TinyOS

Seminar of Distributed Systems 2004/05, Report

Jan S. Rellermeyer
Department of Computer Science, ETH Zurich

jrellermeyer(at)student.ethz.ch

1 Introduction

Sensor networks are low cost and low power devices, designed to collect data, do some local computation and transmit partially processed data via radio frequency or bluetooth. At any time, single nodes of the network might fail or change their position due to environmental influences. Thus sensor networks must be self-organizing and cannot use a globally known topology. The major bottleneck of the motes themselves is the power consumption and the low computational power.
TinyOS has been designed to meet the requirements of sensor networks. Sensor values have to be processed in real time to avoid data loss. As the hardware contains physical parallelism, the OS must provide some kind of multithreaded architecture.

2 Summary

Even systems that call themselves *real time* need some hundreds of processor cycles to perform a context switch. On a small microprocessor like the Atmel AVR, used in both mica-family motes and the BTnode, this is unacceptable and would lead to data loss. TinyOS does not use a stack-based threaded architecture but an event based architecture. This allows to have only one stack and a single execution context. In case a hardware component wants to deliver data, it calls an event that will preempt running tasks.
TinyOS is based on nesC, a C dialect that is imperative at low level and declarative at high level. Applications are component based, code is encapsulated in components that are defined by the interfaces they provide and by those they can consume. Interfaces are bidirectional, they do not only define the commands that have to be implemented by the lower level that implements the interface, but also events that have to be implemented by the higher level that uses the component implementing the interface. The component based model decouples API and implementation and hides the specific properties of an implementation. At compile time, implementations can be substituted by different implementation, either in hardware or in

software, only the wiring has to be changed. This makes TinyOS very flexible.
A core feature of TinyOS is the scheduling, TinyOS uses a simple queue with length 7 and a two level scheduling. Events have high priority and can preempt tasks, that have low priority. To avoid blocking scenarios, events and commands are expected to do only state transmissions and leave complex computations to tasks, that can be preempted if necessary. Hardware interrupts thrown by timers or by the radio module map to first level events.
The paradigm for network transmissions in TinyOS is active messaging. Messages contain a handler address and on arrival this handler is called. Each node is expected to run the same handler code and can either redirect the message to a neighbor, if it is not the receiver, or start some local events as reaction to the message. Transmissions are best effort, if more is needed, it is up to the application to implement more sophisticated features like flow control or encryption. To send packages over the network, TinyOS uses multi-hop routing instead of point-to-point connections to save transmission power. Route discovery is done by 2-hop broadcast and topology discovery is based on shortest path from each node to the base station.
A typical application for TinyOS is TinyDB, a RDBS interface for TinyOS sensor networks. In general, SQL commands are transmitted from the base station down the tree to every node. Intermediate nodes collect the data from the children and transmit the aggregated data to the root node.

3 Conclusion

Sensor networks become more and more popular, in the meantime, global players like Intel are pushing the technology. A yet unsolved problem is the disposal of sensor nodes after they have reached the end of their life cycle.
TinyOS is designed to meet the requirements of small devices with small resources. The major bottleneck is the power consumption of the radio module so better routing algorithms could improve the system significantly. Static resource allocation is a direct result of the small resources but with computation power of current motes, dynamic resource allocation could be afforded. Additionally, systems to change the node's code at runtime might broaden the room for applications as currently nodes have to be collected and manually reprogrammed to change their general behavior. Dynamic component based systems like OSGi are more flexible than TinyOS while still small-footprinted.

References

[1] A. Easwaran: TinyOS. http://www.i2r.a-star.edu.sg/icsd/SecureSensor/papers/TinyOS.pdf

[2] M. Franklin, W. Hong: ETH Zurich Distributed Systems Summer School: Data Streams and Sensor Networks, ETH Zurich, 2004. http://www.dcg.ethz.ch/micsss2004/files/Franklin.zip

[3] J. Hill: A Software Architecture Supporting Networked Sensors, U.C. Berkeley, 2000. http://www.tinyos.net/papers/TinyOS_Masters.pdf

[4] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister: A System Architecture for Networked Sensors, U.C. Berkeley. http://web.cs.berkeley.edu/tos

[5] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister: System Architecture Directions for Networked Sensors. In Proc. 9th Internation Conference on Architectureal Support Programming Languages and Operating Systems (ASPLOS-IX), pages 93-104. ACM Press, New York, Nov. 2000

[6] V. Raghunathan: TinyOS. http://black.csl.uiuc.edu/∼vivek/talks/tinyostalk.pdf

[7] G. Wong: Motes, nesC and TinyOS. http://cs-people.bu.edu/gtw/motes/talk.pdf

[8] www.tinyos.net: TinyOS Tutorial