Sensor Network Management Protocol for State-driven Execution Environment*

Tae-Hyung Kim[1] and Seongsoo Hong[2]

[1] Department of Computer Science and Engineering
Hanyang University, Ansan 426-791, Korea
tkim@cse.hanyang.ac.kr
[2] School of Electrical Engineering and Computer Science
Seoul National University, Seoul 151-742, Korea
sshong@redwood.snu.ac.kr

Abstract. Wireless distributed sensor networks have gained a new importance in a wide spectrum of ubiquitous computing applications. This is largely due to the recent advancement in sensor nodes with low-cost RF communication capability. Although the operating pattern of sensor network applications resembles a traditional distributed computing platform, they present very unique technical challenges and constraints that are ineluctable to developers. Considering the operating environment is constrained by extremely limited hardware resources, we present a finite state machine based operating system for sensor nodes that can meet the seemingly contradictory design requirements of high concurrency and reconfigurability in spite of their limited resources. In this paper, we explore the sensor network management protocol, which is a largely untouched region in sensor network research. For this, we make use of the simple but powerful SenOS operating system.

1 Introduction

Wireless distributed sensor networks have gained a new importance in a wide spectrum of ubiquitous computing applications because they are a prime technical enabler that can provide a means of connection between the computational and the physical worlds. Sensor networks consist of a set of sensor nodes, each equipped with one or more sensing units, a wireless communicating unit, and a local processing unit with a small memory footprint. They have come into general use lately largely due to the recent advancement in sensor nodes with low-cost radio frequency communication capability. Although the operating pattern of sensor network applications resembles a traditional distributed computing platform, they present very unique technical challenges and constraints that are ineluctable to developers.

Wireless sensor nodes are characterized by (1) extremely limited resources including computing power, memory, and supplied electric power, (2) data centric programming styles that view a sensor network as a distributed computing platform consisting

* The work reported in this paper is supported in part by MOST (Ministry of Science and Technology) under the National Research Laboratory (NRL) grant 2000-N-NL-01-C-136.

of tens of thousands of autonomously cooperating nodes, and (3) a disposable computing platform that does not allow recycling of the network. Many embedded real-time systems are naturally amenable to the finite state machine model and networked sensor applications are one such system. Thus, we exploited a finite state machine based execution model to design an ideal operating system for a networked sensor node. One of the benefits of our sensor node operating system, SenOS [2], is that application programmers can take advantage of high-level CASE tools like UML-RT [3] to synthesize executable code for a sensor node.

From the viewpoint of the sensor network protocol stack, much research has concentrated on the first three layers; physical, data link and network layers [1]. The necessity of a transport layer is dubious because global addressing is not the norm in wireless sensor networks for economical reason. But application layer protocols remain a largely unexplored area thus far, in spite of their importance for high level abstraction in sensor network application programming. In this paper, we try to show the operational efficacy of a state-driven execution environment for wireless sensor networks by presenting a sensor network management protocol for SenOS. We present a brief overview of the SenOS architecture and show how a power management protocol can be gracefully embedded into a networked sensor node as an example of a sensor network management protocol.

2 SenOS Architecture

When a finite state machine is implemented, a valid input (or event) triggers a state transition and output generation, which moves the machine from the current state to another state. Such a state transition takes place instantaneously and an output function associated with the state transition is invoked. Using this execution mechanism, a finite state machine sequences a series of actions or handles input events differently depending on the state of the machine.

The SenOS kernel architecture enables such a finite state machine based execution model. It is comprised of three components: (1) the Kernel consisting of a state sequencer and an event queue, (2) a state transition table, and (3) a callback library. The Kernel continuously checks the event queue for event arrivals; if there are one or more inputs in the queue, it takes the first one out of the queue and triggers a state transition if the input is valid. It then invokes an output function associated with the state transition. To do so, the Kernel keeps track of the state of the machine and guards the execution of a callback function with a mutex that can guarantee the run-to-completion semantics.

The call back library provides a set of built-in functions for application programmers, thus determining the capability of a sensor node. The Kernel and callback library should be statically built and stored in the flash ROM of a sensor node whereas the state transition table can be reloaded or modified at runtime. SenOS can host multiple applications by means of multiple co-existing state transition tables and provide concurrency among applications by switching state transition tables. Note that each state

transition table defines an application. During preemption, the Kernel saves the present state of the current application, restores the state of the next application, and changes the current state transition table.

The SenOS architecture also contains a runtime monitor that serves as a dynamic application loader. It contains an interrupt filter that is in fact a generic interrupt service handler invoked upon every external interrupt. When SenOS receives an application reload message via an interrupt from a communication adapter, the Monitor puts the Kernel into a safe state, stops the Kernel, and reloads a new state transition table. Note that the Monitor is allowed to interrupt the Kernel anytime unless it is in a state transition. Since state transition is guarded by a mutex, the safety of a finite state machine is not compromised by such an interruption. Such dynamic reconfigurability of SenOS helps the basic operating system architecture seamlessly be extended to include a sensor network management protocol because an application layer protocol can be modified in the same way as an application program .

3 Sensor Network Management Protocol Example: Power Management

Sensor network management protocol enables application programmers to access sensor node resources for the sake of efficient networked sensor operations. Since sensor nodes have no global identification, system administrators should interact with the network using this protocol. While an individual sensor node cannot be addressed directly, we can justly assume that there is some way to disseminate the user's interests to the node using routing protocols at the network layer. One of the most valuable resources to manage in sensor networks is power, as sensor nodes are mostly battery powered and replenishment is almost impossible, especially in inhospitable environments. In this case, sensor node lifetime is determined by battery lifetime. We use such a power management example as an application layer protocol for sensor network management under SenOS.

The callback library repertoire for power management can be listed as shown in [4]: sensor hardware access functions like getTemp( ), turnOn( ), turnOff( ), location-aware functions like isNeighbor( ), communicating functions like tellNeighbor( ), and event-handling functions like receive( ), every(int period), expire(int settimer).

Using the abovementioned callback functions and the state-driven execution engine of SenOS, we can embed a power management protocol into sensor networks. For instance, Fig. 1 shows a two-tier sensor network using hierarchical clustering as in [5]. If Tier 0 nodes are simply redundant nodes for alternate operation in order to increase the lifetime of the entire cluster at Tier 0, only one node is necessary and sufficient for a required sensing task. Battery life is very difficult to lengthen, especially if we should maintain the low-cost and small-sized constraints, so the increase of sensor field density is an easily scalable option to lengthen the sensor node's lifetime after its deployment. However, we need to have a proper power management protocol that achieves this goal. At a lower level in a cluster, only one super node should be alive

for a given period of time, and all other sensors should remain dormant for power conservation. Choosing such a super node alternately in a cluster can be implemented in a finite state machine with the callback library functions for power management under SenOS.

In general, the basic idea for power conservation is to shut down nodes when they are not needed and wake them up when necessary. But in reality, significant power-wise benefits cannot be achieved by this naïve policy, because frequent sleep-state transitions incur overhead. An application specific algorithm for power management is thus called for. Dynamic power management (DPM) in [6] is such a comprehensive algorithmic power management technique that meets our requirements. It adopts a power-aware sensor node model in which each sub-unit in a node is allowed to have multiple operational modes. For instance, a processing unit can be in active, idle or sleep mode, and an RF communicating unit can be in transmit, receive, standby or off mode. DPM determines the sleep-state transition moment dynamically at run time. The end result of this technique can be elegantly expressed in a finite state machine model. The State-driven SenOS execution model by nature allows a variety of sensor management protocols including power management.

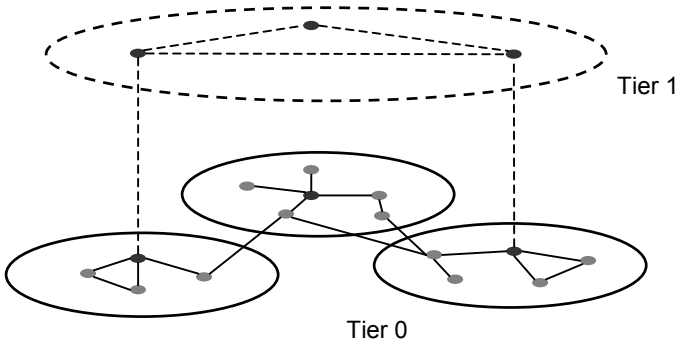

Fig. 1. Two-tier Sensor Network for Power Management Protocol.

4 Conclusion

SenOS [2] is a state machine based operating system for a networked sensor node. It consists of the Kernel, Monitor, replaceable state transition tables, and call back libraries. In this paper, we have explored how to extend SenOS to implement a sensor network management protocol, which is one of the largely untouched regions of sensor network research. Specifically, power management protocol is illustrated to show the dynamic reconfigurability of SenOS. It gives us an important refinement on application layer support for the sensor network protocol stack. Consequently, programmers can write application programs of higher abstraction for sensor networks.

References

1. I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, A Survey on Sensor Networks.
2. S. Hong and T. Kim, SenOS: State-driven Operating System Architecture for Dynamic Sensor Node Reconfigurability, International Conference on Ubiquitous Computing (2003).
3. IBM (former Rational Software), http://www.rational.com.
4. C. Shen, C. Srisathapornphat, and C. Jaikaeo, Sensor Information Networking Architecture and Applications, IEEE Personal Communications (Aug. 2001).
5. D. Estrin, R. Govindan, and J. Heidemann, Embedding the Internet, Communications of the ACM, Vol. 43 (May 2000).
6. A. Sinha and A. Chandrakasan, Dynamic Power Manangement in Wireless Sensor Networks, IEEE Design & Test of Computers (2001).