receiving a message, etc making it a useful tool for research purposes. The effort needed to port *Hive* simulator to another platform should not be too big since the changes are limited to the physical platform code-area. There are a number of features that we want to add to the simulator such as: a GUI in order to make it easier to use - at first limited to selecting nodes or componet types and adding them in a 2D space and later, at a more developed phase, it could permit to create a more realistic scenario by incorporating obstacles, natural phenomena, etc.

Acknowledgment

The authors would like to thank XYZ for their support and dedication.

References

[1] A. Sobeih and M. Viswanathan and D. Marinov and J.C. Hou. J-Sim: An integrated environment for simulation and model checking of network protocols. In *IEEE International Parallel and Distributed Processingi Symposium*, pages 1–6. IEEE Computer Society, 2007.

[2] Philip Levis and Nelson Lee. TOSSIM: A simulator for TinyOS Networks. pages 1–17, September 2003.