TinyOS Meets Wireless Mesh Networks

Muhammad Hamad Alizai, Bernhard Kirchen, Jó Ágila Bitsch Link, Hanno Wirtz, Klaus Wehrle
Communication and Distributed Systems, RWTH Aachen University, Germany
{lastname}@comsys.rwth-aachen.de

Abstract

We present TinyWifi, a nesC code base extending TinyOS to support Linux powered network nodes. It enables developers to build arbitrary TinyOS applications and protocols and execute them directly on Linux by compiling for the new TinyWifi platform. Using TinyWifi as a TinyOS platform, we expand the applicability and means of evaluation of wireless protocols originally designed for sensornets towards inherently similar Linux driven ad hoc and mesh networks.

1 Motivation

Although different in their applications and resource constraints, sensornets and Wi-Fi based multihop networks share inherent similarities: (1) They operate on the same frequency band, (2) experience highly dynamic and bursty links due to radio interferences and other physical influences resulting in unreliable routing paths, (3) each node can only communicate with nodes within its radio range forming a mesh topology, and (4) the intended use cases in both domains demand a reliable and highly scalable communication infrastructure. As a result, the majority of algorithmic concepts [1, 5, 6] and state-of-the-art protocols — including MAC [7], link estimation [2] and routing [3, 4] — originally designed for sensornets are equally relevant in the Wi-Fi domain and vice versa. It is due to the significant implementation and porting effort that the developers are restricted to build and evaluate their prototypes for a single domain and implicitly assume their applicability in the other [2, 4, 6].

We introduce TinyWifi, a TinyOS platform supporting Linux driven devices. It allows direct execution of protocol libraries originally developed for a different networking domain. Applications from highly resource constrained sensornets can easily be compiled for resource rich Wi-Fi based networks , thereby making the very rich and mature protocol-repository of TinyOS available for broader wireless research.

During our evaluation on a 50 node mesh testbed, we observed that TinyWifi is particulary useful for (1) evaluating prototypes, (2) fine-tuning protocol parameters and (3) establishing multiple performance metrics in different wireless domains without reimplementation.

SenSys'10, November 3–5, 2010, Zurich, Switzerland.
ACM 978-1-4503-0344-6/10/11

2 TinyWifi

The goal of TinyWifi is to enable direct execution of TinyOS applications and protocols on Linux driven network nodes with no additional effort. To achieve this, the TinyWifi platform extends the existing TinyOS core to provide the exact same hardware independent functionality as any other platform (see Figure 1). At the same time, it exploits the customary advantages of typical Linux driven network devices such as large memory, more processing power and higher communication bandwidth. In the following we describe the architecture of each component of our TinyWifi Implementation.

2.1 Timers

The TinyOS timing functionality is based on the hardware timers present in current microcontrollers. A sensor-node platform provides multiple realtime hardware timers to specific TinyOS components at the HAL layer - such as alarms, counters, and virtualization. Once configured, these timers trigger an interrupt in the future without the need for continous monitoring.

Although our target devices provide hardware timers as well, user space applications have no access to them. Therefore we use the Linux *itimer* library. Each process running on a Linux kernel is allowed to use a single realtime *itimer*. We introduce a new VirtualizeLinuxTimer component that uses this single *itimer* and provides virtual alarms and timers, which the TinyOS timer multiplexing is then based on. Hence, providing timing functionality similar to any other sensor platform (see Figure 2).

2.2 The Split-Phase Operation

Non-blocking system calls are realized as split-phase operations in TinyOS. A command that starts a system service returns immediately while the completion of that service is signaled later via so-called events. TinyWifi supports both blocking and non-blocking system calls. The support for blocking system calls in Linux is trivial. To mimic hardware such as radio chips that process data in parallel, we use threads. The completion of these parallel processing threads is then indicated via a Linux signal, which in turn triggers the main TinyOS thread.

2.3 Radio Communication

We encapsule TinyOS messages in UDP packets using datagram sockets. Another option would be to pass the data

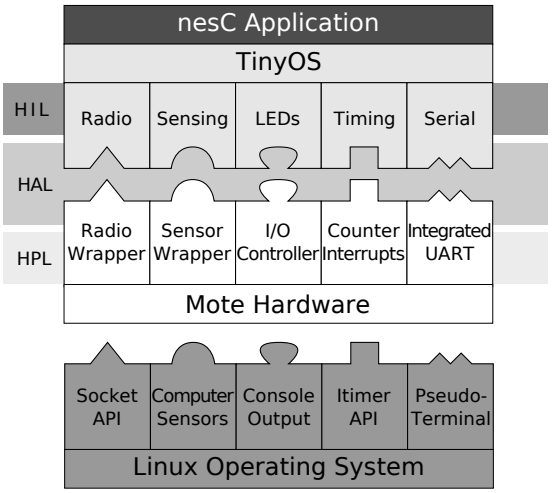

Figure 1. TinyWifi Architecture

The hardware abstraction layer (HAL) translates HIL functionality to the device specific modules of the hardware presentation layer (HPL). TinyOS is independent of the HPL implementation, thus new platforms can be added by providing the corresponding HPL modules.

directly to the Wi-Fi adapter but we avoid this for three reasons: (1) To maximize portability, (2) to minimize interference with different applications on the network, and (3) to allow direct execution of TinyWifi without negotiating special kernel level privileges. Moreover, to establish similar behavior in both wireless domains, we broadcast packets but suppress routing by adjusting the TTL value accordingly. By doing so, packets are only received by TinyWifi nodes in the radio range.

2.4 Serial Communication

The majority of TinyOS applications use the serial communication port, particulary the base station. In order to provide a similar functionality, i.e. serial active messaging on a TinWifi device, a Linux pseudo terminal is used. As with typical motes, an unaltered serial forwarder connected to the pseudo terminal allows for sending and receiving serial data to and from the TinyWifi node.

2.5 Sensing and Debugging

Since we emphasize on protocol evaluation, sensing is a subordinate issue. Nevertheless, we do supply demo sensor implementations to allow for TinyWifi to be used out of the box.

In addition to the *printf* library to output debugging information through the serial interface to an attached PC and displayed in a human readable manner, TinyOS provides *dbg* functions to print additional information. In our TinyWifi implementation we print those messages directly to standard output. Similarly, to indicate the status of a physical mote to a developer, motes are equipped with LEDs. TinyWifi provides pseudo-LEDs: Messages are sent to standard output similar to the debugging mechanism of the TOSSIM simulator.

3 Initial Tests

TinyOS is equipped with a number of useful test applications: *Blink* and *BlinkToRadio* demonstrate the proper functioning of timers and radio communication, respectively. *Oscilloscope* and *Multihop-Oscilloscope* prove an accurate implementation of demo sensors and the serial message interface. We also evaluated TinyWifi in a 50 node wireless mesh testbed by running the Collection Tree Protocol, Beacon Vector Routing and the S4 routing protocol. We were able to successfully build routing trees and receive and display demo sensor measurements from the TinyWifi nodes via a TinyWifi base station.

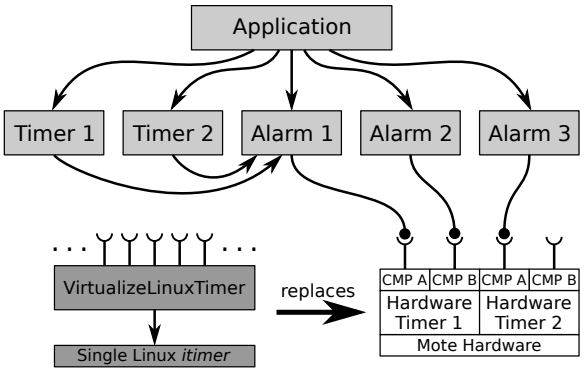

Figure 2. Timers

The TinyWifi timer implementation provides several instances of alarms and timers because Linux only provides a single realtime timer per proccess.

4 Future Work

Our future work is mainly focused in three directions: (1) We aim to use the wireless interface directly in order to optimize active messaging and utilize extra features such as RSSI. (2) We want to thoroughly evaluate the routing and link estimation protocols of TinyOS. Finally, (3) our major focus lies in comparing these protocols with standard protocols used in the in the Wi-Fi domain, such as OLSR and AODV.

Acknowledgments

This research was funded in part by the DFG Cluster of Excellence on Ultra High-Speed Mobile Information and Communication (UMIC), German Research Foundation grant DFG EXC 89.

5 References

[1] M. H. Alizai, O. Landsiedel, J. A. Bitsch Link, S. Goetz, and K. Wehrle. Bursty traffic over bursty links. In *SenSys'09*, Nov. 2009.

[2] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis. Four-bit wireless link estimation. In *HotNets*, 2007.

[3] R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, and I. Stoica. Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In *NSDI*, May 2005.

[4] Y. Mao, F. Wang, L. Qiu, S. S. Lam, and J. M. Smith. S4: Small state and small stretch routing protocol for large wireless sensor networks. In *NSDI*, 2007.

[5] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali. Routing without routes: The backpressure collection protocol. In *IPSN*, 2010.

[6] J. Newsome and D. Song. Gem: Graph embedding for routing and data-centric storage in sensor networks without geographic information. In *SenSys '03*, 2003.

[7] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04*, 2004.