PART I
Overview

1 Introduction

> The creation of genuinely new software has far more in common with developing a new theory of physics than it does with producing cars or watches on an assembly line.
>
> —T. Bollinger

Software that drives the operations of sensors and communication among sensors is basic to any meaningful application of sensor networks. The goal of this book is to provide an understanding of how this software functions; how it allows the sensors to gather information, process it, and interact with each other in networks; and how these networks interact with the physical world. One aim of this book is to provide fundamental information necessary to write efficient sensor network software. A second aim is to provide a balance between theory and applications, so that the subject matter is complete (self-contained).

Wireless sensor network (WSN) applications may consist of diverse sensors with varying capabilities. For example, sensors may range from an extremely constrained 8-bit "mote" to less resource-constrained 32-bit "microservers." These sensors may be organized in different network configurations, which use different communication and data dissemination protocols, most software development platforms consist of libraries that implement message-passing interprocess communication (IPC) primitives, tools to support simulation, emulation, and visualization of networked systems, and services that support networking, sensing, and time synchronization. Given all of this diversity, there is an underlying theme of software development and deployment that cuts across platforms.

1.1 SOME FOUNDATIONAL INFORMATION

This section provides some basic information necessary for understanding the sensors and sensor networks.

1.1.1 Sensors

Typically a sensor is composed of components that sense the environment, process the data, and communicate with other sensors/computers. A sensor responds to a physical stimulus, such as heat, light, sound, or pressure, and produces a measurable electrical

Fundamentals of Sensor Network Programming: Applications and Technology, By S. S. Iyengar, N. Parameshwaran, V. V. Phoha, N. Balakrishnan, and C. D. Okoye

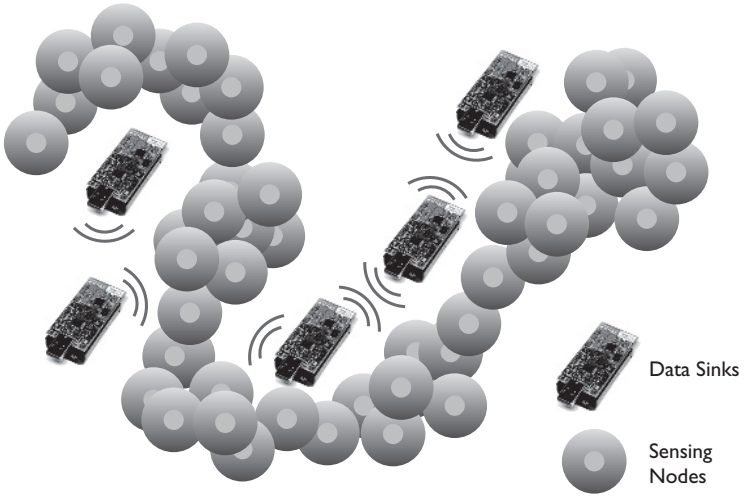

FIGURE 1.1 Networking structure of a distributed sensor network.

signal. Thus a sensor with its own sensing device, a memory, and a processor can typically be programmed with a high-level programming language, such as CorJava. The sensing devices can range from nanosensors to micro- and megasensors. In the remainder of this book when we refer to a *sensor*, we refer to a whole system such as a mote, which may have more than one physical sensor, its memory, processor, and other associated circuitry. Figure 1.1 shows a distributed sensor architecture and various components.

1.1.2 Sensor Networks

A *distributed sensor network* (DSN) is a collection of sensors distributed logically or geographically over an environment in order to collect data. Distributed computing and distributed problem solving are commonly used in DSN in order to abstract relevant information from the data gathered and derive appropriate inferences. This kind of data fusion can be used to compensate for the shortcomings of the individual sensor in real-world enviornments. For more details on sensor networks, see Refs. 1–3.

Most references to the term *sensor network* can denote multiple sensing configurations to be used in multiple contexts. Sensor networks typically consist of numerous sensing devices that may communicate over wired or wireless media, and may have as intrinsic properties limitations in computational capability, communication, or energy reserve. This does not imply that all sensor deployments consist of severely resource-constrained devices; for example, radar, closed-circuit cameras, and other wireline devices are commonly used in sensor network experimentations in academia and military research. These sensing devices possess reasonable computational capability and more importantly, may not have limited energy or constrained communication

abilities. The main crux of this book is focused on the class of sensors having severely constrained computation, communication, and energy resources. These devices range from penny to matchbox in size and are deployed in an ad hoc and nonplanned (random) fashion. Examples of such devices include the mote platforms commonly used in academia.

1.2 NEXT-GENERATION SENSOR NETWORKED TINY DEVICES

1.2.1 Domain-Specific Challenges

Development of software in wireless sensor networks draws on experiences across several domains in computer science and some engineering disciplines such as

1. *Networking.* Networking knowledge is critical in sensor networks, providing information on how large-scale mobile ad hoc wireless networks can be created and managed efficiently.
2. *Power Systems.* Sensor networks, also rely on information from computer science and electrical and nanosystems engineering, in the creation of energy efficient software and hardware components, resulting in improved life of sensor networks.
3. *Data Management.* Experience in large-scale data management and data mining techniques is required in sensor networks since huge heterogenous datastreams are generated from these ubiquitous sensing devices.
4. *Data Fusion.* Since most devices have basic sensing capabilities, the need to create software systems capable of combining data from multiple sources to create more complex representation of the world is necessary; hence the need for data fusion. Fusion systems draw on advances in artificial intelligence, statistical analysis, and distributed systems.

1.2.2 Technology-Driven Methods

A few examples of technology driven methods in sensor networks follow.

1. *Flooding*, such as broadcast of packets in a synchronized network from source to destination until the path is formed to find the topology
2. *Clustering*, including K-means clustering to find K centers and form a cluster to minimize the distance between nodes in a dense region and efficiently form a topology
3. *Short-path algorithms* for data aggregation, such as data aggregation trees to form wireless spanners to efficiently collect data periodically
4. *Distributed algorithms* for energy and reusability loading and fault tolerance in large sensor networks

1.2.3 Wireless Sensor Network Environment

Sensor Network make it possible to monitor, instruct, or control various domains such as homes, buildings, warzones, cities, and forests. Sensor networks can observe the sensing environment at a close range and thus have many advantages, such as ability to monitor smallest details, proximity to places which are difficult to reach by humans, for example difficult terrain or hazardous environment. The major limitations of sensors are their limited power supply, limited communication bandwidth and range, and limited computation ability and memory capacity. Data transmission consumes a large percentage of energy; reducing the amount of data transmitted is the primary focus of data processing. The small bandwidth of the wireless links represents a challenge for data processing. Because of the limited communication radius of a sensor node, data may have to go through multiple hops to reach the final destination. This leads to extra power consumption in sensor nodes on the relay path. Limited processing and memory capacities restrict the complexity of data processing algorithms running at the sensor nodes. The intermediate results and other data are also burdensome to store in the node because of limited memory size. Sensor data are a stream: a real-time, continuous, ordered sequence with limited control over the order in which items arrive and the limitations of low battery life, low bandwidth, and low processing power and operating memory present programming challenges that are unique to the sensor network environment.

1.3 SENSOR NETWORK SOFTWARE

A network architecture and protocols are essential foundations for building software applications.

Developing computational/communication systems for deployment and application for wireless sensor networks has been a challenge since the mid-1990s. More Specifically, wireless ad hoc sensor networks have been largely designed with static and custom architectures for specific tasks, thus providing inflexible operation and interaction capabilities. WSN applications need to be programmed with constrained memory and process-centric resource requirements in mind, in order to write communication code with real-time sensing deadlines, which are critical to a dedicated scheduled measuring task. In short, the problem is the choice of abstraction for the sensor node runtime environment. Our computational framework or paradigm called INSPIRE, defines and supports nanofootprint and real-time deadlines, scheduled tasks for computing, and allows communication and sensing resources at the sensor nodes to be efficiently harnessed in high density event driven application-sensing fashion, through the use of an object oriented framework. A key feature of the runtime abstraction is that all the infrastructure used by the kernel is simulated to provide wireless communications using renewable energy resources with its unique extended lifetime model. This allows it to scale all the code to any processor. The implementation of INSPIRE on a target prototype node occupies less than 10–40 kB

(kilobytes) of code memory; for details, refer to Chapter 10. The distributed source coding implementation is used to measure the sensing activity and memory overheads using traditional sensor applications without constraints, but more importantly, we highlight the reliability of the transmitted data from the measuring applications.

1.3.1 Technology-Driven Software

Individual updates of software are impractical because of the large number of nodes and the relative inaccessibility of deployed nodes. One solution for updating software in sensor nodes is the deployment of a support network of small, mobile, temporarily attachable nodes with virtual connections from a host PC to individual nodes. This scheme allows the use of standard tools to update the software in the individual sensor nodes. For many sensor networks in field applications, such as sensors deployed in unreachable places such as in water or trees, it is desirable to remotely update the software on the sensor nodes.

The following are a few issues to be considered when updating the nodes with software updates:

- Updates need to be planned. The items included in planning are tradeoffs of different updates relative to energy costs, the injection strategy for network configuration, and size reduction techniques that result in quick updates.
- Injection strategies of software. The strategies could include updating individual nodes, or sending updates to a base station or to a number of select nodes that may then disseminate the updates to other nodes.
- How software would be activated. Software may be either automatically activated or based on a set of rules, or manual activation may be required. To meet the requirements for backward/forward version compatibility, control over the order of node activation may be needed.
- Checking the downloaded software for integrity, version mismatch, and platform mismatch, and dynamically checking the operation of the downloaded software after it has been activated.
- Monitoring of update-related faults.
- Security-related issues, such as key distribution, authentication, secrecy, integrity, and authorization.
- Problems related to very small nodes, such as limited code memory, and almost no RAM or EEPROM (random-access or electrically erasable programmable read-only memory) for storing new code. Techniques may need to be developed for incremental building of new code into code memory (usually flash-RAM).
- Version control, that is, prevention version mismatch.
- Heterogeneity of sensor nodes. There can be various forms of heterogeneity; for example, there may be a mix of platforms, or a network may consist of a small number of “spine”/data backbone (shown to be optimal for data delivery) and a large number of lower-power nodes (for data collection). Here the backbone

nodes will have to handle different versions of their code base as well as different codebases.

- Performance. The time required to update nodes as well as tradeoffs between time and energy need to be considered.
- Provisions to recover from faulty updates, with mechanisms to verify the new software both before and during execution.

1.4 PERFORMANCE-DRIVEN NETWORK SOFTWARE PROGRAMMING

There are four basic issues here:

1. *Quality of Service.* In sensor networks quality of service is an important metric to analyze the performance and reliability of different WSN routing algorithms. As the sensor nodes use fixed batteries to sense and communicate, it is necessary to collaboratively use the network resources to minimize power usage and when idling, conserve power by using ultra-low-duty cycling. The communication module of a sensor mote uses a software "stack" and a radio to receive and transmit information. The network stack has many layers, spanning from physical layer to network layer; with various functionalities. By design, a running stack needs to use a small footprint and be power-aware, avoiding unnecessary overheads at every layer. The QoS can be defined as how the stack performs load balancing (reusability index), power-aware sleep scheduling (due to network density), and the reliability of sending sensed data wirelessly (at the datalink layer).
2. *Reusability Index.* This performance-based index can described as the number of times that a given node has been used as a clusterhead to communicate to a base station or a sink during its lifetime. As many of the clusterhead selection algorithms are distributed in nature, they will not overuse a specific node more than the critical number of times. If all nodes are used evenly, then the reliability of the network increases during the entire lifetime of the node.
3. *Sleep Scheduling.* Most of the deployed sensor network applications are dense because of the limited radio transmission range, so even when not transmitting, data nodes are subjected to overhearing and collision. These factors severely impact the total power consumed. So, in a dense deployment if a sufficient number of nodes are awake to receive the multihop traffic, then other nodes can shut off their radios after exchanging the next polling time, to minimize idling. By activating only a subset of nodes and scheduling timeslots for nodes to be active, sleep scheduling saves on power and avoids dropped packets. The end goal of each of these methods is to continue reciving data from the network for as long as possible.

4. *Datalink Reliability.* Data must be not only available but also accurate. In wireless sensor networks a node needs to not only communicate with its neighbors also forward the periodic sensed data over the network. Many of the MAC protocols are designed for efficient ad hoc communications but not for reliable data sensing as the radio does not have a way to filter floor noise or a new sensed value in harsh environments. For this reason, a twoway handshake is necessary between the MAC and the datalink layer, which allows them to reliably capture the new data everytime a data aggregation is performed. With this reliable datalink mechanism the clusterhead can further fuse the data from neighboring sensors and discard any false values.

1.4.1 Routing

In a sensor network stack the network layer is solely responsible for route planning and maintenance. Most of the energy used by the network is due to its routing activity. In implementing routing there are two methods, one at the network layer, which is controlled by distributed algorithms to form clusters and uses efficient clusterhead selection, and another at the MAC layer, which uses multihop routing to forward data at the lower layers by using best-effort QoS.

1.4.2 Data Aggregation

In a large sensor network deployment many parameters are sensed over a wide area and are periodically sent to the central coordinator. As the sensed parameters are the same at every node (similar sensor types are attached), WSN data aggregation allows reduction of the redundancy in a transmission by statistically evaluating the frequency of occurring samples and the trend direction that they have during its lifetime. When sensor nodes sample individually, only the aggregated data are transmitted, thus increasing the local processing and decreasing the radio usage per aggregation cycle. A simple example is using data compression at the nodes to send fewer bits during each transmission.

1.4.3 Security

Security is a constant threat to outdoor wireless environments; thus it is prudent to have an encryption algorithm that allows encryption and decryption of wireless communications. One novel way to implement a security algorithm is to have a oneway function which is NP-complete at the predeployment stage and cannot be decrypted with limited resources in a deployed site of operation. This method is more suitable in other applications of networks; in the case of WSN networks, because of the nature of their distribution one can design a network polynomial key that is not a local function. The broadcast message cannot be decrypted when a few nodes are compromised as it needs to have other parameters that are well distributed and concealed from the intruder.

1.5 UNIQUE CHARACTERISTICS OF PROGRAMMING ENVIRONMENTS FOR SENSOR NETWORKS

Sensor networks differ from both wired and wireless computer networks in many ways. The topology of sensor networks can change rapidly and frequently. The nodes in a sensor network do not have a global identifier such as an IP (Internet Protocol) address, and the number of sensor nodes in a sensor network may be an order of magnitude greater than that in a typical computer network. The memory and the processing capabilities of sensor nodes are limited in comparison to nodes in a computer network. These characteristics lead to a programming environment that is unique. Thus, the programs need to be short and efficient, providing capabilities of interfaces and links of components and modules to each other. Additionally, to save battery power, the nodes may need to have aggressive power management capabilities; thus the programming environment needs to provide mechanisms such as split phase, the nonblocking equivalent of common power-saving techniques such as the sleep command.

1.6 GOALS OF THE BOOK

The goals of this book are to develop programming methodologies unique to sensor networks, and present in an organized fashion techniques for programming of sensors to enable them to work effectively as a group. Thus, although the focus is on programming of the individual sensor, the goal is to enable the sensor to work within a collaborative environment.

1.7 WHY TinyOS AND NesC

TinyOS is an emerging platform that provides a framework for the most common type of sensor application programming. Thus we have a tool that can be implemented on small Crossbow sensors and a wireless sensor network that can be ported to different classrooms and laboratories. NesC provides a C-type, component-based language.

In NesC a module is the lowest level of component abstraction that implements any commands provided in its interface. It may directly address a particular hardware component such as a light sensor, providing methods that abstract the actual operation of that particular hardware component. Several modules may be grouped together using a configuration to form a larger component.

1.8 ORGANIZATION OF THE BOOK

The book is organized as follows. In Part I, we present an overview of the subject of sensor network programming, beginning with a general introduction in the remainder of this chapter (Chapter 1). Chapter 2 gives a general description of the wireless

sensors. It explains the basic components of a sensor, its sensing environment, and the various roles that a sensor can play in a wireless sensor network. Chapter 3 discusses current sensor technology, including the major families and types of sensors currently in use, including the Mica, Telos, Tmote Sky families, and others.

Part II provides a general background for sensor network (SN) programming, beginning with discussions on data structures for sensor computing programming in Chapter 4. Sensor computing programming of individual sensors in a SN environment requires an understanding of data structures, such as arrays, queues, stacks, and lists, which are essential to programming. For network implementation, an understanding of graphs is useful to appreciate routing and message passing. Thus, Chapter 4 explains those data structures, which are essential for programming in a wireless sensor network environment. Chapter 5 explains the tiny operating system (TinyOS) environment, and is essential to understanding the subsequent chapters. It presents the structure of application programming interfaces (APIs) built using a nesC like structure, which facilitates the readability of the examples given in the rest of that chapter. For the sake of completeness and continuity, Chapter 5 also includes a bare-minimum description of nesC programming language. In Chapter 6, on nesC programming, the nesC language is formally introduced and some major concepts in the language are discussed.

Part III discusses and presents examples of sensor network implementation. Chapter 7 provides a basic introduction to sensor programming. It discusses some of the challenges encountered when programming large numbers of sensors and some interfaces provided by TinyOS to alleviate these programming challenges. Chapter 8, on algorithms for wireless sensor networks, is the core and the major focus of the book. It gives detailed descriptions of various algorithms and their implementation in nesC. In Chapter 9, on techniques for protocol programming, we discuss several protocols used in most wireless sensor networks and provide accompanying pseudocode to explain the concepts.

Part IV presents real-world scenarios in sensor network programming. In Chapter 10 we discuss some programming abstractions that simplify the development and deployment of sensors. Chapter 11 presents standards for building WSN applications, with a brief overview of the ZigBee networking standard. Chapter 12 discusses an active sensor approach to distributed algorithms, widely known as INSPIRE (innovation in sensor programming implementation for real-time environments). Chapter 13 explores the performance analysis of networks in some detail with respect to power-aware algorithms. Chapter 14 describes sensor network modeling through design and simulation. This chapter presents an architecture of a sensor simulator and a sensor node that is used in the simulator, and further elaborates that OMNeT++ is a viable discrete-event simulation framework for studying both the networking aspects and the distributed computing aspects of sensor networks. We present the architecture of a sensor node that is used in the simulator and the general architecture of the simulator. Chapter 15 presents a MATLAB implementation of simple data processing and decisionmaking logic to be used to detect and respond to events in an airport baggage-handling system. Chapter 16 consists of closing comments.

1.9 FUTURE DEMANDS ON SENSOR-BASED SOFTWARE

In the future, advances in microelectromechanical systems (MEMSs) will lead to miniature sensing devices of about 20 (μm micrometers) to a millimeter in length. These devices will be self-powered, allowing even more collaboration with other devices. In regard to software, more standards specifying how data can be exported between different sensor networks will be established, allowing a more enriched and integrated sensing experience such as

- Real-time collaboration between navigation systems and traffic monitoring sensors
- Current information about seat availability at local restaurants or physicians offices
- Real-time environmental awareness by a wide range of applications and devices leading to better management of scarce resources, such as smart energy-saving homes.

In this regard, the principles addressed in this book will serve as building blocks for developing large-scale, longlived systems requiring self-organization and adaptivity.

PROBLEMS

1.1 Define the following:
(a) Sensor
(b) Ad hoc network
(c) Distributed sensor network
(d) Wireless sensor network
(e) Reusability index

1.2 Discuss some of the design challenges that set wireless sensor networks apart from conventional networks.

1.3 Crossbow Technology Inc.'s MTS400 multisensor board is one of the most popular multipurpose heterogeneous sensing devices available on the market. Research and prepare a two-page report discussing the specifications and functionality of the MTS400 multisensor board.

1.4 Write a one-page summary of the article by Akyildiz et al. [4].

1.5 Other than those discussed in this introductory chapter, list three advantages and three disadvantages of sensor networks.

1.6 Crossbow Technology Inc.'s MICAz mote and Europe's Smart-Its platform are two popular sensor platforms. Research and contrast the features of MICAz with smart-Its (in terms of size, weight, battery life, onboard sensors, memory, CPU, operating system, processing limits, radio range, etc.).

1.7 What are the unique characteristics of programming environments for sensor network software?

1.8 Other than sleep scheduling, give two techniques to conserve the battery life (energy) of nodes in a sensor network.

1.9 State the issues to be considered when updating computation/communication software in a sensor network.

1.10 Does the data aggregation strategy adopted by a sensor network application affect its operational integrity and security? If "Yes," explain how and if "No," explain why.

1.11 In about five paragraphs discuss any three of your favorite real-world sensor network applications.

1.12 Sensors mounted on moving objects can open many interesting real-world applications. For example, sensors are already mounted on devices such as mobile phones to sense temperature, motion, and other parameters. Suggest some applications where mounting sensors on mobile objects will be useful.

1.13 Interesting scenarios are created when sensors are made much smaller in size and are programmed to become more autonomous. "Smart dust" refers to tiny devices that are capable of limited sensing, computations, and commmications capabilities, with short lifetime. Suggest some applications where one or many "bags" of smart dust can be used.

1.14 When wireless sensors become tiny and are deployed in very large number (such as in several bags of smart dust), interestingly, the overall behavior of such a system will in some way behave like social systems, exhibiting autonomy, self control, limited lifetime, and intracommunications. Identify the management challenges in such a social system. Consider an example application, and propose specific management solutions appropriate for this application.

1.15 Traditional programming views programs as a mapping from input values to output values. Suggest some characteristics of programs written for the wireless devices. [*Hint:* A sensor program spends a considerable amount of time in communication, and thus must be sufficiently ingenious to manage its resources (such as data and power), and cooperate with other sensors to achieve the overall behavior as required by the application.]

1.16 In a computer network, each node communicates with the other nodes using a set of protocols. What will be the limitations of this model when applied to wireless sensor networks? Suppose that we augment the protocols with flexible dialog features where each sensor node engages "intelligently" with the other sensor nodes. What will be the advantages? Discuss the resulting overhead.

1.17 Suppose that we view a WSN as a multiagent system (MAS). Suggest an application where this view will be appropriate. What will be the undesirable aspects inherent in such a MAS model?

1.18 Traditional network systems are designed to satisfy strict specifications. Because of the dynamic and uncertain environments in which WSN is employed, traditional approaches may not be appropriate. Investigate why this may be the case. If self-autonomy is one possible solution, how can it help the sensor network in satisfying the application requierements? What additional complications will this solution create for the application?

1.19 Indentify some aspects of security issues that are unique to WSN but may not be present in the traditional computer network systems.

REFERENCES

1. R. R. Brooks and S .S. Iyengar, *Multi-Sensor Fusion,* Prentice-Hall, Englewood Cliffs, NJ 1997.
2. K. Chakrabarty and S. S. Iyengar, *Scalable Infrastructure for Distributed Sensor Networks,* Springer-Verlag, 2005.
3. S. S. Iyengar and R. R. Brooks, eds., *Distributed Sensor Networks,* CRC Press, Dec. 2004.
4. I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, A survey of sensor networks, *IEEE Commun. Mag.* **40**(8):102–114 (Aug. 2002).