Sri Chandrasekharendra Saraswathi Viswa MahaVidyalaya

(Deemed to be University)

Department of Electronics and Communication Engineering

Lecture Notes

on

Wireless Sensor Networks

(Program Elective – III)

Dr. G. Senthil Kumar

Associate Professor, ECE Department

SCSVMV Deemed University

Email: gsk_ece@kanchiuniv.ac.in

Disclaimer

ESSENTIAL READING BOOKS :

1. Holger Karl & Andreas Willig, "Protocols And Architectures for Wireless Sensor Networks", John Wiley, 2005.
2. Feng Zhao & Leonidas J.Guibas, “Wireless Sensor Networks- An Information Processing Approach", Elsevier, 2007.
3. Waltenegus Dargie , Christian Poellabauer, “Fundamentals of Wireless Sensor Networks - Theory and Practice”, John Wiley & Sons Publications, 2011

RECOMMONDED READING BOOKS :

1. Kazem Sohraby, Daniel Minoli, & Taieb Znati, “Wireless Sensor Networks-Technology, Protocols, and Applications”, John Wiley, 2007.
2. Anna Hac, “Wireless Sensor Network Designs”, John Wiley, 2003

WEB LINKS FOR REFERENCE

1. https://nptel.ac.in/courses/106/105/106105160/
2. https://onlinecourses.swayam2.ac.in/arp19_ap52/preview
3. https://cse.iitkgp.ac.in/~smisra/course/wasn.html

Wireless Sensor Networks (Program Elective – III)

PROGRAMME : B.E. BRANCH : Electronics and Communication Engineering

Semester	Subject Code	Subject Name	Total Contact Hours	Weekly Hours			Credit
				L	T	P	
VII	**BECF187EH0**	**Wireless Sensor Networks (Program Elective – III)**	**50**	**3**	**0**	**0**	**3**

(For Students admitted from 2018 onwards)

COURSE OBJECTIVES

- To make students understand the basics of Wireless sensor Networks.
- To familiarize with learning of the Architecture of WSN.
- To understand the concepts of Networking and Networking in WSN.
- To study the design consideration of topology control and solution to the various problems.
- To introduce the hardware and software platforms and tool in WSN.

Prerequisite:

- Basic knowledge of Data Communication Networks.

COURSE OUTCOMES

At the end of the course, the students will be able to:

CO No.	**Course Outcomes**	**Bloom's Taxonomy (Knowledge/Comprehension/ Application/ Analysis/ Synthesis/Evaluation)**
CO1	Understand challenges and technologies for wireless networks	Knowledge / Comprehension / Application / Analysis
CO2	Understand architecture and sensors	Knowledge / Comprehension / Application / Analysis
CO3	Describe the communication, energy efficiency, computing, storage and transmission	Knowledge / Comprehension/ Application / Analysis / Synthesis / Evaluation
CO4	Establishing infrastructure and simulations	Knowledge / Comprehension / Application / Analysis
CO5	Explain the concept of programming the in WSN environment	Knowledge / Comprehension / Application / Analysis

Mapping with Program Outcome

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	S	S	S	M	L	M					S	S
CO2	S	S	S	M	L	M					S	S
CO3	S	S	S	M	L	M					S	S
CO4	S	S	S	M	L	M					S	S
CO5	S	S	S	M	L	M					S	S

S – Strong, M – Medium, L - Low

Assessment Pattern

Bloom's Category	Continuous Assessment Tests		Terminal Examination (100 marks)
	Ist Internal (30 marks)	IInd Internal (30 marks)	
Knowledge	10	8	30
Comprehension	8	8	30
Application	4	6	15
Analysis	4	4	15
Synthesis	4	2	6
Evaluation	-	2	4

SYLLABUS: WIRELESS SENSOR NETWORKS (Program Elective – III)

UNIT – I: OVERVIEW OF WIRELESS SENSOR NETWORKS [9 Hours]

Single-Node Architecture - Hardware Components - Network Characteristics- unique constraints and challenges, Enabling Technologies for Wireless Sensor Networks - Types of wireless sensor networks.

UNIT – II: ARCHITECTURES [9 Hours]

Network Architecture - Sensor Networks-Scenarios - Design Principle, Physical Layer and Transceiver Design Considerations, Optimization Goals and Figures of Merit, Gateway Concepts, Operating Systems and Execution Environments - introduction to Tiny OS and nesC - Internet to WSN Communication.

UNIT – III: NETWORKING SENSORS [10 Hours]

MAC Protocols for Wireless Sensor Networks, Low Duty Cycle Protocols And Wakeup Concepts – SMAC, - B-MAC Protocol, IEEE 802.15.4 standard and ZigBee, the Mediation Device Protocol, Wakeup Radio Concepts, Address and Name Management, Assignment of MAC Addresses, Routing Protocols Energy-Efficient Routing, Geographic Routing.

UNIT – IV: INFRASTRUCTURE ESTABLISHMENT [8 Hours]

Topology Control, Clustering, Time Synchronization, Localization and Positioning, Sensor Tasking and Control.

UNIT – V: SENSOR NETWORK PLATFORMS AND TOOLS [9 Hours]

Sensor Node Hardware – Berkeley Motes, Programming Challenges, Node-level software platforms, Node- level Simulators, State-centric programming.

ESSENTIAL READING BOOKS :

1. Holger Karl & Andreas Willig, "Protocols And Architectures for Wireless Sensor Networks", John Wiley, 2005.

2. Feng Zhao & Leonidas J.Guibas, “Wireless Sensor Networks- An Information Processing Approach", Elsevier, 2007.

3. Waltenegus Dargie , Christian Poellabauer, “Fundamentals of Wireless Sensor Networks - Theory and Practice”, John Wiley & Sons Publications, 2011

RECOMMONDED READING BOOKS :

1. Kazem Sohraby, Daniel Minoli, & Taieb Znati, “Wireless Sensor Networks-Technology, Protocols, and Applications”, John Wiley, 2007.

2. Anna Hac, “Wireless Sensor Network Designs”, John Wiley, 2003

WEB LINKS FOR REFERENCE

1. https://nptel.ac.in/courses/106/105/106105160/

2.https://onlinecourses.swayam2.ac.in/arp19_ap52/preview

3. https://cse.iitkgp.ac.in/~smisra/course/wasn.html

Notes on
Overview of Wireless Sensor Networks
Unit I

Dr. G. Senthil Kumar,
Associate Professor,
Dept. of ECE, SCSVMV,
email: gsk_ece@kanchiuniv.ac.in

==

OBJECTIVES:

In this lesson, you will be introduced for the types of applications for which wireless sensor networks are intended and a first intuition about the types of technical solutions that are required, both in hardware and in networking technologies. Also, able to understand the capabilities and limitations of the nodes in a sensor network and principal options on how individual sensor nodes can be connected into a wireless sensor network.

CONTENTS:

1. INTRODUCTION

Applications should shape and form the technology for which they are intended. This holds true in particular for wireless sensor networks, which have, to some degree, been a technology-driven development. The first part of this chapter starts out by putting the idea of wireless sensor networks into a broader perspective and gives a number of application scenarios, which motivate particular technical needs. It also generalizes from specific examples to types or classes of applications. Then, the specific challenges for these application types are discussed and why current technology is not up to meeting these challenges.

The second part of this chapter explains the basic part of a wireless sensor network: the nodes as such. It discusses the principal tasks of a node – computation, storage, communication, and sensing/actuation – and which components are required to perform these tasks. Then, the energy consumption of these components is described: how energy can be stored, gathered from the environment, and saved by intelligently controlling the mode of operation of node components. Finally, some examples of sensor nodes are given.

Vision of Ambient Intelligence

The most common form of information processing has happened on large, general-purpose computational devices, ranging from old-fashioned mainframes to modern laptops or palmtops. In many applications, like office applications, these computational devices are mostly used to process information that is at its core centered around a human user of a system, but is at best indirectly related to the physical environment.

In another class of applications, the physical environment is at the focus of attention. Computation is used to exert control over physical processes, for example, when controlling chemical processes in a factory for correct temperature and pressure. Here, the computation is integrated with the control; it is embedded into a physical system. Unlike the former class of systems, such embedded systems are usually not based on human interaction but are rather required to work without it; they are intimately tied to their control task in the context of a larger system. Such embedded systems are a well-known and long-used concept in the engineering sciences (in fact, estimates say that up to 98% of all computing devices are used in an embedded context. Their impact on everyday life is also continuing to grow at a quick pace. Rare is the household where embedded computation is not present to control a washing machine, a video player, or a cell phone. In such applications, embedded systems meet human-interaction-based systems. Technological progress is about to take this spreading of embedded control in our daily lives a step further. There is a tendency not only to equip larger objects like a washing machine with embedded computation and control, but also smaller, even dispensable goods like groceries; in addition, living and working spaces themselves can be endowed with such capabilities. Eventually, computation will surround us in our daily lives, realizing a vision of "Ambient Intelligence" where many different devices will gather and process information from many different sources to both control physical processes and to interact with human users.

To realize this vision, a crucial aspect is needed in addition to computation and control: communication. All these sources of information have to be able to transfer the information to the place where it is needed – an actuator or a user – and they should collaborate in providing as precise a picture of the real world as is required. For some application scenarios, such networks of sensors and actuators are easily built using existing, wired networking technologies. For many other application types, however, the need to wire together all these entities constitutes a considerable obstacle to success: wires constitute a maintenance problem; wires prevent entities from being mobile; and wires can prevent sensors or actuators from being close to the phenomenon that they are supposed to control. Hence, wireless communication between such devices is, in many application scenarios, an inevitable requirement. Therefore, a new class of networks has appeared in the last few years: the so-called Wireless Sensor Network (WSN). These networks consist of individual nodes that areable to interact with their environment by sensing or controlling physical parameters; these nodes have to collaborate to fulfill their tasks as, usually, a single node is incapable of doing so; and they use wireless communication to enable this collaboration. In essence, the nodes without such a network contain at least some computation, wireless communication, and sensing or control functionalities. Despite the fact that these networks also often include actuators, the term wireless sensor network has become the commonly accepted name. Sometimes, other names like "wireless sensor and actuator networks" are also found.

These WSNs are powerful in that they are amenable to support a lot of very different real-world applications; they are also a challenging research and engineering problem because of this very flexibility. Accordingly, there is no single set of requirements that clearly classifies all WSNs, and there is also not a single technical solution that encompasses the entire design space. For example, in many WSN applications, individual nodes in the network cannot easily be connected to a wired power supply but rather have to rely on onboard batteries. In such an application, the energy efficiency of any proposed solution is hence a very important figure of merit as a long operation time is usually desirable. In other applications, power supply might not be an issue and hence other metrics, for example, the accuracy of the delivered results, can become more important. Also, the acceptable size and costs of an individual node can be relevant in many applications. Closely tied to the size is often the capacity of an onboard battery; the price often has a direct bearing on the quality of the node's sensors, influencing the accuracy of the result that can be obtained from a single node. Moreover, the number, price, and potentially low accuracy of individual nodes is relevant when comparing a distributed system of many sensor nodes to a more centralized version with fewer, more expensive nodes of higher accuracy. Simpler but numerous sensors that are close to the phenomenon under study can make the architecture of a system both simpler and more energy efficient as they facilitate distributed sampling – detecting objects, for example, requires a distributed system.

Realizing such wireless sensor networks is a crucial step toward a deeply penetrating Ambient Intelligence concept as they provide, figuratively, the "last 100 meters" of pervasive control. To realize them, a better understanding of their potential applications and the ensuing requirements is necessary, as is an idea of the enabling technologies. These questions are answered in the following sections; a juxtaposition of wireless sensor networks and related networking concepts such as fieldbuses or mobile ad hoc network is provided as well.

2. TYPES OF WIRELESS SENSOR NETWORKS (Through Application Examples)

The claim of wireless sensor network proponents is that this technological vision will facilitate many existing application areas and bring into existence entirely new ones. This claim depends on many factors, but a couple of the envisioned application scenarios shall be highlighted. Apart from the need to build cheap, simple to program and network, potentially long-lasting sensor nodes, a crucial and primary ingredient for developing actual applications is the actual sensing and actuating faculties with which a sensor node can be endowed. For many physical parameters, appropriate sensor technology exists that can be integrated in a node of a WSN. Some of the few popular ones are temperature, humidity, visual and infrared light (from simple luminance to cameras), acoustic, vibration (e.g. for detecting seismic disturbances), pressure, chemical sensors (for gases of different types or to judge soil composition), mechanical stress, magnetic sensors (to detect passing vehicles), potentially even radar. But even more sophisticated sensing capabilities are conceivable, for example, toys in a kindergarten might have tactile or motion sensors or be able to determine their own speed or location.

Actuators controlled by a node of a wireless sensor network are perhaps not quite as multifaceted. Typically, they control a mechanical device like a servo drive, or they might switch some electrical appliance by means of an electrical relay, like a lamp, a bullhorn, or a similar device.

On the basis of nodes that have such sensing and/or actuation faculties, in combination with computation and communication abilities, many different kinds of applications can be constructed, with very different types of nodes, even of different kinds within one application. A brief list of scenarios should make the vast design space and the very different requirements of various applications evident.

Disaster relief applications One of the most often mentioned application types for WSN are disaster relief operations. A typical scenario is wildfire detection: Sensor nodes are equipped with thermometers and can determine their own location (relative to each other or in absolute coordinates). These sensors are deployed over a wildfire, for example, a forest, from an airplane. They collectively produce a “temperature map” of the area or determine the perimeter of areas with high temperature that can be accessed from the outside, for example, by fire fighters equipped with Personal Digital Assistants (PDAs). Similar scenarios are possible for the control of accidents in chemical factories, for example. Some of these disaster relief applications have commonalities with military applications, where sensors should detect, for example, enemy troops rather than wildfires. In such an application, sensors should be cheap enough to be considered disposable since a large number is necessary; lifetime requirements are not particularly high.

Environment control and biodiversity mapping WSNs can be used to control the environment, for example, with respect to chemical pollutants – a possible application is garbage dump sites. Another example is the surveillance of the marine ground floor; an understanding of its erosion processes is important for the construction of offshore wind farms. Closely related to environmental control is the use of WSNs to gain an understanding of the number of plant and animal species that live in a given habitat (biodiversity mapping).

The main advantages of WSNs here are the long-term, unattended, wirefree operation of sensors close to the objects that have to be observed; since sensors can be made small enough to be unobtrusive, they only negligibly disturb the observed animals and plants. Often, a large number of sensors is required with rather high requirements regarding lifetime.

Intelligent buildings Buildings waste vast amounts of energy by inefficient Humidity, Ventilation Air Conditioning (HVAC) usage. A better, real-time, high-resolution monitoring of temperature, airflow, humidity, and other physical parameters in a building by means of a WSN can considerably increase the comfort level of inhabitants and reduce the energy consumption (potential savings of two quadrillion British Thermal Units in the US alone have been speculated. Improved energy efficiency as well as improved convenience are some goals of "intelligent buildings", for which currently wired systems like BACnet, LonWorks, or KNX are under development or are already deployed; these standards also include the development of wireless components or have already incorporated them in the standard. In addition, such sensor nodes can be used to monitor mechanical stress levels of buildings in seismically active zones. By measuring mechanical parameters like the bending load of girders, it is possible to quickly ascertain via a WSN whether it is still safe to enter a given building after an earthquake or whether the building is on the brink of collapse – a considerable advantage for rescue personnel. Similar systems can be applied to bridges. Other types of sensors might be geared toward detecting people enclosed in a collapsed building and communicating such information to a rescue team.

The main advantage here is the collaborative mapping of physical parameters. Depending on the particular application, sensors can be retrofitted into existing buildings (for HVACtype applications) or have to be incorporated into the building already under construction. If power supply is not available, lifetime requirements can be very high – up to several dozens of years – but the number of required nodes, and hence the cost, is relatively modest, given the costs of an entire building.

Facility management In the management of facilities larger than a single building, WSNs also have a wide range of possible applications. Simple examples include keyless entry applications where people wear badges that allow a WSN to check which person is allowed to enter which areas of a larger company site. This example can be extended to the detection of intruders, for example of vehicles that pass a street outside of normal business hours. A widearea WSN could track such a vehicle's position and alert security personnel – this application shares many commonalities with corresponding military applications. Along another line, a WSN could be used in a chemical plant to scan for leaking chemicals. These applications combine challenging requirements as the required number of sensors can be large, they have to collaborate (e.g. in the tracking example), and they should be able to operate a long time on batteries.

Machine surveillance and preventive maintenance One idea is to fix sensor nodes to difficult to reach areas of machinery where they can detect vibration patterns that indicate the need for maintenance. Examples for such machinery could be robotics or the axles of trains. Other applications in manufacturing are easily conceivable.

The main advantage of WSNs here is the cable free operation, avoiding a maintenance problem in itself and allowing a cheap, often retrofitted installation of such sensors. Wired power supply may or may not be available depending on the scenario; if it is not available, sensors should last a long time on a finite supply of energy since exchanging batteries is usually impractical and costly. On the other hand, the size of nodes is often not a crucial issue, nor is the price very heavily constrained.

Precision agriculture Applying WSN to agriculture allows precise irrigation and fertilizing by placing humidity soil composition sensors into the fields. A relatively small number is claimed to be sufficient, about one sensor per 100 m × 100 m area. Similarly, pest control can profit from a high-resolution surveillance of farm land. Also, livestock breeding can benefit from attaching a sensor to each pig or cow, which controls the health status of the animal (by checking body temperature, step counting, or similar means) and raises alarms if given thresholds are exceeded.

Medicine and health-care Along somewhat similar lines, the use of WSN in health care applications is a potentially very beneficial, but also ethically controversial, application. Possibilities range from postoperative and intensive care, where sensors are directly attached to patients – the advantage of doing away with cables is considerable here – to the long-term surveillance of (typically elderly) patients and to automatic drug administration (embedding sensors into drug packaging, raising alarms when applied to the wrong patient, is conceivable). Also, patient and doctor tracking systems within hospitals can be literally life saving.

Logistics In several different logistics applications, it is conceivable to equip goods (individual parcels, for example) with simple sensors that allow a simple tracking of these objects during transportation or facilitate inventory tracking in stores or warehouses.

In these applications, there is often no need for a sensor node to actively communicate; passive readout of data is often sufficient, for example, when a suitcase is moved around on conveyor belts in an airport and passes certain checkpoints. Such passive readout is much simpler and cheaper than the active communication and information processing concept discussed in the other examples; it is realized by so-called Radio Frequency Identifier (RF ID) tags.

On the other hand, a simple RFID tag cannot support more advanced applications. It is very difficult to imagine how a passive system can be used to locate an item in a warehouse; it can also not easily store information about the history of its attached object – questions like "where has this parcel been?" are interesting in many applications but require some active participation of the sensor node.

Telematics Partially related to logistics applications are applications for the telematics context, where sensors embedded in the streets or roadsides can gather information about traffic conditions at a much finer grained resolution than what is possible today. Such a socalled "intelligent roadside" could also interact with the cars to exchange danger warnings about road conditions or traffic jams ahead.

In addition to these, other application types for WSNs that have been mentioned in the literature include airplane wings and support for smart spaces, applications in waste water treatment plants, instrumentation of semiconductor processing chambers and wind tunnels, in "smart kindergartens" where toys interact with children, the detection of floods, interactive museums, monitoring a bird habitat on a remote island, and implanting sensors into the human body (for glucose monitoring or as retina prosthesis).

While most of these applications are, in some form or another, possible even with today's technologies and without wireless sensor networks, all current solutions are "sensor starved". Most applications would work much better with information at higher spatial and temporal resolution about their object of concern than can be provided with traditional sensor technology. Wireless sensor networks are to a large extent about providing the required information at the required accuracy in time with as little resource consumption as possible.

TYPES OF APPLICATIONS *(Cont. - Types of wireless sensor networks through)*

Many of these applications share some basic characteristics. In most of them, there is a clear difference between sources of data – the actual nodes that sense data – and sinks – nodes where the data should be delivered to. These sinks sometimes are part of the sensor network itself; sometimes they are clearly systems "outside" the network (e.g. the firefighter's PDA communicating with a WSN). Also, there are usually, but not always, more sources than sinks and the sink is oblivious or not interested in the identity of the sources; the data itself is much more important.

The interaction patterns between sources and sinks show some typical patterns. The most relevant ones are:

Event detection Sensor nodes should report to the sink(s) once they have detected the occurrence of a specified event. The simplest events can be detected locally by a single sensor node in isolation (e.g. a temperature threshold is exceeded); more complicated types of events require the collaboration of nearby or even remote sensors to decide whether a (composite) event has occurred (e.g. a temperature gradient becomes too steep). If several different events can occur, event classification might be an additional issue.

Periodic measurements Sensors can be tasked with periodically reporting measured values. Often, these reports can be triggered by a detected event; the reporting period is application dependent.

Function approximation and edge detection The way a physical value like temperature changes from one place to another can be regarded as a function of location. A WSN can be used to approximate this unknown function (to extract its spatial characteristics), using a limited number of samples taken at each individual sensor node. This approximate mapping should be made available at the sink. How and when to update this mapping depends on the application's needs, as do the approximation accuracy and the inherent trade-off against energy consumption.

Similarly, a relevant problem can be to find areas or points of the same given value. An example is to find the isothermal points in a forest fire application to detect the border of the actual fire. This can be generalized to finding "edges" in such functions or to sending messages along the boundaries of patterns in both space and/or time.

Tracking The source of an event can be mobile (e.g. an intruder in surveillance scenarios). The WSN can be used to report updates on the event source's position to the sink(s), potentially with estimates about speed and direction as well. To do so, typically sensor nodes have to cooperate before updates can be reported to the sink. These interactions can be scoped both in time and in space (reporting events only within a given time span, only from certain areas, and so on). These requirements can also change dynamically overtime; sinks have to have a means to inform the sensors of their requirements at runtime. Moreover, these interactions can take place only for one specific request of a sink (so-called "one-shot queries"), or they could be long-lasting relationships between many sensors and many sinks.

The examples also have shown a wide diversity in deployment options. They range from well planned, fixed deployment of sensor nodes (e.g. in machinery maintenance applications) to random deployment by dropping a large number of nodes from an aircraft over a forest fire. In addition, sensor nodes can be mobile themselves and compensate for shortcomings in the deployment process by moving, in a post deployment phase, to positions such that their sensing tasks can be better fulfilled. They could also be mobile because they are attached to other objects (in the logistics applications, for example) and the network has to adapt itself to the location of nodes.

Closely related to the maintenance options are the options for energy supply. In some applications, wired power supply is possible and the question is mute. For self-sustained sensor nodes, depending on the required mission time, energy supply can be trivial (applications with a few days of usage only) or a challenging research problem, especially when no maintenance is possible but nodes have to work for years. Obviously, acceptable price and size per node play a crucial role in designing energy supply.

3A. NETWORK CHARACTERISTICS - UNIQUE CONSTRAINTS AND CHALLENGES FOR WSNS

Handling such a wide range of application types will hardly be possible with any single realization of a WSN. Nonetheless, certain common traits appear, especially with respect to the characteristics and the required mechanisms of such systems. Realizing these characteristics with new mechanisms is the major challenge of the vision of wireless sensor networks.

CHARACTERISTIC REQUIREMENTS

The following characteristics are shared among most of the application examples discussed above:

Type of service The service type rendered by a conventional communication network is evident – it moves bits from one place to another. For a WSN, moving bits is only a means to an end, but not the actual purpose. Rather, a WSN is expected to provide meaningful information and/or actions about a given task: "People want answers, not numbers". Additionally, concepts like scoping of interactions to specific geographic regions or to time intervals will become important. Hence, new paradigms of using such a network are required, along with new interfaces and new ways of thinking about the service of a network.

Quality of Service Closely related to the type of a network's service is the quality of that service. Traditional quality of service requirements – usually coming from multimedia-type applications – like bounded delay or minimum bandwidth are irrelevant when applications are tolerant to latency or the bandwidth of the transmitted data is very small in the first place. In some cases, only occasional delivery of a packet can be more than enough; in other cases, very high reliability requirements exist. In yet other cases, delay is important when actuators are to be controlled in a real-time fashion by the sensor network. The packet delivery ratio is an insufficient metric; what is relevant is the amount and quality of information that can be extracted at given sinks about the observed objects or area. Therefore, adapted quality concepts like reliable detection of events or the approximation quality of a, say, temperature map is important.

Fault tolerance Since nodes may run out of energy or might be damaged, or since the wireless communication between two nodes can be permanently interrupted, it is important that the WSN as a whole is able to tolerate such faults. To tolerate node failure, redundant deployment is necessary, using more nodes than would be strictly necessary if all nodes functioned correctly.

Lifetime In many scenarios, nodes will have to rely on a limited supply of energy (using batteries). Replacing these energy sources in the field is usually not practicable, and simultaneously, a WSN must operate at least for a given mission time or as long as possible. Hence, the lifetime of a WSN becomes a very important figure of merit. Evidently, an energy-efficient way of operation of the WSN is necessary.

As an alternative or supplement to energy supplies, a limited power source (via power sources like solar cells, for example) might also be available on a sensor node. Typically, these sources are not powerful enough to ensure continuous operation but can provide some recharging of batteries. Under such conditions, the lifetime of the network should ideally be infinite. The lifetime of a network also has direct trade-offs against quality of service: investing more energy can increase quality but decrease lifetime. Concepts to harmonize these trade-offs are required.

The precise definition of lifetime depends on the application at hand. A simple option is to use the time until the first node fails (or runs out of energy) as the network lifetime. Other options include the time until the network is disconnected in two or more partitions, the time until 50% (or some other fixed ratio) of nodes have failed, or the time when for the first time a point in the observed region is no longer covered by at least a single sensor node (when using redundant deployment, it is possible and beneficial to have each point in space covered by several sensor nodes initially).

Scalability Since a WSN might include a large number of nodes, the employed architectures and protocols must be able scale to these numbers.

Wide range of densities In a WSN, the number of nodes per unit area – the density of the network – can vary considerably. Different applications will have very different node densities. Even within a given application, density can vary over time and space because nodes fail or move; the density also does not have to homogeneous in the entire network (because of imperfect deployment, for example) and the network should adapt to such variations.

Programmability Not only will it be necessary for the nodes to process information, but also they will have to react flexibly on changes in their tasks. These nodes should be programmable, and their programming must be changeable during operation when new tasks become important. A fixed way of information processing is insufficient.

Maintainability As both the environment of a WSN and the WSN itself change (depleted batteries, failing nodes, new tasks), the system has to adapt. It has to monitor its own health and status to change operational parameters or to choose different trade-offs (e.g. to provide lower quality when energy resource become scarce). In this sense, the network has to maintain itself; it could also be able to interact with external maintenance mechanisms to ensure its extended operation at a required quality.

REQUIRED MECHANISMS

To realize these requirements, innovative mechanisms for a communication network have to be found, as well as new architectures, and protocol concepts. A particular challenge here is the need to find mechanisms that are sufficiently specific to the idiosyncrasies of a given application to support the specific quality of service, lifetime, and maintainability requirements. On the other hand, these mechanisms also have to generalize to a wider range of applications lest a complete from-scratch development and implementation of a WSN becomes necessary for every individual application – this would likely render WSNs as a technological concept economically infeasible.

Some of the mechanisms that will form typical parts of WSNs are:

Multihop wireless communication While wireless communication will be a core technique, a direct communication between a sender and a receiver is faced with limitations. In particular, communication over long distances is only possible using prohibitively high transmission power. The use of intermediate nodes as relays can reduce the total required power. Hence, for many forms of WSNs, so-called multihop communication will be a necessary ingredient.

Energy-efficient operation To support long lifetimes, energy-efficient operation is a key technique. Options to look into include energy-efficient data transport between two nodes (measured in J/bit) or, more importantly, the energy-efficient determination of a requested information. Also, nonhomogeneous energy consumption – the forming of "hotspots" – is an issue.

Auto-configuration A WSN will have to configure most of its operational parameters autonomously, independent of external configuration – the sheer number of nodes and simplified deployment will require that capability in most applications. As an example, nodes should be able to determine their geographical positions only using other nodes of the network – socalled "self-location". Also, the network should be able to tolerate failing nodes (because of a depleted battery, for example) or to integrate new nodes (because of incremental deployment after failure, for example).

Collaboration and In-Network processing In some applications, a single sensor is not able to decide whether an event has happened but several sensors have to collaborate to detect an event and only the joint data of many sensors provides enough information. Information is processed in the network itself in various forms to achieve this collaboration, as opposed to having every node transmit all data to an external network and process it "at the edge" of the network.

An example is to determine the highest or the average temperature within an area and to report that value to a sink. To solve such tasks efficiently, readings from individual sensors can be aggregated as they propagate through the network, reducing the amount of data to be transmitted and hence improving the energy efficiency. How to perform such aggregation is an open question.

Data centric Traditional communication networks are typically centered around the transfer of data between two specific devices, each equipped with (at least) one network address – the operation of such networks is thus address-centric. In a WSN, where nodes are typically deployed redundantly to protect against node failures or to compensate for the low quality of a single node's actual sensing equipment, the identity of the particular node supplying data becomes irrelevant. What is important are the answers and values themselves, not which node has provided them. Hence, switching from an address-centric paradigm to a data-centric paradigm in designing architecture and communication protocols is promising.

An example for such a data-centric interaction would be to request the average temperature in a given location area, as opposed to requiring temperature readings from individual nodes. Such a data-centric paradigm can also be used to set conditions for alerts or events ("raise an alarm if temperature exceeds a threshold"). In this sense, the data-centric approach is closely related to query concepts known from databases; it also combines well with collaboration, in-network processing, and aggregation.

Locality Rather a design guideline than a proper mechanism, the principle of locality will have to be embraced extensively to ensure, in particular, scalability. Nodes, which are very limited in resources like memory, should attempt to limit the state that they accumulate during protocol processing to only information about their direct neighbours. The hope is that this will allow the network to scale to large numbers of nodes without having to rely on powerful processing at each single node. How to combine the locality principle with efficient protocol designs is still an open research topic, however.

Exploit trade-offs Similar to the locality principle, WSNs will have to rely to a large degree on exploiting various inherent trade-offs between mutually contradictory goals, both during system/protocol design and at runtime. Examples for such trade-offs have been mentioned already: higher energy expenditure allows higher result accuracy, or a longer lifetime of the entire network trades off against lifetime of individual nodes. Another important trade-off is node density: depending on application, deployment, and node failures at runtime, the density of the network can change considerably – the protocols will have to handle very different situations, possibly present at different places of a single network. Again, not all the research questions are solved here.

Harnessing these mechanisms such that they are easy to use, yet sufficiently general, for an application programmer is a major challenge. Departing from an address-centric view of the network requires new programming interfaces that go beyond the simple semantics of the conventional socket interface and allow concepts like required accuracy, energy/accuracy trade-offs, or scoping.

3B. ENABLING TECHNOLOGIES FOR WIRELESS SENSOR NETWORKS

Building such wireless sensor networks has only become possible with some fundamental advances in enabling technologies. First and foremost among these technologies is the miniaturization of hardware. Smaller feature sizes in chips have driven down the power consumption of the basic components of a sensor node to a level that the constructions of WSNs can be contemplated. This is particularly relevant to microcontrollers and memory chips as such, but also, the radio modems, responsible for wireless communication, have become much more energy efficient. Reduced chip size and improved energy efficiency is accompanied by reduced cost, which is necessary to make redundant deployment of nodes affordable.

Next to processing and communication, the actual sensing equipment is the third relevant technology.

These three basic parts of a sensor node have to accompanied by power supply. This requires, depending on application, high capacity batteries that last for long times, that is, have only a negligible self-discharge rate, and that can efficiently provide small amounts of current. Ideally, a sensor node also has a device for energy scavenging, recharging the battery with energy gathered from the environment – solar cells or vibration-based power generation are conceivable options. Such a concept requires the battery to be efficiently chargeable with small amounts of current, which is not a standard ability. Both batteries and energy scavenging are still objects of ongoing research.

The counterpart to the basic hardware technologies is software. The first question to answer here is the principal division of tasks and functionalities in a single node – the architecture of the operating system or runtime environment. This environment has to support simple retasking, cross-layer information exchange, and modularity to allow for simple maintenance. This software architecture on a single node has to be extended to a network architecture, where the division of tasks between nodes, not only on a single node, becomes the relevant question – for example, how to structure interfaces for application programmers. The third part to solve then is the question of how to design appropriate communication protocols.

4. SINGLE NODE ARCHITECTURE

Building a wireless sensor network first of all requires the constituting nodes to be developed and available. These nodes have to meet the requirements that come from the specific requirements of a given application: they might have to be small, cheap, or energy efficient, they have to be equipped with the right sensors, the necessary computation and memory resources, and they need adequate communication facilities.

HARDWARE COMPONENTS

Sensor node hardware overview

When choosing the hardware components for a wireless sensor node, evidently the application's requirements play a decisive factor with regard mostly to size, costs, and energy consumption of the nodes – communication and computation facilities as such are often considered to be of acceptable quality, but the trade-offs between features and costs is crucial. In some extreme cases, an entire sensor node should be smaller than 1 cc, weigh (considerably) less than 100 g, be substantially cheaper than US$1, and dissipate less than 100 μW. In even more extreme visions, the nodes are sometimes claimed to have to be reduced to the size of grains of dust. In more realistic applications, the mere size of a node is not so important; rather, convenience, simple power supply, and cost are more important.

These diversities notwithstanding, a certain common trend is observable in the literature when looking at typical hardware platforms for wireless sensor nodes. While there is certainly not a single standard available, nor would such a standard necessarily be able to support all application types, this section will survey these typical sensor node architectures. In addition, there are a number of research projects that focus on shrinking any of the components in size, energy consumption, or costs, based on the fact that custom off-the-shelf components do currently not live up to some of the more stringent application requirements. But as this book focuses on the networking aspects of WSNs, these efforts are not discussed here.

A basic sensor node comprises five main components (Figure 2.1):

Controller A controller to process all the relevant data, capable of executing arbitrary code.

Memory Some memory to store programs and intermediate data; usually, different types of memory are used for programs and data.

Sensors and actuators The actual interface to the physical world: devices that can observe or control physical parameters of the environment.

Communication Turning nodes into a network requires a device for sending and receiving information over a wireless channel.

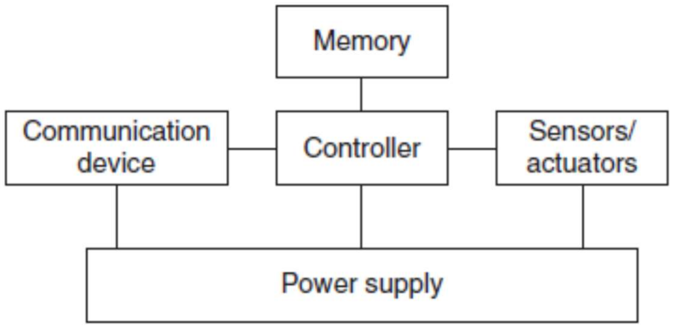


Figure 2.1 Overview of main sensor node hardware components

Power supply As usually no tethered power supply is available, some form of batteries are necessary to provide energy. Sometimes, some form of recharging by obtaining energy from the environment is available as well (e.g. solar cells). Each of these components has to operate balancing the trade-off between as small an energy consumption as possible on the one hand and the need to fulfill their tasks on the other hand. For example, both the communication device and the controller should be turned off as long as possible. To wake up again, the controller could, for example, use a pre-programmed timer to be reactivated after some time. Alternatively, the sensors could be programmed to raise an interrupt if a given event occurs – say, a temperature value exceeds a given threshold or the communication device detects an incoming transmission.

Supporting such alert functions requires appropriate interconnection between individual components. Moreover, both control and data information have to be exchanged along these interconnections. This interconnection can be very simple – for example, a sensor could simply report an analog value to the controller – or it could be endowed with some intelligence of its own, pre-processing sensor data and only waking up the main controller if an actual event has been detected – for example, detecting a threshold crossing for a simple temperature sensor. Such pre-processing can be highly customized to the specific sensor yet remain simple enough to run continuously, resulting in improved energy efficiency.

Controller

Microcontrollers versus microprocessors, FPGAs, and ASICs

The controller is the core of a wireless sensor node. It collects data from the sensors, processes this data, decides when and where to send it, receives data from other sensor nodes, and decides on the actuator's behavior. It has to execute various programs, ranging from time-critical signal processing and communication protocols to application programs; it is the Central Processing Unit (CPU) of the node.

Such a variety of processing tasks can be performed on various controller architectures, representing trade-offs between flexibility, performance, energy efficiency, and costs.

One solution is to use general-purpose processors, like those known from desktop computers. These processors are highly overpowered, and their energy consumption is excessive. But simpler processors do exist, specifically geared toward usage in embedded systems. These processors are commonly referred as microcontrollers. Some of the key characteristics why these microcontrollers are particularly suited to embedded systems are their flexibility in connecting with other devices (like sensors), their instruction set amenable to time-critical signal processing, and their typically low power consumption; they are also convenient in that they often have memory built in. In addition, they are freely programmable and hence very flexible. Microcontrollers are also suitable for WSNs since they commonly have the possibility to reduce their power consumption by going into sleep states where only parts of the controller are active; details vary considerably between different controllers.. One of the main differences to general-purpose systems is that microcontroller-based systems usually do not feature a memory management unit, somewhat limiting the functionality of memory – for example, protected or virtual memory is difficult, if not impossible, to achieve.

A specialized case of programmable processors are Digital Signal Processors (DSPs). They are specifically geared, with respect to their architecture and their instruction set, for processing large amounts of vectoral data, as is typically the case in signal processing applications. In a wireless sensor node, such a DSP could be used to process data coming from a simple analog, wireless communication device to extract a digital data stream. In broadband wireless communication, DSPs are an appropriate and successfully used platform. But in wireless sensor networks, the requirements on wireless communication are usually much more modest (e.g. simpler, easier to process modulations are used that can be efficiently handled in hardware by the communication device itself) and the signal processing tasks related to the actual sensing of data is also not overly complicated. Hence, these advantages of a DSP are typically not required in a WSN node and they are usually not used.

Another option for the controller is to depart from the high flexibility offered by a (fairly general purpose) microcontroller and to use Field-Programmable Gate Arrays (FPGAs) or Application- Specific Integrated Circuits (ASICs) instead. An FPGA can be reprogrammed (or rather reconfigured) “in the field” to adapt to a changing set of requirements; however, this can take time and energy – it is not practical to reprogram an FPGA at the same frequency as a microcontroller could change between different programs. An ASIC is a specialized processor, custom designed for a given application such as, for example, high-speed routers and switches. The typical trade-off here is loss of flexibility in return for a considerably better energy efficiency and performance. On the other hand, where a microcontroller requires software development, ASICs provide the same functionality in hardware, resulting in potentially more costly hardware development.

For a dedicated WSN application, where the duties of a the sensor nodes do not change over lifetime and where the number of nodes is big enough to warrant the investment in ASIC development, they can be a superior solution. At the current stage of WSN technology, however, the bigger flexibility and simpler usage of microcontrollers makes them the generally preferred solution. However, this is not necessarily the final solution as “convenient programmability over several orders of energy consumption and data processing requirements is a worthy research goal”. In addition, splitting processing tasks between some low-level, fixed functionality put into a very energy-efficient ASIC and high-level, flexible, relatively rarely invoked processing on a microcontroller is an attractive design and research option.

Some examples for microcontrollers

Microcontrollers that are used in several wireless sensor node prototypes include the Atmel processor or Texas Instrument’s MSP 430. In older prototypes, the Intel StrongArm processors have also been used, but this is no longer considered as a practical option; it is included here for the sake of completeness. Nonetheless, as the principal properties of these processors and controllers are quite similar, conclusions from these earlier research results still hold to a large degree.

Intel StrongARM The Intel StrongARM is, in WSN terms, a fairly high-end processor as it is mostly geared toward handheld devices like PDAs. The SA-1100 model has a 32-bit Reduced Instruction Set Computer (RISC) core, running at up to 206 MHz.

Texas Instruments MSP 430 Texas Instrument provides an entire family of microcontrollers under the family designation MSP 430. Unlike the StrongARM, it is explicitly intended for embedded applications. Accordingly, it runs a 16-bit RISC core at considerably lower clock frequencies (up to 4 MHz) but comes with a wide range of interconnection possibilities and an instruction set amenable to easy handling of peripherals of different kinds. It features a varying amount of on-chip RAM (sizes are 2–10 kB), several 12-bit analog/digital converters, and a real-time clock. It is certainly powerful enough to handle the typical computational tasks of a typical wireless sensor node (possibly with the exception of driving the radio front end, depending on how it is connected – bit or byte interface – to the controller).

Atmel ATmega The Atmel ATmega 128L [28] is an 8-bit microcontroller, also intended for usage in embedded applications and equipped with relevant external interfaces for common peripherals.

Memory

The memory component is fairly straightforward. Evidently, there is a need for Random Access Memory (RAM) to store intermediate sensor readings, packets from other nodes, and so on. While RAM is fast, its main disadvantage is that it loses its content if power supply is interrupted. Program code can be stored in Read-Only Memory (ROM) or, more typically, in Electrically Erasable Programmable Read-Only Memory (EEPROM) or flash memory (the later being similar to EEPROM but allowing data to be erased or written in blocks instead of only a byte at a time). Flash memory can also serve as intermediate storage of data in case RAM is insufficient or when the power supply of RAM should be shut down for some time. The long read and write access delays of flash memory should be taken into account, as well as the high required energy.

Correctly dimensioning memory sizes, especially RAM, can be crucial with respect to manufacturing costs and power consumption. However, even general rules of thumbs are difficult to give as the memory requirements are very much application dependent.

Communication Device

Choice of transmission medium

The communication device is used to exchange data between individual nodes. In some cases, wired communication can actually be the method of choice and is frequently applied in many sensor networklike settings (using field buses like Profibus, LON, CAN, or others). The communication devices for these networks are custom off-the-shelf components.

The case of wireless communication is considerably more interesting. The first choice to make is that of the transmission medium – the usual choices include radio frequencies, optical communication, and ultrasound; other media like magnetic inductance are only used in very specific cases. Of these choices, Radio Frequency (RF)-based communication is by far the most relevant one as it best fits the requirements of most WSN applications: It provides relatively long range and high data rates, acceptable error rates at reasonable energy expenditure, and does not require line of sight between sender and receiver. Thus, RF-based communication and transceiver will receive the lion share of attention here; other media are only treated briefly at the end of this section.

For a practical wireless, RF-based system, the carrier frequency has to be carefully chosen. Chapter 4 contains a detailed discussion; for the moment, suffice it to say that wireless sensor networks typically use communication frequencies between about 433 MHz and 2.4 GHz.

Transceivers

For actual communication, both a transmitter and a receiver are required in a sensor node. The essential task is to convert a bit stream coming from a microcontroller (or a sequence of bytes or frames) and convert them to and from radio waves. For practical purposes, it is usually convenient to use a device that combines these two tasks in a single entity. Such combined devices are called transceivers. Usually, half-duplex operation is realized since transmitting and receiving at the same time on a wireless medium is impractical in most cases (the receiver would only hear the own transmitter anyway).

A range of low-cost transceivers is commercially available that incorporate all the circuitry required for transmitting and receiving – modulation, demodulation, amplifiers, filters, mixers, and so on. For a judicious choice, the transceiver’s tasks and its main characteristics have to be understood.

Transceiver tasks and characteristics

To select appropriate transceivers, a number of characteristics should be taken into account. The most important ones are:

Service to upper layer A receiver has to offer certain services to the upper layers, most notably to the Medium Access Control (MAC) layer. Sometimes, this service is packet oriented; sometimes, a transceiver only provides a byte interface or even only a bit interface to the microcontroller. In any case, the transceiver must provide an interface that somehow allows the MAC layer to initiate frame transmissions and to hand over the packet from, say, the main memory of the sensor node into the transceiver (or a byte or a bit stream, with additional processing required on the microcontroller). In the other direction, incoming packets must be streamed into buffers accessible by the MAC protocol.

Power consumption and energy efficiency The simplest interpretation of energy efficiency is the energy required to transmit and receive a single bit. Also, to be suitable for use in WSNs, transceivers should be switchable between different states, for example, active and sleeping. The idle power consumption in each of these states and during switching between them is very important.

Carrier frequency and multiple channels Transceivers are available for different carrier frequencies; evidently, it must match application requirements and regulatory restrictions. It is often useful if the transceiver provides several carrier frequencies (“channels”) to choose from, helping to alleviate some congestion problems in dense networks. Such channels or “sub bands” are relevant, for example, for certain MAC protocols (FDMA or multichannel CSMA/ ALOHA techniques.

State change times and energy A transceiver can operate in different modes: sending or receiving, use different channels, or be in different power-safe states. In any case, the time and the energy required to change between two such states are important figures of merit. The turnaround time between sending and receiving, for example, is important for various medium access protocols.

Data rates Carrier frequency and used bandwidth together with modulation and coding determine the gross data rate. Typical values are a few tens of kilobits per second – considerably less than in broadband wireless communication, but usually sufficient for WSNs. Different data rates can be achieved, for example, by using different modulations or changing the symbol rate.

Modulations The transceivers typically support one or several of on/off-keying, ASK, FSK, or similar modulations. If several modulations are available, it is convenient for experiments if they are selectable at runtime even though, for real deployment, dynamic switching between modulations is not one of the most discussed options.

Coding Some transceivers allow various coding schemes to be selected.

Transmission power control Some transceivers can directly provide control over the transmission power to be used; some require some external circuitry for that purpose. Usually, only a discrete number of power levels are available from which the actual transmission power can be chosen. Maximum output power is usually determined by regulations.

Noise figure The noise figure, *NF* of an element is defined as the ratio of the Signal-to-Noise Ratio (SNR) ratio SNRI at the input of the element to the SNR ratio SNRO at the element's output: $NF = SNR_I / SNR_O$

It describes the degradation of SNR due to the element's operation and is typically given in dB: $NF\ dB = SNR_I\ dB - SNR_O\ dB$

Gain The gain is the ratio of the output signal power to the input signal power and is typically given in dB. Amplifiers with high gain are desirable to achieve good energy efficiency.

Power efficiency The efficiency of the radio front end is given as the ratio of the radiated power to the overall power consumed by the front end; for a power amplifier, the efficiency describes the ratio of the output signal's power to the power consumed by the overall power amplifier.

Receiver sensitivity The receiver sensitivity (given in dBm) specifies the minimum signal power at the receiver needed to achieve a prescribed Eb/N0 or a prescribed bit/packet error rate. Better sensitivity levels extend the possible range of a system.

Range While intuitively the range of a transmitter is clear, a formal definition requires some care. The range is considered in absence of interference; it evidently depends on the maximum transmission power, on the antenna characteristics, on the attenuation caused by the environment, which in turn depends on the used carrier frequency, on the modulation/coding scheme that is used, and on the bit error rate that one is willing to accept at the receiver. It also depends on the quality of the receiver, essentially captured by its sensitivity. Typical values are difficult to give here, but prototypes or products with ranges between a few meters and several hundreds of meters are available.

Blocking performance The blocking performance of a receiver is its achieved bit error rate in the presence of an interferer. More precisely, at what power level can an interferer (at a fixed distance) send at a given offset from the carrier frequency such that target BER can still be met? An interferer at higher frequency offsets can be tolerated at large power levels. Evidently, blocking performance can be improved by interposing a filter between antenna and transceiver.

An important special case is an adjacent channel interferer that transmits on neighbouring frequencies. The adjacent channel suppression describes a transceiver's capability to filter out signals from adjacent frequency bands (and thus to reduce adjacent channel interference) has a direct impact on the observed Signal to Interference and Noise Ratio (SINR).

Out of band emission The inverse to adjacent channel suppression is the out of band emission of a transmitter. To limit disturbance of other systems, or of the WSN itself in a multichannel setup, the transmitter should produce as little as possible of transmission power outside of its prescribed bandwidth, centered around the carrier frequency.

Carrier sense and RSSI In many medium access control protocols, sensing whether the wireless channel, the carrier, is busy (another node is transmitting) is a critical information. The receiver has to be able to provide that information. The precise semantics of this carrier sense signal depends on the implementation. For example, the IEEE 802.15.4 standard distinguishes the following modes:

• The received energy is above threshold; however, the underlying signal does not need to comply with the modulation and spectral characteristics.

• A carrier has been detected, that is, some signal which complies with the modulation.

• Carrier detected and energy is present.

Also, the signal strength at which an incoming data packet has been received can provide useful information (e.g. a rough estimate about the distance from the transmitter assuming the transmission power is known); a receiver has to provide this information in the Received Signal Strength Indicator (RSSI).

Frequency stability The frequency stability denotes the degree of variation from nominal center frequencies when environmental conditions of oscillators like temperature or pressure change. In extreme cases, poor frequency stability can break down communication links, for example, when one node is placed in sunlight whereas its neighbor is currently in the shade.

Voltage range Transceivers should operate reliably over a range of supply voltages. Otherwise, inefficient voltage stabilization circuitry is required.

Transceiver Structure

A fairly common structure of transceivers is into the Radio Frequency (RF) front end and the baseband part:

The radio frequency front end performs analog signal processing in the actual radio frequency band, whereas the baseband processor performs all signal processing in the digital domain and communicates with a sensor node's processor or other digital circuitry. Between these two parts, a frequency conversion takes place, either directly or via one or several Intermediate Frequencys (IFs). The boundary between the analog and the digital domain is constituted by Digital/Analog Converters (DACs) and Analog/Digital Converters (ADCs).

The RF front end performs analog signal processing in the actual radio frequency band, for example in the 2.4 GHz Industrial, Scientific, and Medical (ISM) band; it is the first stage of the interface between the electromagnetic waves and the digital signal processing of the further transceiver stages. Some important elements of an RF front ends architecture are sketched in Figure:

The Power Amplifier (PA) accepts upconverted signals from the IF or baseband part and amplifies them for transmission over the antenna. The Low Noise Amplifier (LNA) amplifies incoming signals up to levels suitable for further processing without significantly reducing the SNR [470]. The range of powers of the incoming signals varies from very weak signals from nodes close to the reception boundary to strong signals from nearby nodes; this range can be up to 100 dB. Without management actions, the LNA is active all the time and can consume a significant fraction of the transceiver's energy.

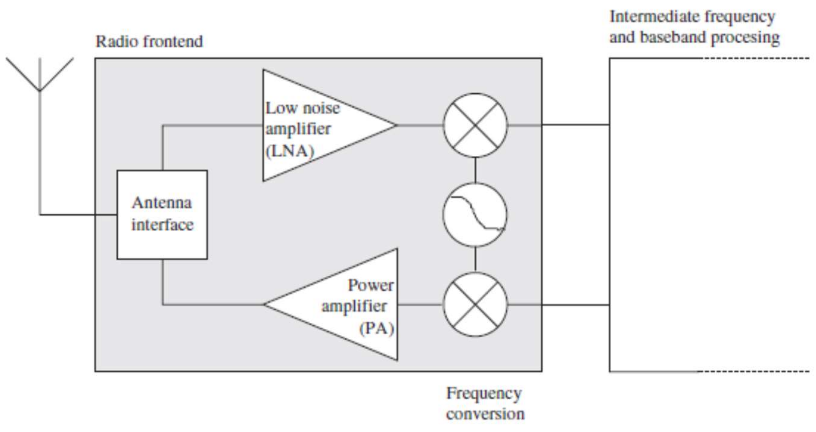


Figure 2.2 RF front end [46]

The elements like local oscillators or voltage-controlled oscillators and mixers are used for frequency conversion from the RF spectrum to intermediate frequencies or to the baseband. The incoming signal at RF frequencies fRF is multiplied in a mixer with a fixed-frequency signal from the local oscillator (frequency fLO). The resulting intermediate-frequency signal has frequency fLO − fRF. Depending on the RF front end architecture, other elements like filters are also present.

Transceiver operational states Many transceivers can distinguish four operational states:

Transmit In the transmit state, the transmit part of the transceiver is active and the antenna radiates energy.

Receive In the receive state the receive part is active.

Idle A transceiver that is ready to receive but is not currently receiving anything is said to be in an idle state. In this idle state, many parts of the receive circuitry are active, and others can be switched off. For example, in the synchronization circuitry, some elements concerned with acquisition are active, while those concerned with tracking can be switched off and activated only when the acquisition has found something. A major source of power dissipation is leakage.

Sleep In the sleep state, significant parts of the transceiver are switched off. There are transceivers offering several different sleep states for a discussion of sleep states for IEEE 802.11 transceivers. These sleep states differ in the amount of circuitry switched off and in the associated recovery times and start-up energy. For example, in a complete power down of the transceiver, the start-up costs include a complete initialization as well as configuration of the radio, whereas in "lighter" sleep modes, the clock driving certain transceiver parts is throttled down while configuration and operational state is remembered.

Advanced radio concepts

Apart from these basic transceiver concepts, a number of advanced concepts for radio communication are the objectives of current research. Three of them are briefly summarized here.

Wakeup radio Looking at the transceiver concepts described above, one of the most power-intensive operations is waiting for a transmission to come in, ready to receive it. During this time, the receiver circuit must be powered up so that the wireless channel can be observed, spending energy without any immediate benefit. While it seems unavoidable to provide a receiver with power during the actual reception of a packet, it would be desirable not to have to invest power while the node is only waiting for a packet to come in. A receiver structure is necessary that does not need power but can detect when a packet starts to arrive. To keep this specialized receiver simple, it suffices for it to raise an event to notify other components of an incoming packet; upon such an event, the main receiver can be turned on and perform the actual reception of the packet.

Spread-spectrum transceivers Simple transceiver concepts, based on modulations like Amplitude Shift Keying (ASK) or Frequency Shift Keying (FSK), can suffer from limited performance, especially in scenarios with a lot of interference. To overcome this limitation, the use of spread-spectrum transceivers has been proposed by some researchers. These transceivers, however, suffer mostly from complex hardware and consequently higher prices, which has prevented them from becoming a mainstream concept for WSNs so far.

Ultrawideband communication UltraWideBand (UWB) communication is a fairly radical change from conventional wireless communication as outlined above. Instead of modulating a digital signal onto a carrier frequency, a very large bandwidth is used to directly transmit the digital sequence as very short impulses (to form nearly rectangular impulses requires considerable bandwidth, because of which this concept is not used traditionally). Accordingly, these impulses occupy a large spectrum starting from a few Hertz up to the range of several GHz. The challenge is to synchronize sender and receiver sufficiently (to an accuracy of trillionth of seconds) so that the impulses can be correctly detected..

For a communication system, the effect is that a very high data rate can be realized over short distances; what is more, UWB communication can relatively easily penetrate obstacles such as doors, which are impermeable to narrowband radio waves. For a WSN, the high data rate is not strictly necessary but can be leveraged to reduce the on-time of the transceivers. The nature of UWB also allows to precisely measure distances (with claimed precision of centimeters).

Nonradio frequency wireless communication While most of the wireless sensor network work has focused on the use of radio waves as communication media, other options exists. In particular, optical communication and ultrasound communication have been considered as alternatives.

Optical The use of optical links between sensor nodes. Its main advantage is the very small energy per bit required for both generating and detecting optical light – simple Light-Emitting Diodes (LEDs) are good examples for high-efficiency senders. The required circuitry for an optical transceiver is also simpler and the device as a whole can be smaller than the radio frequency counterpart. Also, communication can take place concurrently with only negligible interference. The evident disadvantage, however, is that communicating peers need to have a line of sight connection and that optical communication is more strongly influenced by weather conditions.

In some application scenarios, however, sensor nodes are used in environments where radio or optical communication is not applicable because these waves do not penetrate the surrounding medium. One such medium is water, and an application scenario is the surveillance of marine ground floor erosion to help in the construction of offshore wind farms. Sensors are deployed on the marine ground floor and have to communicate amongst themselves. In such an underwater environment, ultrasound is an attractive communication medium as it travels relatively long distances at comparably low power.

Some examples of radio transceivers

To complete this discussion of possible communication devices, a few examples of standard radio transceivers that are commonly used in various WSN prototype nodes should be briefly described. All these transceivers are in fact commodity, off-the-shelf items available via usual distributors. They are all single-chip solutions, integrating transmitter and receiver functionality, requiring only a small number of external parts and have a fairly low-power consumption. In principle, similar equipment is available from a number of manufacturers – as can be expected, there is not one "best product" available, but each of them has particular advantages and disadvantages.

RFM TR1000 family

The TR1000 family of radio transceivers from RF Monolithics2 is available for the 916 MHz and 868 MHz frequency range. It works in a 400 kHz wide band centered at, for example, 916.50 MHz. It is intended for short-range radio communication with up to 115.2 kbps. The modulation is either on-off-keying (at a maximum rate of 30 kbps) or ASK; it also provides a dynamically tunable output power.

Hardware accelerators (Mica motes)

The Mica motes use the RFM TR1000 transceiver and contain also a set of hardware accelerators. On the one hand, the transceiver offers a very low-level interface, giving the microcontroller tight control over frame formats, MAC protocols, and so forth. On the other hand, framing and MAC can be very computation intensive, for example, for computing checksums, for making bytes out of serially received bits or for detecting Start Frame

Delimiters (SFDs) in a stream of symbols. The hardware accelerators offer some of these primitive computations in hardware, right at the disposal of the microcontroller.

IEEE 802.15.4/Ember EM2420 RF transceiver

The IEEE 802.15.4 low-rate Wireless Personal Area Network (WPAN) works in three different frequency bands and employs a DSSS scheme. For one particular RF front-end design, the Ember4 EM2420 RF Transceiver, some numbers on power dissipation are available. For a radiated power of −0.5 dBm (corresponding to ≈0.9 mW) and with a supply voltage of 3.3 V, the transmit mode draws a current of 22.7 mA, corresponding to ≈74.9 mW, whereas in the receive mode, 25.2 mA current are drawn, corresponding to ≈83.2 mW. In the sleep mode, only 12 μA are drawn.

Sensors and Actuators

Without the actual sensors and actuators, a wireless sensor network would be beside the point entirely. But as the discussion of possible application areas has already indicated, the possible range of sensors is vast. It is only possible to give a rough idea on which sensors and actuators can be used in a WSN.

Sensors

Sensors can be roughly categorized into three categories:

Passive, omnidirectional sensors These sensors can measure a physical quantity at the point of the sensor node without actually manipulating the environment by active probing – in this sense, they are passive. Moreover, some of these sensors actually are self-powered in the sense that they obtain the energy they need from the environment – energy is only needed to amplify their analog signal. There is no notion of "direction" involved in these measurements. Typical examples for such sensors include thermometer, light sensors, vibration, microphones, humidity, mechanical stress or tension in materials, chemical sensors sensitive for given substances, smoke detectors, air pressure, and so on.

Passive, narrow-beam sensors These sensors are passive as well, but have a well-defined notion of direction of measurement. A typical example is a camera, which can "take measurements" in a given direction, but has to be rotated if need be.

Active sensors This last group of sensors actively probes the environment, for example, a sonar or radar sensor or some types of seismic sensors, which generate shock waves by small explosions. These are quite specific – triggering an explosion is certainly not a lightly undertaken action – and require quite special attention.

In practice, sensors from all of these types are available in many different forms with many individual peculiarities. Obvious trade-offs include accuracy, dependability, energy consumption, cost, size, and so on – all this would make a detailed discussion of individual sensors quite ineffective. Overall, most of the theoretical work on WSNs considers passive, omnidirectional sensors. Narrow-beam-type sensors like cameras are used in some practical testbeds, but there is no real systematic investigation on how to control and schedule the movement of such sensors. Active sensors are not treated in the literature to any noticeable extent.

Strictly speaking, this assumption of a coverage area is difficult to justify in its simplest form. Nonetheless, it can be practically useful: It is often possible to postulate, on the basis of application specific knowledge, some properties of the physical quantity under consideration, in particular, how quickly it can change with respect to distance. For example, temperature or air pressure are unlikely to vary very strongly within a few meters. Hence, allowing for some inevitable inaccuracies in the measurement, the maximum rate of changeover distance can be used to derive such a "coverage radius" within which the values of a single sensor node are considered "good enough". The precise mathematical tools for such a derivation are spatial versions of the sampling theorems.

Actuators

Actuators are just about as diverse as sensors, yet for the purposes of designing a WSN, they are a bit simpler to take account of: In principle, all that a sensor node can do is to open or close a switch or a relay or to set a value in some way. Whether this controls a motor, a light bulb, or some other physical object is not really of concern to the way communication protocols are designed. Hence, in this book, we shall treat actuators fairly summarily without distinguishing between different types. In a real network, however, care has to be taken to properly account for the idiosyncrasies of different actuators. Also, it is good design practice in most embedded system applications to pair any actuator with a controlling sensor – following the principle to "never trust an actuator".

Power Supply Of Sensor Nodes

For untethered wireless sensor nodes, the power supply is a crucial system component. There are essentially two aspects: First, storing energy and providing power in the required form; second, attempting to replenish consumed energy by “scavenging” it from some node-external power source over time.

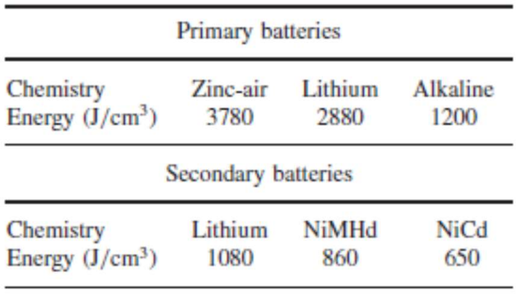
Storing power is conventionally done using batteries. As a rough orientation, a normal AA battery stores about 2.2–2.5 Ah at 1.5 V. Battery design is a science and industry in itself, and energy scavenging has attracted a lot of attention in research.

Storing energy: Batteries

Traditional batteries

The power source of a sensor node is a battery, either non-rechargeable (“primary batteries”) or, if an energy scavenging device is present on the node, also rechargeable (“secondary batteries”).

Table 2.2 Energy densities for various primary and secondary battery types [703]



Primary batteries			
Chemistry	Zinc-air	Lithium	Alkaline
Energy (J/cm^3)	3780	2880	1200
Secondary batteries			
Chemistry	Lithium	NiMHd	NiCd
Energy (J/cm^3)	1080	860	650

In some form or other, batteries are electro-chemical stores for energy – the chemicals being the main determining factor of battery technology. Upon these batteries, very tough requirements are imposed:

Capacity They should have high capacity at a small weight, small volume, and low price. The main metric is energy per volume, J/cm3. Table shows some typical values of energy densities, using traditional, macroscale battery technologies. In addition, research on “microscale” batteries, for example, deposited directly onto a chip, is currently ongoing.

Capacity under load They should withstand various usage patterns as a sensor node can consume quite different levels of power over time and actually draw high current in certain operation modes. Current numbers on power consumption of WSN nodes vary, so it is difficult to provide precise guidelines. But for most technologies, the larger the battery, the more power can be delivered instantaneously. In addition, the rated battery capacity specified by a manufacturer is only valid as long as maximum discharge currents are not exceeded, lest capacity drops or even premature battery failure occurs.

Self-discharge Their self-discharge should be low; they might also have to last for a long time (using certain technologies, batteries are operational only for a few months, irrespective of whether power is drawn from them or not). Zinc-air batteries, for example, have only a very short lifetime (on the order of weeks), which offsets their attractively high energy density.

Efficient recharging Recharging should be efficient even at low and intermittently available recharge power; consequently, the battery should also not exhibit any “memory effect”. Some of the energy-scavenging techniques described below are only able to produce current in the μA region (but possibly sustained) at only a few volts at best. Current battery technology would basically not recharge at such values.

Relaxation Their relaxation effect – the seeming self-recharging of an empty or almost empty battery when no current is drawn from it, based on chemical diffusion processes within the cell – should be clearly understood. Battery lifetime and usable capacity is considerably extended if this effect is leveraged. As but one example, it is possible to use multiple batteries in parallel and “schedule” the discharge from one battery to another, depending on relaxation properties and power requirements of the operations to be supported.

Unconventional energy stores Apart from traditional batteries, there are also other forms of energy reservoirs that can be contemplated. In a wider sense, fuel cells also qualify as an electro-chemical storage of energy, directly producing electrical energy by oxidizing hydrogen or hydrocarbon fuels. Fuel cells actually have excellent energy densities (e.g. methanol as a fuel stores 17.6 kJ/cm3), but currently available systems still require a nonnegligible minimum size for pumps, valves, and so on. A slightly more traditional approach to using energy stored in hydrocarbons is to use miniature versions of heat engines, for example, a turbine. Shrinking such heat engines to the desired sizes still requires a considerable research effort in MicroElectroMechanical Systems (MEMSs); predictions regarding power vary between 0.1–10 W at sizes of about 1 cc. And lastly, even radioactive substances have been proposed as an energy store. Another option are so-called “gold caps”, high-quality and high-capacity capacitors, which can store relatively large amounts of energy, can be easily and quickly recharged, and do not wear out over time.

DC–DC Conversion Unfortunately, batteries (or other forms of energy storage) alone are not sufficient as a direct power source for a sensor node. One typical problem is the reduction of a battery's voltage as its capacity drops. Consequently, less power is delivered to the sensor node's circuits, with immediate consequences for oscillator frequencies and transmission power – a node on a weak battery will have a smaller transmission range than one with a full battery, possibly throwing off any calibrations done for the range at full battery ranges.

A DC – DC converter can be used to overcome this problem by regulating the voltage delivered to the node's circuitry. To ensure a constant voltage even though the battery's supply voltage drops, the DC – DC converter has to draw increasingly higher current from the battery when the battery is already becoming weak, speeding up battery death. Also, the DC – DC converter does consume energy for its own operation, reducing overall efficiency. But the advantages of predictable operation during the entire life cycle can outweigh these disadvantages.

Energy scavenging Some of the unconventional energy stores described above – fuel cells, micro heat engines, radioactivity – convert energy from some stored, secondary form into electricity in a less direct and easy to use way than a normal battery would do. The entire energy supply is stored on the node itself – once the fuel supply is exhausted, the node fails.

To ensure truly long-lasting nodes and wireless sensor networks, such a limited energy store is unacceptable. Rather, energy from a node's environment must be tapped into and made available to the node – energy scavenging should take place.

Photovoltaics The well-known solar cells can be used to power sensor nodes. The available power depends on whether nodes are used outdoors or indoors, and on time of day and whether for outdoor usage. Different technologies are best suited for either outdoor or indoor usage. The resulting power is somewhere between 10 μW/cm2 indoors and 15 mW/cm2 outdoors. Single cells achieve a fairly stable output voltage of about 0.6 V (and have therefore to be used in series) as long as the drawn current does not exceed a critical threshold, which depends, among other factors, on the light intensity. Hence, solar cells are usually used to recharge secondary batteries. Best trade-offs between complexity of recharging circuitry, solar cell efficiency, and battery lifetime are still open questions.

Temperature gradients Differences in temperature can be directly converted to electrical energy. Theoretically, even small difference of, for example, 5 K can produce considerable power, but practical devices fall very short of theoretical upper limits (given by the Carnot efficiency). Seebeck effect-based thermoelectric generators are commonly considered; one example is a generator, which will be commercially available soon, that achieves about 80 μW/cm2 at about 1 V from a 5 Kelvin temperature difference.

Vibrations One almost pervasive form of mechanical energy is vibrations: walls or windows in buildings are resonating with cars or trucks passing in the streets, machinery often has low frequency vibrations, ventilations also cause it, and so on. The available energy depends on both amplitude and frequency of the vibration and ranges from about 0.1 μW/cm3 up to 10, 000 μW/cm3 for some extreme cases (typical upper limits are lower).

Converting vibrations to electrical energy can be undertaken by various means, based on electromagnetic, electrostatic, or piezoelectric principles. Figure shows, as an example, a generator based on a variable capacitor. Practical devices of 1 cm3 can produce about 200 μW/cm3 from 2.25 m/s2, 120 Hz vibration sources, actually sufficient to power simple wireless transmitters.

Pressure variations Somewhat akin to vibrations, a variation of pressure can also be used as a power source. Such piezoelectric generators are in fact used already. One well-known example is the inclusion of a piezoelectric generator in the heel of a shoe, to generate power as a human walks. This device can produce, on average, 330 μW/cm2. It is, however, not clear how such technologies can be applied to WSNs.

Flow of air/liquid Another often-used power source is the flow of air or liquid in wind mills or turbines. The challenge here is again the miniaturization, but some of the work on millimetre scale MEMS gas turbines might be reusable. However, this has so far not produced any notable results.

As these examples show, energy scavenging usually has to be combined with secondary batteries as the actual power sources are not able to provide power consistently, uninterruptedly, at a required level; rather, they tend to fluctuate over time. This requires additional circuitry for recharging of batteries, possibly converting to higher power levels, and a battery technology that can be recharged at low currents. An alternative approach is to align the task execution pattern of the sensor network (which sensor is active when) with the characteristics of energy.

Energy Consumption of Sensor Nodes

Operation states with different power consumption

As the previous section has shown, energy supply for a sensor node is at a premium: batteries have small capacity, and recharging by energy scavenging is complicated and volatile. Hence, the energy consumption of a sensor node must be tightly controlled. The main consumers of energy are the controller, the radio front ends, to some degree the memory, and, depending on the type, the sensors.

One important contribution to reduce power consumption of these components comes from chip-level and lower technologies: Designing low-power chips is the best starting point for an energy-efficient sensor node. But this is only one half of the picture, as any advantages gained by such designs can easily be squandered when the components are improperly operated.

The crucial observation for proper operation is that most of the time a wireless sensor node has nothing to do. Hence, it is best to turn it off. Naturally, it should be able to wake up again, on the basis of external stimuli or on the basis of time. Therefore, completely turning off a node is not possible, but rather, its operational state can be adapted to the tasks at hand. Introducing and using multiple states of operation with reduced energy consumption in return for reduced functionality is the core technique for energy-efficient wireless sensor node. These modes can be introduced for all components of a sensor node, in particular, for controller, radio front end, memory, and sensors. Different models usually support different numbers of such sleep states with different characteristics; some examples are provided in the following sections. For a controller, typical states are “active”, “idle”, and “sleep”; a radio modem could turn transmitter, receiver, or both on or off; sensors and memory could also be turned on or off. The usual terminology is to speak of a “deeper” sleep state if less power is consumed.

While such a graded sleep state model is straightforward enough, it is complicated by the fact that transitions between states take both time and energy. The usual assumption is that the deeper the sleep state, the more time and energy it takes to wake up again to fully operational state (or to another, less deep sleep state). Hence, it may be worthwhile to remain in an idle state instead of going to deeper sleep states even from an energy consumption point of view.

Figure illustrates this notion based on a commonly used model. At time t1, the decision whether or not a component (say, the microcontroller) is to be put into sleep mode should be taken to reduce power consumption from *Pactive* to *Psleep*. If it remains active and the next event occurs at time *tevent*, then a total energy of *Eactive* = *Pactive* (*tevent* − *t1*) has be spent uselessly idling. Putting the component into sleep mode, on the other hand, requires a time τdown until sleep mode has been reached; as a simplification, assume that the average power consumption during this phase is (*Pactive* + *Psleep*)/2. Then, *Psleep* is consumed until *tevent*. In total, *τdown*(*Pactive* + *Psleep*)/2 + (*tevent* − *t1* − *τdown*)*Psleep* energy is required in sleep mode as opposed to (*tevent* − *t1*)*Pactive* when remaining active. The energy saving is thus

$$E_{\text{saved}} = (t_{\text{event}} - t_1)P_{\text{active}} - (\tau_{\text{down}}(P_{\text{active}} + P_{\text{sleep}})/2 + (t_{\text{event}} - t_1 - \tau_{\text{down}})P_{\text{sleep}}). \tag{2.1}$$

Once the event to be processed occurs, however, an additional overhead of

$$E_{\text{overhead}} = \tau_{\text{up}}(P_{\text{active}} + P_{\text{sleep}})/2, \tag{2.2}$$

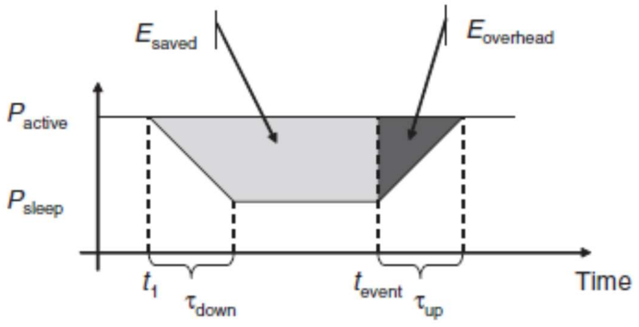


Figure 2.5 Energy savings and overheads for sleep modes

is incurred to come back to operational state before the event can be processed, again making a simplifying assumption about average power consumption during makeup. This energy is indeed an overhead since no useful activity can be undertaken during this time. Clearly, switching to a sleep mode is only beneficial if *Eoverhead* < *Esaved* or, equivalently, if the time to the next event is sufficiently large:

$$(t_{\text{event}} - t_1) > \frac{1}{2}\left(\tau_{\text{down}} + \frac{P_{\text{active}} + P_{\text{sleep}}}{P_{\text{active}} - P_{\text{sleep}}}\tau_{\text{up}}\right). \tag{2.3}$$

Careful scheduling of such transitions has been considered from several perspectives – reference, for example, gives a fairly abstract treatment – and in fact, a lot of medium access control research in wireless sensor networks can be regarded as the problem of when to turn off the receiver of a node.

Microcontroller energy consumption

Basic power consumption in discrete operation states: Embedded controllers commonly implement the concept of multiple operational states as outlined above; it is also fairly easy to control. Some examples probably best explain the idea.

Dynamic voltage scaling

A more sophisticated possibility than discrete operational states is to use a continuous notion of functionality/power adaptation by adapting the speed with which a controller operates. The idea is to choose the best possible speed with which to compute a task that has to be completed by a given deadline. One obvious solution is to switch the controller in full operation mode, compute the task at highest speed, and go back to a sleep mode as quickly as possible.

The alternative approach is to compute the task only at the speed that is required to finish it before the deadline. The rationale is the fact that a controller running at lower speed, that is, lower clock rates, consumes less power than at full speed. This is due to the fact that the supply voltage can be reduced at lower clock rates while still guaranteeing correct operation. This technique is called Dynamic Voltage Scaling (DVS).

Memory

From an energy perspective, the most relevant kinds of memory are on-chip memory of a microcontroller and FLASH memory – off-chip RAM is rarely if ever used. In fact, the power needed to drive on-chip memory is usually included in the power consumption numbers given for the controllers. Read times and read energy consumption tend to be quite similar between different types of FLASH memory. Writing is somewhat more complicated, as it depends on the granularity with which data can be accessed (individual bytes or only complete pages of various sizes). One means for comparability is to look at the numbers for overwriting the whole chip. Considerable differences in erase and write energy consumption exist, up to ratios of 900:1 between different types of memory. Hence, writing to FLASH memory can be a time- and energy-consuming task that is best avoided if somehow possible. For detailed numbers, it is necessary to consult the documentation of the particular wireless sensor node and its FLASH memory under consideration.

Radio transceivers

A radio transceiver has essentially two tasks: transmitting and receiving data between a pair of nodes. To accommodate the necessary low total energy consumption, the transceivers should be turned off most of the time and only be activated when necessary – they work at a low duty cycle. But this incurs additional complexity, time and power overhead that has to be taken into account. To understand the energy consumption behavior of radio transceivers and their impact on the protocol design, models for the energy consumption per bit for both sending and receiving are required.

Modelling energy consumption during transmission

In principle, the energy consumed by a transmitter is due to two sources: one part is due to RF signal generation, which mostly depends on chosen modulation and target distance and hence on the transmission power *Ptx*, that is, the power radiated by the antenna. A second part is due to electronic components necessary for frequency synthesis, frequency conversion, filters, and so on. These costs are basically constant. In addition to the amplifier, other circuitry has to be powered up during transmission as well, for example, baseband processors. This power is referred to as *PtxElec*.

Modelling energy consumption during reception

Similar to the transmitter, the receiver can be either turned off or turned on. While being turned on, it can either actively receive a packet or can be idle, observing the channel and ready to receive. Evidently, the power consumption while it is turned off is negligible. Even the difference between idling and actually receiving is very small and can, for most purposes, be assumed to be zero.

To elucidate, the energy *Ercvd* required to receive a packet has a start-up component *TstartPstart* similar to the transmission case when the receiver had been turned off (startup times are considered equal for transmission and receiving here); it also has a component that is proportional to the packet time n RRcode. During this time of actual reception, receiver circuitry has to be powered up, requiring a (more or less constant) power of PrxElec – for example, to drive the LNA in the RF front end. The last component is the decoding overhead, which is incurred for every bit.

Again, it is worthwhile pointing out that different modulation schemes only implicitly affect this result via the increase in time to transmit the packet.

Some numbers

Providing concrete numbers for exemplary radio transceivers is even more difficult than it is for microcontrollers: The range of commercially available transceivers is vast, with many different characteristics. Transceivers that appear to have excellent energy characteristics might suffer from other shortcomings like poor frequency stability under temperature variations (leading to partitioning of a network when parts of the node are placed in the shade and others in sunlight), poor blocking performance, high susceptibility to interference on neighboring frequency channels, or undesirable error characteristics; they could also lack features that other transceivers have, like tunability to multiple frequencies. Hence, the numbers presented here should be considered very cautiously, even more so since they had been collected from different sources and were likely determined in noncomparable environments (and not all numbers are available for all examples). Still, they should serve to provide some impression of current performance figures for actual hardware.

Another common observation based on these figures is that transmitting and receiving have comparable power consumption, at least for short-range communication. Details differ, of course, but it is an acceptable approximation to assume *PtxElec* = *PrxElec* and even neglecting the amplifier part can be admissible as long as very low transmission powers are used. In fact, for some architectures, receiving consumes more power than transmitting.

Dynamic scaling of radio power consumption

Applying controller-based Dynamic Voltage Scaling (DVS) principles to radio transceivers as well is tempting, but nontrivial. Scaling down supply voltage or frequency to obtain lower power consumption in exchange for higher latency is only applicable to some of the electronic parts of a transceiver, but this would mean that the remainder of the circuitry – the amplifier, for instance, which cannot be scaled down as its radiated and hence its consumed power mostly depends on the communication distance – still has to be run at high power over an extended period of time.

However, the frequency/voltage versus performance trade-off exploited in DVS is not the only possible trade-off to exploit. Any such “parameter versus performance” trade-off that has a convex characteristic should be amenable to an analogous optimization technique. For radio communication, in particular, possible parameters include the choice of modulation and/or code, giving raise to Dynamic Modulation Scaling (DMS), Dynamic Code Scaling (DCS) and Dynamic Modulation- Code Scaling (DMCS) optimization techniques. The claim that such trade-offs do not apply to communication is another one of the “myths” of energy consumption in communication.

The idea of these approaches is to dynamically adapt modulation, coding, or other parameters to maximize system metrics like throughput or, particularly relevant here, energy efficiency. It rests on the hardware’s ability to actually perform such modulation adaptations, but this is a commonly found property of modern transceivers. In addition, delay constraints and time-varying radio channel properties have to be taken into account.

Relationship between computation and communication

Looking at the energy consumption numbers for both microcontrollers and radio transceivers, an evident question to ask is which is the best way to invest the precious energy resources of a sensor node: Is it better to send data or to compute? What is the relation in energy consumption between sending data and computing?

It is clear that communication is a considerably more expensive undertaking than computation. Still, energy required for computation cannot be simply ignored; depending on the computational task, it is usually still smaller than the energy for communication, but still noticeable. This basic observation motivates a number of approaches and design decisions for the networking architecture of wireless sensor networks. The core idea is to invest into computation within the network whenever possible to safe on communication costs, leading to the notion of in-network processing and aggregation.

Power consumption of sensor and actuators

Providing any guidelines about the power consumption of the actual sensors and actuators is next to impossible because of the wide diversity of these devices. For some of them – for example, passive light or temperature sensors – the power consumption can perhaps be ignored in comparison to other devices on a wireless node. For others, in particular, active devices like sonar, power consumption can be quite considerable and must even be considered in the dimensioning of power sources on the sensor node, not to overstress batteries, for example. To derive any meaningful numbers, requires a look at the intended application scenarios and the intended sensors to be used.

In addition, the sampling rate evidently is quite important. Not only does more frequent sampling require more energy for the sensors as such but also the data has to processed and, possibly, communicated somewhere.

CONCLUSION:

This chapter has introduced the types of applications for which wireless sensor networks are intended and a first intuition about the types of technical solutions that are required, both in hardware and in networking technologies. Then, the necessary hardware prerequisites for building wireless sensor networks – the nodes as such. It has shown the principal ways of constructing such nodes and has shown some numbers on the performance and energy consumption of its main components – mainly the controller, the communication device, and the sensors. A wireless sensor node consists of two separate parts: One part that is continuously vigilant, can detect and report events, and has small or even negligible power consumption. This is complemented by a second part that performs actual processing and communication, has higher, nonnegligible power consumption, and has therefore to be operated in a low duty cycle.

There is not a single, "perfect" wireless sensor node – different application requirements will require different trade-offs to be made and different architectures to be used. As a consequence, there will be sensor networks that employ a heterogeneous mix of various node types to fulfil their tasks, for example, nodes with more or less computation power, different types of wireless communication, or different battery sizes. This can have consequences on how to design a wireless sensor network by exploiting this heterogeneity in hardware to assign different tasks to the best-suited nodes.

Notes on
ARCHITECTURES
Unit II

Dr. G. Senthil Kumar,
Associate Professor,
Dept. of ECE, SCSVMV,
email: gsk_ece@kanchiuniv.ac.in

===

OBJECTIVES:

In this lesson, you will be introduced for intelligently controlling the mode of operation of node components. This control has to be exerted by an operating system like execution environment. Then, the basic principles of turning individual sensor nodes into a wireless sensor network is introduced. On the basis of these scenarios and goals, a few principles for the design of networking protocols in wireless sensor networks are derived. Next, the physical layer, that is, functions and components of a sensor node that mediate between the transmission and reception of wireless waveforms and the processing of digital data in the remaining node, including the higher-layer protocol processing.

CONTENTS:

INTRODUCTION

The operating system and programming model is an important consideration. This section describes the tasks of such an operating system along with some examples as well as suitable programming interfaces. On the basis of the high-level application scenarios, more concrete scenarios and the resulting optimization goals of how a network should function are discussed.

On the basis of these scenarios and goals, a few principles for the design of networking protocols in wireless sensor networks are derived – these principles and the resulting protocol mechanisms constitute the core differences of WSNs compared to other network types. To make the resulting capabilities of a WSN usable, a proper service interface is required, as is an integration of WSNs into larger network contexts.

It is a commonly acknowledged truth that the properties of the transmission channel and the physical-layer shape significant parts of the protocol stack. The first goal of this chapter is therefore to provide the reader with a basic understanding of some fundamental concepts related to digital communications over wireless channels. The second important goal is to explain how the specific constraints of wireless sensor networks (regarding, for example, energy and node costs) in turn shape the design of modulation schemes and transceivers. The reader should get an understanding on some of the fundamental trade-offs regarding transmission robustness and energy consumption and how these are affected by the power-consumption properties of transceiver components.

The physical layer is mostly concerned with modulation and demodulation of digital data; this task is carried out by so-called transceivers. In sensor networks, the challenge is to find modulation schemes and transceiver architectures that are simple, low cost, but still robust enough to provide the desired service. The first part of this chapter explains the most important concepts regarding wireless channels and digital communications (over wireless channels); its main purpose is to provide appropriate notions and to give an insight into the tasks involved in transmission and reception over wireless channels. Some simple modulation schemes are discussed as well. In the next part, we discuss the implications of the specific requirements of wireless sensor networks, most notably the scarcity of energy, for the design of transceivers and transmission schemes.

1. OPERATING SYSTEMS AND EXECUTION ENVIRONMENTS

Embedded Operating Systems

The traditional tasks of system are controlling and protecting the access to resources (including support for input/output) and managing their allocation to different users as well as the support for concurrent execution of several processes and communication between these processes. These tasks are, however, an operating only partially required in an embedded system as the executing code is much more restricted and usually much better harmonized than in a general-purpose system. Also, as the description of the microcontrollers has shown, these systems plainly do not have the required resources to support a full-blown operating system.

Rather, an operating system or an execution environment – perhaps the more modest term is the more appropriate one – for WSNs should support the specific needs of these systems. In particular, the need for energy-efficient execution requires support for energy management, for example, in the form of controlled shutdown of individual components or Dynamic Voltage Scaling (DVS) techniques. Also, external components – sensors, the radio modem, or timers – should be handled easily and efficiently, in particular, information that becomes available asynchronously (at any arbitrary point in time) must be handled. All this requires an appropriate programming model, a clear way to structure a protocol stack, and explicit support for energy management – without imposing too heavy a burden on scarce system resources like memory or execution time. These three topics are treated in the following sections, with a case study completing the operating system considerations.

Programming Paradigms and Application Programming Interfaces

Concurrent Programming

One of the first questions for a programming paradigm is how to support concurrency. Such support for concurrent execution is crucial for WSN nodes, as they have to handle data communing from arbitrary sources – for example, multiple sensors or the radio transceiver – at arbitrary points in time. For example, a system could poll a sensor to decide whether data is available and process the data right away, then poll the transceiver to check whether a packet is available, and then immediately process the packet, and so on. (Figure). Such a simple sequential model would run the risk of missing data while a packet is processed or missing a packet when sensor information is processed. This risk is particularly large if the processing of sensor data or incoming packets takes substantial amounts of time, which can easily be the case. Hence, a simple, sequential programming model is clearly insufficient.

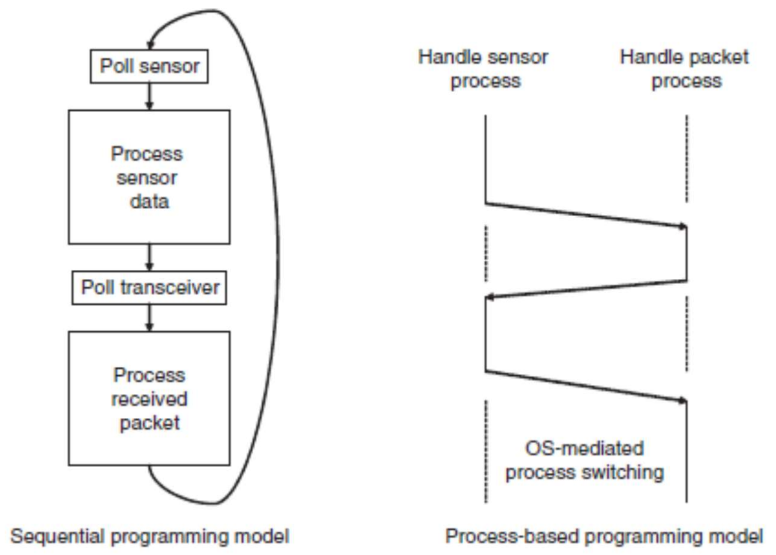


Figure 2.7 Two inadequate programming models for WSN operating systems: purely sequential execution (a) and process-based execution (b)

Process-based concurrency

Most modern, general-purpose operating systems support concurrent (seemingly parallel) execution of multiple processes on a single CPU. Hence, such a process-based approach would be a first candidate to support concurrency in a sensor node as well; it is illustrated in (b) of Figure 2.7. While indeed this approach works in principle, mapping such an execution model of concurrent processes to a sensor node shows, however, that there are some granularity mismatches: Equating individual protocol functions or layers with individual processes would entail a high overhead in switching from one process to another. This problem is particularly severe if often tasks have to be executed that are small with respect to the overhead incurred for switching between tasks – which is typically the case in sensor networks. Also, each process requires its own stack space in memory, which fits ill with the stringent memory constraints of sensor nodes. Event-based programming For these reasons, a somewhat different programming model seems preferable. The idea is to embrace the reactive nature of a WSN node and integrate it into the design of the operating system. The system essentially waits for any event to happen, where an event typically can be the availability of data from a sensor, the arrival of a packet, or the expiration of a timer. Such an event is then handled by a short sequence of instructions that only stores the fact that this event has occurred and stores the necessary information – for example, a byte arriving for a packet or the sensor's value – somewhere. The actual processing of this information is not done in these event handler routines, but separately, decoupled from the actual appearance of events. This event-based programming model is sketched in Figure.

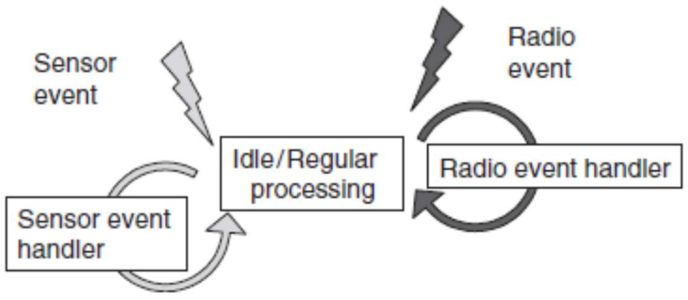


Figure 2.8 Event-based programming model

Such an event handler can interrupt the processing of any normal code, but as it is very simple and short, it can be required to run to completion in all circumstances without noticeably disturbing other code. Event handlers cannot interrupt each other (as this would in turn require complicated stack handling procedures) but are simply executed one after each other. As a consequence, this event-based programming model distinguishes between two different "contexts": one for the time-critical event handlers, where execution cannot be interrupted and a second context for the processing of normal code, which is only triggered by the event handlers.

This event-based programming model is slightly different to what most programmers are used to and commonly requires some getting used to. It is actually comparable, on some levels, to communicating, extended finite state machines, which are used in protocol design formalisms as well as in some parallel programming paradigms. It does offer considerable advantages. The performance of a process-based and an event-based programming model (using TinyOS) are compared on the same hardware and found that performance improved by a factor of 8, instruction/data memory requirements were reduced by factors of 2 and 30, respectively, and power consumption was reduced by a factor of 12.

Interfaces to the Operating System

In addition to the programming model that is stipulated, if not actually imposed, by the operating system, it is also necessary to specify some interfaces to how internal state of the system can be inquired and perhaps set. As the clear distinction between protocol stack and application programs vanishes somewhat in WSNs, such an interface should be accessible from protocol implementations and it should allow these implementations to access each other. This interface is also closely tied with the structure of protocol stacks discussed in the following section.

Such an Application Programming Interface (API) comprises, in general, a "functional interface, object abstractions, and detailed behavioral semantics". Abstractions are wireless links, nodes, and so on; possible functions include state inquiry and manipulation, sending and transmitting of data, access to hardware (sensors, actuators, transceivers), and setting of policies, for example, with respect to energy/quality trade-offs. While such a general API would be extremely useful, there is currently no clear standard – or even an in-depth discussion – arising from the literature. Some first steps in this direction are more concerned with the networking architecture, not so much with accessing functionality on a single node. Until this changes, de facto standards will continue to be used and are likely to serve reasonably well.

Structure of Operating System and Protocol Stack

The traditional approach to communication protocol structuring is to use layering: individual protocols are stacked on top of each other, each layer only using functions of the layer directly below. This layered approach has great benefits in keeping the entire protocol stack manageable, in containing complexity, and in promoting modularity and reuse. For the purposes of a WSN, however, it is not clear whether such a strictly layered approach will suffice (the presentation here follows to some degree reference.

As an example, consider the use of information about the strength of the signal received from a communication partner. This physical layer information can be used to assist in networking protocols to decide about routing changes (a signal becomes weaker if a node moves away and should perhaps no longer be used as a next hop), to compute location information by estimating distance from the signal strength, or to assist link layer protocols in channel-adaptive or hybrid FEC/ARQ schemes. Hence, one single source of information can be used to the advantage of many other protocols not directly associated with the source of this information.

Such cross-layer information exchange is but one way to loosen the strict confinements of the layered approach. Also, WSNs are not the only reason why such liberations are sought. Even in traditional network scenarios, efficiency considerations, the need to support wired networking protocols in wireless systems (e.g. TCP over wireless), the need to migrate functionality into the backbone despite the prescriptions of Internet's end-to-end model, or the desire to support handover mechanisms by physical layer information in cellular networks all have created a considerable pressure for a flexible, manageable, and efficient way of structuring and implementing communication protocols.

When departing from the layered architecture, the prevalent trend is to use a component model. Relatively large, monolithic layers are broken up into small, self-contained "components", "building blocks", or "modules" (the terminology varies). These components only fulfil one well-defined function each – for example, computation of a Cyclic Redundancy Check (CRC) – and interact with each other over clear interfaces. The main difference compared to the layered architecture is that these interactions are not confined to immediate neighbours in an up/down relationship, but can be with any other component.

This component model not only solves some of the structuring problems for protocol stacks, it also fits naturally with an event-based approach to programming wireless sensor nodes. Wrapping of hardware, communication primitives, in-network processing functionalities all can be conveniently designed and implemented as components.

TinyOS uses the notion of explicit wiring of components to allow event exchange to take place between them. While this is beneficial for "push" types of interactions (events are more or less immediately distributed to the receiving component), it does not serve well other cases where a "pull" type of information exchange is necessary. Looking at the case of the received signal strength information described above, the receiving component might not be interested in receiving all such events; rather, it might suffice to be informed asynchronously. A good solution for this is a blackboard, based on publish/subscribe principles, where information can be deposited and anonymously exchanged, allowing a looser coupling between components.

Dynamic Energy and Power Management

Switching individual components into various sleep states or reducing their performance by scaling down frequency and supply voltage and selecting particular modulation and codings were the prominent examples for improving energy efficiency. To control these possibilities, decisions have to be made by the operating system, by the protocol stack, or potentially by an application when to switch into one of these states. Dynamic Power Management (DPM) on a system level is the problem at hand.

One of the complicating factors to DPM is the energy and time required for the transition of a component between any two states. If these factors were negligible, clearly it would be optimal to always & immediately go into the mode with the lowest power consumption possible. As this is not the case, more advanced algorithms are required, taking into account these costs, the rate of updating power management decisions, the probability distribution of time until future events, and properties of the used algorithms.

Probabilistic state transition policies

Consider the problem of policies that regulate the transition between various sleep states. They start out by considering sensors randomly distributed over a fixed area and assume that events arrive with certain temporal distributions (Poisson process) and spatial distributions. This allows them to compute probabilities for the time to the next event, once an event has been processed (even for moving events). They use this probability to select the deepest sleep state out of several possible ones that still fulfil the threshold requirements. In addition, they take into account the possibility of missing events when the sensor as such is also shut down in sleep mode.

Controlling dynamic voltage scaling

To turn the possibilities of DVS into a technical solution also requires some further considerations. For example, it is the rare exception that there is only a single task to be run in an operating system; hence, a clever scheduler is required to decide which clock rate to use in each situation to meet all deadlines. This can require feedback from applications and has been mostly studied in “traditional” applications. Another approach incorporates dynamic voltage scaling control into the kernel of the operating system and achieves energy efficiency improvements in mixed workloads without modifications to user programs.

Trading off fidelity against energy consumption

Most of the just described work on controlling DVS assumes hard deadlines for each task (the task has to be completed by a given time, otherwise its results are useless). In WSNs, such an assumption is often not appropriate. Rather, there are often tasks that can be computed with a higher or lower level of accuracy. The fidelity achieved by such tasks is a candidate for trading it off against other resources. When time is considered, the concept of “imprecise computation” results. In a WSN, the natural trade-off is against energy required to compute a task. Essentially, the question arises again how best to invest a given amount of energy available for a given task. Deliberately embracing such inaccuracies in return for lower energy consumption is a characteristic feature of WSNs; some examples will be discussed in various places in the book. Some approaches to exploit such trade-offs have been described in the literature, for example, in references, but mostly in the context of multimedia systems. Also, discuss the energy-quality trade-off for algorithm design, especially for signal processing purposes (filtering, frequency domain transforms, and classification). The idea is to transform an algorithm such that it quickly approximates the final result and keeps computing as long as energy is available, producing incremental refinements (being a direct counterpart to imprecise computation, where computation can continue as long as time is available). The performance of such (original or transformed) algorithms is studied using their $E - Q$ metric, indicating which (normalized) result quality can be achieved for how much (normalized) energy.

CASE STUDY: *tinyOS* and *nesC*

The use of an event-based programming model as the only feasible way to support the concurrency required for sensor node software while staying within the confined resources and running on top of the simple hardware provided by these nodes. The open question is how to harness the power of this programming model without getting lost in the complexity of many individual state machines sending each other events. In addition, modularity should be supported to easily exchange one state machine against another. The operating system *TinyOS*, along with the programming language *nesC*, addresses these challenges.

TinyOS supports modularity and event-based programming by the concept of components. A component contains semantically related functionality, for example, for handling a radio interface or for computing routes. Such a component comprises the required state information in a frame, the program code for normal tasks, and handlers for events and commands. Both events and commands are exchanged between different components. Components are arranged hierarchically, from low-level components close to the hardware to high-level components making up the actual application. Events originate in the hardware and pass upward from low-level to high-level components; commands, on the other hand, are passed from high-level to low-level components.

Figure shows a timer component that provides a more abstract version of a simple hardware time. It understands three commands (“init”, “start”, and “stop”) and can handle one event ("fire") from another component, for example, a wrapper component around a hardware timer. It issues "setRate" commands to this component and can emit a "fired" event itself.

The important thing to note is that, in staying with the event-based paradigm, both command and event handlers must run to conclusion; they are only supposed to perform very simple triggering duties. In particular, commands must not block or wait for an indeterminate amount of time; they are simply a request upon which some task of the hierarchically lower component has to act. Similarly, an event handler only leaves information in its component's frame and arranges for a task to be executed later; it can also send commands to other components or directly report an event further up.

The actual computational work is done in the tasks. In *TinyOS*, they also have to run to completion, but can be interrupted by handlers. The advantage is twofold: there is no need for stack management and tasks are atomic with respect to each other. Still, by virtue of being triggered by handlers, tasks are seemingly concurrent to each other.

The arbitration between tasks – multiple can be triggered by several events and are ready to execute – is done by a simple, power-aware First In First Out (FIFO) scheduler, which shuts the node down when there is no task executing or waiting.

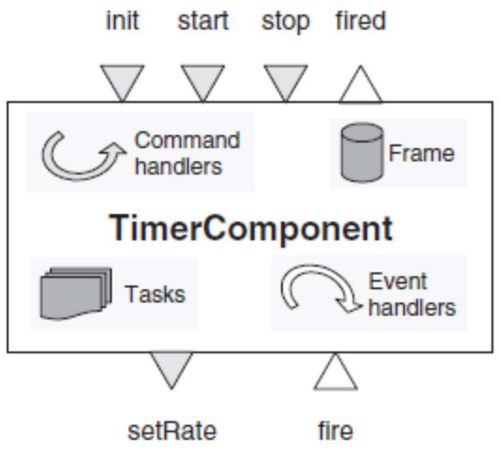


Figure 2.9 Example Timer component (adapted from references [285, 353])

With handlers and tasks all required to run to completion, it is not clear how a component could obtain feedback from another component about a command that it has invoked there – for example, how could an Automatic Repeat Request (ARQ) protocol learn from the MAC protocol whether a packet had been sent successfully or not? The idea is to split invoking such a request and the information about answers into two phases: The first phase is the sending of the command, the second is an explicit information about the outcome of the operation, delivered by a separate event. This split-phase programming approach requires for each command a matching event but enables concurrency under the constraints of run-to-completion semantics – if no confirmation for a command is required, no completion event is necessary.

Having commands and events as the only way of interaction between components (the frames of components are private data structures), and especially when using split-phase programming, a large number of commands and events add up in even a modestly large program. Hence, an abstraction is necessary to organize them. As a matter of fact, the set of commands that a component understands and the set of events that a component may emit are its interface to the components of a hierarchically higher layer; looked at it the other way around, a component can invoke certain commands at its lower component and receive certain events from it. Therefore, structuring commands and events that belong together forms an interface between two components.

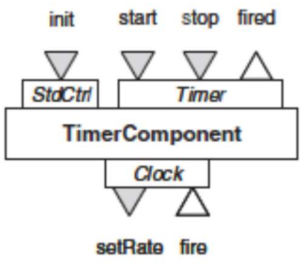


Figure 2.10 Organizing the Timer component using interfaces [285, 353]

Listing 2.1: Defining modules and interfaces [285]

```
interface StdCtrl {
  command result_t init();
}

interface Timer {
  command result_t start (char type, uint32_t interval);
  command result_t stop ();
  event result_t fired();
}

interface Clock {
  command result_t setRate (char interval , char scale);
  event result_t  fire ();
}

module TimerComponent {
  provides {
    interface StdCtrl;
    interface Timer;
  }
  uses interface Clock as Clk;
}
```

The *nesC* language formalizes this intuition by allowing a programmer to define interface types that define commands and events that belong together. This allows to easily express split-phase programming style by putting commands and their corresponding completion events into the same interface. Components then provide certain interfaces to their users and in turn use other interfaces from underlying components.

Figure shows how the Timer component of the previous example can be reorganized into using a clock interface and providing two interfaces *StdCtrl* and Timer. The corresponding *nesC* code is shown in Listing *1*. Note that the component *TimerComponent* is defined here as a module since it is a primitive component, directly containing handlers and tasks.

Such primitive components or modules can be combined into larger configurations by simply "wiring" appropriate interfaces together. For this wiring to take place, only components that have the correct interface types can be plugged together (this is checked by the compiler). Figure shows how the *TimerComponent* and an additional component *HWClock* can be wired together to form a new component *CompleteTimer*, exposing only the *StdCtrl* and Timer interfaces to the outside; Listing *2* shows the corresponding *nesC* code. Note that both modules and configurations are components.

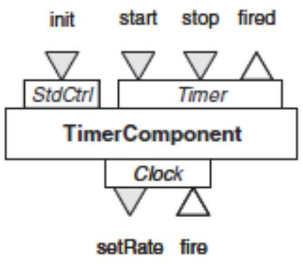


Figure 2.11 Building a larger configuration out of two components [285, 353]

Listing 2.2: Wiring components to form a configuration [285]

```
configuration CompleteTimer {
  provides {
    interface StdCtrl;
    interface Timer;
  }
  implementation {
    components TimerComponent , HWClock;
    StdCtrl - TimerComponent.HWClock;
    Timer - TimerComponent.Timer;
    TimerComponent.Clk - HWClock.Clock;
  }
}
```

Using these component definition, implementation, and connection concepts, *TinyOS* and *nesC* together form a powerful and relatively easy to use basis to implement both core operating system functionalities as well as communication protocol stacks and application functions. Programmers do use these paradigms and arrive at relatively small, highly specialized components that are then combined as needed, proving the modularity claim. Also, code size and memory requirements are quite small.

Overall, *TinyOS* can currently be regarded as the standard implementation platform for WSNs. It is also becoming available for an increasing number of platforms other than the original “motes” on which it had been developed. On top of the *TinyOS* operating system, a vast range of extensions, protocols, and applications have been developed. A virtual machine concept describes on top of *TinyOS* that provides a high-level interface to concisely represent programs; it is particularly beneficial for over-the-air reprogramming and retasking of an existing network. Conceiving of the sensor network as a relational database is made possible by the *TinyDB* project.

Other examples

Apart from TinyOS, there are a few other execution environments or operating systems for WSN nodes. One example is Contiki10, which has been ported to various hardware platforms and actually implements a TCP/IP stack on top of a platform with severely restricted resources. Other examples are ecos and the Mantis project.

Some Examples of Sensor Nodes

There are quite a number of actual nodes available for use in wireless sensor network research and development. Again, depending on the intended application scenarios, they have to fulfill quite different requirements regarding battery life, mechanical robustness of the node’s housing, size, and so on.

The “Mica Mote” family

Starting in the late 1990s, an entire family of nodes has evolved out of research projects at the University of California at Berkeley, partially with the collaboration of Intel, over the years. They are commonly known as the Mica motes11, with different versions (Mica, Mica2, Mica2Dot) having been designed. They are commercially available via the company Crossbow12 in different versions and different kits. TinyOS is the usually used operating system for these nodes.

All these boards feature a microcontroller belonging to the Atmel family, a simple radio modem (usually a TR 1000 from RFM), and various connections to the outside. In addition, it is possible to connect additional "sensor boards" with, for example, barometric or humidity sensors, to the node as such, enabling a wider range of applications and experiments. Also, specialized enclosures have been built for use in rough environments, for example, for monitoring bird habitats. Sensors are connected to the controller via an I2C bus or via SPI, depending on the version.

EYES nodes

The nodes developed by Infineon in the context of the European Union – sponsored project "Energyefficient Sensor Networks" (EYES) 13 are another example of a typical sensor node (Figure 2.13). It is equipped with a Texas Instrument MSP 430 microcontroller, an Infineon radio modem TDA 5250, along with a SAW filter and transmission power control; the radio modem also reports the measured signal strength to the controller. The node has a USB interface to a PC and the possibility to add additional sensors/actuators.

BTnodes

The "Btnodes" have been developed at the ETH Z¨urich out of several research projects. They feature an Atmel ATmega 128L microcontroller, 64 + 180 kB RAM, and 128 kB FLASH memory. Unlike most other sensor nodes (but similar to some nodes developed by Intel), they use Bluetooth as their radio technology in combination with a Chipcon CC1000 operating between 433 and 915 MHz.

A other examples shall highlight typical approaches; an overview of current developments can be found, in the textbook.

Commercial solutions

Apart from these academic research prototypes, there are already a couple of sensor-node-type devices commercially available, including appropriate housing, certification, and so on. Some of these companies include "ember" (www.ember.com) or "Millenial" (www.millenial.net). The market here is more dynamic than can be reasonably reflected in a textbook and the reader is encouraged to watch for up-to-date developments.

2. NETWORK ARCHITECTURE

The architecture of wireless sensor networks draws upon many sources. Historically, a lot of related work has been done in the context of self-organizing, mobile, ad hoc networks. While these networks are intended for different purposes, they share the need for a decentralized, distributed form of organization. From a different perspective, sensor networks are related to real-time computing and even to some concepts from peer-to-peer computing, active networks, and mobile agents/swarm intelligence.

SENSOR NETWORK SCENARIOS

Types of Sources and Sinks

Several typical interaction patterns found in WSNs – event detection, periodic measurements, function approximation and edge detection, or tracking – it has also already briefly touched upon the definition of "sources" and "sinks". A source is any entity in the network that can provide information, that is, typically a sensor node; it could also be an actuator node that provides feedback about an operation.

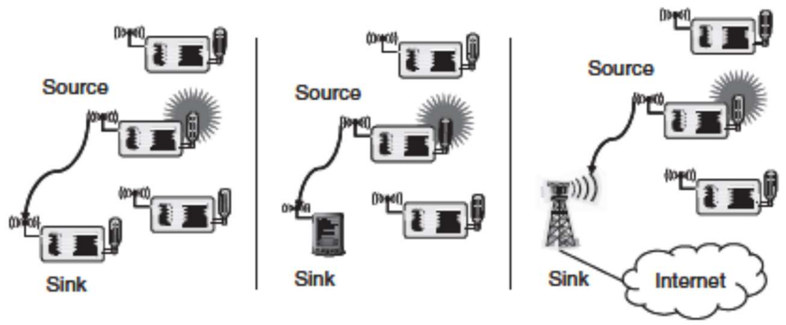


Figure 3.1 Three types of sinks in a very simple, single-hop sensor network

A sink, on the other hand, is the entity where information is required. There are essentially three options for a sink: it could belong to the sensor network as such and be just another sensor/actuator node or it could be an entity outside this network. For this second case, the sink could be an actual device, for example, a handheld or PDA used to interact with the sensor network; it could also be merely a gateway to another larger network such as the Internet, where the actual request for the information comes from some node "far away" and only indirectly connected to such a sensor network. These main types of sinks are illustrated by Figure 1, showing sources and sinks in direct communication. It is important, whether sources or sinks move, but what they do with the information is not a primary concern of the networking architecture.

Single-hop versus Multihop Networks

From the basics of radio communication and the inherent power limitation of radio communication follows a limitation on the feasible distance between a sender and a receiver. Because of this limited distance, the simple, direct communication between source and sink is not always possible, specifically in WSNs, which are intended to cover a lot of ground (e.g. in environmental or agriculture applications) or that operate in difficult radio environments with strong attenuation (e.g. in buildings).

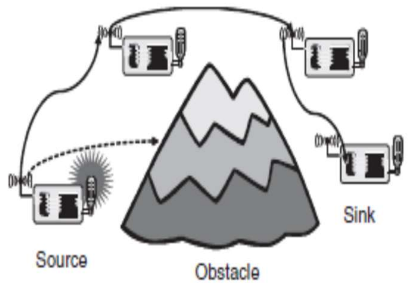


Figure 3.2 Multihop networks: As direct communication is impossible because of distance and/or obstacles, multihop communication can circumvent the problem

To overcome such limited distances, an obvious way out is to use relay stations, with the data packets taking multi hops from the source to the sink. This concept of multihop networks (illustrated in Figure 2) is particularly attractive for WSNs as the sensor nodes themselves can act as such relay nodes, foregoing the need for additional equipment. Depending on the particular application, the likelihood of having an intermediate sensor node at the right place can actually be quite high – for example, when a given area has to be uniformly equipped with sensor nodes anyway – but nevertheless, there is not always a guarantee that such multihop routes from source to sink exist, nor that such a route is particularly short.

While multihopping is an evident and working solution to overcome problems with large distances or obstacles, it has also been claimed to improve the energy efficiency of communication. The intuition behind this claim is that, as attenuation of radio signals is at least quadratic in most environments (and usually larger), it consumes less energy to use relays instead of direct communication:

When targeting for a constant SNR at all receivers (assuming for simplicity negligible error rates at this SNR), the radiated energy required for direct communication over a distance *d* is cdα (c some constant, $\alpha \geq 2$ the path loss coefficient); using a relay at distance *d/2* reduces this energy to *2c(d/2)α.*

But this calculation considers only the radiated energy, not the actually consumed energy – in particular, the energy consumed in the intermediate relay node. Even assuming that this relay belongs to the WSN and is willing to cooperate, when computing the total required energy it is necessary to take into account the complete power consumption. It is an easy exercise to show that energy is actually wasted if intermediate relays are used for short distances d. Only for large d does the radiated energy dominate the fixed energy costs consumed in transmitter and receiver electronics – the concrete distance where direct and multihop communication are in balance depends on a lot of device-specific and environment-specific parameters. Nonetheless, this relationship is often not considered. The classification of the misconception that multihopping saves energy as the number one myth about energy consumption in wireless communication. Great care should be taken when applying multihopping with the end of improved energy efficiency.

It should be pointed out that only multihop networks operating in a store and forward fashion are considered here. In such a network, a node has to correctly receive a packet before it can forward it somewhere. Alternative, innovative approaches attempt to exploit even erroneous reception of packets, for example, when multiple nodes send the same packet and each individual transmission could not be received, but collectively, a node can reconstruct the full packet. Such cooperative relaying techniques are not considered here.

Multiple Sinks and Sources

So far, only networks with a single source and a single sink have been illustrated. In many cases, there are multiple sources and/or multiple sinks present. In the most challenging case, multiple sources should send information to multiple sinks, where either all or some of the information has to reach all or some of the sinks. Figure 3 illustrates these combinations.

Three types of mobility

In the scenarios discussed above, all participants were stationary. But one of the main virtues of wireless communication is its ability to support mobile participants. In wireless sensor networks, mobility can appear in three main forms:

Node mobility The wireless sensor nodes themselves can be mobile. The meaning of such mobility is highly application dependent. In examples like environmental control, node mobility should not happen; in livestock surveillance (sensor nodes attached to cattle, for example), it is the common rule.

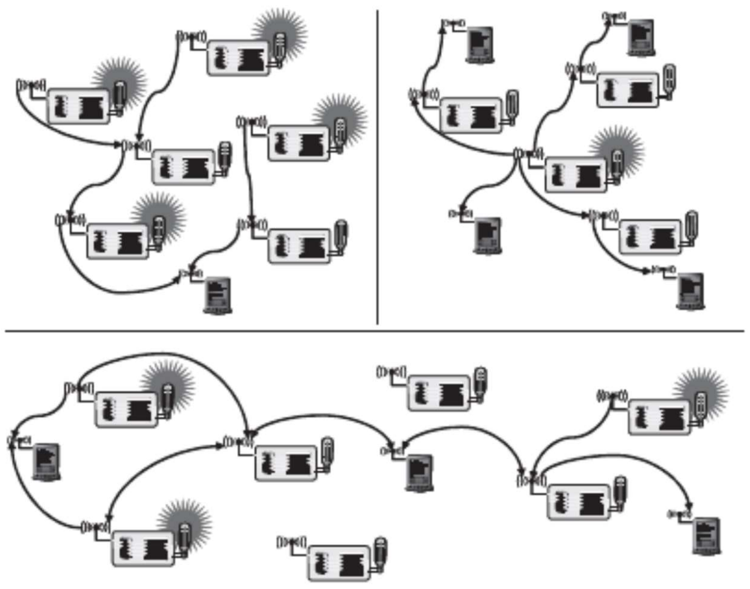


Figure 3.3 Multiple sources and/or multiple sinks. Note how in the scenario in the lower half, both sinks and active sources are used to forward data to the sinks at the left and right end of the network

In the face of node mobility, the network has to reorganize itself frequently enough to be able to function correctly. It is clear that there are trade-offs between the frequency and speed of node movement on the one hand and the energy required to maintain a desired level of functionality in the network on the other hand.

Sink mobility The information sinks can be mobile (Figure 4). While this can be a special case of node mobility, the important aspect is the mobility of an information sink that is not part of the sensor network, for example, a human user requested information via a PDA while walking in an intelligent building.

In a simple case, such a requester can interact with the WSN at one point and complete its interactions before moving on. In many cases, consecutive interactions can be treated as separate, unrelated requests. Whether the requester is allowed interactions with any node or only with specific nodes is a design choice for the appropriate protocol layers. A mobile requester is particularly interesting, however, if the requested data is not locally available but must be retrieved from some remote part of the network. Hence, while the requester would likely communicate only with nodes in its vicinity, it might have moved to some other place. The network, possibly with the assistance of the mobile requester, must make provisions that the requested data actually follows and reaches the requester despite its movements.

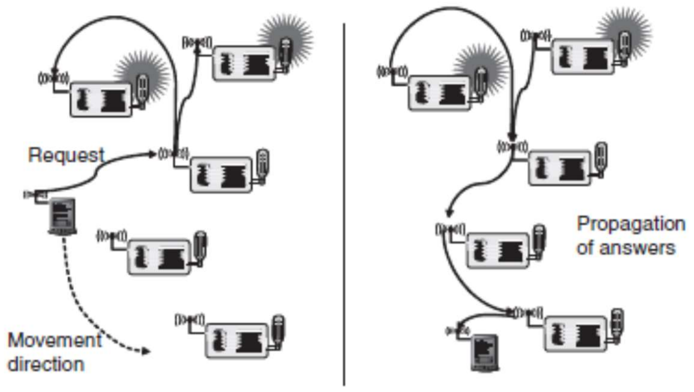


Figure 3.4 A mobile sink moves through a sensor network as information is being retrieved on its behalf

Event mobility In applications like event detection and in particular in tracking applications, the cause of the events or the objects to be tracked can be mobile. In such scenarios, it is (usually) important that the observed event is covered by a sufficient number of sensors at all time. Hence, sensors will wake up around the object, engaged in higher activity to observe the present object, and then go back to sleep. As the event source moves through the network, it is accompanied by an area of activity within the network – this has been called the frisbee model (which also describes algorithms for handling the "wakeup wavefront"). This notion is described by Figure 5, where the task is to detect a moving elephant and to observe it as it moves around.

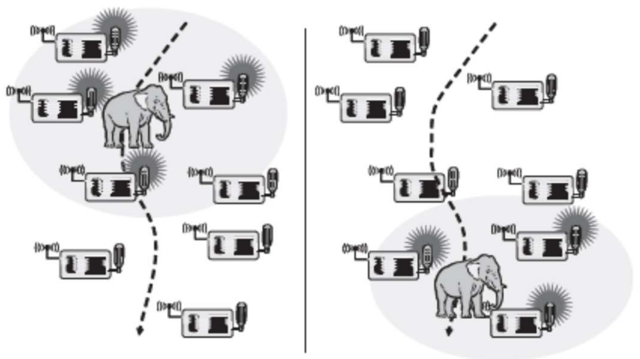


Figure 3.5 Area of sensor nodes detecting an event – an elephant [378] – that moves through the network along with the event source (dashed line indicate the elephant's trajectory; shaded ellipse the activity area following or even preceding the elephant)

Nodes that do not actively detect anything are intended to switch to lower sleep states unless they are required to convey information from the zone of activity to some remote sink (not shown in Figure 5). Communication protocols for WSNs will have to render appropriate support for these forms of mobility. In particular, event mobility is quite uncommon, compared to previous forms of mobile or wireless networks.

OPTIMIZATION GOALS AND FIGURES OF MERIT

For all these scenarios and application types, different forms of networking solutions can be found. The challenging question is how to optimize a network, how to compare these solutions, how to decide which approach better supports a given application, and how to turn relatively imprecise optimization goals into measurable figures of merit? While a general answer appears impossible considering the large variety of possible applications, a few aspects are fairly evident.

Quality of Service

WSNs differ from other conventional communication networks mainly in the type of service they offer. These networks essentially only move bits from one place to another. Possibly, additional requirements about the offered Quality of Service (QoS) are made, especially in the context of multimedia applications. Such QoS can be regarded as a low-level, networking-device-observable attribute – bandwidth, delay, jitter, packet loss rate – or as a high-level, user-observable, so-called subjective attribute like the perceived quality of a voice communication or a video transmission. While the first kind of attributes is applicable to a certain degree to WSNs as well (bandwidth, for example, is quite unimportant), the second one clearly is not, but is really the more important one to consider! Hence, high-level QoS attributes corresponding to the subjective QoS attributes in conventional networks are required.

But just like in traditional networks, high-level QoS attributes in WSN highly depend on the application. Some generic possibilities are:

Event detection/reporting probability What is the probability that an event that actually occurred is not detected or, more precisely, not reported to an information sink that is interested in such an event? For example, not reporting a fire alarm to a surveillance station would be a severe shortcoming.

Clearly, this probability can depend on/be traded off against the overhead spent in setting up structures in the network that support the reporting of such an event (e.g. routing tables) or against the run-time overhead (e.g. sampling frequencies).

Event classification error If events are not only to be detected but also to be classified, the error in classification must be small.

Event detection delay What is the delay between detecting an event and reporting it to any/all interested sinks?

Missing reports In applications that require periodic reporting, the probability of undelivered reports should be small.

Approximation accuracy For function approximation applications (e.g. approximating the temperature as a function of location for a given area), what is the average/maximum absolute or relative error with respect to the actual function? Similarly, for edge detection applications, what is the accuracy of edge descriptions; are some missed at all?

Tracking accuracy Tracking applications must not miss an object to be tracked, the reported position should be as close to the real position as possible, and the error should be small. Other aspects of tracking accuracy are, for example, the sensitivity to sensing gaps.

Energy Efficiency

Much of the discussion has already shown that energy is a precious resource in wireless sensor networks and that energy efficiency should therefore make an evident optimization goal. It is clear that with an arbitrary amount of energy, most of the QoS metrics defined above can be increased almost at will (approximation and tracking accuracy are notable exceptions as they also depend on the density of the network). Hence, putting the delivered QoS and the energy required to do so into perspective should give a first, reasonable understanding of the term energy efficiency.

The term “energy efficiency” is, in fact, rather an umbrella term for many different aspects of a system, which should be carefully distinguished to form actual, measurable figures of merit. The most commonly considered aspects are:

Energy per correctly received bit How much energy, counting all sources of energy consumption at all possible intermediate hops, is spent on average to transport one bit of information (payload) from the source to the destination? This is often a useful metric for periodic monitoring applications.

Energy per reported (unique) event Similarly, what is the average energy spent to report one event? Since the same event is sometimes reported from various sources, it is usual to normalize this metric to only the unique events (redundant information about an already known event does not provide additional information).

Delay/energy trade-offs Some applications have a notion of "urgent" events, which can justify an increased energy investment for a speedy reporting of such events. Here, the trade-off between delay and energy overhead is interesting.

Network lifetime The time for which the network is operational or, put another way, the time during which it is able to fulfill its tasks (starting from a given amount of stored energy). It is not quite clear, however, when this time ends. Possible definitions are:

> ***Time to first node death*** When does the first node in the network run out of energy or fail and stop operating?
>
> ***Network half-life*** When have 50% of the nodes run out of energy and stopped operating? Any other fixed percentile is applicable as well.
>
> ***Time to partition*** When does the first partition of the network in two (or more) disconnected parts occur? This can be as early as the death of the first node (if that was in a pivotal position) or occur very late if the network topology is robust.

Time to loss of coverage Usually, with redundant network deployment and sensors that can observe a region instead of just the very spot where the node is located, each point in the deployment region is observed by multiple sensor nodes. A possible figure of merit is thus the time when for the first time any spot in the deployment region is no longer covered by any node's observations. If k redundant observations are necessary (for tracking applications, for example), the corresponding definition of loss of coverage would be the first time any spot in the deployment region is no longer covered by at least k different sensor nodes.

Time to failure of first event notification A network partition can be seen as irrelevant if the unreachable part of the network does not want to report any events in the first place. Hence, a possibly more application-specific interpretation of partition is the inability to deliver an event. This can be due to an event not being noticed because the responsible sensor is dead or because a partition between source and sink has occurred.

It should be noted that simulating network lifetimes can be a difficult statistical problem. Obviously, the longer these times are, the better does a network perform. More generally, it is also possible to look at the (complementary) distribution of node lifetimes (with what probability does a node survive a given amount of time?) or at the relative survival times of a network (at what time are how many percent of the nodes still operational?). This latter function allows an intuition about many WSN-specific protocols in that they tend to sacrifice long lifetimes in return for an improvement in short lifetimes – they "sharpen the drop" (Figure 6).

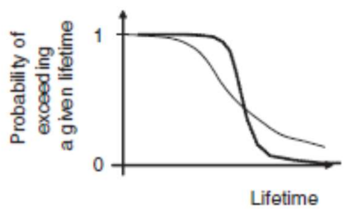


Figure 3.6 Two probability curves of a node exceeding a given lifetime – the dotted curve trades off better minimal lifetime against reduced maximum lifetime

All these metrics can of course only be evaluated under a clear set of assumptions about the energy consumption characteristics of a given node, about the actual "load" that the network has to deal with (e.g. when and where do events happen), and also about the behaviour of the radio channel.

Scalability

The ability to maintain performance characteristics irrespective of the size of the network is referred to as scalability. With WSN potentially consisting of thousands of nodes, scalability is an evidently indispensable requirement. Scalability is ill served by any construct that requires globally consistent state, such as addresses or routing table entries that have to be maintained. Hence, the need to restrict such information is enforced by and goes hand in hand with the resource limitations of sensor nodes, especially with respect to memory.

The need for extreme scalability has direct consequences for the protocol design. Often, a penalty in performance or complexity has to be paid for small. Architectures and protocols should implement appropriate scalability support rather than trying to be as scalable as possible. Applications with a few dozen nodes might admit more efficient solutions than applications with thousands of nodes; these smaller applications might be more common in the first place. Nonetheless, a considerable amount of research has been invested into highly scalable architectures and protocols.

Robustness

Related to QoS and somewhat also to scalability requirements, wireless sensor networks should also exhibit an appropriate robustness. They should not fail just because a limited number of nodes run out of energy, or because their environment changes and severs existing radio links between two nodes – if possible, these failures have to be compensated for, for example, by finding other routes. A precise evaluation of robustness is difficult in practice and depends mostly on failure models for both nodes and communication links.

3. DESIGN PRINCIPLES FOR WSNs

Appropriate QoS support, energy efficiency, and scalability are important design and optimization goals for wireless sensor networks. But these goals themselves do not provide many hints on how to structure a network such that they are achieved. A few basic principles have emerged, which can be useful when designing networking protocols. Nonetheless, the general advice to always consider the needs of a concrete application holds here as well – for each of these basic principles, there are examples where following them would result in inferior solutions.

DISTRIBUTED ORGANIZATION

Both the scalability and the robustness optimization goal, and to some degree also the other goals, make it imperative to organize the network in a distributed fashion. That means that there should be no centralized entity in charge – such an entity could, for example, control medium access or make routing decisions, similar to the tasks performed by a base station in cellular mobile networks. The disadvantages of such a centralized approach are obvious as it introduces exposed points of failure and is difficult to implement in a radio network, where participants only have a limited communication range. Rather, the WSNs nodes should cooperatively organize the network, using distributed algorithms and protocols. Self-organization is a commonly used term for this principle.

When organizing a network in a distributed fashion, it is necessary to be aware of potential shortcomings of this approach. In many circumstances, a centralized approach can produce solutions that perform better or require less resources (in particular, energy). To combine the advantages, one possibility is to use centralized principles in a localized fashion by dynamically electing, out of the set of equal nodes, specific nodes that assume the responsibilities of a centralized agent, for example, to organize medium access. Such elections result in a hierarchy, which has to be dynamic:

The election process should be repeated continuously lest the resources of the elected nodes be overtaxed, the elected node runs out of energy, and the robustness disadvantages of such – even only localized – hierarchies manifest themselves. The particular election rules and triggering conditions for re-election vary considerably, depending on the purpose for which these hierarchies are used.

IN-NETWORK PROCESSING

When organizing a network in a distributed fashion, the nodes in the network are not only passing on packets or executing application programs, they are also actively involved in taking decisions about how to operate the network. This is a specific form of information processing that happens in the network, but is limited to information about the network itself. It is possible to extend this concept by also taking the concrete data that is to be transported by the network into account in this information processing, making in-network processing a first-rank design principle.

Several techniques for in-network processing exist, and by definition, this approach is open to an arbitrary extension – any form of data processing that improves an application is applicable.

Aggregation

Perhaps the simplest in-network processing technique is aggregation. Suppose a sink is interested in obtaining periodic measurements from all sensors, but it is only relevant to check whether the average value has changed, or whether the difference between minimum and maximum value is too big. In such a case, it is evidently not necessary to transport are readings from all sensors to the sink, but rather, it suffices to send the average or the minimum and maximum value. The transmitting data is considerably more expensive than even complex computation shows the great energy-efficiency benefits of this approach. The name aggregation stems from the fact that in nodes intermediate between sources and sinks, information is aggregated into a condensed form out of information provided by nodes further away from the sink (and potentially, the aggregator's own readings).

Clearly, the aggregation function to be applied in the intermediate nodes must satisfy some conditions for the result to be meaningful; most importantly, this function should be composable. A further classification of aggregate functions distinguishes duplicate-sensitive versus insensitive, summary versus exemplary, monotone versus nonmonotone, and algebraic versus holistic. Functions like average, counting, or minimum can profit a lot from aggregation; holistic functions like the median are not amenable to aggregation at all.

Figure illustrates the idea of aggregation. In the left half, a number of sensors transmit readings to a sink, using multihop communication. In total, 13 messages are required (the numbers in the figure indicate the number of messages traveling across a given link). When the highlighted nodes perform aggregation – for example, by computing average values (shown in the right half of the figure) – only 6 messages are necessary.

Challenges in this context include how to determine where to aggregate results from which nodes, how long to wait for such results, and determining the impact of lost packets.

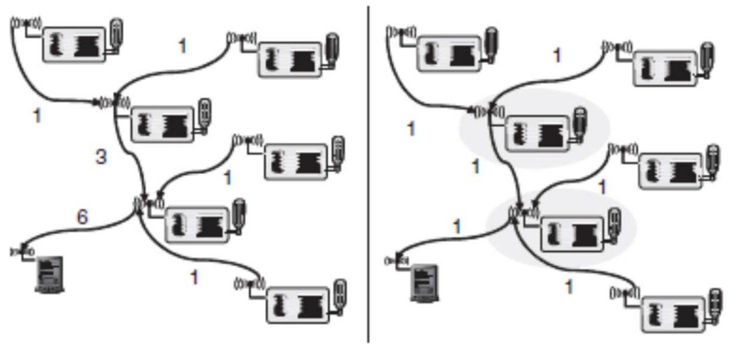


Figure 3.7 Aggregation example

Distributed Source Coding and Distributed Compression

Aggregation condenses and sacrifices information about the measured values in order not to have to transmit all bits of data from all sources to the sink. Is it possible to reduce the number of transmitted bits (compared to simply transmitting all bits) but still obtain the full information about all sensor readings at the sink?

While this question sounds surprising at first, it is indeed possible to give a positive answer. It is related to the coding and compression problems known from conventional networks, where a lot of effort is invested to encode, for example, a video sequence, to reduce the required bandwidth. The problem here is slightly different, in that we are interested to encode the information provided by several sensors, not just by a single camera; moreover, traditional coding schemes tend to put effort into the encoding, which might be too computationally complex for simple sensor nodes.

How can the fact that information is provided by multiple sensors be exploited to help in coding? If the sensors were connected and could exchange their data, this would be conceivable (using relatively standard compression algorithms), but of course pointless. Hence, some implicit, joint information between two sensors is required. Recall here that these sensors are embedded in a physical environment – it is quite likely that the readings of adjacent sensors are going to be quite similar; they are correlated. Such correlation can indeed be exploited such that not simply the sum of the data must be transmitted but that overhead can be saved here.

Slepian-Wolf theorem–based work is an example of exploiting spatial correlation that is commonly present in sensor readings, as long as the network is sufficiently dense, compared to the derivate of the observed function and the degree of correlation between readings at two places. Similarly, temporal correlation can be exploited in sensor network protocols.

Distributed and Collaborative Signal Processing

The in-networking processing approaches discussed so far have not really used the ability for processing in the sensor nodes, or have only used this for trivial operations like averaging or finding the maximum. When complex computations on a certain amount of data is to be done, it can still be more energy efficient to compute these functions on the sensor nodes despite their limited processing power, if in return the amount of data that has to be communicated can be reduced.

An example for this concept is the distributed computation of a Fast Fourier Transform (FFT). Depending on where the input data is located, there are different algorithms available to compute an FFT in a distributed fashion, with different trade-offs between local computation complexity and the need for communication. In principle, this is similar to algorithm design for parallel computers. However, here not only the latency of communication but also the energy consumption of communication and computation are relevant parameters to decide between various algorithms. Such distributed computations are mostly applicable to signal processing type algorithms; typical examples are beamforming and target tracking applications.

Mobile code/Agent-based Networking

With the possibility of executing programs in the network, other programming paradigms or computational models are feasible. One such model is the idea of mobile code or agent-based networking. The idea is to have a small, compact representation of program code that is small enough to be sent from node to node. This code is then executed locally, for example, collecting measurements, and then decides where to be sent next. This idea has been used in various environments; a classic example is that of a software agent that is sent out to collect the best possible travel itinerary by hopping from one travel agent's computer to another and eventually returning to the user who has posted this inquiry.

Also, virtual machines for WSNs have been proposed that have a native language that admits a compact representation of the most typical operations that mobile code in a WSN would execute, allowing this code to be small.

ADAPTIVE FIDELITY AND ACCURACY

Making the fidelity of computation results contingent upon the amount of energy available for that particular computation. This notion can and should be extended from a single node to an entire network. As an example, consider a function approximation application. Clearly, when more sensors participate in the approximation, the function is sampled at more points and the approximation is better. But in return for this, more energy has to be invested. Similar examples hold for event detection and tracking applications and in general for WSNs.

Hence, it is up to an application to somehow define the degree of accuracy of the results (assuming that it can live with imprecise, approximated results) and it is the task of the communication protocols to try to achieve at least this accuracy as energy efficiently as possible. Moreover, the application should be able to adapt its requirements to the current status of the network – how many nodes have already failed, how much energy could be scavenged from the environment, what are the operational conditions (have critical events happened recently), and so forth. Therefore, the application needs feedback from the network about its status to make such decisions.

DATA CENTRICITY

Address Data, Not Nodes

In traditional communication networks, the focus of a communication relationship is usually the pair of communicating peers – the sender and the receiver of data. In a wireless sensor network, on the other hand, the interest of an application is not so much in the identity of a particular sensor node, it is much rather in the actual information reported about the physical environment. This is especially the case when a WSN is redundantly deployed such that any given event could be reported by multiple nodes – it is of no concern to the application precisely which of these nodes is providing data. This fact that not the identity of nodes but the data are at the center of attention is called data-centric networking. For an application, this essentially means that an interface is exposed by the network where data, not nodes, is addressed in requests. The set of nodes that is involved in such a data-centric address is implicitly defined by the property that a node can contribute data to such an address.

As an example, consider the elephant-tracking example. In a data-centric application, all the application would have to do is state its desire to be informed about events of a certain type – “presence of elephant” – and the nodes in the network that possess “elephant detectors” are implicitly informed about this request. In an identity-centric network, the requesting node would have to find out somehow all nodes that provide this capability and address them explicitly. As another example, it is useful to consider the location of nodes as a property that defines whether a node belongs to a certain group or not. The typical example here is the desire to communicate with all nodes in a given area, say, to retrieve the (average) temperature measured by all nodes in the living room of a given building.

Data-centric networking allows very different networking architectures compared to traditional, identity-centric networks. For one, it is the ultimate justification for some in-network processing techniques like data fusion and aggregation. Data-centric addressing also enables simple expressions of communication relationships – it is no longer necessary to distinguish between one-to-one, one to- many, many-to-one, or many-to-many relationships as the set of participating nodes is only implicitly defined. In addition to this decoupling of identities, data-centric addressing also supports a decoupling in time as a request to provide data does not have to specify when the answer should happen – a property that is useful for event-detection applications, for example.

Apart from providing a more natural way for an application to express its requirements, datacentric networking and addressing is also claimed to improve performance and especially energy efficiency of a WSN. One reason is the hope that data-centric solutions scale better by being implementable using purely local information about direct neighbours. Another reason could be the easier integration of a notion of adaptive accuracy into a data-centric framework as the data as well as its desired accuracy can be explicitly.

IMPLEMENTATION OPTIONS FOR DATA-CENTRIC NETWORKING

There are several possible ways to make this abstract notion of data-centric networks more concrete. Each way implies a certain set of interfaces that would be usable by an application. The three most important ones are briefly sketched here.

Overlay networks and distributed hash tables

There are some evident similarities between well-known peer-to-peer applications like file sharing and WSN: In both cases, the user/requester is interested only in looking up and obtaining data, not in its source; the request for data and its availability can be decoupled in time; both types of networks should scale to large numbers. In peer-to-peer networking, the solution for an efficient lookup of retrieval of data from an unknown source is usually to form an overlay network, implementing a Distributed Hash Table (DHT). The desired data can be identified via a given key (a hash) and the DHT will provide one (or possibly several) sources for the data associated with this key. The crucial point is that this data source lookup can be performed efficiently, requiring O(log n) steps where n is the number of nodes, even with only distributed, localized information about where information is stored in the peer-to-peer network.

Despite these similarities, there are some crucial differences. First of all, it is not clear how the rather static key of a DHT would correspond to the more dynamic, parameterized requests in a WSN. Second, and more importantly, DHTs, coming from an IP-networking background, tend to ignore the distance/the hop count between two nodes and consider nodes as adjacent only on the basis of semantic information about their stored keys. This hop-count-agnostic behaviour is unacceptable for WSNs where each hop incurs considerable communication overhead.

Publish/Subscribe

The required separation in both time and identity of a sink node asking for information and the act of providing this information is not well matched with the synchronous characteristics of a request/reply protocol. What is rather necessary is a means to express the need for certain data and the delivery of the data, where the data as such is specified and not the involved entities.

This behaviour is realized by the publish/subscribe approach: Any node interested in a given kind of data can subscribe to it, and any node can publish data, along with information about its kind as well. Upon a publication, all subscribers to this kind of data are notified of the new data. The elephant example is then easily expressed by sink nodes subscribing to the event "elephant detected"; any node that is detecting an elephant can then, at any later time, publish this event. If a subscriber is no longer interested, it can simply unsubscribe from any kind of event and will no longer be notified of such events. Evidently, subscription and publication can happen at different points in time and the identities of subscribers and publishers do not have to be known to each other.

Implementing this abstract concept of publishing and subscribing to information can be done in various ways. One possibility is to use a central entity where subscriptions and publications are matched to each other, but this is evidently inappropriate for WSNs. A distributed solution is preferable but considerably more complicated.

Also relevant is the expressiveness of the data descriptions (their "names") used to match publications and subscriptions. A first idea is to use explicit subjects or keywords as names, which have to be defined up front – published data only matches to subscriptions with the same keyword (like in the "elephant detected" example above). This subject-based approach can be extended into hierarchical schemes where subjects are arranged in a tree; a subscription to a given subject then also implies interest in any descendent subjects. A more general naming scheme allows to formulate the matching condition between subscriptions and publications as general predicates over the content of the publication and is hence referred to as content-based publish/subscribe approach.

In practice, general predicates on the content are somewhat clumsy to handle and restricted expressions (also called filters) of the form (attribute, value, operator) are preferable, where attribute corresponds to the subjects from above (e.g. temperature) and can assume values, value is a concrete value like "25 ◦ C" or a placeholder (ALL or ANY), and operator is a relational operator like "=", "<", "≤". Moreover, this formalism also lends itself very conveniently to the expression of accuracy requirements or periodic measurement support.

Databases

A somewhat different view on WSN is to consider them as (dynamic) databases. This view matches very well with the idea of using a data-centric organization of the networking protocols. Being interested in certain aspects of the physical environment that is surveyed by a WSN is equivalent to formulating queries for a database.

To cast the sensor networks into the framework of relational databases, it is useful to regard the sensors as a virtual table to which relational operators can be applied. Then, extracting the average temperature reading from all sensors in a given room can be simply written as shown in Listing – it should come as no surprise to anybody acquainted with the Standard Query Language (SQL). Such SQL-based querying of a WSN can be extended to an easy-to-grasp interface to wireless sensor networks, being capable of expressing most salient interaction patterns with a WSN. It is, however, not quite as clear how to translate this interface into actual networking protocols that implement this interface and can provide the results for such queries. In a traditional relational database, this implementation of a query is done by determining an execution plan; the same is necessary here. Here, however, the execution plan has to be distributed and has to explicitly take communication costs into account.

Listing 3.1: Example of an SQL-based request for sensor readings [528]

```
SELECT AVG(temperature)
FROM sensors
WHERE location = "Room 123"
```

EXPLOIT LOCATION INFORMATION

Another useful technique is to exploit location information in the communication protocols whenever such information is present. Since the location of an event is a crucial information for many applications, there have to be mechanisms that determine the location of sensor nodes (and possibly also that of observed events). Once such information is available, it can simplify the design and operation of communication protocols and can improve their energy efficiency considerably.

EXPLOIT ACTIVITY PATTERNS

Activity patterns in a wireless sensor network tend to be quite different from traditional networks. While it is true that the data rate averaged over a long time can be very small when there is only very rarely an event to report, this can change dramatically when something does happen. Once an event has happened, it can be detected by a larger number of sensors, breaking into a frenzy of activity, causing a well-known event shower effect. Hence, the protocol design should be able to handle such bursts of traffic by being able to switch between modes of quiescence and of high activity.

EXPLOIT HETEROGENEITY

Related to the exploitation of activity patterns is the exploitation of heterogeneity in the network. Sensor nodes can be heterogenous by constructions, that is, some nodes have larger batteries, farther-reaching communication devices, or more processing power. They can also be heterogenous by evolution, that is, all nodes started from an equal state, but because some nodes had to perform more tasks during the operation of the network, they have depleted their energy resources or other nodes had better opportunities to scavenge energy from the environment (e.g. nodes in shade are at a disadvantage when solar cells are used).

Whether by construction or by evolution, heterogeneity in the network is both a burden and an opportunity. The opportunity is in an asymmetric assignment of tasks, giving nodes with more resources or more capabilities the more demanding tasks. For example, nodes with more memory or faster processors can be better suited for aggregation, nodes with more energy reserves for hierarchical coordination, or nodes with a farther-reaching radio device should invest their energy mostly for long-distance communication, whereas, shorter-distance communication can be undertaken by the other nodes. The burden is that these asymmetric task assignments cannot usually be static but have to be re-evaluated as time passes and the node/network state evolves. Task reassignment in turn is an activity that requires resources and has to be balanced against the potential benefits.

COMPONENT-BASED PROTOCOL STACKS AND CROSS-LAYER OPTIMIZATION

Finally, a consideration about the implementation aspects of communication protocols in WSNs is necessary. For a component-based as opposed to a layering-based model of protocol implementation in WSN. What remains to be defined is mainly a default collection of components, not all of which have to be always available at all times on all sensor nodes, but which can form a basic "toolbox" of protocols and algorithms to build upon.

All wireless sensor networks will require some – even if only simple – form of physical, MAC and link layer protocols; there will be wireless sensor networks that require routing and transport layer functionalities. Moreover, "helper modules" like time synchronization, topology control, or localization can be useful. On top of these "basic" components, more abstract functionalities can then be built. As a consequence, the set of components that is active on a sensor node can be complex, and will change from application to application.

Protocol components will also interact with each other in essentially two different ways. One is the simple exchange of data packets as they are passed from one component to another as it is processed by different protocols. The other interaction type is the exchange of cross-layer information. This possibility for cross-layer information exchange holds great promise for protocol optimization, but is also not without danger.

4. PHYSICAL LAYER AND TRANSCEIVER DESIGN CONSIDERATIONS

The physical layer is mostly concerned with modulation and demodulation of digital data; this task is carried out by so-called transceivers. Some of the most crucial points influencing PHY design in wireless sensor networks are:

• Low power consumption.

• As one consequence: small transmit power and thus a small transmission range.

• As a further consequence: low duty cycle. Most hardware should be switched off or operated in a low-power standby mode most of the time.

• Comparably low data rates, on the order of tens to hundreds kilobits per second, required.

• Low implementation complexity and costs.

• Low degree of mobility.

• A small form factor for the overall node.

In this section, we discuss some of the implications of these requirements. In general, in sensor networks, the challenge is to find modulation schemes and transceiver architectures that are simple, low-cost but still robust enough to provide the desired service.

Energy Usage Profile

The choice of a small transmit power leads to an energy consumption profile different from other wireless devices like cell phones. These pivotal differences have been discussed in various places already but deserve a brief summary here. First, the radiated energy is small, typically on the order of 0 dBm (corresponding to 1mW). On the other hand, the overall transceiver (RF front end and baseband part) consumes much more energy than is actually radiated; A transceiver working at frequencies beyond 1 GHz takes 10 to 100mW of power to radiate 1 mW. These numbers coincide well with the observation that many practical transmitter designs have efficiencies below 10% at low radiated power.

A second key observation is that for small transmit powers the transmit and receive modes consume more or less the same power; it is even possible that reception requires more power than transmission; depending on the transceiver architecture, the idle mode's power consumption can be less or in the same range as the receive power. To reduce average power consumption in a low-traffic wireless sensor network, keeping the transceiver in idle mode all the time would consume significant amounts of energy. Therefore, it is important to put the transceiver into sleep state instead of just idling. It is also important to explicitly include the received power into energy dissipation models, since the traditional assumption that receive energy is negligible is no longer true.

A third key observation is the relative costs of communications versus computation in a sensor node. Clearly, a comparison of these costs depends for the communication part on the BER requirements, range, transceiver type, and so forth, and for the computation part on the processor type, the instruction mix, and so on.

Choice of Modulation Scheme

A crucial point is the choice of modulation scheme. Several factors have to be balanced here: the required and desirable data rate and symbol rate, the implementation complexity, the relationship between radiated power and target BER, and the expected channel characteristics.

To maximize the time a transceiver can spend in sleep mode, the transmit times should be minimized. The higher the data rate offered by a transceiver/modulation, the smaller the time needed to transmit a given amount of data and, consequently, the smaller the energy consumption.

A second important observation is that the power consumption of a modulation scheme depends much more on the symbol rate than on the data rate. For example, power consumption measurements of an IEEE 802.11b Wireless Local Area Network (WLAN) card showed that the power consumption depends on the modulation scheme, with the faster Complementary Code Keying (CCK) modes consuming more energy than DBPSK and DQPSK; however, the relative differences are below 10% and all these schemes have the same symbol rate. It has also been found that for the µAMPS-1 nodes the power consumption is insensitive to the data rate.

Obviously, the desire for “high” data rates at “low” symbol rates calls for m-ary modulation schemes. However, there are trade-offs:

• m$^{\text{-ary}}$ modulation requires more complex digital and analog circuitry than 2-ary modulation, for example, to parallelize user bits into m-ary symbols.

• Many m-ary modulation schemes require for increasing m an increased Eb/N0 ratio and consequently an increased radiated power to achieve the same target BER; others become less and less bandwidth efficient. However, in wireless sensor network applications with only low to moderate bandwidth requirements, a loss in bandwidth efficiency can be more tolerable than an increased radiated power to compensate E_b/N_0 losses.

• It is expected that in many wireless sensor network applications most packets will be short, on the order of tens to hundreds of bits. For such packets, the start-up time easily dominates overall energy consumption, rendering any efforts in reducing the transmission time by choosing m^{-ary} modulation schemes irrelevant.

The choice of modulation scheme depends on several interacting aspects, including technological factors (in the example: α, β), packet size, target error rate, and channel error model. The optimal decision would have to properly balance the modulation scheme and other measures to increase transmission robustness, since these also have energy costs:

• With retransmissions, entire packets have to be transmitted again.

• With FEC coding, more bits have to be sent and there is additional energy consumption for coding and decoding. While coding energy can be neglected, and the receiver needs significant energy for the decoding process.

• The cost of increasing the radiated power depends on the efficiency of the power amplifier but the radiated power is often small compared to the overall power dissipated by the transceiver, and additionally this drives the PA into a more efficient regime.

Dynamic Modulation Scaling

Even if it is possible to determine the optimal scheme for a given combination of BER target, range, packet sizes and so forth, such an optimum is only valid for short time; as soon as one of the constraints changes, the optimum can change, too. In addition, other constraints like delay or the desire to achieve high throughput can dictate to choose higher modulation schemes.

Therefore, it is interesting to consider methods to adapt the modulation scheme to the current situation. Such an approach, called dynamic modulation scaling, uses the symbol rate B and the number of levels per symbol m as parameters. This model expresses the energy required per bit and also the achieved delay per bit (the inverse of the data rate), taking into account that higher modulation levels need higher radiated energy. With modulation scaling, a packet is equipped with a delay constraint, from which directly a minimal required data rate can be derived. Since the symbol rate is kept fixed, the approach is to choose the smallest m that satisfies the required data rate and which thus minimizes the required energy per bit.

Such delay constraints can be assigned either explicitly or implicitly. One approach explored in the paper is to make the delay constraint depend on the packet backlog (number of queued packets) in a sensor node: When there are no packets present, a small value for m can be used, having low energy consumption. As backlog increases, m is increased as well to reduce the backlog quickly and switch back to lower values of m.

Antenna Considerations

The desired small form factor of the overall sensor nodes restricts the size and the number of antennas. As explained above, if the antenna is much smaller than the carrier's wavelength, it is hard to achieve good antenna efficiency, that is, with ill-sized antennas one must spend more transmit energy to obtain the same radiated energy. Secondly, with small sensor node cases, it will be hard to place two antennas with suitable distance to achieve receive diversity. The antennas should be spaced apart at least 40–50% of the wavelength used to achieve good effects from diversity. For 2.4 GHz, this corresponds to a spacing of between 5 and 6 cm between the antennas, which is hard to achieve with smaller cases.

In addition, radio waves emitted from an antenna close to the ground – typical in some applications – are faced with higher path-loss coefficients than the common value $\alpha = 2$ for free-space communication. Typical attenuation values in such environments, which are also normally characterized by obstacles (buildings, walls, and so forth), are about $\alpha = 4$. Moreover, depending on the application, antennas must not protrude from the casing of a node, to avoid possible damage to it. These restrictions, in general, limit the achievable quality and characteristics of an antenna for wireless sensor nodes.

Nodes randomly scattered on the ground, for example, deployed from an aircraft, will land in random orientations, with the antennas facing the ground or being otherwise obstructed. This can lead to non-isotropic propagation of the radio wave, with considerable differences in the strength of the emitted signal in different directions. This effect can also be caused by the design of an antenna, which often results in considerable differences in the spatial propagation characteristics (so-called lobes of an antenna).

CONCLUSION:

The separation of functionalities is justified from the hardware properties as is it supported by operating systems like *TinyOS*. These trade-offs form the basis for the construction of networking functionalities, geared toward the specific requirements of wireless sensor network applications.

The wireless sensor networks and their networking architecture will have many different guises and shapes. For many applications, but by no means all, multihop communication is the crucial enabling technology, and most of the WSN research as well as the following part of this book are focused on this particular form of wireless networking. Four main optimization goals – WSN-specific forms of quality of service support, energy efficiency, scalability, and robustness – dominate the requirements for WSNs and have to be carefully arbitrated and balanced against each other. To do so, the design of WSNs departs in crucial aspects from that of traditional networks, resulting in a number of design principles. Most importantly, distributed organization of the network, the use of in-network processing, a data-centric view of the network, and the adaptation of result fidelity and accuracy to given circumstances are pivotal techniques to be considered for usage.

The large diversity of WSNs makes the design of a uniform, general-purpose service interface difficult; consequently, no final solutions to this problem are currently available. Similarly, the integration of WSNs in larger network contexts, for example, to allow Internet-based hosts a simple access to WSN services, is also still a fairly open problem. The physical layer is mostly concerned with modulation and demodulation of digital data; this task is carried out by so-called transceivers. In sensor networks, the challenge is to find modulation schemes and transceiver architectures that are simple, low cost, but still robust enough to provide the desired service.

NOTES on NETWORKING SENSORS Unit III

Dr. G. Senthil Kumar,
Associate Professor,
Dept. of ECE, SCSVMV,
email: gsk_ece@kanchiuniv.ac.in

==

OBJECTIVES:

In this lesson, you will be introduced for Medium Access Control (MAC) protocols that solve a seemingly simple task: they coordinate the times where a number of nodes access a shared communication medium. Second, Naming and addressing schemes are used to denote and to find things. In networking, names and addresses often refer to individual nodes as well as to data items stored in them. In a multihop network, intermediate nodes have to relay packets from the source to the destination node. Such an intermediate node has to decide to which neighbor to forward an incoming packet not destined for itself. Typically, routing tables that list the most appropriate neighbor for any given packet destination are used. The construction and maintenance of these routing tables is the crucial task of a distributed routing protocol.

CONTENTS:

INTRODUCTION

This chapter presents the fundamentals of MAC protocols and explains the specific requirements and problems these protocols have to face in wireless sensor networks. The single most important requirement is energy efficiency and there are different MAC-specific sources of energy waste to consider: overhearing, collisions, overhead, and idle listening. We discuss protocols addressing one or more of these issues. One important approach is to switch the wireless transceiver into a sleep mode. Therefore, there are trade-offs between a sensor network's energy expenditure and traditional performance measures like delay and throughput. An "unoverseeable" number of protocols differ, among others, in the types of media they use and in the performance requirements for which they are optimized.

In this chapter, we first give a brief introduction to MAC protocols in general and to the particular requirements and challenges found in wireless sensor networks. Most notably, the issue of energy efficiency is the prime consideration in WSN MAC protocols, and therefore, we concentrate on schemes that explicitly try to reduce overall energy consumption. One of the main approaches to conserve energy is to put nodes into sleep state whenever possible. Protocols striving for low duty cycle or wakeup concepts are designed to accomplish this. Other important classes of useful MAC protocols are contention-based and schedule-based protocols. The IEEE 802.15.4 protocol combines contention- and schedule-based elements and can be expected to achieve significant commercial impact.

Medium Access Control (MAC) protocols is the first protocol layer above the Physical Layer (PHY) and consequently MAC protocols are heavily influenced by its properties. The fundamental task of any MAC protocol is to regulate the access of a number of nodes to a shared medium in such a way that certain application-dependent performance requirements are satisfied. Some of the traditional performance criteria are delay, throughput, and fairness, whereas in WSNs, the issue of energy conservation becomes important.

Within the OSI reference model, the MAC is considered as a part of the Data Link Layer (DLL), but there is a clear division of work between the MAC and the remaining parts of the DLL. The MAC protocol determines for a node the points in time when it accesses the medium to try to transmit a data, control, or management packet to another node (unicast) or to a set of nodes (multicast, broadcast). Two important responsibilities of the remaining parts of the DLL are error control and flow control.

Then, Addresses/names are always tied to a representation, which has a certain length when considered as a string of bits. As opposed to other types of networks, representation size is a critical issue in wireless sensor networks, since addresses are present in almost any packet. However, coordination among nodes is needed to assign reasonably short addresses. A second key aspect is content-based addressing, where not nodes or network interfaces but data is addressed. Content-based addressing can be integrated with data-centric routing and is also a key enabler of in-network processing.

Naming and addressing are two fundamental issues in networking. We can say very roughly that names are used to denote things (for example, nodes, data, transactions) whereas addresses supply the information needed to find these things; they help, for example, with routing in a multihop network. This distinction is not sharp; sometimes addresses are used to denote things too – an IP address contains information to both find a node (the network part of an address) and to identify a node – more precisely: a network interface within a node – within a single subnetwork (the host part).

In traditional networks like the Internet or ad hoc networks, frequently independent nodes or stations as well as the data hosted by these are named and addressed. This is adequate for the intended use of these networks: They connect many users and let them exchange data or access servers. The range of possible user data types is enormous and the network can support these tasks best by making the weakest assumption about the data – all data is just a pile of bits to be moved from one node to another. In wireless sensor networks, the nodes are not independent but collaborate to solve a given task and to provide the user with an interface to the external world. Therefore, it might be appropriate to shift the view from naming nodes toward naming aspects of the physical world or naming data. Here, the focus on aspects like address allocation, address representation, and proper use of different addressing/naming schemes in wireless sensor networks.

Next, this chapter discusses mechanisms for routing and forwarding when the destination of a packet is identified by a unique node identifier, by a set of such identifiers, or when all the nodes in the network shall receive a packet. One particular type of identifier will be position information, which can identify both individual nodes and groups of nodes.

1. MAC PROTOCOLS FOR WIRELESS SENSOR NETWORKS

Fundamentals of (Wireless) MAC Protocols

In this section, we discuss some fundamental aspects and important examples of wireless MAC protocols, since the protocols used in wireless sensor networks inherit many of the problems and approaches already existing for this more general field. With the advent of wireless sensor networks, energy has been established as one of the primary design concerns.

Requirements and design constraints for wireless MAC protocols

Traditionally, the most important performance requirements for MAC protocols are throughput efficiency, stability, fairness, low access delay (time between packet arrival and first attempt to transmit it), and low transmission delay (time between packet arrival and successful delivery), as well as a low overhead. The overhead in MAC protocols can result from per-packet overhead (MAC headers and trailers), collisions, or from exchange of extra control packets. Collisions can happen if the MAC protocol allows two or more nodes to send packets at the same time. Collisions can result in the inability of the receiver to decode a packet correctly, causing the upper layers to perform a retransmission. For time-critical applications, it is important to provide deterministic or stochastic guarantees on delivery time or minimal available data rate. Sometimes, preferred treatment of important packets over unimportant ones is required, leading to the concept of priorities.

The operation and performance of MAC protocols is heavily influenced by the properties of the underlying physical layer. Since WSNs use a wireless medium, they inherit all the well-known problems of wireless transmission. One problem is time-variable, and sometimes quite high, error rates, which is caused by physical phenomena like slow and fast fading, path loss, attenuation, and man-made or thermal noise. Depending on modulation schemes, frequencies, distance between transmitter and receiver, and the propagation environment, instantaneous bit error rates in the range of $10^{-3} \ldots 10^{-2}$ can easily be observed.

The received power *Prcvd* decreases with the distance between transmitting and receiving node. This path loss combined with the fact that any transceiver needs a minimum signal strength to demodulate signals successfully leads to a maximum range that a sensor node can reach with a given transmit power. If two nodes are out of reach, they cannot hear each other. This gives rise to the well-known hidden-terminal/exposed-terminal problems. The hidden-terminal problem occurs specifically for the class of Carrier Sense Multiple Access (CSMA) protocols, where a node senses the medium before starting to transmit a packet. If the medium is found to be busy, the node defers its packet to avoid a collision and a subsequent retransmission. Consider the example in Figure 1. Here, we have three nodes A, B, and C that are arranged such that A and B are in mutual range, B and C are in mutual range, but A and C cannot hear each other. Assume that A starts to transmit a packet to B and sometime later node C also decides to start a packet transmission. A carrier-sensing operation by C shows an idle medium since C cannot hear A's signals. When C starts its packet, the signals collide at B and both packets are useless. Using simple CSMA in a hidden-terminal scenario thus leads to needless collisions.

In the exposed-terminal scenario, B transmits a packet to A, and some moment later, C wants to transmit a packet to D. Although this would be theoretically possible since both A and D would receive their packets without distortions, the carrier-sense operation performed by C suppresses C's transmission and bandwidth is wasted. Using simple CSMA in an exposed terminal scenario thus leads to needless waiting. Two solutions to the hidden-terminal and exposed-terminal problems are busy-tone solutions and the RTS/CTS handshake used in the IEEE 802.11 WLAN standard and first presented in the MACA /MACAW protocols.

On wired media, it is often possible for the transmitter to detect a collision at the receiver immediately and to abort packet transmission. This feature is called collision detection (CD) and is used in Ethernet's CSMA/CD protocol to increase throughput efficiency. Such a collision detection works because of the low attenuation in a wired medium, resulting in similar SNRs at transmitter and receiver. Consequently, when the transmitter reads back the channel signal during transmission and observes a collision, it can infer that there must have been a collision at the receiver too. More importantly, the absence of a collision at the transmitter allows to conclude that there has been no

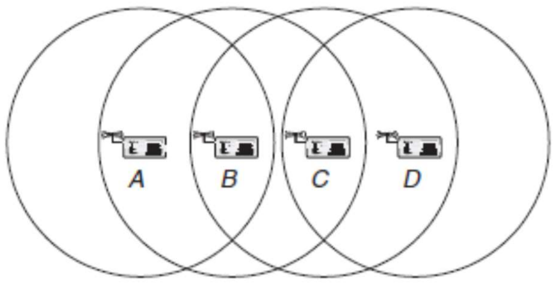


Figure 5.1 Hidden-terminal scenario (circles indicate transmission & interference range)

collision at the receiver during the packet transmission. In a wireless medium, neither of these two conclusions holds true – the interference situation at the transmitter does not tell much about the interference situation at the receiver. Furthermore, simple wireless transceivers work only in a half-duplex mode, meaning that at any given time either the transmit or the receive circuitry is active but not both. Therefore, collision detection protocols are usually not applicable to wireless media.

Another important problem arises when there is no dedicated frequency band allocated to a wireless sensor network and the WSN has to share its spectrum with other systems. Because of license-free operations, many wireless systems use the so-called ISM bands, with the 2.4 GHz ISM band being a prime example.

Finally, the design of MAC protocols depends on the expected traffic load patterns. If a WSN is deployed to continuously observe a physical phenomenon, for example, the time-dependent temperature distribution in a forest, a continuous and low load with a significant fraction of periodic traffic can be expected. On the other hand, if the goal is to wait for the occurrence of an important event and upon its occurrence to report as much data as possible, the network is close to idle for a long time and then is faced with a bulk of packets that are to be delivered quickly. A high MAC efficiency is desirable during these overload phases.

Important Classes of Mac Protocols

A huge number of (wireless) MAC protocols have been devised during the last thirty years. They can be roughly classified into the following classes: fixed assignment protocols, demand assignment protocols, and random access protocols. Fixed assignment protocols In this class of protocols, the available resources are divided between the nodes such that the resource assignment is long term and each node can use its resources exclusively without the risk of collisions. Long term means that the assignment is for durations of minutes, hours, or even longer, as opposed to the short-term case where assignments have a scope of a data burst, corresponding to a time horizon of perhaps (tens of) milliseconds. To account for changes in the topology – for example, due to nodes dying or new nodes being deployed, mobility, or changes in the load patterns – signalling mechanisms are needed in fixed assignment protocols to renegotiate the assignment of resources to nodes. This poses questions about the scalability of these protocols.

Typical protocols of this class are TDMA, FDMA, CDMA, and SDMA. The Time Division Multiple Access (TDMA) scheme subdivides the time axis into fixed-length super frames and each super frame is again subdivided into a fixed number of time slots. These time slots are assigned to nodes exclusively and hence the node can transmit in this time slot periodically in every super frame. TDMA requires tight time synchronization between nodes to avoid overlapping of signals in adjacent time slots. In Frequency Division Multiple Access (FDMA), the available frequency band is subdivided into a number of subchannels and these are assigned to nodes, which can transmit exclusively on their channel. This scheme requires frequency synchronization, relatively narrowband filters, and the ability of a receiver to tune to the channel used by a transmitter. Accordingly, an FDMA transceiver tends to be more complex than a TDMA transceiver. In Code Division Multiple Access (CDMA) schemes, the nodes spread their signals over a much larger bandwidth than needed, using different codes to separate their transmissions. The receiver has to know the code used by the transmitter; all parallel transmissions using other codes appear as noise. Crucial to CDMA is the code management. Finally, in Space Division Multiple Access (SDMA), the spatial separation of nodes is used to separate their transmissions. SDMA requires arrays of antennas and sophisticated signal processing techniques and cannot be considered a candidate technology for WSNs.

Demand assignment protocols

In demand assignment protocols, the exclusive allocation of resources to nodes is made on a short-term basis, typically the duration of a data burst. This class of protocols can be broadly subdivided into centralized and distributed protocols. In central control protocols (examples are the HIPERLAN/2 protocol, DQRUMA, or the MASCARA protocol; polling schemes can also be subsumed under this class), the nodes send out requests for bandwidth allocation to a central node that either accepts or rejects the requests. In case of successful allocation, a confirmation is transmitted back to the requesting node along with a description of the allocated resource, for example, the numbers and positions of assigned time slots in a TDMA system and the duration of allocation. The node can use these resources exclusively. The submission of requests from nodes to the central station is often done contention based, that is, using a random access protocol on a dedicated (logical) signalling channel. Another option is to let the central station poll its associated nodes. In addition, the nodes often piggyback requests onto data packets transmitted in their exclusive data slots, thus avoiding transmission of separate request packets. The central node needs to be switched on all the time and is responsible for resource allocation. Resource deallocation is often done implicitly: when a node does not use its time slots any more, the central node can allocate these to other nodes.

This way, nodes do not need to send extra deallocation packets. Summarizing, the central node performs a lot of activities, it must be constantly awake, and thus needs lots of energy. This class of protocols is a good choice if a sufficient number of energy-unconstrained nodes are present and the duties of the central station can be moved to these.

An example of distributed demand assignment protocols are token-passing protocols like IEEE 802.4 Token Bus. The right to initiate transmissions is tied to reception of a small special token frame. The token frame is rotated among nodes organized in a logical ring on top of a broadcast medium. Special ring management procedures are needed to include and exclude nodes from the ring or to correct failures like lost tokens. Token-passing protocols have also been considered for wireless or error-prone media, but they tend to have problems with the maintenance of the logical ring in the presence of significant channel errors. In addition, since token circulation times are variable, a node must always be able to receive the token to avoid breaking the logical ring. Hence, a nodes transceiver must be switched on most of the time. In addition, maintaining a logical ring in face of frequent topology changes is not an easy task and involves significant signalling traffic besides the token frames themselves.

Random access protocols

The nodes are uncoordinated, and the protocols operate in a fully distributed manner. Random access protocols often incorporate a random element, for example, by exploiting random packet arrival times, setting timers to random values, and so on. One of the first and still very important random access protocols is the ALOHA or slotted ALOHA protocol, developed at the University of Hawaii. In the pure ALOHA protocol, a node wanting to transmit a new packet transmits it immediately. There is no coordination with other nodes and the protocol thus accepts the risk of collisions at the receiver. To detect this, the receiver is required to send an immediate acknowledgment for a properly received packet. The transmitter interprets the lack of an acknowledgment frame as a sign of a collision, backs off for a random time, and starts the next trial. ALOHA provides short access and transmission delays under light loads; under heavier loads, the number of collisions increases, which in turn decreases the throughput efficiency and increases the transmission delays. In slotted ALOHA, the time is subdivided into time slots and a node is allowed to start a packet transmission only at the beginning of a slot. A slot is large enough to accommodate a maximum-length packet. Accordingly, only contenders starting their packet transmission in the same slot can destroy a node’s packet. If any node wants to start later, it has to wait for the beginning of the next time slot and has thus no chance to destroy the node’s packet. In short, the synchronization reduces the probability of collisions and slotted ALOHA has a higher throughput than pure ALOHA.

In the class of CSMA protocols, a transmitting node tries to be respectful to ongoing transmissions. First, the node is required to listen to the medium; this is called carrier sensing. If the medium is found to be idle, the node starts transmission. If the medium is found busy, the node defers its transmission for an amount of time determined by one of several possible algorithms. For example, in nonpersistent CSMA, the node draws a random waiting time, after which the medium is sensed again. Before this time, the node does not care about the state of the medium. In different persistent CSMA variants, after sensing that the medium is busy, the node awaits the end of the ongoing transmission and then behaves according to a backoff algorithm. In many of these backoff algorithms, the time after the end of the previous frame is subdivided into time slots. In p-persistent CSMA, a node starts transmission in a time slot with some probability p and with probability 1 – p it waits for another slot.3 If some other node starts to transmit in the meantime, the node defers and repeats the whole procedure after the end of the new frame. A small value of p makes collisions unlikely, but at the cost of high access delays. The converse is true for a large value of p.

In the backoff algorithm executed by the IEEE 802.11 Distributed Coordination Function (DCF), a node transmitting a new frame picks a random value from the current contention window and starts a timer with this value. The timer is decremented after each slot. If another node starts in the meantime, the timer is suspended and resumed after the next frame ends and contention continues. If the timer decrements to zero, the node transmits its frame. When a transmission error occurs (indicated, for example, by a missing acknowledgment frame), the size of the contention window is increased according to a modified binary exponential backoff procedure.4 While CSMA protocols are still susceptible to collisions, they have a higher throughput efficiency than ALOHA protocols, since ongoing packets are not destroyed when potential competitors hear them on the medium.

As explained above, carrier-sense protocols are susceptible to the hidden-terminal problem since interference at the receiver cannot be detected by the transmitter. This problem may cause packet collisions. The energy spent on collided packets is wasted and the packets have to be retransmitted. Several approaches have appeared to solve or at least to reduce the hidden-terminal problem; we present two important ones: the busy-tone solution and the RTS/CTS handshake.

In the original busy-tone solution, two different frequency channels are used, one for the data packets and the other one as a control channel. As soon as a node starts to receive a packet destined to it, it emits an unmodulated wave on the control channel and ends this when packet reception is finished. A node that wishes to transmit a packet first senses the control channel for the presence of a busy tone. If it hears something, the node backs off according to some algorithm, for example similar to nonpersistent CSMA. If it hears nothing, the node starts packet transmission on the data channel. This protocol solves both the hidden- and exposed-terminal problem, given that the busy-tone signal can be heard over the same distance as the data signal. If the busy tone is too weak, a node within radio range of the receiver might start data transmission and destroy the receiver's signal. If the busy tone is too strong, more nodes than necessary suppress their transmissions. The control channel does not need much bandwidth but a narrow bandwidth channel requires good frequency synchronization. A solution with two busy tones, one sent by the receiver and the other by the transmitter node. Another variant of the busy-tone approach is used by PAMAS.

The RTS/CTS handshake as used in IEEE 802.11 is based on the MACAW protocol and is illustrated in Figure 2. It uses only a single channel and two special control packets. Suppose that node B wants to transmit a data packet to node C. After B has obtained channel access (for example after sensing the channel as idle), it sends a Request To Send (RTS) packet to C, which includes a duration field indicating the remaining length of the overall transaction (i.e., until the point where B would receive the acknowledgment for its data packet). If C has properly received the RTS packet, it sends a Clear To Send (CTS) packet, which again contains a duration field. When B receives the CTS packet, it starts transmission of the data packet and finally C answers with an acknowledgment packet. The acknowledgment is used to tell B about the success of the transmission; lack of acknowledgment is interpreted as collision (the older MACA protocol lacks the acknowledgment). Any other station A or D hearing either the RTS, CTS, data or acknowledgment packet sets an internal timer called Network Allocation Vector (NAV) to the remaining duration indicated in the respective frame and avoids sending any packet as long as this timer is not expired. Specifically, nodes A and D send no CTS answer packets even when they have received a RTS packet correctly. This way, the ongoing transmission is not distorted.

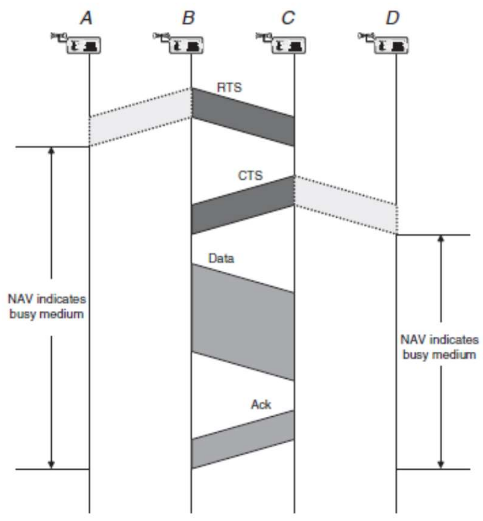


Figure 5.2 RTS/CTS handshake in IEEE 802.11

Does this scheme eliminate collisions completely? No, there still exist some collision scenarios. First, in the scenario described above, nodes A and C can issue RTS packets to B simultaneously. However, in this case, only the RTS packets are lost and no long data frame has been transmitted. Two further problems are illustrated in Figure 3: In the left part of the figure, nodes A and B run the RTS-CTS-Data-Ack sequence, and B’s CTS packet also reaches node C. However, at almost the same time, node D sends an RTS packet to C, which collides at node C with B’s CTS packet. This way, C has no chance to decode the duration field of the CTS packet and to set its NAV variable accordingly. After its failed RTS packet, D sends the RTS packet again to C and C answers with a CTS packet. Node C is doing so because it cannot hear A’s ongoing transmission and has no proper NAV entry. C’s CTS packet and A’s data packet collide at B. In the figure’s right part, the problem is created by C starting its RTS packet to D immediately before it can sense B’s CTS packet, which C consequently cannot decode properly. One solution approach is to ensure that CTS packets are longer than RTS packets. For an explanation, consider the right part of Figure 3. Here, even if B’s CTS arrives at C immediately after C starts its RTS, it lasts long enough that C has a chance to turn its transceiver into receive mode and to sense B’s signal. An additional protocol rule states that in such a case node C has to defer any further transmission for a sufficiently long time to accommodate one maximum-length data packet. Hence, the data packet between A and B can be transmitted without distortion. It is not hard to convince oneself that the problem in the left half of Figure 3 is eliminated too.

A further problem of the RTS/CTS handshake is its significant overhead of two control packets per data packet, not counting the acknowledgment packet. If the data packet is small, this overhead might not pay off and it may be simpler to use some plain CSMA variant. For long packets, the overhead of the RTS/CTS handshake can be neglected, but long packets are more likely to be hit by channel errors and must be retransmitted entirely, wasting precious energy (channel errors often hit only a few bits). A good compromise is to fragment a large packet like, for example, in

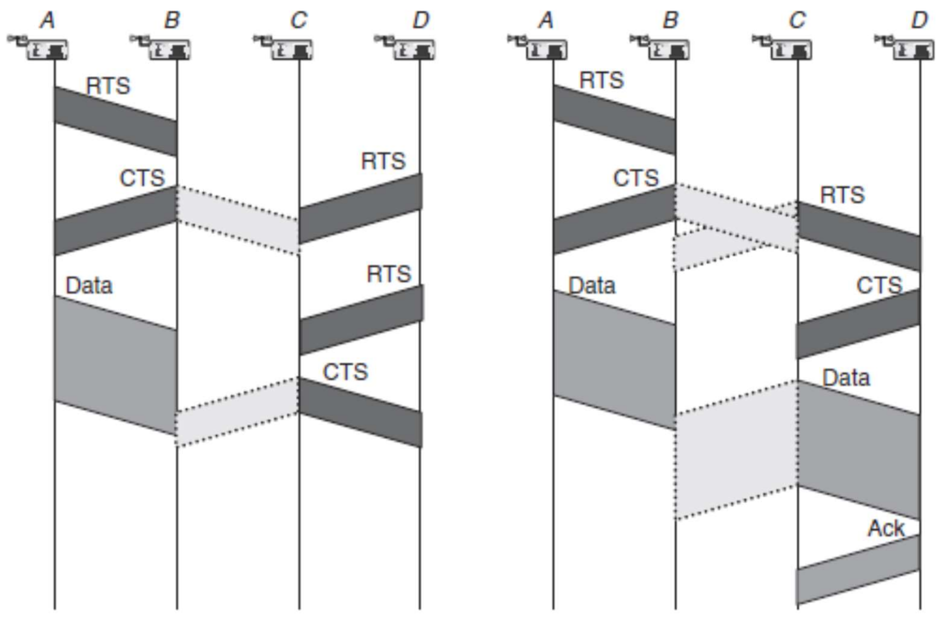


Figure 5.3 Two problems in RTS/CTS handshake [668]

IEEE 802.11 or in the S-MAC protocol and to use the RTS/CTS only once for the whole set of fragments.

MAC PROTOCOLS FOR WIRELESS SENSOR NETWORKS

In this section, we narrow down the specific requirements and design considerations for MAC protocols in wireless sensor networks.

Balance of requirements

For the case of WSNs, the balance of requirements is different from traditional (wireless) networks. Additional requirements come up, first and foremost, the need to conserve energy.

The importance of energy efficiency for the design of MAC protocols is relatively new and many of the “classical” protocols like ALOHA and CSMA contain no provisions toward this goal. Some papers covering energy aspects in MAC protocols are references. Other typical performance figures like fairness, throughput, or delay tend to play a minor role in sensor networks. Fairness is not important since the nodes in a WSN do not represent individuals competing for bandwidth, but they collaborate to achieve a common goal. The access/transmission delay performance is traded against energy conservation, and throughput is mostly not an issue either.

Further important requirements for MAC protocols are scalability and robustness against frequent topology changes, as caused for example by nodes powering down temporarily to replenish their batteries by energy scavenging, mobility, deployment of new nodes, or death of existing nodes. The need for scalability is evident when considering very dense sensor networks with dozens or hundreds of nodes in mutual range.

Energy problems on the MAC layer

A nodes transceiver consumes a significant share of energy. Recall that a transceiver can be in one of the four main states: transmitting, receiving, idling, or sleeping. The energy-consumption properties of some transceiver designs in the different operational states. In a nutshell, the lessons are: Transmitting is costly, receive costs often have the same order of magnitude as transmit costs, idling can be significantly cheaper but also about as expensive as receiving, and sleeping costs almost nothing but results in a “deaf” node. Applying these lessons to the operations of a MAC protocol, we can derive the following energy problems and design goals.

Collisions collisions incur useless receive costs at the destination node, useless transmit costs at the source node, and the prospect to expend further energy upon packet retransmission. Hence, collisions should be avoided, either by design (fixed assignment/TDMA or demand assignment protocols) or by appropriate collision avoidance/hidden-terminal procedures in CSMA protocols. However, if it can be guaranteed for the particular sensor network application at hand that the load is always sufficiently low, collisions are no problem.

Overhearing Unicast frames have one source and one destination node. However, the wireless medium is a broadcast medium and all the source’s neighbors that are in receive state hear a packet and drop it when it is not destined to them; these nodes overhear the packet. For higher node densities overhearing avoidance can save significant amounts of energy. On the other hand, overhearing is sometimes desirable, for example, when collecting neighborhood information or estimating the current traffic load for management purposes.

Protocol overhead Protocol overhead is induced by MAC-related control frames like, for example, RTS and CTS packets or request packets in demand assignment protocols, and furthermore by per-packet overhead like packet headers and trailers.

Idle listening A node being in idle state is ready to receive a packet but is not currently receiving anything. This readiness is costly and useless in case of low network loads; for many radio modems, the idle state still consumes significant energy. Switching off the transceiver is a solution; however, since mode changes also cost energy, their frequency should be kept at "reasonable" levels. TDMA-based protocols offer an implicit solution to this problem, since a node having assigned a time slot and exchanging (transmitting/receiving) data only during this slot can safely switch off its transceiver in all other time slots. Most of the MAC protocols developed for wireless sensor networks attack one or more of these problems to reduce energy consumption, as we will see in the next sections.

A design constraint somewhat related to energy concerns is the requirement for low complexity operation. Sensor nodes shall be simple and cheap and cannot offer plentiful resources in terms of processing power, memory, or energy. Therefore, computationally expensive operations like complex scheduling algorithms should be avoided. The desire to use cheap node hardware includes components like oscillators and clocks. Consequently, the designer of MAC protocols should bear in mind that very tight time synchronization (as needed for TDMA with small time slots) would require frequent resynchronization of neighbouring nodes, which can consume significant energy.

In the following sections, we discuss protocols that explicitly attack the idle listening problem by applying periodic sleeping or even wakeup radio concepts. Some other protocols are classified into either contention-based or schedule-based protocols. This distinction is to be understood by the number of possible contenders upon a transmit opportunity toward a receiver node. The IEEE 802.15.4 protocol, which combines elements of schedule- and contention-based protocols and can be expected to achieve some commercial impact.

LOW DUTY CYCLE PROTOCOLS AND WAKEUP CONCEPTS

Low duty cycle protocols try to avoid spending (much) time in the idle state and to reduce the communication activities of a sensor node to a minimum. In an ideal case, the sleep state is left

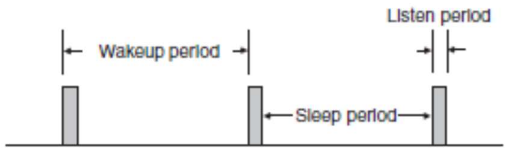


Figure 5.4 Periodic wakeup scheme

only when a node is about to transmit or receive packets. A concept for achieving this, the wakeup radio, is discussed. However, such a system has not been built yet, and thus, there is significant interest to find alternative approaches.

In several protocols, a periodic wakeup scheme is used. Such schemes exist in different flavours. One is the cycled receiver approach, illustrated in Figure 4. In this approach, nodes spend most of their time in the sleep mode and wake up periodically to receive packets from other nodes. Specifically, a node A listens onto the channel during its listen period and goes back into sleep mode when no other node takes the opportunity to direct a packet to A. A potential transmitter B must acquire knowledge about A's listen periods to send its packet at the right time – this task corresponds to a rendezvous. This rendezvous can, for example, be accomplished by letting node A transmit a short beacon at the beginning of its listen period to indicate its willingness to receive packets. Another method is to let node B send frequent request packets until one of them hits A's listen period and is really answered by A. However, in either case, node A only receives packets during its listen period. If node A itself wants to transmit packets, it must acquire the target's listen period. A whole cycle consisting of sleep period and listen period is also called a wakeup period. The ratio of the listen period length to the wakeup period length is also called the node's duty cycle. From this discussion, we already can make some important observations:

• By choosing a small duty cycle, the transceiver is in sleep mode most of the time, avoiding idle listening and conserving energy.

• By choosing a small duty cycle, the traffic directed from neighbouring nodes to a given node concentrates on a small time window (the listen period) and in heavy load situations significant competition can occur.

• Choosing a long sleep period induces a significant per-hop latency, since a prospective transmitter node has to wait an average of half a sleep period before the receiver can accept packets. In the multihop case, the per-hop latencies add up and create significant end-to-end latencies.

• Sleep phases should not be too short lest the start-up costs outweigh the benefits. In other protocols like, for example, S-MAC, there is also a periodic wakeup but nodes can both transmit and receive during their wakeup phases. When nodes have their wakeup phases at the same time, there is no necessity for a node wanting to transmit a packet to be awake outside these phases to rendezvous its receiver.

Subsequently, we discuss some variations of this approach. They differ in various aspects, for example, the number of channels required or in the methods by which prospective transmitters can learn the listen periods of the intended receivers.

Sparse topology and energy management (STEM)

The Sparse Topology and Energy Management (STEM) protocol does not cover all aspects of a MAC protocol but provides a solution for the idle listening problem. STEM targets networks that are deployed to wait for and report on the behaviour of a certain event, for example, when studying the paths of elephants in a habitat. From the perspective of a single sensor, most of

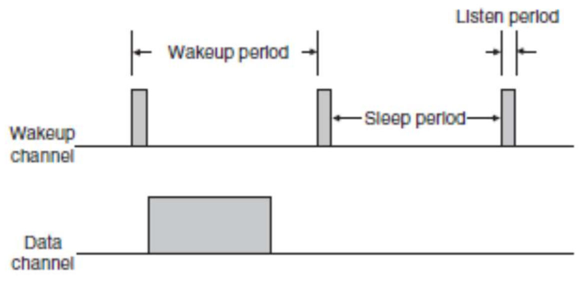


Figure 5.5 STEM duty cycle for a single node [742, Fig. 3]

the time there are no elephants and the sensor has nothing to report. However, once an elephant appears, the sensor reports its readings periodically. More abstractly, the network has a monitor state, where the nodes idle and do nothing, and also a transfer state, where the nodes exhibit significant sensing and communication activity. STEM tries to eliminate idle listening in the monitor state and to provide a fast transition into the transfer state, if required. In the transfer state, different MAC protocols can be employed. The term "topology" in

STEMs name comes from the observation that as nodes enter and leave the sleep mode network topology changes. An important requirement for such topology-management schemes is that the network stays connected (or bi-connected or fulfils even higher connectivity requirements) even if a subset of nodes is in the sleep mode.

For an explanation of STEM (Figure 5), two different channels are used, requiring two transceivers in each node: the wakeup channel and the data channel. The data channel is always in sleep mode, except when transmitting or receiving data packets. The underlying MAC protocol is executed solely on the data channel during the transfer states. On the wakeup channel the time is divided into fixed-length wakeup periods of length T . A wakeup period is subdivided into a listen period of length TRx T and a sleep period, where the wakeup channel transceiver enters sleep mode, too. If a node enters the listen period, it simply switches on its receiver for the wakeup channel and waits for incoming signals. If nothing is received during time TRx, the node returns into sleep mode. Otherwise the transmitter and receiver start a packet transfer on the data channel. There are two different variants for the transmitter to acquire the receiver’s attention:

• In STEM-B, the transmitter issues so-called beacons on the wakeup channel periodically and without prior carrier sensing. Such a beacon indicates the MAC addresses of transmitter and receiver. As soon as the receiver picks up the beacon, it sends an acknowledgment frame back on the wakeup channel (causing the transmitter to stop beacon transmission), switches on the transceiver for the data channel, and both nodes can proceed to execute the regular MAC protocol on the data channel, like for example an RTS/CTS handshake. Any other node receiving the beacon on the wakeup channel recognizes that the packet is not destined for it and goes back to sleep mode. The transmitter sends these beacons at least for one full wakeup period to be sure to hit the receivers listen period.

• In STEM-T, the transmitter sends out a simple busy tone on the control channel (the T in STEM-T comes from “tone”) for a time long enough to hit the receiver’s listen period. Since the busy tone carries no address information, all the transmitter’s neighbors (the receiver as well as other nodes) will sense the busy tone and switch on their data channel, without sending an acknowledgment packet. The other nodes can go back to sleep when they can deduce from the packet exchange on the data channel that they are not involved in the data transfer. A transceiver capable of generating and sensing busy tones can be significantly cheaper and less energy-consuming than a transceiver usable for data transmission but requires proper frequency synchronization. In STEM-B, several transmitters might transmit their beacons simultaneously, leading to beacon collisions. A node waking up and receiving just some energy on the wakeup channel without being able to decode it behaves exactly as in STEM-T: It sends no acknowledgment, switches on its data channel, and waits what happens. The transmitter in this case transmits the beacons for the maximum time (since it hears no acknowledgment), then switches to the data channel, and tries to start the conversation with the receiver node.

It is noteworthy that in STEM a node entering the listen period remains silent, that is, transmits no packet. The opposite approach has been chosen, where node just waking up announces its willingness to receive a packet by transmitting a query beacon packet. In the approach taken by STEM, the transmitter has to send beacons or a busy tone for an average time of ≈ T and in the worst case for a maximum of ≈ T . If packet transmissions are a rare event, it pays off to avoid the frequent (and mostly useless) query beacons and to put some extra burden on the transmitter to reach its receiver. Therefore, in low load situations, STEM-T is preferable over STEM-B. With respect to energy expenditure, STEM-T can have advantages, since the acknowledgment packet is saved and the length of the listen period TRx can be significantly shorter in STEM-T than for STEM-B, since it suffices to detect energy, whereas in STEM-B this time has to accommodate at least one full beacon packet.

S-MAC

The S-MAC (Sensor-MAC) protocol provides mechanisms to circumvent idle listening, collisions, and overhearing. As opposed to STEM, it does not require two different channels. S-MAC adopts a periodic wakeup scheme, that is, each node alternates between a fixed-length listen period and a fixed-length sleep period according to its schedule, compare Figure 6. However, as opposed to STEM, the listen period of S-MAC can be used to receive and transmit packets. S-MAC attempts to coordinate the schedules of neighbouring nodes such that their listen periods start at the same time. A node x's listen period is subdivided into three different phases:

• In the first phase (SYNCH phase), node x accepts SYNCH packets from its neighbors. In these packets, the neighbors describe their own schedule and x stores their schedule in a table (the schedule table). Node x's SYNCH phase is subdivided into time slots and x's neighbours contend according to a CSMA scheme with additional backoff, that is, each neighbor y wishing to transmit a SYNCH packet picks one of the time slots randomly and starts to transmit if no signal was received in any of the previous slots. In the other case, y goes back into sleep mode and waits for x's next wakeup. In the other direction, since x knows a neighbor y's schedule, x can wake at appropriate times and send its own SYNCH packet to y (in broadcast mode). It is not required that x broadcasts its schedule in every of y's wakeup periods. However, for reasons of time synchronization and to allow new nodes to learn their local network topology, x should send SYNCH packets periodically. The according period is called synchronization period.

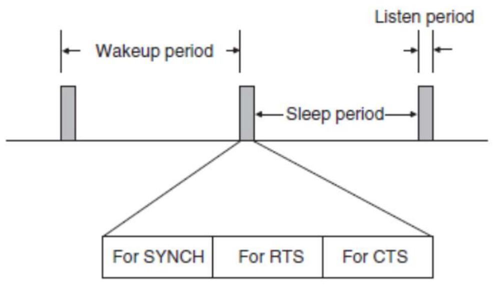


Figure 5.6 S-MAC principle

• In the second phase (RTS phase), x listens for RTS packets from neighbouring nodes. In S-MAC, the RTS/CTS handshake described is used to reduce collisions of data packets due to hidden-terminal situations. Again, interested neighbors contend in this phase according to a CSMA scheme with additional back-off.

• In the third phase (CTS phase), node x transmits a CTS packet if an RTS packet was received in the previous phase. After this, the packet exchange continues, extending into x's nominal sleep time.

In general, when competing for the medium, the nodes use the RTS/CTS handshake, including the virtual carrier-sense mechanism, whereby a node maintains a NAV variable. The NAV mechanism can be readily used to switch off the node during ongoing transmissions to avoid overhearing. When transmitting in a broadcast mode (for example SYNCH packets), the RTS and CTS packets are dropped and the nodes use CSMA with backoff.

If we can arrange that the schedules of node x and its neighbors are synchronized, node x and all its neighbors wake up at the same time and x can reach all of them with a single SYNCH packet. The S-MAC protocol allows neighbouring nodes to agree on the same schedule and to create virtual clusters. The clustering structure refers solely to the exchange of schedules; the transfer of data packets is not influenced by virtual clustering.

The S-MAC protocol proceeds as follows to form the virtual clusters: A node x, newly switched on, listens for a time of at least the (globally known) synchronization period. If x receives any SYNCH packet from a neighbor, it adopts the announced schedule and broadcasts it in one of the neighbours' next listen periods. In the other case, node x picks a schedule and broadcasts it. If x receives another node's schedule during the broadcast packet's contention period, it drops its own schedule and follows the other one. It might also happen that a node x receives a different schedule after it already has chosen one, for example, because bit errors destroyed previous SYNCH packets. If node x already knows about the existence of neighbors who adopted its own schedule, it keeps its schedule and in the future has to transmit its SYNCH and data packets according to both schedules. On the other hand, if x has no neighbor sharing its schedule, it drops its own and adopts the other one. Since there is always a chance to receive SYNCH packets in error, node x periodically listens for a whole synchronization period to relearn its neighborhood. This makes the virtual cluster formation fairly robust.

By this approach, a large multihop network is partitioned into "islands of schedule synchronization". Border nodes have to follow two or more different schedules for broadcasting their SYNCH packets and for forwarding data packets. Thus, they expend more energy than nodes only having neighbors of the same "schedule regime".

The periodic wakeup scheme adopted by S-MAC allows nodes to spend much time in the sleep mode, but there is also a price to pay in terms of latency. Without further modifications, the per-hop latency of S-MAC will be approximately equal to the sleep period on average when all nodes follow the same schedule. It describe the adaptive-listening scheme, which roughly halves the per-hop latency.

RTS or CTS packet belonging to a packet exchange from neighbor node y to node z. From the duration field of these packets, x can infer the time t0 when the packet exchange ends. Since it might happen that x is the next hop for z's packet, node x schedules an extra listen period around time t0 and z tries to send an extra RTS at time t0, ignoring x's normal wakeup cycle. Under ideal circumstances, x is awake when z sends the RTS and the packet can take the next hop quickly.

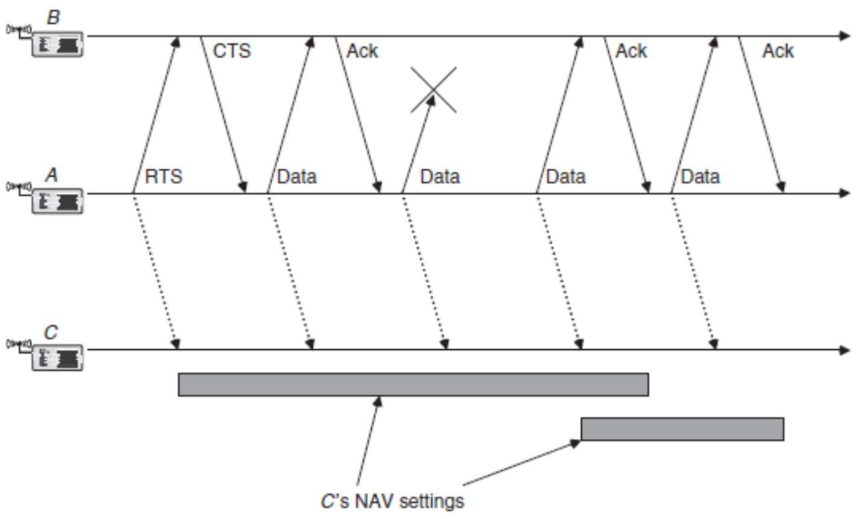


Figure 5.7 S-MAC fragmentation and NAV setting

S-MAC also adopts a message-passing approach (illustrated in Figure 7), where a message is a larger data item meaningful to the application. In-network processing usually requires the aggregating node to receive a message completely. On the other hand, on wireless media, it is advisable to break a longer packet into several shorter ones. S-MAC includes a fragmentation scheme working as follows. A series of fragments is transmitted with only one RTS/CTS exchange between the transmitting node A and receiving node B. After each fragment, B has to answer with an acknowledgment packet. All the packets (data, ack, RTS, CTS) have a duration field and a neighbouring node C is required to set its NAV field accordingly. In S-MAC, the duration field of all packets carries the remaining length of the whole transaction, including all fragments and their acknowledgments. Therefore, the whole message shall be passed at once. If one fragment needs to be retransmitted, the remaining duration is incremented by the length of a data plus ack packet, and the medium is reserved for this prolonged time. However, there is the problem of how a nonparticipating node shall learn about the elongation of the transaction when he has only heard the initial RTS or CTS packets.

This scheme has some similarities to the fragmentation scheme used in IEEE 802.11 but there are important differences. In IEEE 802.11, the RTS and CTS frame reserve the medium only for the time of the first fragment, and any fragment reserves only for the next fragment. If one packet needs to be retransmitted, the initiating node has to give up the channel and recontend for it in the same way as for a new packet. The approach taken by S-MAC reduces the latency of complete messages by suppressing intertwined transmissions of other packets.

Therefore, in a sense, this protocol is unfair because single nodes can block the medium for long time. However, the fairness requirement has a different weight in a wireless sensor network than it has in a data network where users want to have fair medium access.

S-MAC has one major drawback: it is hard to adapt the length of the wakeup period to changing load situations, since this length is essentially fixed, as is the length of the listen period.

The T-MAC protocol presented is similar to S-MAC but adaptively shortens the listen period. If a node x senses no activity on the medium for a specified duration, it is allowed to go back into sleep mode prematurely. Therefore, if no node wants to transmit to x, the listen period can be ended quickly, whereas in S-MAC, the listen period has a fixed length.

THE MEDIATION DEVICE PROTOCOL

The mediation device protocol is compatible with the peer-to-peer communication mode of the IEEE 802.15.4 low-rate WPAN standard. It allows each node in a WSN to go into sleep mode periodically and to wake up only for short times to receive packets from neighbor nodes. There is no global time reference, each node has its own sleeping schedule, and does not take care of its neighbors sleep schedules.

Upon each periodic wakeup, a node transmits a short query beacon, indicating its node address and its willingness to accept packets from other nodes. The node stays awake for some short time following the query beacon, to open up a window for incoming packets. If no packet is received during this window, the node goes back into sleep mode.

When a node wants to transmit a packet to a neighbor, it has to synchronize with it. One option would be to have the sender actively waiting for query beacon, but this wastes considerable energy for synchronization purposes only. The dynamic synchronization approach achieves this synchronization without requiring the transmitter to be awake permanently to detect the destinations query beacon. To achieve this, a mediation device (MD) is used. We first discuss the case where the mediation device is not energy constrained and can be active all the time; this scenario is illustrated in Figure 8. Because of its full duty cycle, the mediation device can receive the query beacons from all nodes in its vicinity and learn their wakeup periods.

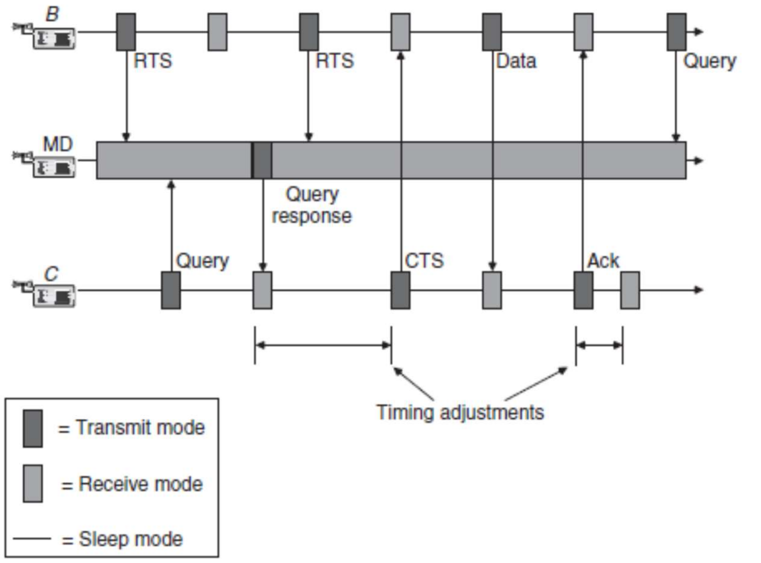


Figure 5.8 Mediation device protocol with unconstrained mediators [115, Chap. 4, Fig. 3]

Suppose that node A wants to transmit a packet to node B. Node A announces this to the mediation device by sending periodically request to send (RTS) packets, which the MD captures. Node A sends its RTS packets instead of its query beacons and thus they have the same period. Again, there is a short answer window after the RTS packets, where A listens for answers. After the MD has received A's RTS packet, it waits for B's next query beacon. The MD answers this with a query response packet, indicating A's address and a timing offset, which lets B know when to send the answering clear to send (CTS) to A such that the CTS packet hits the short answer window after A's next RTS packet. Therefore, B has learned A's period. After A has received the CTS packet, it can send its data packet and wait for B's immediate acknowledgment. After the transaction has finished, A restores its periodic wakeup cycle and starts to emit query beacons again. Node B also restores its own periodic cycle and thus decouples from A's period.

This protocol has some advantages. First, it does not require any time synchronization between the nodes, only the mediation device has to learn the periods of the nodes. Second, the protocol is asymmetric in the sense that most of the energy burden is shifted to the mediation device, which so far is assumed to be power unconstrained. The other nodes can be in the sleep state most of the time and have to spend energy only for the periodic beacons. Even when a transmitter wants to synchronize with the receiver, it does not have to wait actively for the query beacon, but can go back to sleep and wait for the mediation device to do the synchronization work. This way very low duty cycles can be supported. This protocol has also some drawbacks: The nodes transmit their query beacons without checking for ongoing transmissions and, thus, the beacons of different nodes may collide repeatedly when nodes have the same period and their wakeup periods overlap. If the wakeup periods are properly randomized and the node density is sufficiently low, this collision probability can be low too. However, in case of higher node densities or unwanted synchronization between the nodes, the number of collisions can be significant. A possible solution to this is the following: When the MD registers collisions, it might start to emit a dedicated reschedule control frame. All colliding nodes can hear this frame as long as the MD repeats it often enough. Reception of this frame causes each node to randomly pick a new period from a certain interval [a, b] indicated in the reschedule frame. If the MD continues to perceive collisions, it can enlarge the interval accordingly.

The main drawbacks, however, are the assumptions that: (i) the mediation device is energy unconstrained, which does not conform to the idea of a “simply thrown out” wireless sensor network, and (ii) there are sufficient mediation devices to cover all nodes. The distributed mediation device protocol deals with these problems in a probabilistic manner. It lets nodes randomly wake up and serve as MD for a certain time and afterward lets them go back to their regular periodic wakeup behavior. A problem with this approach is that nodes A and B may have two or more MD devices in their vicinity, causing a collision of several query responses. By properly randomizing the times where nodes decide to serve as MD, the probability of this can be kept low.

WAKEUP RADIO CONCEPTS

The ideal situation would be if a node were always in the receiving state when a packet is transmitted to it, in the transmitting state when it transmits a packet, and in the sleep state at all other times; the idle state should be avoided. The wakeup radio concept strives to achieve this goal by a simple, “powerless” receiver that can trigger a main receiver if necessary.

One proposed wakeup MAC protocol assumes the presence of several parallel data channels, separated either in frequency (FDMA) or by choosing different codes in a CDMA schemes. A node wishing to transmit a data packet randomly picks one of the channels and performs a carrier sensing operation. If the channel is busy, the node makes another random channel choice and repeats the carrier-sensing operation. After a certain number of unsuccessful trials, the node backs off for a random time and starts again. If the channel is idle, the node sends a wakeup signal to the intended receiver, indicating both the receiver identification and the channel to use. The receiver wakes up its data transceiver, tunes to the indicated channel, and the data packet transmission can proceed. Afterward, the receiver can switch its data transceiver back into sleep mode. This wakeup radio concept has the significant advantage that only the low-power wakeup transceiver has to be switched on all the time while the much more energy consuming data transceiver is nonsleeping if and only if the node is involved in data transmissions. Furthermore, this scheme is naturally traffic adaptive, that is, the MAC becomes more and more active as the traffic load increases. Periodic wakeup schemes do not have this property.

However, there are also some drawbacks. First, to our knowledge, there is no real hardware yet for such an ultralow power wakeup transceiver. Second, the range of the wakeup radio and the data radio should be the same. If the range of the wakeup radio is smaller than the range of the data radio, possibly not all neighbor nodes can be woken up. On the other hand, if the range of the wakeup radio is significantly larger, there can be a problem with local addressing schemes: These schemes do not use globally or networkwide-unique addresses but only locally unique addresses, such that no node has two or more one-hop neighbors with the same address. Put differently: A node's MAC address should be unique within its two-hop neighborhood. Since the packets exchanged in the neighbor discovery phase have to use the data channel, the two hop neighborhood as seen on the data channel might be different from the two-hop neighbourhood on the wakeup channel. Third, this scheme critically relies on the wakeup channel's ability to transport useful information like node addresses and channel identifications; this might not always be feasible for transceiver complexity reasons and additionally requires methods to handle collisions or transmission errors on the wakeup channel. If the wakeup channel does not support this feature, the transmitter wakes up all its neighbors when it emits a wakeup signal, creating an overhearing situation for most of them. If the transmitting node is about to transmit a long data packet, it might be worthwhile to prepend the data packet with a short filter packet announcing the receiving node's address. All the other nodes can go back to sleep mode after receiving the filter packet. Instead of using an extra packet, all nodes can read the bits of the data packet until the destination address appeared. If the packet's address is not identical to its own address, the node can go back into sleep mode.

2A. IEEE 802.15.4 MAC PROTOCOL / STANDARD

The Institute of Electrical and Electronics Engineers (IEEE) finalized the IEEE 802.15.4 standard in October 2003. The standard covers the physical layer and the MAC layer of a low-rate Wireless Personal Area Network (WPAN). Sometimes, people confuse IEEE 802.15.4 with ZigBee5, an emerging standard from the ZigBee alliance. ZigBee uses the services offered by IEEE 802.15.4 and adds network construction (star networks, peer-to-peer mesh networks, cluster-tree networks), security, application services, and more.

The targeted applications for IEEE 802.15.4 are in the area of wireless sensor networks, home automation, home networking, connecting devices to a PC, home security, and so on. Most of these applications require only low-to-medium bitrates (up to some few hundreds of kbps), moderate average delays without too stringent delay guarantees, and for certain nodes it is highly desirable to reduce the energy consumption to a minimum. The physical layer offers bitrates of 20 kbps (a single channel in the frequency range 868–868.6 MHz), 40 kbps (ten channels in the range between 905 and 928 MHz) and 250 kbps (16 channels in the 2.4 GHz ISM band between 2.4 and 2.485 GHz with 5-MHz spacing between the center frequencies). There are a total of 27 channels available, but the MAC protocol uses only one of these channels at a time; it is not a multichannel protocol.

The MAC protocol combines both schedule-based as well as contention-based schemes. The protocol is asymmetric in that different types of nodes with different roles are used.

Network architecture and types/roles of nodes

The standard distinguishes on the MAC layer two types of nodes:

• A Full Function Device (FFD) can operate in three different roles: it can be a PAN coordinator (PAN = Personal Area Network), a simple coordinator or a device.

• A Reduced Function Device (RFD) can operate only as a device. A device must be associated to a coordinator node (which must be a FFD) and communicates only with this, this way forming a star network. Coordinators can operate in a peer-to-peer fashion and multiple coordinators can form a Personal Area Network (PAN). The PAN is identified by a 16-bit

PAN Identifier and one of its coordinators is designated as a PAN coordinator. A coordinator handles among others the following tasks:

• It manages a list of associated devices. Devices are required to explicitly associate and disassociate with a coordinator using certain signalling packets.

• It allocates short addresses to its devices. All IEEE 802.15.4 nodes have a 64-bit device address. When a device associates with a coordinator, it may request assignment of a 16-bit short address to be used subsequently in all communications between device and coordinator. The assigned address is indicated in the association response packet issued by the coordinator.

• In the beaconed mode of IEEE 802.15.4, it transmits regularly frame beacon packets announcing the PAN identifier, a list of outstanding frames, and other parameters. Furthermore, the coordinator can accept and process requests to reserve fixed time slots to nodes and the allocations are indicated in the beacon.

• It exchanges data packets with devices and with peer coordinators. In the remainder of this section, we focus on the data exchange between coordinator and devices in a star network; a possible protocol for data exchange between coordinators is described. We start with the beaconed mode of IEEE 802.15.4.

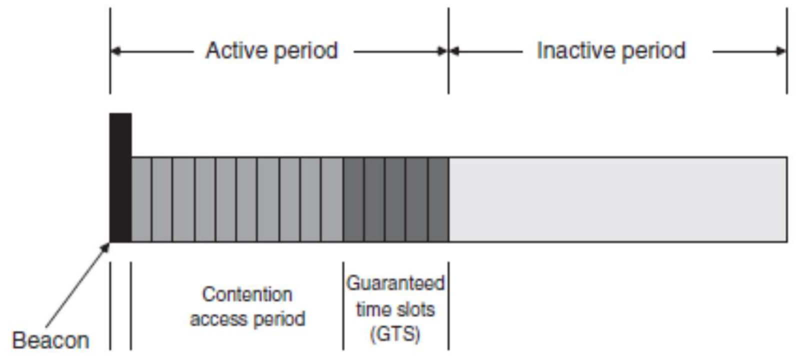


Figure 5.14 Superframe structure of IEEE 802.15.4

Super frame structure

The coordinator of a star network operating in the beaconed mode organizes channel access and data transmission with the help of a super frame structure displayed in Figure 14. All super frames have the same length. The coordinator starts each super frame by sending a frame beacon packet. The frame beacon includes a super frame specification describing the length of the various components of the following super frame:

• The super frame is subdivided into an active period and an inactive period. During the inactive period, all nodes including the coordinator can switch off their transceivers and go into sleep state. The nodes have to wake up immediately before the inactive period ends to receive the next beacon. The inactive period may be void.

• The active period is subdivided into 16 time slots. The first time slot is occupied by the beacon frame and the remaining time slots are partitioned into a Contention Access Period (CAP) followed by a number (maximal seven) of contiguous Guaranteed Time Slots (GTSs).

The length of the active and inactive period as well as the length of a single time slot and the usage of GTS slots are configurable.

The coordinator is active during the entire active period. The associated devices are active in the GTS phase only in time slots allocated to them; in all other GTS slots they can enter sleep mode. In the CAP, a device can shut down its transceiver if it has neither any own data to transmit nor any data to fetch from the coordinator.

It can be noted already from this description that coordinators do much more work than devices and the protocol is inherently asymmetric. The protocol is optimized for cases where energy constrained sensors are to be attached to energy-unconstrained nodes.

GTS management

The coordinator allocates GTS to devices only when the latter send appropriate request packets during the CAP. One flag in the request indicates whether the requested time slot is a transmit slot or a receive slot. In a transmit slot, the device transmits packets to the coordinator and in a receive slot the data flows in the reverse direction. Another field in the request specifies the desired number of contiguous time slots in the GTS phase.

The coordinator answers the request packet in two steps: An immediate acknowledgment packet confirms that the coordinator has received the request packet properly but contains no information about success or failure of the request.

After receiving the acknowledgment packet, the device is required to track the coordinator's beacons for some specified time (called a *GTSDescPersistenceTime*). When the coordinator has sufficient resources to allocate a GTS to the node, it inserts an appropriate GTS descriptor into one of the next beacon frames. This GTS descriptor specifies the short address of the requesting node and the number and position of the time slots within the GTS phase of the super frame. A device can use its allocated slots each time they are announced by the coordinator in the GTS descriptor. If the coordinator has insufficient resources, it generates a GTS descriptor for (invalid) time slot zero, indicating the available resources in the descriptors length field. Upon receiving such a descriptor, the device may consider renegotiation. If the device receives no GTS descriptor within a *GTSDescPersistenceTime* time after sending the request, it concludes that the allocation request has failed.

A GTS is allocated to a device on a regular basis until it is explicitly deallocated. The deallocation can be requested by the device by means of a special control frame. After sending this frame, the device shall not use the allocated slots any further. The coordinator can also trigger deallocation based on certain criteria. Specifically, the coordinator monitors the usage of the time slot: If the slot is not used at least once within a certain number of super frames, the slot is deallocated. The coordinator signals deallocation to the device by generating a GTS descriptor with start slot zero.

Data transfer procedures

Let us first assume that a device wants to transmit a data packet to the coordinator. If the device has an allocated transmit GTS, it wakes up just before the time slot starts and sends its packet immediately without running any carrier-sense or other collision-avoiding operations. However, the device can do so only when the full transaction consisting of the data packet and an immediate acknowledgment sent by the coordinator as well as appropriate *InterFrame* Spaces (IFSs) fit into the allocated time slots. If this is not the case or when the device does not have any allocated slots, it sends its data packet during the CAP using a slotted CSMA protocol, described below. The coordinator sends an immediate acknowledgment for the data packet.

The other case is a data transfer from the coordinator to a device. If the device has allocated a receive GTS and when the packet/acknowledgment/IFS cycle fits into these, the coordinator simply transmits the packet in the allocated time slot without further coordination. The device has to acknowledge the data packet.

The more interesting case is when the coordinator is not able to use a receive GTS. The handshake between device and coordinator is sketched in Figure 15. The coordinator announces a buffered packet to a device by including the devices address into the pending address field of the beacon frame. In fact, the device's address is included as long as the device has not retrieved the packet or a certain timer has expired. When the device finds its address in the pending address field, it sends a special data request packet during the CAP. The coordinator answers this packet with an acknowledgment packet and continues with sending the data packet. The device knows upon receiving the acknowledgment packet that it shall leave its transceiver on and prepares for the incoming data packet, which in turn is acknowledged. Otherwise, the device tries again to send the data request packet during one of the following super frames and optionally switches off its transceiver until the next beacon.

Slotted CSMA-CA protocol

When nodes have to send data or management/control packets during the CAP, they use a slotted CSMA protocol. The protocol contains no provisions against hidden-terminal situations, for example

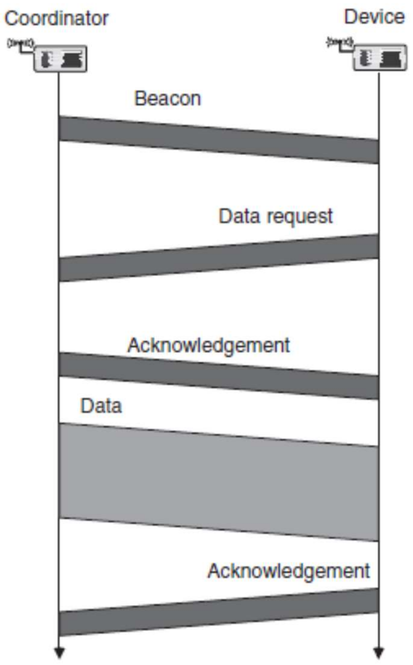


Figure 5.15 Handshake between coordinator and device when the device retrieves a packet [468, Fig. 8]

there is no RTS/CTS handshake. To reduce the probability of collisions, the protocol uses random delays; it is thus a CSMA-CA protocol (CSMA with Collision Avoidance). Using such random delays is also part of the protocols described. We describe the protocol operation in some more detail in Figure 16 also.

The time slots making up the CAP are subdivided into smaller time slots, called back off periods. One back off period has a length corresponding to 20 channel symbol times and the slots considered by the slotted CSMA-CA protocol are just these back off periods

The device maintains three variables NB, CW, and BE. The variable NB counts the number of back offs, CW indicates the size of the current congestion window, and BE is the current back off exponent. Upon arrival of a new packet to transmit, these variables are initialized with NB = 0, CW = 2, and BE = *macMinBE* (with *macMinBE* being a protocol parameter), respectively. The device awaits the next back off period boundary and draws an integer random number r from the interval [0, 2BE − 1]. The device waits for r back off periods and performs a carrier-sense operation (denoted as Clear Channel Assessment (CCA) in the standard). If the medium is idle, the device decrements CW, waits for the next back off period boundary, and senses the channel again. If the channel is still idle, the device assumes that it has won contention and starts transmission of its data packet. If either of the CCA operations shows a busy medium, the number of back offs NB and the back off exponent BE are incremented and CW is set back to CW = 2. If NB exceeds a threshold, the device drops the frame and declares a failure. Otherwise, the device again draws an integer r from [0, 2BE − 1] and waits for the indicated number of backoff slots. All subsequent steps are repeated.

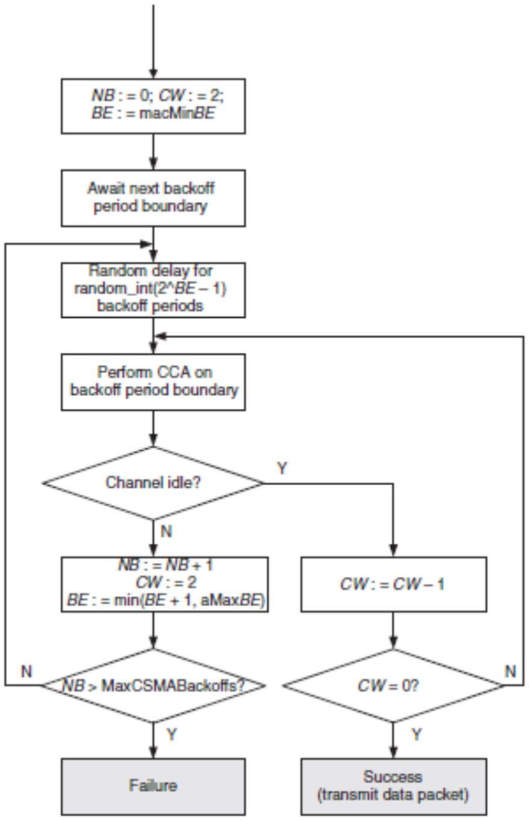

NB := 0; CW := 2;
BE := macMinBE
Await next backoff period boundary
Random delay for random_int(2^BE – 1) backoff periods
Perform CCA on backoff period boundary
Channel idle?
Y
N
NB := NB + 1
CW := 2
BE := min(BE + 1, aMaxBE)
CW := CW – 1
N
NB > MaxCSMABackoffs?
CW = 0?
N
Y
Y
Failure
Success (transmit data packet)

Figure 5.16 Schematic of the slotted CSMA-CA algorithm (simplified version of [468, Fig. 61])

Nonbeaconed mode

The IEEE 802.15.4 protocol offers a nonbeaconed mode besides the beaconed mode. Some important differences between these modes are the following:

• In the nonbeaconed mode, the coordinator does not send beacon frames nor is there any GTS mechanism. The lack of beacon packets takes away a good opportunity for devices to acquire time synchronization with the coordinator.

• All packets from devices are transmitted using an unslotted (because of the lack of time synchronization) CSMA-CA protocol. As opposed to the slotted CSMA-CA protocol, there is no synchronization to back off period boundaries and, in addition, the device performs only a single CCA operation. If this indicates an idle channel, the device infers success./

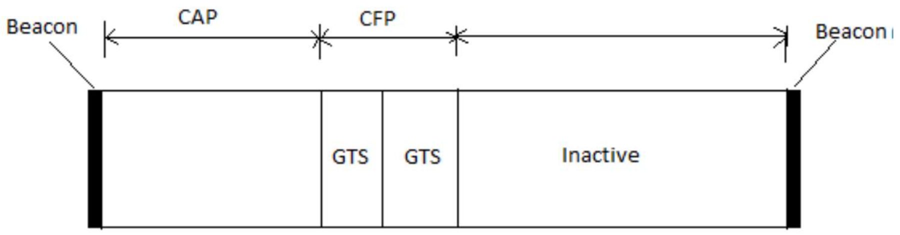
• Coordinators must be switched on constantly but devices can follow their own sleep schedule. Devices wake up for two reasons: (i) to send a data/control packet to the coordinators, or (ii) to fetch a packet destined to itself from the coordinator by using the data request/acknowledgment/ data/acknowledgment handshake (fetch cycle) discussed above. The data request packet is sent through the unslotted CSMA-CA mechanism and the following acknowledgment is sent without any further ado. When the coordinator has a data packet for the device, it transmits it using the unslotted CSMA-CA access method and the device sends an immediate acknowledgment for the data. Therefore, the device must stay awake for a certain time after sending the data request packet. The rate by which the device initiates the fetch cycle is application dependent.

B-MAC

B-MAC (Berkeley MAC) is a carrier sense media access protocol for wireless sensor networks that provides a flexible interface to obtain ultra-low power operation, effective collision avoidance, and high channel utilization. To achieve low power operation, B-MAC employs an adaptive preamble sampling scheme to reduce duty cycle and minimize idle listening. B-MAC is designed for low traffic, low power communication, and is one of the most widely used protocols (e.g. it is part of TinyOS). The BMAC module type implements the B-MAC protocol.

ZigBee MAC PROTOCOL

ZigBee MAC protocol uses CSMA/CA or TDMA for accessing the shared medium. In ZigBee MAC data is packed into super frame. Super frame structure is shown in figure. The super frame may have an active and an inactive portion. During the inactive portion, the coordinator will not interact with its PAN and may enter a lowpower mode.



The active portion consists of contention access period (CAP) and contention free period (CFP). Any device that communicates during the CAP will compete with other devices using a slotted CSMA/CA mechanism. On the other hand, the CFP contains guaranteed time slots (GTSs), a TDMA approach. The GTSs always placed at the end of the active super frame starting at a slot boundary just following the CAP. The network coordinator may allocate up to seven of these GTSs. A GTS can occupy more than one slot period. Synchronization is provided by beacon management. If TDMA mechanism is applied, device uses CFP field that contains GTSs. ZigBee MAC protocol while using CSMA/CA mechanism listen the channel continuously hence energy consumption is high. When it uses GTS management by providing a time slot to a device for transmission, only in period of time slot device has to transmit and for rest of the period it goes in sleep mode. Thus the energy consumption is reduced considerably.

After observing the simulation results it is obvious that ZigBee MAC protocol with GTS management is better if energy efficiency and throughput are more dominating factors. On the other hand T-MAC dominates ZigBee with GTS at low data rates in terms of energy consumption. But at low data rates throughput of T-MAC is lower than that of ZigBee with GTS. ZigBee with GTS has a problem of latency at higher data rates and synchronization, however many solution for resolving these problems are provided. So as per overall performance ZigBee MAC protocol is better for WBAN

2B. ADDRESS AND NAME MANAGEMENT IN WIRELESS SENSOR NETWORKS

Naming and addressing are two fundamental issues in networking. We can say very roughly that names are used to denote things (for example, nodes, data, transactions) whereas addresses supply the information needed to find these things; they help, for example, with routing in a multihop network. This distinction is not sharp; sometimes addresses are used to denote things too – an IP address contains information to both find a node (the network part of an address) and to identify a node – more precisely: a network interface within a node – within a single subnetwork (the host part).

In traditional networks like the Internet or ad hoc networks, frequently independent nodes or stations as well as the data hosted by these are named and addressed. This is adequate for the servers. The range of possible user data types is enormous and the network can support these tasks best by making the weakest assumption about the data – all data is just a pile of bits to be moved from one node to another. In wireless sensor networks, the nodes are not independent but collaborate to solve a given task and to provide the user with an interface to the external world. Therefore, it might be appropriate to shift the view from naming nodes toward naming aspects of the physical world or naming data.

The issue of naming and addressing is often tightly integrated with those parts of a protocol stack using them, for example, routing or address resolution protocols. These protocols are not the subject of this chapter but treated in subsequent chapters. Here we focus on aspects like address allocation, address representation, and proper use of different addressing/naming schemes in wireless sensor networks.

Fundamentals

Use of addresses and names in (sensor) networks

In most computer and sensor networks, the following types of names, addresses, and identifiers can be found.

Unique node identifier A unique node identifier (UID) is a persistent data item unique for every node. An example of a UID might be a combination of a vendor name, a product name, and a serial number, assigned at manufacturing time. Such a UID may or may not have any function in the protocol stack.

MAC address A MAC address is used to distinguish between one-hop neighbors of a node. This is particularly important in wireless sensor networks using contention-based MAC protocols, since by including a MAC addresses into unicast MAC packets a node can determine which packets are not destined to it and go into sleep mode while such a packet is in transit. This overhearing avoidance is an important method of conserving energy at the MAC layer.

Network address A network address is used to find and denote a node over multiple hops and therefore network addresses are often connected to routing.

Network identifiers In geographically overlapping wireless (sensor) networks of the same type and working in the same frequency band, it is also important to distinguish the networks by means of network identifiers. An example of medical body area sensor networks for clinical patients in the same room have to be distinguished to prevent confusion of sensor data belonging to different patients.

Resource identifiers **A name or resource identifier** is represented in user-understandable terms or in a way that "means something" to the user. For example, upon reading the name www.xemacs.org, an experienced user knows that (i) the thing the name refers to is likely a web server and (ii) the user can find information about a great text editor. In contrast, upon looking at the IP address 199.184.165.136, hardly any user draws either conclusion. Names can refer to nodes, groups of nodes, data items, or similar abstractions. A single node can have many names and addresses. For example, the WWW server www.xemacs.org has the name www.xemacs.org, it has the IP address 199.184.165.136 and, assuming that the server is attached to an Ethernet, it has a 48-bit IEEE MAC address. The mapping between user-friendly names like www.xemacs.org and the addresses relevant for network operation is carried out by binding services. This mapping is also often referred to as name resolution. In our example, the domain name service (DNS) provides the mapping from the name to the IP address while the address resolution protocol (ARP) maps the IP address to a MAC address.

The MAC addresses are indispensable if the MAC protocol shall employ overhearing avoidance and go into sleep mode as often as possible. However, do MAC addresses need to be globally or networkwide unique? No, since the scope of a MAC protocol is communication between neighboring nodes and it is sufficient that addresses are locally unique within a two-hop neighborhood. This requirement ensures that no two neighbors of a selected node have the same MAC address. As discussed above, locally unique addresses potentially are short but need an address assignment protocol.

How about higher-layer addresses, specifically network layer addresses, which for traditional routing protocols must be globally or networkwide unique? We will discuss briefly that fulfilling this requirement is a formidable task. We will argue also that this requirement is not really necessary in wireless sensor networks since after all the whole network is not a collection of individual nodes belonging to individual users but the nodes collaborate to process signals and events from the physical environment. The key argument is that users ultimately are interested in the data and not in the individual or groups of nodes delivering them. Taking this a step further, the data can also influence the operation of protocols, which is the essence of data-centric networking. Data-centric or content-based addressing schemes are thus important.

ASSIGNMENT OF MAC ADDRESSES

In this section, we discuss assignment methods for MAC addresses. The assignment of globally unique MAC addresses is undesirable in sensor networks with mostly small packets.

An a priori assignment of networkwide unique addresses is feasible only if it can be done with reasonable effort. But there is still the problem that the overhead to represent addresses can be considerable although not as large as in globally unique addresses. For example, up to 16,384 nodes can be addressed with 14 bits and this number is much friendlier than 48 bits used for globally unique IEEE addresses.

Therefore, we concentrate on dynamic and distributed assignment of networkwide and local addresses. The protocols discussed in this section differ in the amount and scope of collaboration with other nodes.

Distributed assignment of networkwide addresses

Let us start with a very simple approach: A node randomly picks an address from a given address range and hopes that this address is unique. For ease of exposition, we assume that this address range is given by the integers between 0 and $2^m - 1$ and an address can thus be represented with m bits. The address space has a size of $n = 2^m$ addresses.

A node chooses its address without any prior information, in which case it is best to use a uniform distribution on the address range since this has maximum entropy. However, this approach is not without problems, as is shown in the following example.

Example (Random address assignment) Suppose that we have k nodes and each of these nodes picks uniformly and independently a random address from 0 to $2^m - 1$. What is the probability that these nodes choose a conflict-free assignment? A similar problem is known as the "birthday problem"[3] and can be answered by simple combinatorial arguments. For k = 1 this probability is one. For k = 2, the second node picks with probability $\frac{n-1}{n}$ an address different from the first node's choice. For k = 3, the third node picks with probability $\frac{(n-1).(n-2)}{n^2}$ an address different from the first two and so forth. Hence, we have the probability P(n, k) to find a conflict-free assignment

$$P(n,k) = 1 \cdot \frac{n-1}{n} \cdot \ldots \cdot \frac{n-k+1}{n} = \frac{1}{n^k} \cdot \frac{n!}{(n-k)!} = \frac{k!}{n^k} \cdot \binom{n}{k},$$

which, by Stirlings approximation (n! ≈ √2π · nn+1/2 · e^{-n} [255, Chap. II]), is approximately given by:

$$P(n,k) \approx e^{-k} \cdot \left(\frac{n}{n-k}\right)^{(n-k)+1/2}.$$

For an address field of m = 14 bits size, corresponding to n = 214 = 16384 distinct addresses, we show in Figure 3 the probability P(n, k) for different values of k. Already, for quite small values of k, the probability of conflicts becomes close to one. For example: for k = 275 the conflict probability is already larger than 90% but only ≈1.7% of the address space is used! Therefore, this method of random assignment quickly leads to address conflicts. To preserve networkwide uniqueness, either a conflict- resolution protocol is needed or more clever assignment schemes should be chosen.

Can we do better? A node can try to obtain information about already-allocated addresses by overhearing packets in its vicinity and avoiding these addresses. In many sensor network applications, where nodes transmit their sensor data to a local coordinator aggregating and processing the data, overhearing can avoid many conflicts with other local nodes transmitting to the same coordinator.

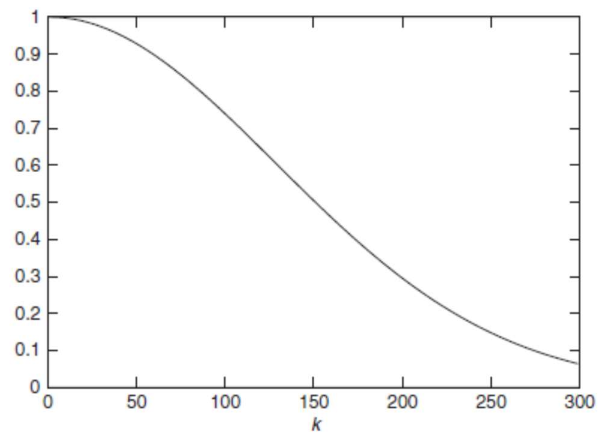


Figure 7.3 "Birthday probability" that k out of $n = 2^{14}$ station pick random addresses without conflicts

With random address assignment we are faced with address collisions with high probability. How do we deal with them? The first solution is to simply accept them and do nothing. Other techniques have been investigated in the context of IP address assignment in MANETs:

• An address autoconfiguration protocol suitable for MANETs. A node starts by randomly selecting a temporary address and a proposed fixed address and sends out an address request control packet, carrying the chosen temporary and fixed addresses. The temporary address is allocated from a dedicated address pool, being disjoint from the pool of true node addresses. The underlying routing protocol tries to find a path to a node having the same fixed address. If there exists such a node (and a path to it), an address reply packet is generated and sent toward the temporary address. Upon receiving this reply, the node knows that the chosen fixed address is allocated and tries another address. If no address reply is received within a certain time, the node repeats the address request packet a configurable number of times to compensate for possibly lost address reply packets. If still no address reply is received after all trials are exhausted, the node accepts the chosen IP address. This protocol breaks down if the delays cannot be bounded, for example, after network partitions. If in sensor networks this scheme is applied to MAC addresses instead of network addresses, then other nodes do not have any routing information, and the address request/reply packets must be flooded into the network. Further problems of this approach are discussed in reference.

• The address assignment problem as a distributed agreement problem, a well-known problem in distributed systems. A requesting node (the requester) contacts a neighboring node already having an address, the initiator. The initiator keeps a table of all known address assignments and picks an unused address. The initiator then disseminates the proposed new address to all nodes in the network and collects the answers. All nodes put the proposed address into a list of candidate addresses. If a node finds the address either in the candidate list or in its local list of known assignments, it answers with a reject packet, otherwise it answers with an accept packet. If all known nodes have answered with an accept packet, the initiator assigns the address to the requester and informs all other nodes in the network that the assignment now is permanent. Otherwise, the initiator picks another address and tries again. This approach is similar to a two-phase commit protocol and clearly produces too much overhead in terms of transmitted packets and buffer space requirements to be feasible in wireless sensor networks.

• A hierarchical address autoconfiguration algorithm for IPv6 addresses intended for MANETs is described. Some nodes in the network become leader nodes and choose a subnet ID randomly. A DAD is executed between leader nodes to guarantee uniqueness of subnet IDs. Other nodes create their addresses from the subnet ID of their leader and a local address (for example, based on the nodes MAC address). A leader is elected for each network partition, assigning addresses to newly arriving nodes. Mergers of networks are detected by introducing separate network identifiers. The observation that the networkwide uniqueness requirement translates into a distributed consensus problem gives some insight into lower bounds on the complexity and communication overhead involved in this assignment problem. The price in terms of communication overhead is to be paid upon an address assignment trial, for example, when the network must be flooded because the requesting node has no routable address. Alternatively, if a proactive routing protocol is used, the nodes possess tables of used addresses that can be consulted quickly or can infer the presence of duplicate nodes because of receiving bogus routing messages carrying the node's source address. However, on-demand routing protocols are more popular in sensor networks than are proactive protocols because of their overhead.

3. ROUTING PROTOCOLS

The many faces of forwarding and routing

Whenever a source node cannot send its packets directly to its destination node but has to rely on the assistance of intermediate nodes to forward these packets on its behalf, a multihop network

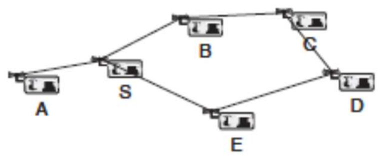


Figure 11.1 A simple example of routing in a multihop network – node S sends packets to node D

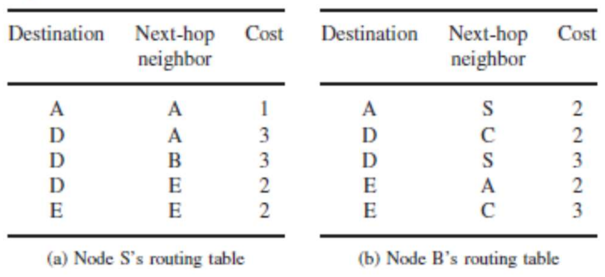
results – an example is shown in Figure 1. In such a network, an intermediate node (as well as the source node) has to decide to which neighbouring node an incoming packet should be passed on so that it eventually reaches the destination – for example, node S sending to node A would not do. This act of passing on is called forwarding, and several different options exist how to organize this forwarding process.

The simplest forwarding rule is to flood the network: Send an incoming packet to all neighbors. As long as source and destination node are in the same connected component of the network, the packet is sure to arrive at the destination. To avoid packets circulating endlessly, a node should only forward packets it has not yet seen (necessitating, for example, unique source identifier and sequence numbers in the packet). Also, packets usually carry some form of expiration date (time to live, maximum number of hops) to avoid needless propagation of the packet (e.g. if the destination node is not reachable at all).

An alternative to forwarding the packet to all neighbors is to forward it to an arbitrary one. Such gossiping results in the packet randomly traversing the network in the hope of eventually finding the destination node. Clearly, the packet delay can be substantially larger. Flooding and gossiping are two extreme ends of a design spectrum; alternatively, the source could send out more than a single packet on a random walk or each node could forward an incoming packet to a subset of its neighbors – for example, as determined by a topology-control algorithm, equivalent to flooding on a reduced topology. This last option is sometimes called controlled flooding.

While these forwarding rules are simple, their performance in terms of number of sent packets or delay, . for example, is likely poor. These shortcomings are due to ignoring the network's topology. In the example of Figure 1, without knowing that node A is even further away from the destination node D, the source node S has no means of avoiding it when forwarding its own packet. Hence, some information about the suitability of a neighbor in the forwarding process would be required. A neighbour's suitability is captured by the cost it incurs to send a packet to its destination via this particular neighbor. These costs can be measured in various metrics, for example, the minimal number of hops or the minimal total energy it requires to reach the destination via the given neighbor. Each node collects these costs in routing tables; Table shows two examples.

Table 11.1 Routing tables for some nodes from Figure 11.1, using hop count as cost metric



Destination	Next-hop neighbor	Cost
A	A	1
D	A	3
D	B	3
D	E	2
E	E	2

(a) Node S's routing table

Destination	Next-hop neighbor	Cost
A	S	2
D	C	2
D	S	3
E	A	2
E	C	3

(b) Node B's routing table

Determining these routing tables is the task of the routing algorithm with the help of the routing protocol. In wired networks, these protocols are usually based on link state or distance vector algorithms (Dijkstra's or Bellman–Ford). In a wireless, possibly mobile, multihop network, different approaches are required. Routing protocols here should be distributed, have low overhead, be self-configuring, and be able to cope with frequently changing network topologies. This question of ad hoc routing has received a considerable amount of attention in the research literature and a large number of ad hoc routing protocols have been developed. A commonly used taxonomy classifies these protocols as either (i) table-driven or proactive protocols, which are "conservative" protocols in that they do try to keep accurate information in their routing tables, or (ii) on-demand protocols, which do not attempt to maintain routing tables at all times but only construct them when a packet is to be sent to a destination for which no routing information is available. As usual, the borders are not sharp between these classes and there are some ideas for hybrid solutions. Examples for table-driven protocols are Destination-Sequenced Distance Vector (DSDV), Clusterhead Gateway Switch Routing (CGSR), and Wireless Routing Protocol (WRP). Popular on-demand protocols are, among others, Dynamic Source Routing (DSR), AODV, Temporally Ordered Routing Algorithm (TORA), Associativity-Based Routing (ABR), and Signal Stability Routing (SSR). A common problem for many of these ad hoc routing protocols is that they require flooding of control messages to explore the network topology and to find destination nodes**.**

The full range of ad hoc networking is too broad to be covered here in full detail and not all the research in this context is relevant to the case of wireless sensor networks (e.g. routing of multimedia traffic in ad hoc networks). Rather, the exposition in this chapter will concentrate on the most crucial aspect: energy efficiency. This pertains both to the selection of energy-efficient routes as well as to the overhead imposed by the construction of the routing tables themselves. Secondary aspects that are briefly touched upon are stability and dependability of the routes as well as routing table size (nodes with limited memory cannot store large routing tables). In particular, the issues related to mobile ad hoc networks where all nodes move around will be considered at best superficially; the case of a mobile sink is briefly discussed at the end of this chapter.

In addition to energy efficiency, resiliency also can be an important consideration for WSNs. For example, when nodes rely on energy scavenging for their operation, they might have to power off at unforeseeable points in time until enough energy has been harvested again. Consequently, it may be desirable to use not only a single path between a sender and receiver but to at least explore multiple paths. Such multiple paths provide not only redundancy in the path selection but can also be used for load balancing, for example, to evenly spread the energy consumption required for forwarding.

Apart from the unicast case, where one node sends packets to another, uniquely identified node, both broadcasting (sending to all nodes in a network) and multicasting (sending to a specified group of nodes) are important tasks in WSNs. One special way to define such a group is by specifying a geographic region such that all nodes in the region should receive the packet. This requires nodes to know about their positions, and once such knowledge is available, it can be used both to assist conventional routing and as a definition for target groups in a multicast sense.

All these options discussed so far are in a sense node-centric in that certain nodes are addressed by source nodes and packets should be delivered to these nodes. An alternative view on routing is enabled by data-centric network where the set of target nodes is only implicitly described by providing certain attributes that these nodes have to fulfil (geographic routing can indeed be conceived of as data-centric routing in this sense). These routing approaches are very important in WSNs as they reflect natural usage cases – in particular, collection of data and dissemination of events to interested nodes.

Gossiping and agent-based unicast forwarding

Basic idea

This section deals with forwarding schemes that attempt to work without routing tables, either because the overhead to create these tables is deemed prohibitive (when a node only issues a command, for example, and does not expect any answers) or because these tables are to be constructed in the first place. The simplest option is flooding – forwarding each new, incoming message – but more efficient schemes are desirable. The topology-control reducing the forwarding set can considerably improve efficiency. The approaches taken here try to find a forwarding set without recurring to topology-control mechanisms but try to solve it strictly locally. It draws a parallel between the distribution of data in a replicated database system and epidemics occurring in human populations. Various options are described; one is "rumor mongering": Once a site receives an update, it periodically, randomly chooses another site to propagate this update to; it stops doing so after the update has already been received by a sufficient number of sites (supposedly similar to the way rumors or epidemics are propagating in a population). The goal is to spread updates to all nodes as fast as possible while minimizing the message overhead. The question is to select neighbors for gossiping the rumor at hand (how often, which neighbors, etc.) This same idea of randomly choosing forwarding nodes can also be applied to wireless sensor networks. There is, in fact, one advantage of wireless communication over wired communication that comes to bear in this context: a single transmission can be received by all neighbouring nodes in radio ranges, thus incurring transmission costs only once for many neighbors. This property has been called the wireless multicast advantage. Evidently, whether this advantage is actually relevant heavily depends on the deployed MAC protocol and on the relative costs of sending and receiving.

Randomized forwarding

On the basis of this consideration, the question how information spreads in a wireless network by such a gossiping mechanism. The key parameter of their mechanism is the probability with which a node retransmits a newly incoming message. In the simplest case, this probability is constant. They show that there is a critical probability value below which the gossip – typically – dies out quickly and reaches only a small number of nodes. If, on the other hand, nodes use a probability larger than the critical threshold to retransmit messages, then most of the gossips reach (almost) all of the nodes in the network. Typical value for the critical threshold are about 65 to 75 %. The existence of this threshold shows that gossiping exhibits a typical phase transition behaviour, in accordance with what can be expected from a percolation-theoretical treatment of the problem. The nodes near the boundary of the sensor network's deployment region are critical as they have, on average, a smaller number of neighbors than nodes in the center of the region. They discuss various possible remedies, for example, (i) to have the neighbors of a node with few neighbors retransmit with higher probability, (ii) to prevent a gossip from dying out too fast by retransmitting messages over the first few hops with probability 1, or (iii) to retransmit a message (despite having decided not to do so) if the node does not overhear the message repeated from at least one of its neighbors (the actual minimum number is an optimization problem). Using such enhancements, the ratio of nodes that receive a gossip is considerably increased.

They propose a couple of heuristics that let a node decide when to repeat a received or overheard packet. They look at rules that are based on counters (do not retransmit when a message has been overheard a certain number of times), distance based (do not retransmit if the distance to the sender is small), or location based (determine the additional coverage that could be obtained by retransmitting, based on the location of the nodes that have already sent the message).

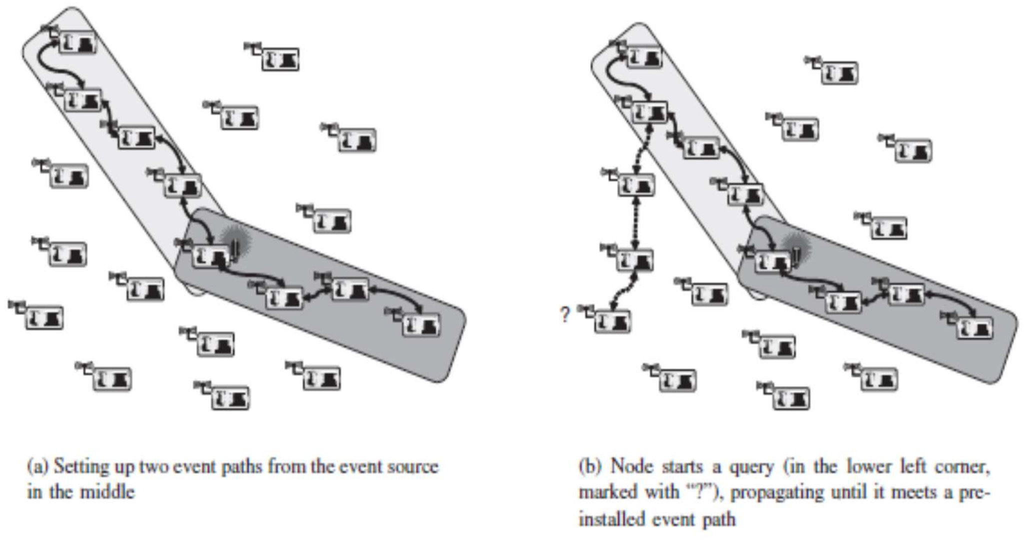
Random walks

Limiting flooding by only probabilistically forwarding a packet is only one option. Another approach is to think of a data packet as an "agent" that wanders through the network in search of its destination. In the simplest form, this is a purely random walk, where a packet is randomly forwarded to an arbitrary neighbor. Hence, the agents are sent via unicast, not via local broadcast, to their next hop. Instead of a single "agent", several of them can be injected into the network by the source to shorten the time to arrival by parallelism. The probabilistic properties of random walks have been extensively studied, but without any additional measures, a purely random walk is too inefficient to be useful for WSNs. Two examples of such extensions to random walks shall be briefly discussed.

Rumor routing

This approach in the context of event notification: Assume some sensors are interested in certain events (e.g. temperature exceeding a given value) and a sensor can observe it. Classical options are to flood either the query for the event or the notifications that an event has occurred through the entire network. The "rumor routing" approach proposed here does not flood the network with information about an event occurrence but only installs a few paths in the network by sending out one or several agents. Each of these agents propagates from node to node and installs routing information about the event in each node that it visited. This is illustrated in Figure 2(a) where the node in the middle detects an event and installs two event paths in the network (shaded areas). Once a node tries to query an event (or to detect whether an event actually exists), it also sends out one or more agents. Such a search agent is forwarded through the network until it intersects with a preinstalled event path and then knows how to find an event. In Figure 2(b), the node in the lower left corner sends out such a search, which happens to propagate upward until it intersects with one event path. All these agent propagations are limited to avoid endless circling of data.

The rationale behind this technique is the relatively high probability that two random lines in a square intersect each other; state a probability of about 69 %. While neither the event paths nor the search paths will in reality be straight lines, the approximation is claimed to be good enough. Using five instead of one event paths increases this probability to about 99.7 %. In effect, rumor routing allows to trade off effort in path creation and/or search against probability of detecting an event.



(a) Setting up two event paths from the event source in the middle

(b) Node starts a query (in the lower left corner, marked with "?"), propagating until it meets a pre-installed event path

Figure 11.2 Rumor routing example

There are a few more functionalities included in rumor routing. For example, agents spread information about more than one event if they have crossed an event path for another event. Also, an agent uses opportunities to shorten existing event paths if they know about shorter paths.

Random walks with known destination

A different perspective on random walks is taken. They consider the problem of a WSN where lots of nodes are redundantly deployed but some of these nodes are randomly turned off and later on again (e.g. due to energy scavenging), giving rise to a dynamic graph. The idea is to use random walks to ensure that all possible paths in the network are used with equal probability, spreading the forwarding burden over all nodes. To do so, only local computations should be required for each node.

The concrete scenario under investigation is a rectangular grid of nodes where the source is in the upper left corner and the destination in the lower right corner; nodes in between are randomly active. For such a situation, formulas are developed to compute the probability of passing an incoming packet either down or to the right, based on a distributed computation of the number of paths from the source to an intermediate node and from the intermediate node to the destination. Compared to assigning both the lower node and the node to the right a probability of 50% each, the random walks based on these formulas indeed result in a much more uniform traffic density in the network.

ENERGY-EFFICIENT UNICAST

Overview

At a first glance, energy-efficient unicast routing appears to be a simple problem: take the network graph, assign to each link a cost value that reflects the energy consumption across this link, and pick any algorithm that computes least-cost paths in a graph. The shortest path algorithm to obtain routes with minimal total transmission power. What qualifies as a good cost metric in general is, however, anything but clear and depends on the precise intention of energy-efficient unicast routing. In fact, there are various aspects how energy or power efficiency can be conceived of in a routing context. Figure 3 shows an example scenario for a communication between nodes A and H including link energy costs and available battery capacity per node.

Minimize energy per packet (or per bit) The most straightforward formulation is to look at the total energy required to transport a packet over a multihop path from source to destination (including all overheads). The goal is then to minimize, for each packet, this total amount of energy by selecting a good route.

Minimizing the hop count will typically not achieve this goal as routes with few hops might include hops with large transmission power to cover large distances – but be aware of distance-independent, constant offsets in the energy-consumption model. Nonetheless, this cost metric can be easily included in standard routing algorithms. It can lead to widely differing energy consumption on different nodes.

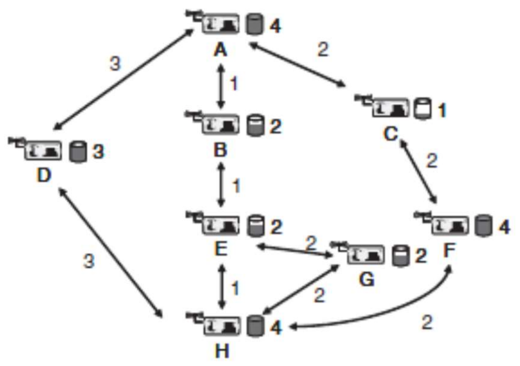


Figure 11.3 Various example routes for communication between nodes A and H, showing energy costs per packet for each link and available battery capacity for each node (adapted from reference [17])

In the example of Figure 3, the minimum energy route is A-B-E-H, requiring 3 units of energy. The minimum hop count route would be A-D-H, requiring 6 units of energy.

Maximize network lifetime A WSN's task is not to transport data, but to observe (and possibly control). Hence, energy-efficient transmission is at best a means to an end and the actual end should be the optimization goal: the network should be able to fulfil its duty for as long as possible.

Which event to use to demarcate the end of a network's lifetime is, however, not clear either. Several options exist:

• Time until the first node fails.

• Time until there is a spot that is not covered by the network (loss of coverage, a useful metric only for redundantly deployed networks).

• Time until network partition (when there are two nodes that can no longer communicate with each other).

While these aspects are related, they require different solutions. For the network partition, for example, nodes in the graph's minimal cut set should have equal energy consumption (or rather, supplies) to ensure maximum time to partition. Also, their solutions can be infeasible – for example, maximizing the time to network partition is reported as NP-complete.

Moreover, maximizing the time until the first node runs out of energy does not have a constant competitive ratio with the optimal off-line algorithm that knows the arrivals of future packets (when optimizing the number of messages the network can successfully carry, a competitive ratio logarithmic in the number of nodes can be shown). Because of these theoretical limitations, only approximative solutions are practically relevant.

Routing considering available battery energy While maximizing the network lifetime is clearly a useful goal, it is not immediately obvious how to reach this goal using observable parameters of an actual network. As the finite energy supply in nodes' batteries is the limiting factor to network lifetime, it stands to reason to use information about battery status in

routing decisions. Some of the possibilities are:

Maximum Total Available Battery Capacity Choose that route where the sum of the available battery capacity is maximized, without taking needless detours (called, slightly incorrectly, "maximum available power").

Looking only at the intermediate nodes in Figure 3, route A-B-E-G-H has a total available capacity of 6 units, but that is only because of the extra node G that is not really needed – such detours can of course arbitrarily increase this metric. Hence, AB- E-G-H should be discarded as it contains A-B-E-H as a proper subset. Eventually, route A-C-F-H is selected.

Minimum Battery Cost Routing (MBCR) Instead of looking directly at the sum of available battery capacities along a given path, MBCR instead looks at the "reluctance" of a node to route traffic. This reluctance increases as its battery is drained; for example, reluctance or routing cost can be measured as the reciprocal of the battery capacity. Then, the cost of a path is the sum of this reciprocals and the rule is to pick that path with the smallest cost. Since the reciprocal function assigns high costs to nodes with low battery capacity, this will automatically shift traffic away from routes with nodes about to run out of energy.

In the example of Figure 3, route A-C-F-H is assigned a cost of 1/1 + 1/4 = 1.25, but route A-D-H only has cost 1/3. Consequently, this route is chosen, protecting node C from needless effort.

Min–Max Battery Cost Routing (MMBCR) This scheme follows a similar intention, to protect nodes with low energy battery resources. Instead of using the sum of reciprocal battery levels, simply the largest reciprocal level of all nodes along a path is used as the cost for this path. Then, again the path with the smallest cost is used. In this sense, the optimal path is chosen by minimizing over a maximum. The same effect is achieved by using the smallest battery level along a path and then maximizing over these path values. This is then a maximum/minimum formulation of the problem. In the example of Figure 3, route A-D-H will be selected.

Conditional Max–Min Battery Capacity Routing (CMMBCR) Another option is to conditionalize upon the actual battery power levels available. If there are routes along which all nodes have a battery level exceeding a given threshold, then select the route that requires the lowest energy per bit. If there is no such route, then pick that route which maximizes the minimum battery level.

Minimize variance in power levels To ensure a long network lifetime, one strategy is to use up all the batteries uniformly to avoid some nodes prematurely running out of energy and disrupting the network. Hence, routes should be chosen such that the variance in battery levels between different routes is reduced.

Minimum Total Transmission Power Routing (MTPR) Without actually considering routing as such, situation of several nodes transmitting directly to their destination, mutually causing interference with each other. A given transmission is successful if its SINR exceeds a given threshold. The goal is to find an assignment of transmission power values for each transmitter (given the channel attenuation metric) such that all transmissions are successful and that the sum of all power values is minimized. MTPR is of course also applicable to multihop networks.

A direct performance comparison between these concepts is difficult as they are trying to fulfil different objectives. Moreover, while these objectives are fairly easy to formulate, it is not trivial to implement them in a distributed protocol that judiciously balances the overhead necessary to collect routing information with the performance gained by clever routing choices. The following section describes some concrete protocols that tackle this challenge; It show that a non-power-aware protocol can actually have (in many circumstances) a better energy-consumption behaviour than some straightforward power-aware solutions.

MULTIPATH UNICAST ROUTING

Overview

The unicast routing protocols discussed so far tried to construct a single energy-efficient path (with whatever interpretation of this term) between a sink and a receiver, typically by giving a clever meaning to the "cost" of a link. These costs try to balance, for example, energy required for communication across this link against the battery capacity of the nodes involved. Focusing on choosing the best possible path, however, limits the opportunities for making such trade-offs. Extending the focus to multiple paths and trying to balance, for example, energy consumption across multiple path is therefore an option worthwhile exploring. Moreover, multiple paths provide redundancy in that they can serve as "hot standbys" to quickly switch to when a node or a link on a primary path fails.

Such multipath routing protocols construct several paths between a given sender and receiver. The basic goal is to find k paths that do not have either links or nodes in common (apart from source and destination node, of course. Once the paths have been established by the routing protocol, the forwarding phase can then dynamically decide which path (or even paths) to choose to transmit a packet. This can increase the robustness of the forwarding process toward link or node failures.

Applying multipath routing to wireless networks, both general ad hoc and sensor networks, is a well-studied problem. Some of the more WSN-relevant papers are briefly described here.

Sequential Assignment Routing (SAR)

As a basic rule of thumb, computing such k-disjoint paths requires about k times more overhead than a single-path routing protocol. It try to reduce the multipath-induced overhead by focusing the disjointness requirements to that part of a network where they truly matter – near the data sink, as the nodes close to the sink are (often) those that likely are going to fail first because of depleted battery resources. Hence, they only require paths to use different neighbors of the sink. The Sequential Assignment Routing (SAR) algorithm achieves this objective by constructing trees outward from each sink neighbor; in the end, most nodes will then be part of several such trees. A packet's actual path is then selected by the source on the basis of information about the available battery resources along the path and the performance metrics (e.g. delay) of a given path.

Constructing energy-efficient secondary paths

When using multiple paths as standby paths to quickly switch to when the primary path fails, an obvious concern is that of the energy efficiency of these secondary paths compared to the (hopefully) optimal primary path. Consider the question how to construct the secondary paths from this perspective, without worrying about battery capacity or similar metrics along the various paths. Their first observation is that strictly requiring node disjointness between the various paths tends to produce rather inefficient secondary paths as large detours can be necessary. To overcome this problem and yet retain the robustness advantages of multiple paths, they suggest the construction of so-called “braided” paths (sometimes also called “meshed” multipaths). These braided paths are only required to leave out some (even only one) node(s) of the primary path but are free to use other nodes on the primary path. This relaxed disjointness requirement results in paths that can “stay close” to the primary path and are therefore likely to have a similar, close to optimal energy efficiency as the primary path. Figure 5 illustrates these two redundant paths’ concepts.

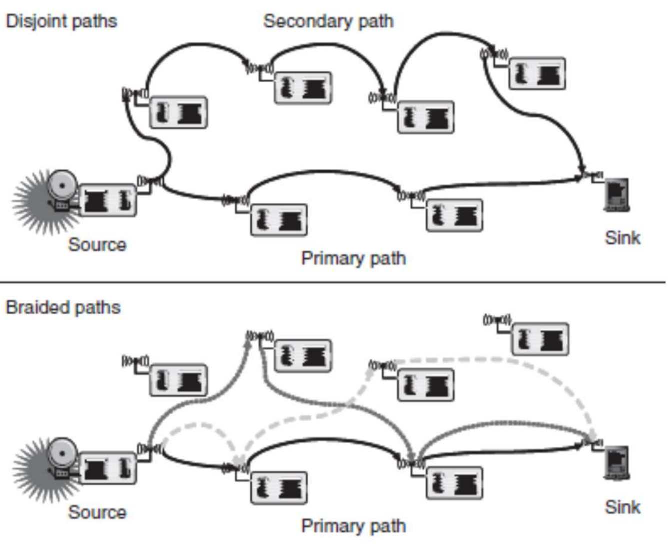


Figure 11.5 Disjoint and braided paths around a primary path

Constructing these two different types of redundant paths is simple in a centralized fashion; a distributed construction is described as a modification to the reinforcement mechanism (popularized by directed diffusion). For disjoint paths, the data sink not only reinforces the primary path via its best neighbor toward the data source but also sends out an “alternate path” reinforcement to its second-best neighbor (or several such neighbors, for multiple standby paths). This alternate path reinforcement is then forwarded toward the best neighbor that is not already on the primary path. For braided paths, each node on the primary path (including the sink) sends out such an alternate path reinforcement, which only has to avoid the next upstream node on the primary path but is then free to use nodes on the primary path.

Which of these two schemes is advantageous clearly depends on the node failure patterns. The authors look at both independent node failures and so-called “patterned” failures (all nodes within a circle of known radius around randomly selected points fail; the appearance of points follows a Poisson distribution). The main figure of merit is the “resilience”, the percentage of cases where the failure of the primary path is compensated for by an alternative path.

Simultaneous transmissions over multiple paths

When using multiple paths as a standby for the primary path, failover times might be improved compared to strictly single-path solutions. Nevertheless, there is some delay in detecting the need to use a secondary path. Depending on which node makes this decision – only the source node or any node on the primary path – there can be more or less overhead involved.

To further shorten the time to delivery and to increase the delivery ratio of a given packet, it is also conceivable to use all or several of the multiple paths simultaneously. The simplest idea is to assume node-disjoint paths and to send several copies of a given packet over these different paths to the destination. Clearly, this trades off resource consumption against packet error rates. A performance comparison of such a packet replication scheme with other multipath schemes, for example, one that uses additional FEC to protect against packet errors. It combine the basic idea of sending packet replicas with FEC by proposing to split a packet and its error correction redundancy over several paths, to be recombined at the receiver. The degree of redundancy and the number of paths can be tuned to the expected error behaviour, trading off overhead against residual packet error rate.

Randomly choosing one of several paths

When maintaining multiple paths, it actually makes sense also to use paths that are less energy efficient than the optimal one. One reason to do so is to share the load among all nodes in order to use the available battery capacity in the network better.

A relatively straightforward way of doing so is described. Each node maintains an energy cost estimate for each of its neighbors (toward the destination, packets are not routed "away" from their destination). When forwarding a packet, the next hop is randomly chosen proportional to the energy consumption of the path over this neighbor. To the upstream node, the appropriately weighted average of these costs (i.e., the harmonic mean of the costs) is reported.

More formally: suppose node v has neighbors $v1$ to vn that advertise cost $c1, \ldots, cn$, respectively. Node v will advertise $c = n / \sum_{i=1}^{n} c_i$ as its own cost and will forward an incoming packet to neighbor i with probability $(1 / c_i) / (1 / \sum_{i=1}^{n} c_i)$.

This routing approach is extended by introducing the notion of altruists. An altruistic node is one that is willing to do more work on behalf of its neighbors, for example, because it has a tethered power supply. It show that such asymmetric nodes can be efficiently exploited by the routing protocol, simply by occasionally broadcasting "altruistic announcements" into the network.

Trade-off analysis

Clearly, supporting such multiple paths in a network implies a trade-off between robustness (the probability that paths are available even after node failure) and energy efficiency (as both the management of these paths and the nonoptimal choices made for packet forwarding decisions imply increased energy expenditure) – irrespective of the concrete routing protocol in use. This trade-off is analysed the robustness gained by multiple paths with those owing to simply increasing transmission power.

Their basic observation, made in a simplified scenario of five nodes, is that it is not possible to simultaneously optimize both robustness and energy efficiency of a given set of paths, but rather that only the notion of Pareto optimality can be applied. They do observe, however, that single path solutions that require a larger transmission power tend to dominate multipath solutions with low transmission power.

To test these basic observations, the authors conducted a set of simulation experiments, comparing various degrees of redundancy via braided multipaths. As one might expect, for low failure rates, the robustness of even two paths is perfectly sufficient. The two controlled parameters are the degree of redundancy via additional paths and the maximum transmission power, enabling the system to bridge across failed nodes if necessary. Using these two factors to influence Pareto optimality with respect to the robustness and energy efficiency objectives shows, interestingly, that the single-path schemes actually perform "best". Overall, the results of this paper highlight the need to carefully choose between various sources of redundancy.

4. GEOGRAPHIC ROUTING

The idea behind the relatively large class of geographic routing protocols is twofold:

• For many applications, it is necessary to address physical locations, for example, as "any node in a given region" or "the node at/closest to a given point". When such requirements exist, they have to be supported by a proper routing scheme.

• When the position of source and destination is known as are the positions of intermediate nodes, this information can be used to assist in the routing process. To do so, the destination node has to be specified either geographically (as above) or as some form of mapping – a location service – between an otherwise specified destination (e.g. by its identifier) and its (conjectured) current position is necessary. The possible advantage is a much simplified routing protocol with significantly smaller or even non-existing routing tables as physical location carries implicit information to which neighbor to forward a packet to.

The first aspect – sending data to arbitrary nodes in a given region – is usually referred to as geocasting. It was originally introduced in an Internet context; a survey can be found. The second aspect is called position-based routing (in particular in combination with a location service); it was probably first introduced as "Cartesian routing".

In wireless sensor networks, usually the geocasting aspect of geographic routing is considerably more important. Since nodes are considered as interchangeable and are only distinguished by external aspects, in particular their position, a location service is usually not necessary. Hence, this chapter concentrates on the geocasting aspect, with position-based routing aspects treated where necessary. The presentation given here partially follows, in its broad structure.

Basics of Position-Based Routing

Some simple forwarding strategies

Most forward within r

Assume a node wants to send a data packet to a node at known position and assume also that every node in the network knows its own position and that of its neighbors. In a simple greedy forwarding approach, the packet is forwarded to that neighbor that is located closest to the destination (the destination's position is included in the packet), minimizing the remaining distance that the packet has to travel. Formally, the next hop of node *v* toward destination *d* is chosen as,

$$\text{next hop}(v) = \text{argmin}_{u \in N(v)}\{|ud|\},$$

where $|ud|$ indicates the distance between nodes u and d and $N(v)$ is the set of neighbors of node v. This scheme is called most forward within r, where r indicates the maximum transmission range and thus the neighborhood. This method is necessarily loop free.

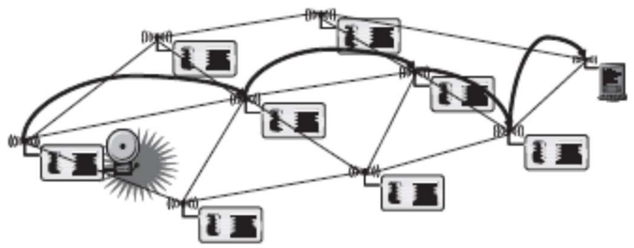


Figure 11.11 Simple greedy geographic forwarding

Figure illustrates this scheme and immediately shows one principal shortcoming: by ignoring topology information, geographic routing is, in general, not able to find the shortest possible path (in hop count). This trade-off between simplified routing scheme and reduced efficiency is, in general, unavoidable.

Nearest with forward progress An alternative to the greedy forwarding is to choose the nearest neighbor that still results in some progress toward the destination. The rationale is to reduce the collision rate and thus to maximize the expected progress per hop; it is not clear how this scheme would interact with an actual MAC layer.

Directional routing Yet another possibility is to forward to nodes that are closer in direction rather than closer in distance. Compass routing is an example, where that neighbor is chosen that is closest to the direct line between transmitter or destination. (A variation would be to choose the angularly closest node; this is not identical.)

Distance Routing Effect Algorithm for Mobility (DREAM) is another example of this idea. Unlike the most progress within r scheme, however, a direction-based scheme like DREAM is not necessarily loop free. To ensure loop freeness in direction-based algorithm, memory about which nodes have already been forwarded by a node has to be used in the nodes.

The problem of dead ends What is more, these simple strategies also cannot deal with dead ends. Figure 12 illustrates how an obstacle that blocks the direct path between source S and destination D interrupts communication even though S and D are actually connected by the network.

An apparently simple fix for a situation where no forward progress can be made is to use the "least unappealing" node, that is, the neighbouring node that loses the least progress [804]. However, as Figure 12 shows, this heuristic can lead to packets looping back and forth between the nodes near the obstacle.

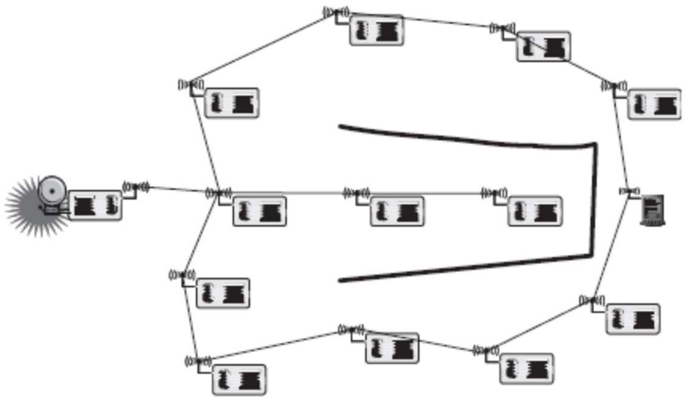


Figure 11.12 Simple greedy geographic forwarding fails in presence of obstacles

The obstacle problem is also not solved by randomly choosing a node that is closer to the destination than the transmitter is randomly forwarding to any node results in random walks. Hence, improvements over these simple schemes are required.

Restricted flooding

Figure 12 also shows that even an extended greedy forwarding where a source forwards to some or all of the nodes that are closer to the destination than itself (so-called geographically restricted flooding) will not remedy the shortcoming – the scheme will not be able to find detours.

Restricted flooding is, on the other hand, quite suited to compensate for mobility of the destination. Assume that the destination moves at a given speed v and that the distance between transmitting node and destination is known, it is a question of simple trigonometry to find an angle α such that D will receive the packet (with given probability) when all neighbors in this angle, centered around the line between transmitter and destination, will receive the packet.

Right-hand rule to recover greedy routing – GPSR

Figure 12 not only illustrates the problem of greedy forwarding in dead ends but also gives an intuition about a possible solution. When being stuck in a dead end, or even in a labyrinth, one certain way of escaping from the labyrinth is to keep the right hand to the wall and keep walking. The practical consequence is to backtrack the packet out of the dead end, counter clockwise around the obstacle; it will eventually find a node closer to the destination.

This intuition has been turned into various protocols, for example, Compass Routing II, "face-2", or, later, the Greedy Perimeter Stateless Routing (GPSR) protocol. GPSR forwards a packet as long as possible using greedy forwarding with the "most forward" rule. If a packet cannot make any more progress, the packet is switched to another routing mode: perimeter routing. A perimeter is a set of nodes defining a face (the largest possible region of the plane that is not cut by any edge of the graph; faces can be exterior or interior). Perimeter routing essentially consists of sending the packet around the face using the right-hand rule. To do so, the packet carries

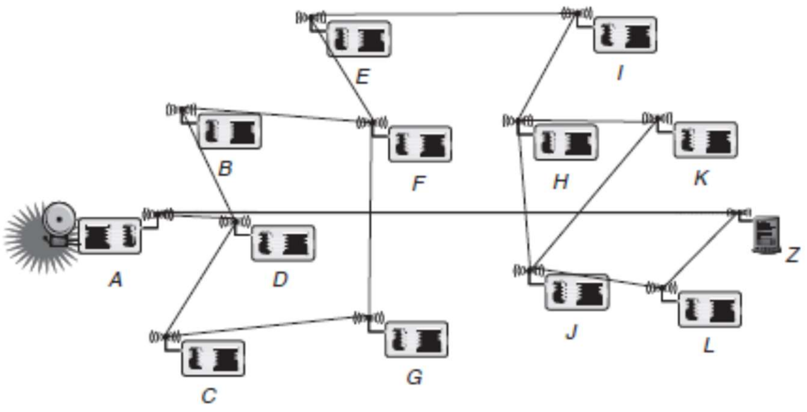


Figure 11.13 Example for GPSR

information where it entered a given face. This node v and the connecting line between v and the destination are used to decide whether the packet should leave the face and proceed to the next one (when the edge from the current node to the next node on the face intersects the connecting line between v and the destination node). Also, the packet can return to greedy forwarding if the distance of the current node to the destination and node v has been effectively reduced (but see the next section regarding performance guarantees of such fallback heuristics).

Figure 13 illustrates how a packet would be routed from node A to node Z. While at node A, the packet can be greedily forwarded to node D. At node D, greedy forwarding fails (both B and C are further away from Z than D itself), so the packet has to be routed around the perimeter of the interior face defined by BFGCD. That is, it is forwarded to B and from there to F. Here, edge FG intersects line DZ and routing can proceed to the next face (note that greedy forwarding to G would not help here). The packet proceeds around the perimeter of the exterior face via E and I to H, from there via K to J and thence to L and Z (the last steps via greedy forwarding).

Since this face-based procedure is based on properties of the plane, it only applies to planar graphs. In general, wireless network graphs are not planar, requiring the construction of a planar subgraph first. It suggests to use a Gabriel graph; reference discusses both Relative Neighborhood Graph (RNG) and Gabriel graph. Both these subgraphs can be constructed in a distributed fashion assuming that node positions are known.

Performance guarantees of combined greedy/face routing

When combining face routing and greedy routing, face routing is tasked with routing around obstacles or out of dead ends while greedy routing tries to make quick progress toward the destination. One would thus like to switch to greedy routing as soon as possible once the obstacle has been cleared. It is, however, nontrivial to select this face-to-greedy switching point correctly or even to provide performance guarantees about the behaviour of such an algorithm. A simple heuristic for such a fallback like switching to greedy mode whenever a node has been found that is closer to the destination than the node where face routing started is in fact not worst-case optimal.

In fact, the first combined greedy/face routing algorithm that is provably worst-case optimal was described, but in order to show the worst-case optimality, quickly switching back to greedy routing could not be used. The proved performance bound was that face routing reaches the destination in $O(c^2)$ steps, where c is the cost of the optimal path from source to destination. The idea here is to adaptively grow an area in which next hops are searched. This performance is worst-case optimal since a graph can be constructed on which no geometric algorithm (without routing tables) can do any better.

The result has been improved by presenting the Greedy and (Other Adaptive) Face Routing (GOAFR)+ algorithm that is worst-case optimal and at the same time efficient in the average case. The crucial point is when to fall back to greedy mode – too soon loses worst-case optimality, too late wastes average-case performance. Two techniques realize this behaviour:

• The algorithm maintains a bounding circle, centered at the destination node, that prevents the face search from needlessly exploring in the wrong direction. This circle is reduced at every step in the greedy forwarding phase and can be enlarged in face routing if, with the current circle restrictions, no progress toward the destination can be made.

• A packet maintains two counters, p and q. When switching to face-based forwarding, both counters are set to 0. Counter p contains the number of nodes on the face perimeter that are closer to the destination than is the node where face search started; q counts nodes farther away. The algorithm falls back to greedy search if $p > \sigma q$ (for some properly chosen constant σ), that is, when substantially more nodes are closer to the destination on this face than are further away. This algorithm is worst-case optimal. It is also efficient in the average case as shown by simulation-based comparison against other algorithms, notably GPSR. An interesting observation is that the difference between these algorithms is largest in the phase transition from a barely connected to a very dense network (where it is either trivial or impossible to find good paths).

Combination with ID-based routing, hierarchies

Purely position-based routing can be problematic in the immediate vicinity of the destination node, for example, when the destination has moved around or the location information is not very accurate. Identity-based routing protocols solve this issues relatively smoothly but have difficulties maintaining state information over long distances. Hence, a natural combination would use (even coarse-grained) position information to forward a packet into the vicinity of the destination where then an identity-based protocol (like any mobile ad hoc networking protocol) would take over. An example for such a hybrid, hierarchical approach is the "Terminodes" project's routing protocol.

GEOCASTING

Geocasting – sending data to a subset of nodes that are located in an indicated region – is evidently an example of multicasting and thus would not require any further attention. Similar to the case of position-based routing, position information of the designated region and the intermediate nodes can be exploited to increase efficiency. Thus, a few dedicated geocasting protocols shall be briefly described in the following.

A broad classification can be made into protocols that are essentially based on some form of geographically restricted flooding even outside the destination region and protocols that are based on some unicast routing protocol to transport a packet into the destination region. Within that region, clearly some form of flooding is required as all nodes in that region are supposed to receive the data. Most of the examples discussed here are based on restricted flooding; GeoTORA is one example based on unicast routing.

Location Based Multicast

A simple way to implement geocasting is to base it on flooding but somehow restrict the area where packets are forwarded. The Location-Based Multicast (LBM) protocol does just that. There is a forwarding zone such that only nodes within the forwarding zone forward a received data packet. This zone can be defined in various ways:

Static zone The smallest rectangle that contains both the source and the entire destination region, with its sides parallel to the axes of the coordinate system. (Alternative geometric definitions are of course possible as well, for example, the destination region and two tangents to it defined by the source node's location).

Adaptive zone Each forwarding node recalculates the zone definition, using its own position as the source. This way, nodes that would be included in the static zone but would represent a detour once the intermediate node has been reached are excluded from forwarding. Since this can, however, again lead to dead end situations, this rule is only applied if an intermediate node actually has neighbors within its newly calculated forwarding zone; otherwise it forwards the packet to all neighbors.

Adaptive distances While the previous two schemes contained the forwarding zone explicitly in each packet, this scheme recomputes it in each step, on the basis of information about the destination region and coordinates of the previous hop (or the source). The idea here is that a node u forwards a packet to its neighbors if its distance to the center of the destination region is smaller than the distance of the previous hop v to the center (the packet has made progress). If not, the packet is only forwarded if the node is actually within the destination region (to ensure that all destinations receive the packet). The importance of not only looking at the overhead caused by a geocasting protocol but also at its accuracy, defined as the ratio of the nodes in the geocast region that actually received the packet. The adaptive algorithms, in fact, achieve a good trade-off between reduced overhead and maintained accuracy.

Finding the right direction: Voronoi diagrams and convex hulls

To correctly decide which neighbors of a forwarding node are the "right" direction is not an immediately obvious task for directional routing approaches like Compass routing or LBM.

Voronoi diagrams: Given a node S that has to forward a message, the destination region D (or the region of uncertainty where the destination node is located), and the set N(S) of neighbors of S. Construct the Voronoi diagram for N(S) (not including S itself). Then, a given neighbor A ∈ N(S) is closest to some node in D if and only if its Voronoi polygon intersects D. Hence, these neighbors should be selected as next hops. Figure 16 shows an example.

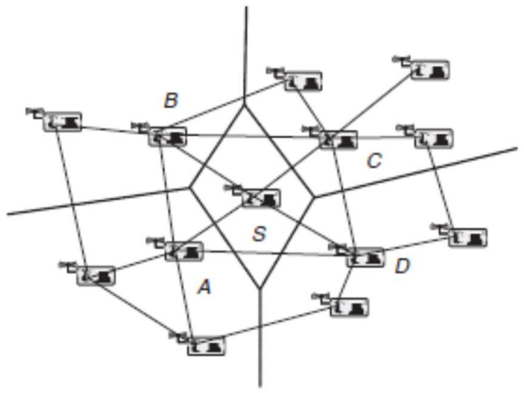


Figure 11.16 Illustration of the Voronoi diagram-based neighbor selection scheme [795] – node S uses the Voronoi cells to decide which neighbor to use for a given destination area

A similar rule can be developed to improve the "most forward within r" rule. The problem is to find those neighbors that make most progress toward some point in the destination region D. To this end, construct the two tangents from S to the region D. Call the intersection points of the tangents with D U and V. For U and V , determine those neighbors of S on the convex hull of N(S) that represent the biggest progress toward U and V ; call them U' and V' , respectively. The set of next hop nodes is then all the nodes in the convex hull of N(V) between and including U' and V' . The convex hull is used to ensure maximum progress; it also can be efficiently constructed.

Tessellating the plane

Apart from locally computed Voronoi diagrams, other, perhaps simpler, tessellations of the plane can also be considered. The biggest simplification would be to use a fixed tessellation into regions where each point in space is uniquely mapped to one region. Here, the plane is divided into square grids where each grid has an elected gateway in charge of it. Only those gateway nodes propagate packets among different grids, resulting in a need to control the size of such a grid.

Mesh-based geocasting

Geocast Adaptive Mesh Environment for Routing (GAMER), a mesh-based protocol for geocasting, which improves upon other mesh-based geocasting protocols by adapting the density of the created mesh according to the mobility of the nodes in the network.

Geocasting using a unicast protocol – GeoTORA

As a last example of standard geocast, let us consider how to modify a unicast protocol to obtain a geocast protocol. The starting point is the Temporally Ordered Routing Algorithm (TORA) unicast ad hoc routing protocol. The intuition behind TORA is to conceive of the graph as a "landscape" where different nodes have different heights above ground. If the destination of a unicast routing protocol is the lowest point in this landscape (e.g. at height zero) and if there are no local minima, then the forwarding process is trivial: simply pass on the packet downward. Formally, this intuition is captured by imposing a Directed Acyclic Graph (DAG) onto the original graph by orienting its edges. This DAG only has a single sink (a node without outgoing edges), which is the destination node. The essence of the TORA protocol is then in ensuring that this DAG structure is maintained despite link failures or node mobility.

On this basis, how to modify TORA to support anycasting (sending a packet to any arbitrary member of a given group). This can be achieved by simply assigning height 0 to all nodes in this anycast group. The DAG can still be constructed, using essentially the same rules as in TORA.

Once anycasting is in place, the extension to geocasting is also relatively simple: any node in the destination region joins the anycast group and, in addition, locally floods a received packet within the destination region, similar to other flooding protocols. It is also necessary to handle the case of an empty geocast region or of an empty geocast region with a node moving into it.

Trajectory-based forwarding (TBF)

In the previous approaches, the destination region was – intuitively – conceived of as a more or less convex region somewhere "far away". But this is not the only possible interpretation of geocasting as the somewhat different approach of Trajectory-Based Forwarding (TBF) shows (Figure 17). Instead of trying to send a packet to some region far away, the region of interest can actually be a path in the network. This path, or trajectory of the packet, can be embedded into the packet as a parametric description of the curve that the packet is supposed to follow; the parameter could be time or, preferably, the length of the path that the packet has followed. In this sense, trajectory-based routing combines aspects from source routing (as the trajectory is defined by the source) and geocasting. Different forms of such trajectories can be useful for different purposes, for example, a tree form for broadcasting or a "boomerang" (where the packet visits all nodes in the network and returns to the source node) for management of a network.

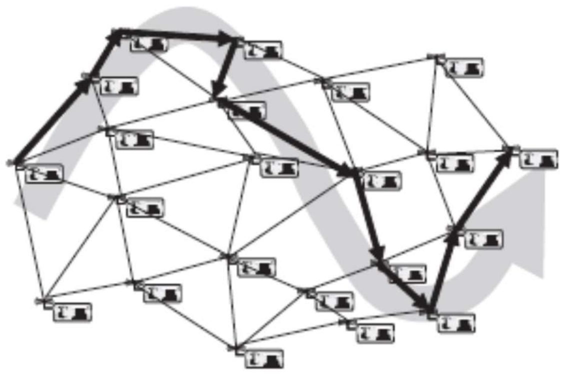


Figure 11.17 Trajectory-based forwarding

Given such a parametric description of a trajectory, the forwarding of a packet can follow different rules. For example, a node could forward a packet to its neighbor that minimizes the distance from the prescribed trajectory or to the one that results in the most advance on the trajectory (without deviating too much from it). Selecting the best forwarding policy depends on the actual application requirements.

CONCLUSION:

Several different MAC protocols for wireless sensor networks have been discussed. All of them are designed with the goal to conserve energy; other goals like small delays or high throughput are often traded off for energy conservation. There is no generic "best" MAC protocol; the proper choice depends on the application, the expected load patterns, the expected deployment (sparse versus dense sensor networks), and the specifics of the underlying hardware's energy-consumption behaviour, for example, the relative costs of transmitting, receiving, switching between modes, wakeup times, and wakeup energy from sleep mode as well as the specific computation costs for executing the MAC protocol.

In any network including sensor networks, there are different levels of addresses and names, for example MAC addresses and network-layer addresses. MAC addresses are used to distinguish between immediate neighbors and network-layer addresses are used to identify (groups of) nodes in a multihop network. A prime concern regarding naming and addressing in sensor networks is the overhead and energy consumption incurred with these schemes. Energy can, for example, be wasted by having inefficient address representations, by running expensive address assignment and deallocation protocols, or by requiring several binding/address resolution protocols.

At the lowest level are MAC addresses, which in contention-based MAC protocols are indispensable to realize energy savings from overhearing avoidance. If required by the MAC protocol, they need to be present in all data packets and can induce significant overhead, especially if the user data is small. As opposed to schedule-based MAC protocols, MAC addresses are always needed in contention-based MAC protocols since a transmitted packet can potentially have many receivers. A general trade-off exists here between stricter uniqueness requirements and the size of the address. On the other hand, locally unique addresses (which are sufficient for the MAC layer) require an address assignment protocol. However, if most of the sensor nodes are stationary, the savings achieved by (efficient representations of) locally unique addresses pay off quickly. Running address assignment protocols with stricter than local uniqueness requirements (say, networkwide uniqueness) quickly becomes impractical in wireless sensor networks since a distributed consensus problem has to be solved, which inevitably has substantial overhead. Furthermore, the address representation size of locally unique addresses depends only on the network density but not on the absolute number of nodes in the network. This is not the case for networkwide unique addresses. Networkwide or globally unique addresses are needed by traditional routing protocols to denote and find individual nodes. In wireless sensor networks, however, content-based addressing provides an attractive alternative. A key to their usefulness is the integration of content-based addresses with routing and their ability to enable in-network processing.

Supporting energy-efficient unicast and multicast communication in a wireless sensor network is a crucial optimization task and its solution draws upon insights from many different disciplines. Both the design of algorithms and their evaluation is a challenging task, requiring great care in selecting the proper assumptions and algorithmic principles, but they also pay off handsomely in extended capacity of lifetime of the network. The mechanisms and schemes described in this chapter were mostly based on the assumption that nodes have a clearly defined address or at least location that could be used to designate the target of the communication. For wireless sensor networks, these mechanisms are important but they are complemented by mechanism that deal with the collection and dissemination of data directly.

Notes on
Infrastructure Establishment
Unit IV

Dr. G. Senthil Kumar,
Associate Professor,
Dept. of ECE, SCSVMV,
email: gsk_ece@kanchiuniv.ac.in

==

OBJECTIVES:

In a densely deployed wireless network, a single node has many neighbouring nodes with which direct communication would be possible when using sufficiently large transmission power. High transmission power requires lots of energy, many neighbors are a burden for a MAC protocol, and routing protocols suffer from volatility. To overcome these problems, topology control can be applied. The idea is to deliberately restrict the set of nodes that are considered neighbors of a given node.

Time is an important aspect for many applications and protocols found in wireless sensor networks. The time synchronization problem is a standard problem in distributed systems. In wireless sensor networks, new constraints have to be considered.

This chapter gives an overview of the methods to determine the symbolic location of a wireless sensor node. The properties of such methods and the principal possibilities for a node to determine information about its whereabouts are discussed. At the end of the chapter, the reader will understand the principal design trade-offs for positioning and gain an appreciation for the overhead involved in obtaining this information.

CONTENTS:

INTRODUCTION

One perhaps typical characteristic of wireless sensor networks is the possibility of deploying many nodes in a small area, for example, to ensure sufficient coverage of an area or to have redundancy present in the network to protect against node failures. While these are clear advantages of a dense network deployment – density as measured, for example, by the average number of neighbors that a single node has – there are also disadvantages. In a relatively crowded network (Figure 10.1), many typical wireless networking problems are aggravated by the large number of neighbors: many nodes interfere with each other, there are a lot of possible routes, nodes might needlessly use large transmission power to talk to distant nodes directly (also limiting the reuse of wireless bandwidth), and routing protocols might have to recompute routes even if only small node movements have happened.

Some of these problems can be overcome by topology-control techniques. This can be done by controlling transmission power, by introducing hierarchies in the network and signalling out some nodes to take over certain coordination tasks, or by simply turning off some nodes for a certain time. Instead of using the possible connectivity of a network to its maximum possible extent, a deliberate choice is made to restrict the topology of the network. The topology of a network is determined by the subset of active nodes and the set of active links along which direct communication can occur.

The time synchronization problem is a standard problem in distributed systems. Nodes can measure time using local clocks, driven by oscillators. Because of random phase shifts and drift rates of oscillators, the local time reading of nodes would start to differ – they loose synchronization – without correction. In wireless sensor networks, new constraints have to be considered, for example, the energy consumption of the algorithms, the possibly large number of nodes to be synchronized, and the varying precision requirements. This chapter gives an introduction to the time synchronization problem in general and discusses the specifics of wireless sensor networks. Following this, some of the protocols proposed for sensor networks are discussed in more detail.

In this section, we explain why time synchronization is needed and what the exact problems are, followed by a list of features that different time synchronization algorithms might have. We also discuss the particular challenges and constraints for time synchronization algorithms in wireless sensor networks. Time plays an important role in the operation of distributed systems in general and in wireless sensor networks in particular, since these are supposed to observe and interact with physical phenomena.

A simple example shall illustrate the need for accurate timing information. An acoustic wavefront generated by a sound source a large distance away impinges onto an array of acoustic sensors and the angle of arrival is to be estimated. Each of the sensors knows its own position exactly and records the time of arrival of the sound event.

In many circumstances, it is useful or even necessary for a node in a wireless sensor network to be aware of its location in the physical world. For example, tracking or event-detection functions are not particularly useful if the WSN cannot provide any information where an event has happened. This chapter gives an overview of the methods to determine the symbolic location – "in the living room" – and the numeric position – "at coordinates (23.54, 11.87)" – of a wireless sensor node. The mathematical basics for positioning are introduced and the single-hop and multihop positioning case are described using several example systems. To do so, usually, the reporting nodes' location has to be known. Manually configuring location information into each node during deployment is not an option. Similarly, equipping every node with a Global Positioning System (GPS) receiver fails because of cost and deployment limitations (GPS, e.g. does not work indoors). This chapter introduces various techniques of how sensor nodes can learn their location automatically, either fully autonomically by relying on means of the WSN itself or by using some assistance from external infrastructure.

To efficiently and optimally utilize scarce resources (e.g., limited on-board battery and limited communication bandwidth) in a sensor network, sensor nodes must carefully tasked and controlled to carry out the required set of tasks. A utility-cost-based approach to distributed sensor network management is to address the balance between utility and resource costs. Utility – the total utility of the data. Cost – power supply, communication bandwidth.

A sensor may take on a particular role depending on the application task requirement and resource availability such as node power levels. Example: Nodes, denoted by SR, may participate in both sensing and routing. Nodes, denoted by S, may perform sensing only and transmit their data to other nodes. Nodes, denoted by R, may decide to act only as routing nodes, especially if their energy reserved is limited. Nodes, denoted by I, may be in idle or sleep mode, to preserve energy. As soon as query from a subscriber arrives at the coordinator, the latter carries out the statistical analysis of the retrieved data, collects knowledge, and displays the decision to the subscriber.

1. TOPOLOGY CONTROL

Motivation and basic ideas

One perhaps typical characteristic of wireless sensor networks is the possibility of deploying many nodes in a small area, for example, to ensure sufficient coverage of an area or to have redundancy present in the network to protect against node failures. While these are clear advantages of a dense network deployment – density as measured, for example, by the average number of neighbors that a single node has – there are also disadvantages. In a relatively crowded network (Figure 1), many typical wireless networking problems are aggravated by the large number of neighbors: many nodes interfere with each other, there are a lot of possible routes, nodes might needlessly use large transmission power to talk to distant nodes directly (also limiting the reuse of wireless bandwidth), and routing protocols might have to recompute routes even if only small node movements have happened.

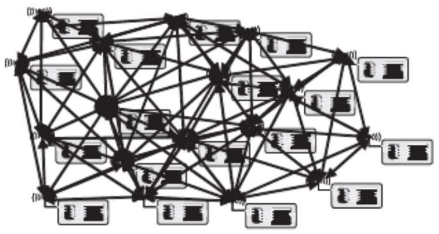


Figure 10.1 Topology in a densely deployed wireless sensor network

Some of these problems can be overcome by topology-control techniques. Instead of using the possible connectivity of a network to its maximum possible extent, a deliberate choice is made to restrict the topology of the network. The topology of a network is determined by the subset of active nodes and the set of active links along which direct communication can occur. Formally speaking, a topology-control algorithm takes a graph G = (V ,E) representing the network – where V is the set of all nodes in the network and there is an edge (v1, v2) ∈ E ⊆ V 2 if and only if nodes v1 and v2 can directly communicate with each other – and transforms it to a graph T = (VT ,ET) such that VT ⊆ V and ET ⊆ E.

Options for topology control

To compute a modified graph T out of a graph G representing the original network G, a topology control algorithm has a few options:

• The set of active nodes can be reduced (VT ⊂ V), for example, by periodically switching off nodes with low energy reserves and activating other nodes instead, exploiting redundant deployment in doing so.

• The set of active links/the set of neighbors for a node can be controlled. Instead of using all links in the network, some links can be disregarded and communication is restricted to crucial links.

When a flat network topology (all nodes are considered equal) is desired, the set of neighbors of a node can be reduced by simply not communicating with some neighbors. There are several possible approaches to choose neighbors, but one that is obviously promising for a WSN is to limit the reach of a node's transmissions – typically by power control, but also by using adaptive modulations (using faster modulations is only possible over shorter distances) – and using the improved energy efficiency when communicating only with nearby neighbors.

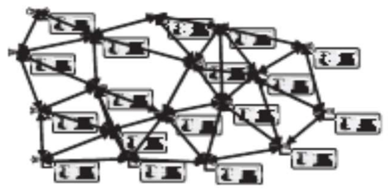


Figure 10.2 Sparser topology after reducing transmission power

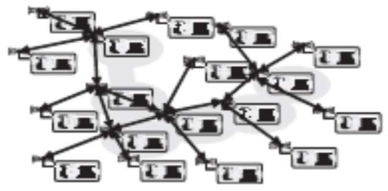


Figure 10.3 Restricting the topology by using a backbone

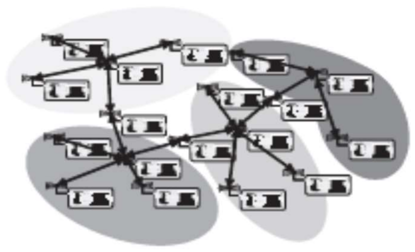


Figure 10.4 Using clusters to partition a graph

Figure 2 illustrates how the dense topology from Figure 1 can be reduced by applying power control. In essence, power control attempts to optimize the trade-off between the higher likelihood of finding a (useful) receiver at higher power values on the one hand and the increased chance of collisions/interference/reduced spatial reuse on the other hand.

• Active links/neighbors can also be rearranged in a hierarchical network topology where some nodes assume special roles. One example, illustrated in Figure 3, is to select some nodes as a “backbone” (or a “spine”) for the network and to only use the links within this backbone and direct links from other nodes to the backbone. To do so, the backbone has to form a dominating set: a subset $D \subset V$ such that all nodes in V are either in D itself or are one-hop neighbors of some node $d \in D$ ($\forall\, v \in V : v \in D \vee \exists\, d \in D : (v, d) \in E$). Then, only the links between nodes of the dominating set or between other nodes and a member of the active set are maintained. For a backbone to be useful, it should be connected.

A related, but slightly different, idea is to partition the network into clusters (Figure4). Clusters are subsets of nodes that together include all nodes of the original graph such that, for each cluster, certain conditions hold (details vary). The most typical problem formulation is to find clusters with cluster heads – a representative of a cluster such that each node is only one hop away from its cluster head. When the (average) number of nodes in a cluster should be minimized, this is equivalent to finding a maximum (dominating) independent set (a subset $C \subset V$ such that $\forall\, v \in V - C : \exists\, c \in C : (v, c) \in E$ and no two nodes in C are joined by an edge in E – $\forall\, c1, c2 \in C : (c1, c2)$

In such a clustered network, only links within a cluster are maintained (typically only those involving the cluster head) as also selected links between clusters to ensure connectivity of the whole network”.

Both problems are intrinsically hard and various approximations and relaxations have been studied. These three main options for topology control – flat networks with a special attention to power control on the one hand, hierarchical networks with backbones or clusters on the other hand – will be treated. First, a few desirable aspects of topology-control algorithms should be discussed.

Aspects of topology-control algorithms

There are a few basic metrics to judge the efficacy and quality of a topology-control algorithm:

Connectivity Topology control should not disconnect a connected graph G. In other words, if there is a (multihop) path in G between two nodes u and v, there should also be some such path in T (clearly, it does not have to be the same path).

Stretch factors Removing links from a graph will likely increase the length of a path between any two nodes u and v. The hop stretch factor is defined as the worst increase in path length for any pair of nodes u and v between the original graph G and the topology-controlled path *T*. Formally,

$$\text{hop stretch factor} = \max_{u,v \in V} \frac{|(u, v)_T|}{|(u, v)_G|} \tag{10.1}$$

where $(u, v)_G$ is the shortest path in graph G and $|(u, v)|$ is its length.
Similarly, the **energy stretch factor** can be defined:

$$\text{energy stretch factor} = \max_{u,v \in V} \frac{E_T(u, v)}{E_G(u, v)} \tag{10.2}$$

where EG(u, v) is the energy consumed along the most energy-efficient path in graph G. Clearly, topology-control algorithms with small stretch factors are desirable. It particular, stretch factors in *O(1)* can be advantageous.

Graph metrics The intuitive examples above already indicated the importance of a small number of edges in T and a low maximum degree (number of neighbors) for each node.

Throughput The reduced network topology should be able to sustain a comparable amount of traffic as the original network (this can be important even in wireless sensor networks with low average traffic, in particular, in case of event showers). One metric to capture this aspect is throughput competitiveness (the largest φ ≤ 1 such that, given a set of flows from node si to node di with rate ri that are routable in G, the set with rates φri can be routed in T).

Robustness to mobility When neighborhood relationships change in the original graph G (for example, because nodes move around or the radio channel characteristics change), some other nodes might have to change their topology information (for example, to reactivate links). Clearly, a robust topology should only require a small amount of such adaptations and avoid having the effects of a reorganization of a local node movement ripple through the entire network.

Algorithm overhead It almost goes without saying that the overhead imposed by the algorithm itself should be small (low number of additional messages, low computational overhead). Also, distributed implementation is practically a condition sine qua none.

In the present context of WSNs, connectivity and stretch factors are perhaps the most important characteristics of a topology-control algorithm, apart from the indispensable distributed nature and low overhead. Connectivity as optimization goal, however, deserves a short caveat.

A caveat to connectivity

Consider a simple example of power control. Five thousand nodes are uniformly, randomly deployed over a an area of 1000 by 1000 m. The transmission range of each node can be set to a precise radius of r m (i.e. all nodes at most r m apart can communicate directly, and no other nodes can). This model is known as the disk graph model; the special case of r = 1 is called the Unit disk graph. For one such example network and a given transmission range, the network is either connected or not. Determining connectivity for 100 different, randomly generated networks gives a rough estimate of the probability of connectivity as a function of the transmission range, shown as the dotted line in Figure 5.

As expected, the probability of connectivity is zero for small transmission ranges and raises relatively sharply, until it levels off and slowly approaches probability 1 at about 30m transmission range.

The same experiment allows to consider an additional metric. For each repetition, the size of the largest connected component can be computed, and this size, averaged over the 100 repetitions, is shown in Figure 5, also as a function of the transmission range. Clearly, even for relatively small transmission ranges, almost all nodes are connected into a single component, even though the probability of connectivity is still practically zero because there are three nodes in an unfortunate position. For example, for transmission range 25 m, the average size of the largest component is 4997 – that is, only 3 out of 5000 nodes are not connected – whereas the probability of connectivity is still practically zero. Evidently, for WSN, connectivity might not be the relevant metric, but rather, a large value of the maximum component size would be more important.

That being said, the overwhelming part of research has gone into studying connectivity properties, with the importance of component sizes being only slowly realized.

Another important aspect is coverage – making sure that all points in the plane are covered by an observing node. If the few nodes missing for connectivity are important for coverage as well, it might actually be inevitable to invest the energy required to connect them.

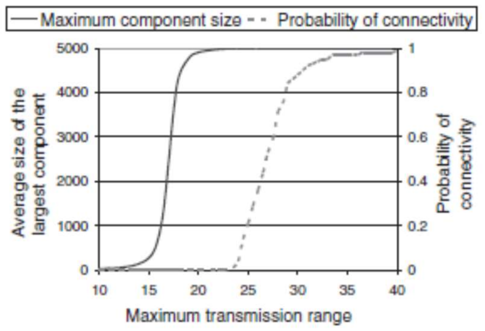


Figure 10.5 Optimization goal of topology control: Probability of connected network versus average size of the largest component

Figure 5 also shows another effect. The average size of the largest component and, to a slightly smaller degree, also the probability of having a connected network do not slowly increase with the maximum transmission range (or, equivalently, the density of the network). Rather, both metrics increase sharply from zero to their maximum values once a certain critical threshold for the transmission range is exceeded. This effect is known for a large number of aspects of (random) graphs in general and called a phase transition. The existence of such thresholds is provable for purely random graphs and plausible for (unit) disk graphs as used to model wireless networks.

We discuss various examples for such thresholds and recommend to set operational parameters of a network just slightly larger than the critical threshold to obtain the desired behaviour without wasting resources. Phase transition phenomena are often characterized using percolation theory techniques.

CLUSTERING

The previous Section has introduced a hierarchy into a network by designating some nodes as belonging to a backbone, a dominating set. Another idea for a hierarchy is to locally mark some nodes as having a special role, for example, controlling neighbouring nodes. In this sense, local groups or clusters of nodes can be formed; the “controllers” of such groups are often referred to as cluster heads. The hoped-for advantages of such clustering are similar to that of a backbone, but with additional emphasis on local resource arbitration (e.g. in MAC protocols), shielding higher layers of dynamics in the network (making routing tables more stable since all traffic is routed over the cluster heads), and making higher-layer protocols more scalable (since the size and complexity of the network as seen by higher layers is in a sense reduced by clustering). In addition, cluster heads are natural places to aggregate and compress traffic converging from many sensors to a single station.

Formally, given a graph G = (V ,E), clustering is simply the identification of a set of subsets of nodes Vi , i = 1, . . . , n such that ∪i=1,...,nVi = V. A number of questions about the detailed properties required from these sets distinguish various clustering approaches:

Are there cluster heads? The partitioning of V into several clusters does not mandate anything about the internal structure of a cluster; in principle, all nodes can be equal (one example is described. Typically, however, for each set Vi there is a unique node ci, the cluster head, that represents the set and can take on various tasks. We shall almost exclusively deal with examples using cluster heads.

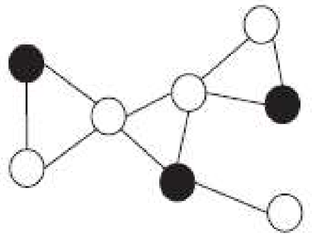


Figure 10.17 An example graph with a maximum independent set [59]

May cluster heads be neighbors? In principle, again, it is perfectly acceptable for two cluster heads (of two different clusters) to be direct neighbors. It is, however, often desirable to have cluster heads separated. Formally, cluster heads should form an independent set: a subset C ⊂ V such that no two nodes in C are joined by an edge in E −∀ c1, c2 ∈ C : (c1, c2) ∉ E./ Finding an arbitrary such set is trivial; the interesting case is maximum independent sets, which contain as many nodes as possible without violating the independence property, resulting in as many clusters around these cluster heads as possible. Figure 17 shows an example graph with one maximum independent set; others are possible (and easy to find) as the maximum independent set is, in general, not unique.

An important property of such a maximal independent set is that it is also dominating (easily proven via the contraposition). This property essentially justifies the importance of this problem formulation: maximum independent sets naturally partition the network and also form a subset of nodes that can control the network.

Determining maximum independent sets is, as expected, NP-complete. It does admit a PTAS for scalar graphs and unit disk graphs. For bounded degree graphs, it is approximable within $(\Delta + 3)/5$ for small Δ, and within $O(\Delta log\ log\Delta/log\Delta)$ for larger values.

While the maximum independent set formulation is elegant and simple, it does not necessarily reflect the actually desired configuration of the clusters. Consider, for example, a graph $G = (\{v_0, . . . , v_n\}, \{(v_0, v_i)|i = 1, . . . , n\})$ (i.e. one node connected to n other nodes that are not connected with each other). The maximum independent set for this graph is v1, . . . , vn, resulting in n clusters, one of size 2, the others of size 1. Much more practical, in most circumstances, would be to use node 0 as the head of only a single cluster. Such an intuition about networks is reflected in most of the later-on described heuristics even though the actual optimization objectives are usually not fully formalized. Objectives like uniform spread of clusters over a given area are often considered important.

May clusters overlap? When forming clusters out of the maximal independent set shown in Figure 17, the question arises to which cluster to assign non cluster head nodes, particularly those nodes that are adjacent to two cluster heads. One option would be to assign these nodes to both clusters, resulting in overlapping clusters. If that is not desirable, some decision rule is required to unambiguously assign nodes to cluster heads. Figure 18 highlights these possibilities.

How do clusters communicate? Whether clusters overlap or not, a node that is adjacent to two cluster heads can naturally assist in the communication between two clusters – it forms a gateway (other names are bridge, boundary node, or similar terms). The idea is that intracluster communication can be routed via the cluster heads, who then use the gateways for any intercluster communication.

There may be cases, however, where two cluster heads are separated by two nodes, and no single node can fulfil the duties of a gateway. In such a situation, two nodes from each cluster together can act as a so-called distributed gateway to enable the communication between clusters. This idea is shown in Figure 19.

The cluster heads together with the (distributed) gateways again form a connected dominating set and thus a backbone of the entire network. This equivalence can also steer the gateway selection, as for example it might not be necessary to connect all neighbouring clusters via gateways (although this is often done regardless of global optimization opportunities) or the choice between different gateways can be optimized by preferring nodes to serve as gateways that can connect more than two clusters, being in the intersection of several clusters. Choosing the optimal set of gateways to connect the given cluster heads into a connected sets is again a Steiner tree problem.

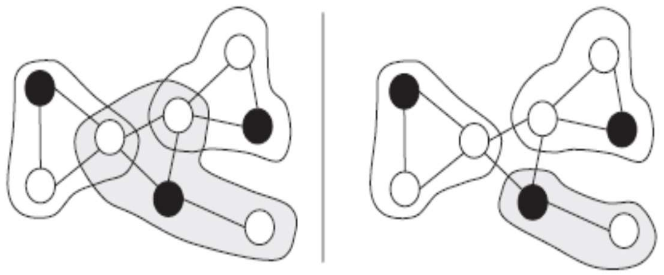


Figure 10.18 Maximum independent set induces overlapping or nonoverlapping clusters (example adapted from reference [59])

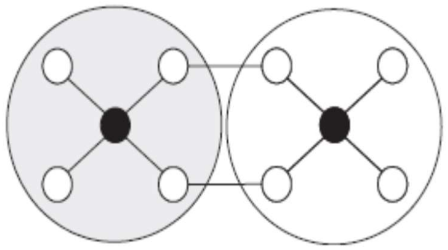


Figure 10.19 Two clusters connected by two distributed gateways

How many gateways exist between clusters? There can be several options to connect two cluster heads via several (distributed) gateways (examples are described). Depending on the optimization goal for the eventual connected dominating set, some degree of redundancy in the intercluster communication may be desirable.

What is the maximal diameter of a cluster? The presence of cluster heads and the goal of constructing a maximum independent set point to a maximum cluster diameter of two – each node in a cluster is at most two hops away from any other node. This is not necessarily the case: sometimes, one-hop clusters are considered (which often do not have cluster heads); sometimes, multihop clusters with larger diameters are used.

Is there a hierarchy of clusters? Cluster heads impose a hierarchy of nodes onto the network. Usually, such a two-level hierarchy is considered sufficient. Nonetheless, it is possible to consider the clusters as such as nodes in a new, induced graph, along with the links between clusters as edges in this graph. To this graph, again, clustering (or other dominating set approaches) can be applied.

A basic idea to construct independent sets

A first, simple idea to construct – hopefully large – independent sets exploits the inherently local nature of being independent – if selected nodes can restrain all their neighbors from being selected as well, independence ensues. The idea is thus for every node to communicate with its neighbors and to locally select nodes to join the set of independent nodes (to become cluster heads in the end).

To do so, all nodes need a property that can be locally determined, easily exchanged with all neighbors, and unambiguously ranked by each node (ties can be broken locally). A simple example for such a property is a unique identifier of each node, sorted for example in ascending order, where ties cannot happen at all. Using the identifier has actually been the first proposal for a distributed clustering algorithm.

Irrespective of the precise choice of the property used for ranking nodes, a basic distributed algorithm to compute independent sets starts out by marking all nodes as being ready to become cluster heads, but as yet undecided. During the course of the algorithm, this status is switched to either “cluster head” or “cluster member” (comparable to the colors white, black, and gray). In the first step, each node determines its local ranking property and exchanges it with all of its neighbors. Once this information is available, a node can decide to become a clusterhead if it has the largest rank (or the smallest, depending on definition) among all its as-yet-undecided neighbors. It changes its state accordingly and announces its new state to its neighbors. Nodes that learn about a clusterhead in their neighborhood switch to cluster member state and in turn announce that to their neighbors. Note that this is the crucial step: Once a node with a large rank becomes a cluster member to some other node, it can “unblock” nodes with lower rank in its vicinity to become clusterheads on their own. The algorithm terminates once all nodes have decided to become either a clusterhead or a cluster member.

This algorithm is illustrated with a simple linear network in Figure 20. Note how, in step 1, nodes 2 and 5 cannot become clusterheads because their neighbouring nodes 3 and 6 have not yet decided and would, potentially, take precedence over them. Once nodes 3 and 6 have learned about node 7 being a clusterhead in their vicinity, they decide to become cluster members and propagate this information to nodes 2 and 5. Then, these nodes can become clusterheads in step 3. This essential algorithm has been considered with several small variations. One variation is whether to actually hold back nodes from forming clusters as long as the clusterhead decision might still be revised, or to allow intermediate clusters to be formed, which will later be reclustered

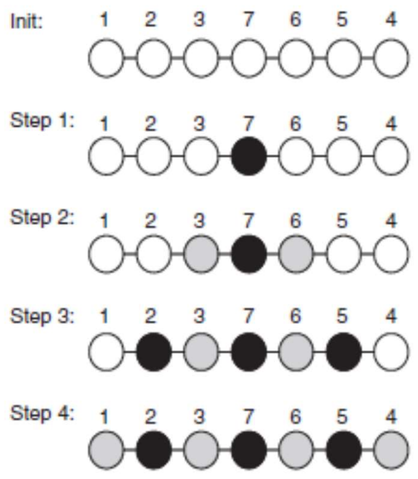


Figure 10.20 Basic algorithm for determining independent sets, using node identifiers as rank (white nodes are undecided, black nodes are clusterheads, gray ones are cluster members)

and nodes might join another clusterhead. This variation might be particularly useful in mobile networks. Another important variation is how to rank nodes. The using of smallest (or largest) identifiers has been the first proposal (describing, for example, the "linked cluster architecture", with some provisions made for how to exchange connectivity information between nodes). Ranking nodes according to their degree, using the identifiers only to break ties, was proposed and investigated. Essentially the same idea has been used, where clusters are grown around nodes with the highest degree, but no clusterhead is elected.

A generalization and some performance insights

Other rankings besides identifiers or node degrees are conceivable. Generalize these approaches by introducing weights for each node and formulate the clustering problem as the Maximum Weight Independent Set (MWIS) problem. Here, the goal is to find an independent set of nodes such that the sum of the weights of the nodes in this set is maximized. As it generalizes the maximum independent set problem, MWIS is NP-hard as well.

The algorithm described is straightforward and quite similar to the algorithm described above. It is actually a centralized algorithm, in each round choosing the node with the largest weight as a clusterhead and assigning all neighbors to this clusterhead; all these nodes are removed from the set of nodes that have to be considered. The algorithm terminates when all nodes have been assigned to some cluster. The result is a set of independent, dominating clusterheads, with nonoverlapping clusters. The algorithms of the previous section are obtained through proper choice of node weight.

The concrete performance of this algorithm depends on the actual choice of weights. It is possible to provide a lower bound on its performance, as long as all the weights are nonnegative. To do so, the notion of "performance" of a clustering algorithm has to be made more precise; the obvious choice here is how well it approximates the maximum weighted set that it is supposed to find. In other words, what is the ratio between the maximum weight of the best independent set and the weight of the set found by the algorithm, given a graph G and a node weighting w. Using this performance definition, it show that this generalized algorithm always finds independent sets at least as heavy as maximum weight/Δ, where Δ is the maximum degree of the graph. This is nontrivial, as it holds irrespective of the actually used node weighting.

What is more, they also show that this is the best bound on a performance ratio that can be proven for any polynomial time algorithm for nontrivial classes of graphs, as long as P ≠NP. In this sense, these simple algorithms are actually optimal.

Nonetheless, the actual performance of an algorithm (and not the worst-case bound) does considerably depend on the concrete weighting in use. The authors compare a "lowest ID" weighting with a weighting that gives preference to slowly moving nodes in a mobile ad hoc network; the metrics of interest are the number of reaffiliations of nodes to new clusters and the number of elections of new clusterheads as the result of mobility. In both these metrics, a mobility-aware weighting outperforms an identifier-based weighting (degree-based approach are known to perform not well in such situations). Reference [58] extends upon this work.

Connecting clusters

Once the clusterheads have been determined, by whatever algorithm, it is usually also necessary to determine the (possibly distributed) gateways between the clusters. Put simply, this problem is reduced again to the Steiner tree problem.

But the situation here is simpler than in the general Steiner tree setting as some properties of the clusterheads are known. In particular, they form a dominating, independent set where all nodes are separated by at most three hops (in case of two ordinary cluster members meeting at the edge of two clusters). For such a setting, they have shown that a connected backbone results if each clusterhead connects to all other clusterheads that are at most three hops away. While for some networks, this might mean more connections than necessary, but there are networks where all this links are needed to ensure connectivity. In addition to this basic connectivity consideration, other aspects like load balancing between multiple gateways can be considered.

Rotating clusterheads

Being a clusterhead means taking over additional tasks: organizing medium access within the cluster or participating in routing decisions. Hence, the battery of clusterheads will tend to be exhausted sooner. Often, it is considered desirable that all nodes have roughly equal battery capacities at any point in time.6 Hence, the duty of being a clusterhead should be shared among all nodes. Such sharing is in fact a viable option as there is usually not only a single solution to a maximum independent set problem but rather a number of different, (nearly) equally good ones.

To be able to rotate the clusterheads, the clustering algorithm cannot run only once but must be repeatedly executed. These repetitions can happen periodically or can be triggered by node mobility, for example. Of course, choosing periods and triggers judiciously is an important optimization problem, depending for example, on average node speed, battery draining rate, and so on.

Using virtual identifiers for rotation

As an example for clusterhead rotations, consider the extensions to the node-identifier-based or node-degree-based algorithms introduced. To enable the ID-based algorithm to rotate clusterheads, the identifier is replaced, on each node independently, by a queue of virtual identifiers that are used in a round-robin fashion in the actual clustering algorithm. The node degree heuristic is adapted by forcing a clusterhead to step down if its degree has changed more than a given threshold in between two runs of the clustering algorithm. Low-Energy Adaptive Clustering Hierarchy (LEACH) Another early and popular example for rotating clusterheads is the Low-Energy Adaptive Clustering Hierarchy (LEACH) protocol. Its target scenario is a sensor network with a known number of nodes and known area, with a dedicated data sink to which all data is to be reported. As the data can be aggregated (e.g. by averaging), the introduction of clusterheads stands to reason. These nodes shall collect data readings from their cluster members and transmit it directly, at high transmission power, to the data sink in a single hop. As this is an energy-intensive operation, it makes sense to protect the clusterheads from being drained by rotating their role among all nodes.

A simple, lightweight protocol for clusterhead election and rotation is desirable, and the idea here is to use a simple random choice of clusterheads, foregoing the entire overhead for determining optimal clusterheads as their role is a temporary one anyway. Nodes independently decide to act as clusterheads and announce this to their neighbouring nodes. These nodes then join that clusterhead in their vicinity with minimal communication costs (if there is more than a single one); nodes that do not hear a clusterhead announcement but do not want to become clusterheads themselves have to communicate with the data sink directly.

For such a random choice of clusterheads, the optimal ratio of clusterheads out of the total number of nodes is required. Taking into account the high costs for communication with a remote data sink, operation without clusterheads will result in low energy efficiency. Adding even a few will quickly improve overall energy efficiency, despite the additional effort for aggregation that these clusterheads incur. When increasing the ratio further, the advantages of clustering slowly diminish (in the extreme case, each node is a clusterhead for itself, voiding any aggregation or multihopping benefits). Hence, there is an optimal number at a relatively low ratio; for a typical example scenario, it determine an optimal number of 5 %, but this does depend on the particular setup and has to be determined beforehand.

Once such an optimal percentage P of clusterheads is known, the actual LEACH algorithm proceeds in 1/P rounds (assuming, for simplicity, that 1/P is an integer value). In each round, a set of clusterheads of expected size nP (n the total number of nodes) of nodes is elected from the set G of nodes that have not yet served as a clusterhead (initially, and after every 1/P rounds, G encompasses all nodes). At the beginning of round r, each node in G becomes a clusterhead with probability $P/(1 - P \cdot (r \bmod 1/P))$. This probability increases with every round, such that in round $1/P - 1$, all as-yet-unelected nodes will become a clusterhead with probability 1, ensuring that every node is serving as a clusterhead exactly once in some round. In round 1/P , the process starts afresh.

It further discuss the suitability of the resulting clustering structure for transmission scheduling and how this scheme can be used to determine multiple levels of clustering. Overall, this is a simple and elegant solution to the rotation problem, but requiring that all clusterheads can directly talk to a data sink should (and can) be replaced by some more elaborate mechanisms.

Some more algorithm examples

On the basis of these principal considerations, a few more algorithms shall be described in slightly more detail.

A Weighted Clustering Algorithm

The node weights (or ranks) discussed so far have been fairly simple: identifiers, node degree, or (inverse) node speed. None of these parameters can fully express all aspects of a node's suitability to serve as a clusterhead. Moreover, there might be constraints imposed by other system layers on the topology selection; for example, Bluetooth only allows a clusterhead (a master) to control clusters of at most seven members (slaves). In general, it might be desirable to prescribe a desirable size of a cluster in number of nodes that a clusterhead can efficiently control. It describe a clustering algorithm that takes the following aspects into account to compute node weights:

• A cluster should not exceed a maximum size δ

• Battery power (being a clusterhead means increased effort, which should be balanced over all nodes)

• Mobility (slow nodes are preferred)

• Closeness of neighbors (clusters with short distances between members are preferred).

The actual algorithm is then essentially identical to the ones discussed above where small weights take precedence (ties are broken arbitrarily). An interesting aspect of this algorithm is that it will, all else being equal, rotate the role of clusterheads among several nodes to ensure sharing of the load between several nodes.

An emergent algorithm for cluster establishment

Most of the algorithms described so far were distributed in that there was no central entity that knew about the complete state of the network and computed the final solution. They were, in this sense, localized – nodes only drew upon information known to themselves or to their neighbors – but they still had a clear goal explicitly incorporated into the algorithm (e.g. nodes with highest degree become clusterheads).

An alternative approach to construct localized algorithms does away with such explicit goals. These are so-called emergent algorithms or protocols. In this algorithm, every node can be in three states: unclustered (unaware of any cluster), clusterhead, or follower (to potentially more than one clusterhead; only at the end a node finally decides for a single clusterhead). Unclustered nodes turn themselves into clusterheads spontaneously (after random delays) if there is no cluster in their vicinity; these clusterheads recruit their neighbors as followers. The interesting idea now is that clusterheads can abdicate if there is a follower node that would make a better clusterhead, for example, one that would have more followers and less overlap with other clusters. Such a superior node will be promoted to clusterhead status by the old, abdicating clusterhead. In effect, the clusterhead role moves around in the network. Nodes terminate the algorithm after a predefined time.

The interesting property of this algorithm is that it achieves a packing efficiency that approaches closest hexagonal packing of clusters in a given area. Its runtime is constant, independent of the size of the network (as enough clusterheads are spawned in a distributed fashion). It does outperform algorithms like "lowest ID".

Multihop clusters

The clusters discussed so far have all been derived from the maximum independent set formulation, with clusterheads forming a dominating set as well. Consequently, the maximum diameter of a cluster is two, resulting in relatively small clusters. Depending on the purpose of clustering, larger clusters can be useful even though not every node is a neighbor of a clusterhead then – routing or aggregation protocols, for example, can profit even from larger clusters whereas cluster support for MAC protocols is mostly based on the dominance property of the clusterheads.

A crucial problem here is to limit the cluster size from both above and below. An early treatment of this topic can be found, where an expanding ring search is used. In this search, the depth limit is successively increased until the cluster exceeds a given size threshold. it also discuss the problem; their contribution is discussed. Another example for such multihop clusters, which also discusses the relationship to MAC protocols, is described.

Fixing the size of clusters by growth budgets

A slightly different tilt is given to the clustering problem when trying to prescribe the size of a cluster, that is, the number of nodes within it, rather than its maximum depth or diameter. The basic idea (incorporated in their "rapid algorithm") is quite simple. Given a cluster target size B, a clusterhead asks its neighbors to adopt $B_i \geq 0$ nodes into the cluster, where $B - 1 = \sum B_i$ (the clusterhead itself counts as a member as well, thus $B - 1$). Each node, on being asked to adopt x nodes, becomes a member of the cluster and again asks its neighbors to find another set of nodes of size x − 1 (implicitly, a spanning tree of the cluster is formed as well). Such a search terminates when the budget has been used up or when there are no more nodes to adopt into a cluster. This readjustment can percolate up to the clusterhead and shift budget to other parts of the cluster.

Evidently, the first algorithm uses fewer messages, namely O(B), than the persistent algorithm, which has a polynomial message complexity. Other algorithms, like expanding ring search, can achieve similar goals but have worse complexities.

When to use multihop clusters

The question when to actually use multihopping within a cluster is considered. They assume a heterogenous system model where clusterheads communicate directly (over longer distances) with a remote data collection entity and sensors send their data to the clusterheads, where they can possibly be aggregated. These sensors can communicate with their clusterhead either directly or via multihop communication. The authors provide an expression for the critical distance beyond which multihopping should be used; this distance only depends on radio parameters (in particular, path-loss coefficient) and is independent of the network characteristics. Moreover, a scheme to compute the optimum number of clusterheads in such a scenario is also provided.

Multiple layers of clustering

Once clusters and their gateways have been determined, they induce a new graph where clusters are the nodes of the graph and any two nodes are connected if there exists a gateway between the clusters. To this induced graph, again a clustering algorithm can be applied, electing new clusterheads and connecting neighbouring nodes by gateways. Evidently, this process can be repeated recursively. One hoped-for advantage of such multiple layers of clustering is to contain topology changes, for example, relevant for routing protocols, better and only to modify information in a local vicinity.

One of the first papers to describe is multilayer clustering. There, a heterogeneous setup is assumed where only some nodes can relay traffic. Naturally, these nodes form the first-level clusterheads, attempting to control the size of each cluster by forming, merging, and splitting clusters. These clusters in turn can elect clusterheads, forming higher-layer clusters. The height of the hierarchy should be kept small, that is, clusters should be of uniform size. An interesting aspect is how the gateways between clusters are formed. Relay-capable nodes that are at the edge of a cluster and detect such nodes of another cluster in their vicinity can invite them to form a “virtual gateway”. To increase redundancy and stability of the topology, these gateways can also incorporate additional relay-capable nodes moving in range, merge with another gateway, or split up into two if nodes move out of range. They start out by considering a standard clustering problem in a graph, with the following three additional requirements: (i) Cluster size $|V_i|$ for any cluster V_i is bounded by a given constant k, $k \leq |V_i| < 2k$ (one cluster is allowed to be smaller than k to avoid some special cases), (ii) two clusters should only have a small, constant number of nodes in common, (iii) each node should only belong to a small set of different clusters. In fact, there are graphs where these requirements cannot be met but they are feasible for unit disk graphs (and similar graphs). These requirements are satisfied by an algorithm that traverses a breadth-first spanning tree of a given graph and connects nodes in subtrees of a node u into clusters, possibly using u to connect these subtree clusters together if they are too small. This process continues up the tree.

It suggest to the use of multiple levels of clustering to save energy in a scenario where sensor nodes are to report sensor readings to a remote processing center. They start out by a simple, randomized clusterhead election protocol where a node volunteers as a clusterhead with probability p. Clusters are of size k. Any node that is not covered by such a cluster also becomes a “forced” clusterhead. On the basis of quite standard assumptions about energy consumption and node deployment, closed-form solutions for both p and k are analytically derived. This randomized algorithm can be fairly easily extended to multiple levels: From the (level 1) clusterheads, again some of them elect themselves as level 2 clusterheads and announce this fact to their level 1 clusterhead neighbors at most k2 hops away; these then join such a level 2 cluster. The extension to more layers is obvious. Again, optimal values for pi and ki are determined, minimizing the energy spent to communicate data readings to a processing center. It is assumed that each node sends its data to a next level clusterhead, which aggregates the data from all its children before forwarding them. The authors claim that this scheme outperforms other clustering schemes in the resulting energy efficiency and that the algorithm has lower complexity than most other ones.

Passive clustering

In terms of energy consumption, one of the most expensive operations in a network is flooding: disseminating a particular piece of operation to all nodes. Flooding happens, for example, in routing protocols when routes have to be computed, but it also occurs when a new data sink announces its interest in certain kinds of observable data.

Usually, flooding is implemented by every node repeating every packet that it has received, with the exception of already received ones to avoid cycles. But not every node would have to retransmit a packet; because of the broadcast nature of the wireless channel, retransmission by a minimum dominating set – such as clusterheads and gateways connecting them – would suffice. This clustering overhead can be reduced if the information flow that is happening anyway during a flooding operation is leveraged to compute a clustering structure on the fly. Actively sending out any message for clustering as such is avoided; the approach discussed here is hence called passive clustering. The necessary information exchange is achieved by adding state information about each sender into any packet that is sent anyway, namely "initial", "clusterhead", "gateway", and "ordinary node". This distributes information about the state of neighbouring nodes; it suffices to build a clustering structure that well approximates maximum independent sets with optimal gateway choice and is competitive with ID-based or degree-based algorithms.

The procedure works as follows. Suppose a node starts a flood; it will be stamped as coming from an "initial" node. The first node receiving and forwarding this packet will become a clusterhead and announces this fact by appropriately stamping the forwarded packet. Any initial nodes receiving such a packet will turn into "ordinary nodes" or into gateways.

The decision to become a gateway depends on the number of clusterheads and other gateways that a node has already heard from. Intuitively, a node that has heard from two or more clusterheads should become a gateway to connect these two clusterheads but only if there is no other gateway nearby already fulfilling this role.

Hence, after the initial declaration of a clusterhead, up to two nodes can declare themselves as gateways (depending on the choice of a and β) and forward the flood packet; all other nodes (hearing from these two gateways) will declare themselves as ordinary nodes and stop forwarding the packet.

In effect, a set of clusterheads and gateways is constructed while performing the flooding operation, limiting the required overhead. While there is no means to guarantee a nearly optimal performance, simulations show that for practical networks the resulting clustering structure is quite similar to that produced by active clustering schemes. 6. Sensor Tasking and Control.

2. TIME SYNCHRONIZATION

Introduction to the time synchronization problem

In this section, we explain why time synchronization is needed and what the exact problems are, followed by a list of features that different time synchronization algorithms might have. We also discuss the particular challenges and constraints for time synchronization algorithms in wireless sensor networks.

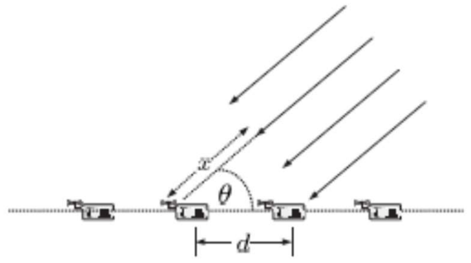


Figure 8.1 Determination of angle of arrival of a distant sound event by an array of acoustic sensors

The need for time synchronization in wireless sensor networks

Time plays an important role in the operation of distributed systems in general and in wireless sensor networks in particular, since these are supposed to observe and interact with physical phenomena.

A simple example shall illustrate the need for accurate timing information (Figure 1). An acoustic wavefront generated by a sound source a large distance away impinges onto an array of acoustic sensors and the angle of arrival is to be estimated. Each of the sensors knows its own position exactly and records the time of arrival of the sound event. In the specific setup shown in the figure, the angle θ can be determined when the lengths d and x are known, using the trigonometric relationship $x = d \cdot \sin \theta$, and accordingly $\theta = \arcsin \frac{x}{y}$.1 The sensor distance d can be derived from the known position of the sensors and the distance x can be derived from the time difference Δt between the sensor readings and the known speed of sound $c \approx 330$ m/s, using $x = c \cdot \Delta_t$. Assuming d = 1 m and $\Delta_t = 0.001$ s gives $\theta \approx 0.336$ (in radians). If the clocks of the sensors are only within 500 μs accurate, the true time difference can be in the range between 500 and 1500 μs, and thus the estimates for θ can vary between $\theta \approx 0.166$ and $\theta \approx 0.518$. Therefore, a seemingly small error in time synchronization can lead to significantly biased estimates.

There are at least two ways to get a more reliable estimate. The first one (and the one focused on in this chapter) is to keep the sensors clocks as tightly synchronized as possible, using dedicated time synchronization algorithms. The second one is to combine the readings of multiple sensors and to “average out” the estimation errors. There are many other applications requiring accurate time synchronization, for example, beamforming. However, not only WSN applications but also many of the networking protocols used in sensor networks need accurate time. Prime examples are MAC protocols based on TDMA or MAC protocols with coordinated wakeup, like the one used in the IEEE 802.15.4 WPAN standard. Sensor nodes running a TDMA protocol need to agree on boundaries of time slots; otherwise their transmissions would overlap and collide.

It is important to note that the time needed in sensor networks should adhere to physical time, that is two sensor nodes should have the same idea about the duration of 1 s and additionally a sensor node’s second should come as close as possible to 1 s of real time or coordinated universal time (UTC).2 The physical time has to be distinguished from the concept of logical time that allows to determine the ordering of events in a distributed system but does not necessarily show any correspondence to real time.

Node clocks and the problem of accuracy

Almost all clock devices of sensor nodes and computers share the same common structure. The node possesses an oscillator of a specified frequency and a counter register, which is incremented in hardware after a certain number of oscillator pulses. The node’s software has only access to the value of this register and the time between two increments determines the achievable time resolution: events happening between two increments cannot be distinguished from their timestamps.

The value of the hardware clock of node i at real time t can be represented as $H_i(t)$. It can be understood as an abstraction of the counter register providing an ever-increasing time value. A common approach to compute a local software clock $L_i(t)$ from this value is to apply an affine transformation to the hardware clock

$$L_i(t) := \theta i \cdot H_i(t) + \varphi i .$$

φi is called phase shift and θi is called drift rate. Given that it is often neither possible nor desirable to influence the oscillator or the counter register, one can change the coefficients θi

and φi to do clock adjustment. Now we can define the notion of precision of the clocks within a network, where we distinguish two cases:

External synchronization The nodes 1, 2, . . . , n are said to be accurate at time t within a bound δ if $|L_i(t) - t| < \delta$ holds for all nodes $i \in \{1, 2, \dots, n\}$.

Internal synchronization The nodes 1, 2, . . . , n are said to agree on the time with a bound of δ if $|L_i(t) - L_j(t)| < \delta$ holds for all $i, j \in \{1, 2, \dots, n\}$.

To achieve external synchronization, a reliable source of real time/UTC time must be available, for example, a GPS receiver. Clearly, if nodes 1, 2, . . . , n are externally synchronized with bound δ, they are also internally synchronized with bound 2δ. There are three problems:

• Nodes are switched on at different and essentially random times, and therefore, without correction, their initial phases φi are random too.

• Oscillators often have a priori a slight random deviation from their nominal frequency, called drift and sometimes clock skew. This can be due to impure crystals but oscillators also depend on several environmental conditions like pressure, temperature, and so on, which in a deployed sensor network might well differ from laboratory specifications. The clock drift is often expressed in parts per million (ppm) and gives the number of additional or missing oscillations a clock makes in the amount of time needed for one million oscillations at the nominal rate. In general, cheaper oscillators – like those used in designs for cheap sensor nodes – have larger drifts with higher probability.

• The oscillator frequency is time variable. There are short-term variations – caused by temperature changes, variations of electric supply voltage, air pressure, and so on – as well as long-term variations due to oscillator aging. It is often safe to assume that the oscillator frequency is reasonably stable over times in the range of minutes to tens of minutes. On the other hand, this also implies that time synchronization algorithms should resynchronize once every few minutes to keep track of changing frequencies. This implies that even if two nodes have the same type of oscillator and are started at the same time with identical logical clocks, the difference $|L_i(t) - L_j(t)|$ can become arbitrarily large as t increases. Therefore, a time synchronization protocol is needed.

Properties and structure of time synchronization algorithms Time synchronization protocols can be classified according to certain criteria:

Physical time versus logical time In wireless sensor networks, applications and protocols mostly require physical time.

External versus internal synchronization Algorithms may or may not require time synchronization with external timescales like UTC.

Global versus local algorithms A global algorithm attempts to keep all nodes of a sensor network (partition) synchronized. The scope of local algorithms is often restricted to some geographical neighborhood of an interesting event. In global algorithms, nodes are therefore required to keep synchronized with not only single-hop neighbors but also with distant nodes (multihop). Clearly, an algorithm giving global synchronization also gives local synchronization.

Absolute versus relative time Many applications like the simple example presented in Section 8.1.1 need only accurate time differences and it would be sufficient to estimate the drift instead of phase offset. However, absolute synchronization is the more general case as it includes relative synchronization as a special case.

Hardware- versus software-based algorithms Some algorithms require dedicated hardware like GPS receivers or dedicated communication equipment while software-based algorithms use plain message passing, using the same channels as for normal data packets.

A priori versus a posteriori synchronization In a priori algorithms, the time synchronization protocol runs all the time, even when there is no external event to observe. In a posteriori synchronization (also called post-facto synchronization, the synchronization process is triggered by an external event.

Deterministic versus stochastic precision bounds Some algorithms can (under certain conditions) guarantee absolute upper bounds on the synchronization error between nodes or with respect to external time. Other algorithms can only give stochastic bounds in the sense that the synchronization error is with some probability smaller than a prescribed bound.

Local clock update discipline How shall a node update its local clock parameters φi and θi? An often-found requirement is that backward jumps in time should be avoided, that is for $t < t'$ it shall not happen that $L_i(t) > L_i(t')$ after an adjustment.4 An additional requirement might be to avoid sudden jumps, that is the difference $L_i(t') - L_i(t)$ for times t immediately before and t' immediately after readjustment should be small. The most important performance metrics of time synchronization algorithms are the following:

Precision For deterministic algorithms, the maximum synchronization error between a node and real time or between two nodes is interesting; for stochastic algorithms, the mean error, the error variance, and certain quantiles are relevant.

Energy costs The energy costs of a time synchronization protocol depend on several factors: the number of packets exchanged in one round of the algorithm, the amount of computation needed to process the packets, and the required resynchronization frequency.

Memory requirements To estimate drift rates, a history of previous time synchronization packets is needed. In general, a longer history allows for more accurate estimates at the cost of increased memory consumption.

Fault tolerance How well can the algorithm cope with failing nodes, with error-prone and time variable communication links, or even with network partitions? Can the algorithm handle mobility? It is useful to decompose time synchronization protocols for wireless sensor networks into four conceptual building blocks, the first three of which are already identified:

• The resynchronization event detection block identifies the points in time where resynchronization is triggered. In most protocols, resynchronizations are triggered periodically with a period depending on the maximum drift rate. A single resynchronization process is called a round. If rounds can overlap in time, sequence numbers are needed to distinguish them and to let a node ignore all but the newest resynchronization rounds.

• The remote clock estimation block acquires clock values from remote nodes/remote clocks. There are two common variants. First, in the time transmission technique, a node i sends its local clock $L_i(t)$ at time t to a neighboring node j , which receives it at local time $L_j(t')$ (with $t' > t$). Basically, node j assumes $t \approx t'$ and uses $L_i(t)$ as estimation for the time $L_i(t')$. This estimation can be made more precise by removing known factors from the difference $t' - t$, for example the time that i packet occupies the channel and the propagation delay. Second, in the remote clock reading technique, a node j sends a request message to another node i, which answers with a response packet. Node j estimates i's clock from the round-trip time of the message and the known packet transmission times. Finally, node j may inform node i about the outcome. This technique is discussed in more detail below.

• The clock correction block computes adjustments of the local clock based on the results of the remote clock estimation block.

• The synchronization mesh setup block determines which nodes synchronize with each other in a multihop network. In fully connected networks, this block is trivial.

Time synchronization in wireless sensor networks

In wireless sensor networks, there are some specifics that influence the requirements and design of time synchronization algorithms:

• An algorithm must scale to large multihop networks of unreliable and severely energy-constrained nodes. The scalability requirement refers to both the number of nodes as well as to the average node degree/node density.

• The precision requirements can be quite diverse, ranging from microseconds to seconds.

• The use of extra hardware only for time synchronization purposes is mostly ruled out because of the extra cost and energy penalties incurred by dedicated circuitry.

• The degree of mobility is low. An important consequence is that a node can reach its neighbors at any time, whereas in networks with high degree of mobility, intermittent connectivity and sporadic communication dominates (there are some publications explicitly targeting MANETs).

• There are mostly no fixed upper bounds for packet delivery delay, owing to the MAC protocol, packet errors, and retransmissions.

• The propagation delay between neighboring nodes is negligible. A distance of 30 m needs 10^{-7} s for speed of light c ≈ 300.000.000 m/s.

• Manual configuration of single nodes is not an option. Some protocols require this, for example, the network time protocol (NTP), where each node must be configured with a list of time servers.

• It will turn out that the accuracy of time synchronization algorithms critically depends on the delay between the reception of the last bit of a packet and the time when it is timestamped. Optimally, timestamping is done in lowest layers, for example, the MAC layer. This feature is much easier to implement in sensor nodes with open firmware and software than it would be using commodity hardware like commercial IEEE 802.11 network cards. Many of the traditional time synchronization protocols try to keep the nodes synchronized all the time, which is reasonable when there are no energy constraints and the topology is sufficiently stable. Accordingly, energy must be spent all the time for running time synchronization protocols. For several sensor network applications this is unnecessary, for example, when the main task of a network is to monitor the environment for rare events like forest fires. With post-facto synchronization (or a posteriori synchronization), a time synchronization on demand can be achieved. Here, nodes are unsynchronized most of the time. When an interesting external event is observed at time t0, a node i stores its local timestamp L_i (t_0) and triggers the synchronization protocol, which for example, provides global synchronization with UTC time. After the protocol has finished at some later time t_1, node i has learned about its relative offset Δ to UTC time, that is, $t_1 = L_i (t_1) + \Delta$. Node i can use this information to relate the past event at t_0 also to UTC time. After node i has delivered the information about the event, it can go into sleep mode again, dropping synchronization. In a nutshell, post-facto synchronization is synchronization on demand and for a short time, to report about an important event.

Before discussing some of the proposals for time synchronization protocols suitable for sensor networks, let us briefly discuss some of the "obvious" solutions and why they do not fit.

• Equip each node with a GPS receiver: GPS receivers still cost some few dollars, need a separate antenna, need energy continuously to keep in synch (acquiring initial synchronization takes minutes!), and have form factors not compatible with the idea of very small sensor nodes. Furthermore, to be useful, a GPS receiver needs a line of sight to at least four of the GPS satellites, which is not always achievable in hilly terrains, forests, or in indoor applications. One application of GPS in a sensor network is for wildlife tracking.

• Equip each node with some receiver for UTC signals:5 the same considerations apply as for a GPS receiver.

• Let some nodes at the edge of the sensor network send strong timing signals: Such a solution can be used indoors and in flat terrain but requires a separate frequency and thus a separate transceiver on each node to let the time server not distort ongoing transmissions.

Protocols based on sender/receiver synchronization

In this kind of protocols, one node, called the receiver, exchanges data packets with another node, called the sender, to let the receiver synchronize to the sender's clock. One of the classic protocols for this is the network time protocol (NTP), widely used in the Internet. In general, sender/receiver based protocols require bidirectional links between neighboring nodes.

Lightweight time synchronization protocol (LTS)

The lightweight time synchronization (LTS) protocol attempts to synchronize the clocks of a sensor network to the clocks held by certain reference nodes, which, for example, may have GPS receivers. The protocol has control knobs that allow to trade off energy expenditure and achievable accuracy, and it gives stochastic precision bounds under certain assumptions about the underlying hardware and operating system. LTS makes no restrictions with respect to the local clock update discipline and it does not try to estimate actual drift rates.

LTS subdivides time synchronization into two building blocks:

• A pair-wise synchronization protocol synchronizes two neighboring nodes.

• To keep all nodes or the set of interesting nodes synchronized to a common reference, a spanning tree from the reference node to all nodes is constructed. If the single-hop synchronization errors are independent and identically distributed and have mean zero, the leaf nodes of the tree also have an expected synchronization error of zero but the variance is the sum of the variances along the path from the reference node to the leaf node. Therefore, this variance can be minimized by finding a minimum-height spanning tree.

Pair-wise synchronization

We first explain the pair-wise synchronization protocol (Figure 2). The protocol uses a remote clock reading technique. Suppose a node i wants to synchronize its clock to that of a node j . After the resynchronization is triggered at node i, a synchronization request packet is formatted and timestamped at time t_1 with time L_i (t1). Node i hands the packet over to the operating system and the protocol stack, where it stays for some time. The medium access delay can be highly variable and make up for a significant fraction of this time. Often, this delay is a random variable. When node i is sending the first bit at time t_2, node j receives the last bit of the packet at time $t_3 = t_2 + \tau + tp$, where τ is the propagation delay and t_p is the packet transmission time (packet length divided by bitrate). Sometime later (interrupt latency), at time t4, the packet arrival is signalled to node j 's operating system or application through an interrupt and it is timestamped at time t_5 with L_j (t_5). At t_6, node j has formatted its answer packet, timestamps it with L_j (t_6), and hands it over to its operating system and networking stack. This packet includes also the previous timestamps L_j (t_5) and L_i (t_1). Node i receives the packet reception interrupt at time t_7 (which is t_6 plus operating system/networking overhead, medium access delay, propagation delay, packet transmission time, and interrupt latency) and timestamps it at time t_8 with L_i (t_8). Let us now analyze how node i infers its clock correction. More precisely, node i wants to estimate $O = \Delta(t_1) := L_i\ (t_1) - L_j\ (t_1)$. To do this, we make the assumption that there is no drift between the clocks in the time between t_1 and t_8, that is $O = \Delta(t*)$ for all $t* \in [t_1, t_8]$, and in fact node i estimates O by estimating Δ(t5). From the figure, the timestamp L_j (t_5), which node i's gets back, is generated at some unknown time between t_1 and t_8. However, we can reduce this uncertainty by the following observations:

• There is one propagation delay τ plus one packet transmission time tp between t_1 and t_5.
• There is another time $\tau + t_p$ between t_5 and t_8 for the response packet. For stationary nodes, we can safely assume that propagation delays are the same in both directions.

The maximum synchronization error of this scheme is |I |/2 if the times τ and tp are known with high precision. The actual synchronization error can essentially be attributed to different interrupt latencies at i and j , to different times between getting a receive interrupt and timestamping the packet, and to different channel access times. These uncertainties can be reduced significantly if the requesting node can timestamp its packet as lately as possible, best immediately before transmitting or right after obtaining medium access.

Networkwide synchronization

Given the ability to carry out pair-wise synchronizations, LTS next solves the task to synchronize all nodes of a (connected) sensor network with a reference node. If a specific node i has a distance of hi hops to the reference node, and if the synchronization error is normally distributed with parameters μ = 0 and σ' = 2σ at each hop, and if furthermore the hops are independent, the synchronization error of i is also normally distributed with variance $\sigma 2 = 4h_{i\sigma}{}^2$. On the basis of this observation, LTS aims to construct a spanning tree of minimum height and only node pairs along the edges of the tree are synchronized. If the synchronization process along the spanning tree takes a lot of time, the drift between the clocks will introduce additional errors.

Two different variants are proposed, a centralized and a distributed one.

Centralized multihop LTS

The reference node – for example, a node with a GPS receiver or another high-quality time reference – constructs a spanning tree T and starts synchronization: First the reference node synchronizes with its children in T , then the children with their children, and so forth. Hence, each node must know its children. There are several algorithms available for distributed construction of a spanning tree; for LTS, two specific ones are discussed, namely the distributed depth-first search (DDFS) and the Echo algorithms. The reference node also has to take care of frequent resynchronization to compensate for drift. It is assumed that the reference node knows four parameters: the maximum height h of the spanning tree, the maximum drift ρ such that Equation 8.1 is satisfied for all nodes in the network, the singlehop standard deviation $2 \cdot \sigma$ (discussed above), and the desired accuracy δ. The goal is to always have a synchronization error of leaf nodes smaller than δ with 99% probability.8 Immediately after resynchronization, a leaf node's accuracy is smaller than $h \cdot 2.3 \cdot 2 \cdot \sigma$ and it is allowed to grow at most to level δ. With maximum drift rate ρ, this growth takes $\delta{-}2{\cdot}2.3{\cdot}h{\cdot}\sigma\ \rho$ time. The actual choice of the synchronization period should be somewhat smaller to account for the drift occurring during a single resynchronization, possibly harming the initial accuracy $2.3 \cdot 2 \cdot h \cdot \sigma$.

A critical issue is the communication costs. A single pair-wise synchronization costs three packets, and synchronizing a network of n nodes therefore costs on the order of 3n packets, not taking channel errors or collisions into consideration. Additionally, significant energy is needed to construct the spanning tree, and it is proposed to repeat this construction upon each synchronization round to achieve some fault tolerance. For reasons of fault tolerance, it is also beneficial to have multiple reference nodes: If one of them fails or if the network becomes partitioned, another one can take over. A leader election protocol is useful to support dynamic reference nodes.

Distributed multihop LTS

The second variant is the distributed multihop LTS protocol. No spanning tree is constructed, but each node knows the identities of a number of reference nodes along with suitable routes to them. It is the responsibility of the nodes to initiate resynchronization periodically. Consider the situation shown in Figure 8.3 and assume that node 1 wants to synchronize with the reference node R. Node 1 issues a synchronization request toward R, which results in a sequence of pair-wise synchronizations: node 4 synchronizes with node R, node 3 synchronizes with node 4, and so forth until node 1 is reached. Two things are noteworthy:

• As a by-product, nodes 2, 3, and 4 also achieve synchronization with node R.

• Given the same accuracy requirement δ and the same drift rate ρ for all nodes, the resynchronization frequency for a node i. Therefore, in the figure, nodes 1 and 6 have the shortest resynchronization period. If these two nodes always request resynchronization with node R, the intermediate nodes 2, 3, 4, and 5 never have to request resynchronization by themselves. A node should choose the closest reference node to minimize its synchronization error. This way, a minimum weight tree is not constructed explicitly, but it is the responsibility of the routing protocol to find good paths.

Another optimization of LTS is also explained in Figure 3, using dashed lines. Suppose again that node 5 wants to synchronize. As explained above, one option would be to let node 5 join an ongoing synchronization request at node 3. On the other hand, it might be necessary to keep nodes 7 and 8 also synchronized with R. To achieve this, node 5 can issue its request through nodes 7 and 8 to R and synchronize the intermediate hops as a by-product. This is called path diversification. The properties of the LTS variants were investigated by simulating 500 randomly distributed nodes in a 120m (120m rectangle, each node having a transmit range of 10 m. The single reference node is placed at the center. It is shown that the distributed multihop LTS is more costly in terms of exchanged packets when all nodes of a network have to be synchronized (between 40 and 100% overhead to the central algorithm), even when optimizations like path diversification or joining ongoing synchronizations are employed (reducing overhead to 15 to 60 %). However, if only a fraction of nodes has to be synchronized, the distributed algorithms can restrict its overhead to the interesting set and conserve all the other nodes' energy whereas the centralized algorithms always include all nodes and thus have fixed costs that occur whether or not time synchronization is currently requested.

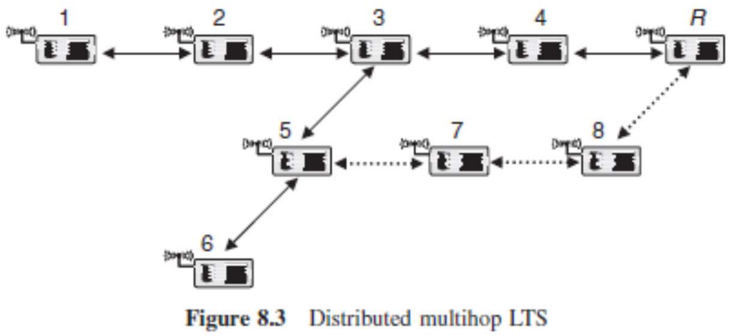


Figure 8.3 Distributed multihop LTS

How to increase accuracy and estimate drift

We take the opportunity and use the pair-wise synchronization protocol of LTS to explain how the synchronization error between nodes can be decreased and how the drift of node x's clock with respect to a reference node R's clock can be estimated. Both can be transformed into standard estimation problems.

Increasing accuracy

Assuming that the drift between x and R is negligible for a certain time and node x wants to estimate the phase offset to R's clock. Node x can increase the accuracy of its estimation by repeating the packet exchange, obtaining multiple estimates $O(t_0)$,$O(t_1)$, . . . , $O(t_n-1)$. Let A be the initial phase offset at time t_0 and let us assume that the synchronization errors observed at the different times t_0, . . . , t_n-1 are independent. One can therefore model each $O(t_k)$ as:

$$O(t_k) = A + w(t_k)$$

where the $w(t_k)$ are sampled from a white Gaussian noise process with zero mean and standard deviation $\sigma_x = 2 \cdot \sigma$, that is all $w(t_k)$ are independent.

If the node responding to a request packet is known to send its answer as quickly as possible, another approach making fewer assumptions about the observations and the noise can be used. Referring to Figure 8.2, we can observe the following: When node i makes the k-th observation Ok, it measures the time difference between times t_1,k and t_8,k , that is the times between timestamping the request and the response packets. Clearly, the observation k with the minimum difference t_8,k − t_1,k has the smallest uncertainty and is the most precise one.

Drift estimation

Clearly, it is impossible to estimate the drift from just one resynchronization. Therefore, let O(t0),O(t1), . . . , O(tn−1) be the estimated offsets obtained by node x for the resynchronizations carried out at times t0, t1, . . . , tn−1. We assume that node x's drift ρx is constant through the interval [t_0, tn−1]. Let us first consider the case where the pair-wise synchronization protocol runs repeatedly but node x does not readjust its clock after making an observation.

Timing-sync protocol for sensor networks (TPSN)

The Timing-Sync Protocol for Sensor Networks (TPSN) is another interesting sender receiver based protocol. Again, we first explain the approach to pair-wise synchronization before turning to the multihop case.

Pair-wise synchronization

The pair-wise synchronization protocol of TPSN has some similarities with LTS. It operates in an asymmetric way: Node i synchronizes to the clock of another node j but not vice versa (Figure 8.4). The operation is as follows:

• Node i initiates resynchronization at time t0. It formats a synchronization pulse packet and hands it over to the operating system and networking stack at time t1.

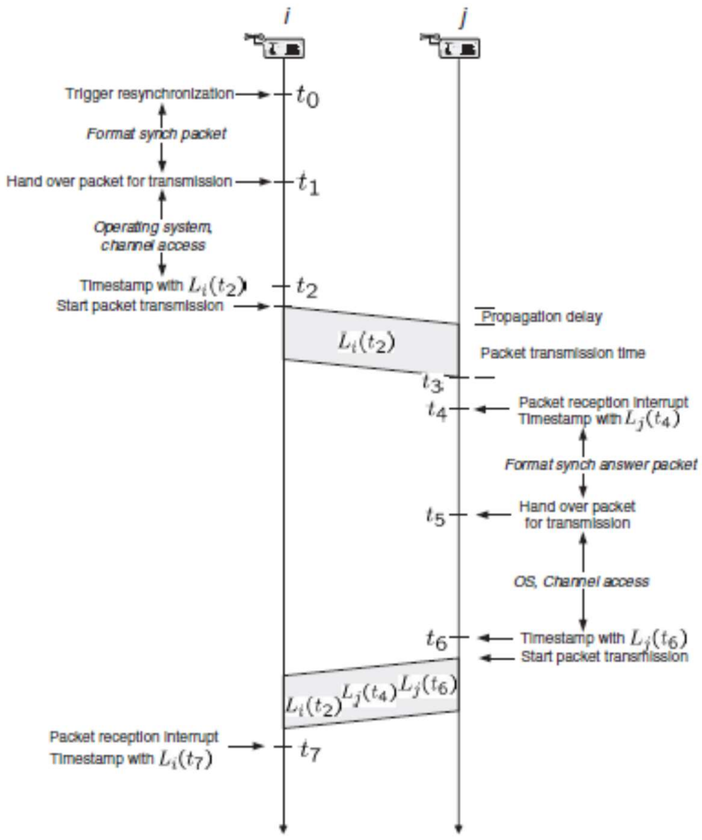

i
j
Trigger resynchronization
t_0
Format synch packet
Hand over packet for transmission
t_1
Operating system, channel access
Timestamp with $L_i(t_2)$
t_2
Start packet transmission
Propagation delay
$L_i(t_2)$
Packet transmission time
t_3
t_4
Packet reception interrupt Timestamp with $L_j(t_4)$
Format synch answer packet
t_5
Hand over packet for transmission
OS, Channel access
t_6
Timestamp with $L_j(t_6)$
Start packet transmission
$L_i(t_2)$ $L_j(t_4)$ $L_j(t_6)$
Packet reception interrupt Timestamp with $L_i(t_7)$
t_7

Figure 8.4 Sender/receiver synchronization in TPSN

• The networking stack executes the MAC protocol and determines a suitable time for transmission of the packet, say t_2. Immediately before transmission, the packet is timestamped with Li (t2). By timestamping the packet immediately before transmission and not already when the packet has been formatted in the application layer, two sources of uncertainty are removed: the operating system/networking stack and the medium access delay. The remaining uncertainty is the small time between timestamping the packet and the true start of its transmission. This delay is created, for example, by the need to recompute the packet checksum immediately before sending it.

• After propagation delay and packet transmission time, the last bit arrives at the receiver at time t3, and some time after this the packet receive interrupt is triggered, say at time t4. The receiver timestamps the packet already in the interrupt routine with Lj (t4).

• Node j formats an acknowledgement packet and hands it over at time t5 to the operating system and networking stack. Again, the networking stack executes the MAC protocol and sends the packet at time t6. Immediately before transmission, the packet is timestamped with Lj (t6), and the packet carries also the other timestamps Li (t2) and Lj (t4).

The key feature of this approach is that node i timestamps the outgoing packet as lately as possible and node j timestamps the incoming packet as early as possible. This requires support from the MAC layer, which is easier to achieve in sensor nodes than with commodity hardware like IEEE 802.11 network interface cards. This protocol allows arbitrary jumps in node i's local clock.

Networkwide synchronization

The networkwide synchronization algorithm of TPSN essentially builds a spanning tree where each node knows its level in the tree and the identity of its parent. The root node is assigned level 0 and it is its responsibility to trigger the construction of the tree. All reachable nodes in the network synchronize with the root node. If the root node has access to a precise external timescale like UTC, all nodes therefore synchronize to UTC.

The protocol works as follows. To start the tree construction, the root node sends a level discovery packet containing its level 0. All one-hop neighbors of the root node assign themselves a level of one plus the level indicated in the received level discovery packet and accept the root as their parent. Subsequently, the level 1 nodes send their own level discovery packets of level 1 and so forth. The level 1 nodes choose a random delay to avoid excessive MAC collisions. Once a node has received a level discovery packet, the packet originator is accepted as parent and all subsequent packets are dropped. After a node has found a parent, it periodically resynchronizes to the parent's clock.

A node might fail to receive level discovery packets because of MAC collisions or because it is deployed after initial tree construction. If a node i does not receive any level discovery packet within a certain amount of time, it asks its one-hop neighborhood about an already existing tree by issuing a level request packet. The neighboring nodes answer by sending their own level. Node I collects the answers from some time window and chooses the neighbor with the smallest level as its parent.

The tree maintenance is integrated with resynchronization. To account for drift, a node i must run the pair-wise algorithm with its parent j periodically. If this fails subsequently for a number of times, node i concludes that its parent has moved or passed away. If the level of i is two or larger, it sends a level request packet, collects the answers for some time and assigns itself a new level from the lowest-level answer packet. If i is at level one, it concludes that the root node has died. There are several possibilities to resolve this situation. One of them is to run a leader election protocol among level 1 nodes. This approach has the following properties:

• The resulting spanning tree is not necessarily a minimal one, since MAC collisions and random delays may lead to a situation where a node receives a level discovery from a higher-level node first. However, there is a trade-off between the synchronization accuracy (longer paths imply larger average error) and the overhead for tree construction. Algorithms for finding minimal spanning trees are more elaborate.

• If two nodes i and j are geographically close together and receive the same level ν level discovery in the tree setup phase, both assign themselves level ν + 1 and try to resend the level discovery packet. One of them wins contention. Since both are close together, their one-hop neighborhoods are almost identical. As a result, all so-far-unsynchronized neighbors accept node i as their parent and create significant resynchronization load for i, whereas node j spends almost no energy because it has no children. To avoid unfairness, the tree construction should be repeated periodically, which in turn creates network load.

• The average magnitude of the synchronization error between a level ν node and the root node increases with ν, but gracefully. For one hop, the average synchronization error is ≈17 μs and for five hops ≈23 μs.

• It is possible to achieve post-facto synchronization. In this case, no spanning tree is constructed. Consider a scenario in which a node i0 wants to communicate an event (which happened at time t) to another node in over a number of intermediate hops i1, i2, . . . , in^{-1}. Node i0 sends the packet with its local timestamp Li0 (t) to i1. Subsequently, node i1 synchronizes its clock to that of node i0 and forwards the packet to node i2, and so forth. Finally, all nodes including node in have synchronized to node i0 and in has the packet with timestamp Li0 (t) and can thus decide about the age of the event.

3 LOCALIZATION AND POSITIONING

In many circumstances, it is useful or even necessary for a node in a wireless sensor network to be aware of its location in the physical world. For example, tracking or event-detection functions are not particularly useful if the WSN cannot provide any information where an event has happened. To do so, usually, the reporting nodes' location has to be known. Manually configuring location information into each node during deployment is not an option. Similarly, equipping every node with a Global Positioning System (GPS) receiver fails because of cost and deployment limitations (GPS, e.g. does not work indoors).

This chapter introduces various techniques of how sensor nodes can learn their location automatically, either fully autonomically by relying on means of the WSN itself or by using some assistance from external infrastructure.

Properties of localization and positioning procedures

The simple intuition of "providing location information to a node" has a number of facets that should be classified to make the options for a location procedure clear. The most important properties are,

Physical position versus symbolic location Does the system provide data about the physical position of a node (in some numeric coordinate system) or does a node learn about a symbolic location – for example, "living room", "office 123 in building 4"? Is it, in addition, possible to match physical position with a symbolic location name (out of possibly several applicable ones)?

While these two concepts are different, there is no consistent nomenclature in the literature – position and location are often used interchangeably. The tendency is to use "location" as the more general term. We have to rely on context to distinguish between these two contexts.

Absolute versus relative coordinates An absolute coordinate system is valid for all objects and embedded in some general frame of reference. For example, positions in the Universal Transverse Mercator (UTM) coordinates form an absolute coordinate system for any place on earth. Relative coordinates, on the other hand, can differ for any located object or set of objects – a WSN where nodes have coordinates that are correct with respect to each other but have no relationship to absolute coordinates is an example.

To provide absolute coordinates, a few anchors are necessary (at least three for a two dimensional system). These anchors are nodes that know their own position in the absolute coordinate system. Anchors can rotate, translate, and possibly scale a relative coordinate system so that it coincides with the absolute coordinate system. These anchors are also commonly called "beacons" or "landmarks" in the literature.

Localized versus centralized computation Are any required computations performed locally, by the participants, on the basis of some locally available measurements or are measurements reported to a central station that computes positions or locations and distributes them back to the participants? Apart from scaling and efficiency considerations (both with respect to computational and communication overhead), privacy issues are important here as it might not be desirable for a participant to reveal its position to a central entity.

Accuracy and precision The two most important figures of merit for a localization system are the accuracy and the precision of its results. Positioning accuracy is the largest distance between the estimated and the true position of an entity (high accuracy indicates a small maximal mismatch). Precision is the ratio with which a given accuracy is reached, averaged over many repeated attempts to determine a position. For example, a system could claim to provide a 20-cm accuracy with at least 95% precision. Evidently, accuracy and precision values only make sense when considered together, forming the accuracy/precision characteristic of a system.

Scale A system can be intended for different scales, for example – in indoor deployment – the size of a room or a building or – in outdoor deployment – a parking lot or even worldwide operation. Two important metrics here are the area the system can cover per unit of infrastructure and the number of locatable objects per unit of infrastructure per time interval.

Limitations For some positioning techniques, there are inherent deployment limitations. GPS, for example, does not work indoors; other systems have only limited ranges over which they operate.

Costs Positioning systems cause costs in time (infrastructure installation, administration), space (device size, space for infrastructure), energy (during operation), and capital (price of a node, infrastructure installation).

Figure 1 illustrates the positioning problem. The figures in this chapter use the "access point" icon to indicate anchors for easy distinction. It should be pointed out, however, that normal sensor nodes can just as well be used as anchors, as long as they have are aware of their position. In addition, a positioning or localization system can be used to provide the recognition or classification of objects; this property is less important in the WSN context or, if used, usually not considered a part of the localization system.

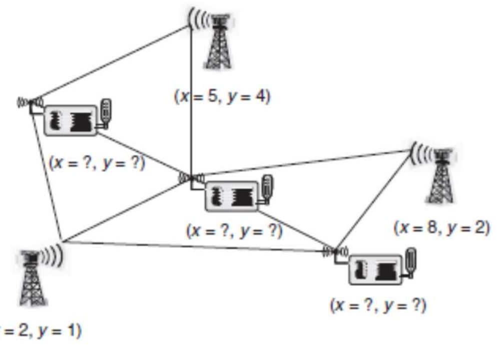


Figure 9.1 Determining the position of sensor nodes with the assistance from some anchor points; not all nodes are necessarily in contact with all anchors

Possible approaches

Three main approaches exist to determine a node's position: Using information about a node's neighborhood (proximity-based approaches), exploiting geometric properties of a given scenario (triangulation and trilateration), and trying to analyze characteristic properties of the position of a node in comparison with premeasured properties (scene analysis).

Proximity

The simplest technique is to exploit the finite range of wireless communication. It can be used to decide whether a node that wants to determine its position or location is in the proximity of an anchor. While this only provides coarse-grain information, it can be perfectly sufficient. One example is the natural restriction of infrared communication by walls, which can be used to provide a node with simple location information about the room it is in.

Proximity-based systems can be quite sophisticated and can even be used for approximate positioning when a node can analyze proximity information of several overlapping anchors. They can also be relatively robust to the uncertainties of the wireless channel – deciding whether a node is in the proximity of another node is tantamount to deciding connectivity, which can happen on relatively long time scales, averaging out short-term fluctuations.

Trilateration and triangulation

Lateration versus angulation

In addition to mere connectivity/proximity information, the communication between two nodes often allows to extract information about their geometric relationship. For example, the distance between two nodes or the angle in a triangle can be estimated – how this is done is discussed in the following two subsections. Using elementary geometry, this information can be used to derive information about node positions. When distances between entities are used, the approach is called lateration; when angles between nodes are used, one talks about angulation.

For lateration in a plane, the simplest case is for a node to have precise distance measurements to three noncolinear anchors. The extension to a three-dimensional space is trivial (four anchors are needed); all the following discussion will concentrate on the planar case for simplicity. Using distances and anchor positions, the node's position has to be at the intersection of three circles around the anchors (Figure 2).

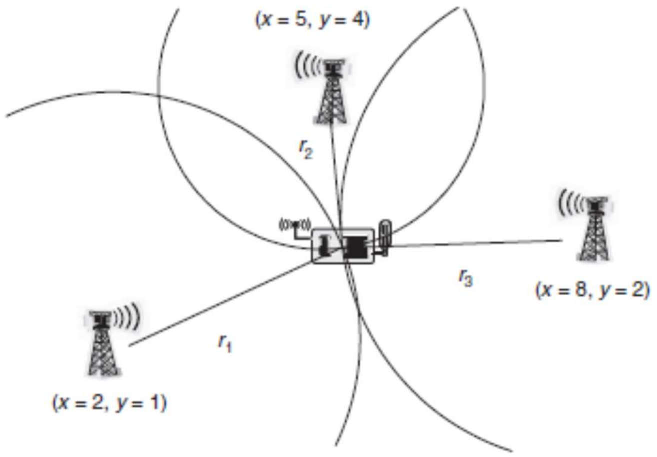


Figure 9.2 Triangulation by intersecting three circles

The problem here is that, in reality, distance measurements are never perfect and the intersection of these three circles will, in general, not result in a single point. To overcome these imperfections, distance measurements form more that three anchors can be used, resulting in a multilateration problem. Multilateration is a core solution technique, used and reused in many concrete systems described below. Angulation exploits the fact that in a triangle once the length of two sides and two angles are known the position of the third point is known as the intersection of the two remaining sides of the triangle. The problem of imprecise measurements arises here as well and can also be solved using multiple measurements.

Determining distances

To use (multi-)lateration, estimates of distances to anchor nodes are required. This ranging process1 ideally leverages the facilities already present on a wireless node, in particular, the radio communication device. The characteristics of wireless communication are partially determined by the distance between sender and receiver, and if these characteristics can be measured at the receiver, they can serve as an estimator of distance. The most important characteristics are Received Signal Strength Indicator (RSSI), Time of Arrival (ToA), and Time Difference of Arrival (TDoA).

Received signal strength indicator

Assuming that the transmission power P_{tx}, the path loss model, and the path loss coefficient α are known, the receiver can use the received signal strength P_{rcvd} to solve for the distance d in a path loss equation like

$$P_{\mathrm{rcvd}} = c\frac{P_{\mathrm{tx}}}{d^{\alpha}} \Leftrightarrow d = \sqrt[\alpha]{\frac{c P_{\mathrm{tx}}}{P_{\mathrm{rcvd}}}}.$$

This is appealing since no additional hardware is necessary and distance estimates can even be derived without additional overhead from communication that is taking place anyway. The disadvantage, however, is that RSSI values are not constant but can heavily oscillate, even when sender and receiver do not move. Here, the signal attenuation along an indirect path, which is higher than along a direct path, can lead to incorrectly assuming a longer distance than what is actually the case. As this is a structural problem, it cannot be combated by repeated measurements. Hence, when using RSSI as a ranging technique, it is necessary to accept and deal with considerable ranging errors or to treat the outcome of the ranging process as a stochastic result to begin with.

Time of arrival

Time of Arrival (ToA) (also sometimes called “time of flight”) exploits the relationship between distance and transmission time when the propagation speed is known. Assuming both sender and receiver know the time when a transmission – for example, a short ultrasound pulse – starts, the time of arrival of this transmission at the receiver can be used to compute propagation time and, thus, distance. To relieve the receiver of this duty, it can return any received “measurement pulse” in a deterministic time; the original sender then only has to measure the round trip time assuming symmetric paths.

Depending on the transmission medium that is used, time of arrival requires very high resolution clocks to produce results of acceptable accuracy. For sound waves, these resolution requirements are modest; they are very hard for radio wave propagation. One disadvantage of sound is that its propagation speed depends on external factors such as temperature or humidity – careful calibration is necessary but not obvious.

Time difference of arrival

To overcome the need for explicit synchronization, the Time Difference of Arrival (TDoA) method utilizes implicit synchronization by directly providing the start of transmission information to the receiver. This can be done if two transmission mediums of very different propagation speeds are used – for example, radio waves propagating at the speed of light and ultrasound, with a different in speed of about six orders of magnitude. Hence, when a sender starts an ultrasound and a radio transmission simultaneously, the receiver can use the arrival of the radio transmission to start measuring the time until arrival of the ultrasound transmission, safely ignoring the propagation time of the radio communication.

The obvious disadvantage of this approach is the need for two types of senders and receivers on each node. The advantage, on the other hand, is a considerably better accuracy compared to RSSI-based approaches.

Determining angles

As an alternative to measuring distances between nodes, angles can be measured. Such an angle can either be the angle of a connecting line between an anchor and a position-unaware node to a

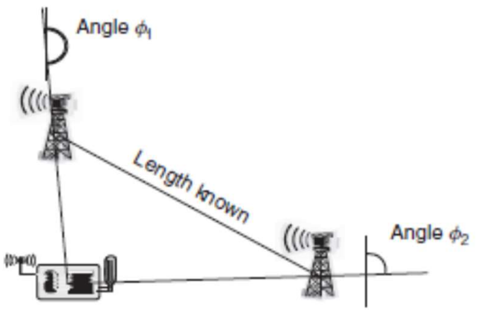


Figure 9.4 Angulation based on two anchors

given reference direction ("0◦ north"). It can also be the angle between two such connecting lines if no reference direction is commonly known to all nodes (Figure 4). A traditional approach to measuring angles is to use directional antennas (antennas that only send to/receive from a given direction), rotating on their axis, similar to a radar station or a conventional lighthouse. This makes angle measurements conceptually simple, but such devices are quite inappropriate for sensors nodes; they can be useful for supporting infrastructure anchors.

Another technique is to exploit the finite propagation speed of all waveforms. With multiple antennas mounted on a device at known separation and measuring the time difference between a signal's arrival at the different antennas, the direction from which a wavefront arrived at the device can be computed. The smaller the antenna separation, the higher the precision of the time differences has to be, which results in strenuous timing requirements given the desirable small size of sensor nodes. Overall, angulation is a less frequently discussed technique compared to lateration.

Scene analysis

A quite different technique is scene analysis. The most evident form of it is to analyze pictures taken by a camera and to try to derive the position from this picture. This requires substantial computational effort and is hardly appropriate for sensor nodes. But apart from visual pictures, other measurable characteristic "fingerprints" of a given location can be used for scene analysis, for example, radio wave propagation patterns. One option is to use signal strength measurements of (one or more anchors) transmitting a known signal strength and compare the actually measured values with those stored in a database of previously off-line measured values for each location – the RADAR system is one example that uses this approach to determine positions in a building. Using other physical characteristics such as multipath behaviour is also conceivable.

While scene analysis is interesting for systems that have a dedicated deployment phase and where off-line measurements are acceptable, this is not always the case for WSNs. Hence, this approach is not the main focus of attention.

Single-hop localization

Using these basic building blocks of distance/range or angle measurements and the mathematical basics, quite a number of positioning or locationing systems have been developed. This section concentrates on systems where a node with unknown position can directly communicate with anchors – if anchors are used at all. The following section contains systems where, for some nodes, multihop communication to anchors is necessary. These single-hop systems usually predate wireless sensor networks but provide much of the basic technology upon which multihop systems are built.

Active Badge

The "Active Badge Location System" is the first system designed and built for locating simple, portable devices – badges – within a building. It uses diffused infrared as transmission medium and exploits the natural limitation of infrared waves by walls as a delimeter for its location granularity. A badge periodically sends a globally unique identifier via infrared to receivers, at least one of which is installed in every room. This mapping of identifiers to receivers (and hence rooms) is stored on a central server, which can be queried for the location of a given badge. It is possible to run additional queries, such as which badge is in the same room as a particular given badge. As soon as badges are directly connected to persons, privacy issues play a crucial role as well.

Active office

After the Active Badge system introduced locating techniques, the positioning of indoor devices. Here, ultrasound is used, with receivers placed at well-known position, mounted in array at the ceiling of a room; devices for which the position is to be determined act as ultrasound senders.

When the position of a specific device shall be determined, a central controller sends a radio message, containing the device's address. The device, upon receiving this radio message, sends out a short ultrasound pulse. This pulse is received by the receiver array that measures the time of arrival and computes the difference between time of arrival of the ultrasound pulse and the time of the radio pulse (neglecting propagation time for the radio wave). Using this time, a distance estimate is computed for every receiver and a multilateration problem is solved (on the central controller), computing a position estimate for the mobile device. Sending the radio pulse is repeated every 200 ms, allowing the mobile devices to sleep for most of the time.

The system also compensates for imprecision in the distance estimates by discarding outliers based on statistical tests. The obtained accuracy is very good, with at least 95% of averaged position estimates lying within 8 cm of the true position. With several senders on a mobile device, the accuracy is even high enough to provide orientation information.

RADAR

The RADAR system is also geared toward indoor computation of position estimates. Its most interesting aspect is its usage of scene analysis techniques, comparing the received signal characteristics from multiple anchors with premeasured and stored characteristic values. Both the anchors and the mobile device can be used to send the signal, which is then measured by the counterpart device(s). While this is an intriguing technique, the necessary off-line deployment phase for measuring the "signal landscape" cannot always be accommodated in practical systems.

Cricket

In the Active Badge and active office systems described above, the infrastructure determines the device positions. Sometimes, it is more convenient if the devices themselves can compute their own positions or locations – for example, when privacy issues become relevant. The "Cricket" system is an example for such a system. It is also based on anchors spread in a building, which provide combined radio wave and ultrasound pulses to allow measuring of the TDoA (signal strength information had been found to be not reproducible enough to work satisfactorily). From this information, symbolic location information within the building is extracted. A simple randomized protocol is used to overcome this obstacle.

Overlapping connectivity

It describe an example for an outdoor positioning system that operates without any numeric range measurements. Instead, it tries to use only the observation of connectivity to a set of anchors to determine a node's position (Figure 5). The underlying assumption is that transmissions (of known and fixed transmission power) from an anchor can be received within a circular area of known radius. Anchor nodes periodically send out transmissions identifying themselves (or, equivalently, containing their positions). Once a node has received these announcements from all anchors of which it is in reach (typically waiting for a few periods to smooth out the effect of random packet losses), it can determine that it is in the intersection of the circles around these anchors. The estimated position is then the arithmetic average of the received anchors' positions. Moreover, assuming that the node knows about all the anchors that are deployed, the fact that some anchor announcements are not received implies that the node is outside the respective circles. This information further allows to restrict the node's possible position.

The achievable absolute accuracy depends on the number of anchors – more anchors allow a finer-grained resolution of the area. At 90% precision, the relative accuracy is one-third the separation distance between two adjacent anchors – assuming that the anchors are arranged in a regular mesh and that the coverage area of each anchor is a perfect circle. In a 10m × 10m area, the average error is 1.83 m; in 90% of the cases, positioning error is less than 3 m.

Accuracy degrades if the real coverage range deviates from a perfect sphere (as it usually does in reality). In addition, the transmission range has to be chosen carefully to result in a minimal positioning error, given a set of anchors.

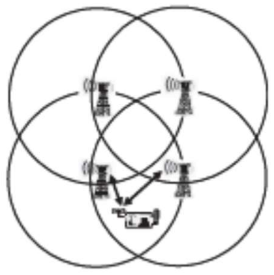


Figure 9.5 Positioning using connectivity information to multiple anchors [106]

Approximate point in triangle

The previous approach has used a range-free connectivity detection to decide whether a node is inside or outside a circle around a given anchor. In fact, more information can be extracted from pure connectivity information. The idea is to decide whether a node is within or outside of a triangle formed. Using this information, a node can intersect the triangles and estimate its own position, similar to the intersection of circles. Figure 6 illustrates the idea. The node has detected that it is inside the triangles BDF, BDE, and CDF and also that it is outside the triangle ADF (and ABF, AFC, and others). Hence, it can estimate its own position to be somewhere within the dark gray area – for example, this area's center of gravity.

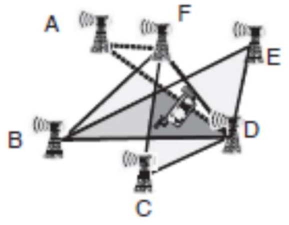


Figure 9.6 Position estimates using overlapping triangles

The interesting question is how to decide whether a node is inside or outside the triangle formed by any three arbitrarily selected anchors. The intuition is to look at what happens when a node inside a triangle is moved: Irrespective of the direction of the movement, the node must be closer to at least one of the corners of the triangle than it was before the movement. Conversely, for a node outside a triangle, there is at least one direction for which the node's distance to all corners increases.

Moving a sensor node to determine its position is hardly practical. But one possibility to approximate movements is for a node to inquire all its neighbors about their distance to the given three corner anchors, compared to the enquiring node's distance. If, for all neighbors, there is at least one corner such that the neighbor is closer to the corner than the enquiring node, it is assumed to be inside the triangle, else outside – this is illustrated in Figure 9.7. Deciding which of two nodes is closer to an anchor can be approximated by comparing their corresponding RSSI values.

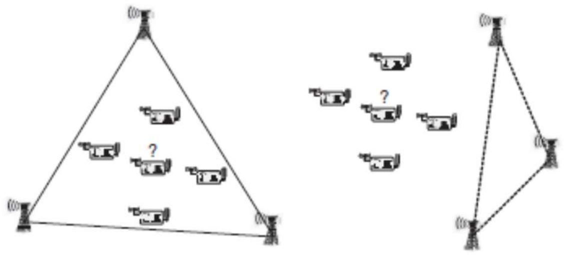


Figure 9.7 Testing whether a node is in a triangle or not using APIT (enquiring node is marked with a "?")

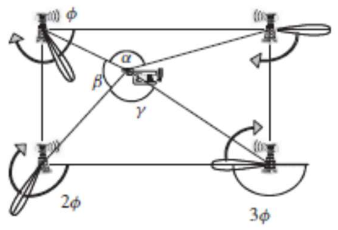


Figure 9.8 Rotating beacons provide angle of arrival information via timing offsets [586]

Both the RSSI comparison and the finite amount of neighbors introduce errors in this decision. For example, for a node close to the edge of the triangle, there is a chance that the next neighbor in the direction toward the edge is already outside the triangle, incorrectly leading the enquiring node to assume this also gives more cases and details. Therefore, the approach is likely to work better in dense networks where the probability of such kinds of errors is reduced. Note that it can still be regarded as a range-free algorithm since only relative signal strength received by two nodes is compared, but no direct relationship is presupposed between RSSI values and distance. Nonetheless, nonmonotonic RSSI behaviour over distance is a source of error for this approach. Because of these potential errors, it is only an Approximate Point in Triangle (APIT) test.

Using angle of arrival information

One example method to obtain angular information in a sensor network is described. They use anchors nodes that use narrow, rotating beams where the rotation speed is constant and known to all nodes. Nodes can then measure the time of arrival of each such beam, compute the differences between two consecutive signals, and determine the angles α, β, and γ from Figure 8 using straightforward geometric relationships. The challenge here is mainly to ensure that the beams are narrow enough (less than 15◦ are recommended) so that nodes have a clear triggering point for the time measurements and to handle effects of multipath propagation. An advantage of this approach is that it is unaffected by the network density and causes no traffic in the network; the sensor nodes themselves can remain quite simple. In simulations, excellent accuracy is reported, limiting the positioning error to about 2m in a 75m × 75m area.

Positioning in multihop environments

All the approaches and concepts that a node trying to determine its position can directly communicate with – in general – several anchor nodes. This assumption is not always true in a wireless sensor network – not every node is in direct contact with at least three anchors. Mechanisms are necessary that can somehow cope with the limited geographic availability of (relatively) precise ranging or position information. Such mechanisms and approaches are described here. In some form or another, they rest upon the fact that for a sufficiently connected graph with known length of the edges, it is possible to reconstruct its embedding in the plane (or in three-dimensional space).

Connectivity in a multihop network

A semidefinite program feasibility formulation A first approach to the multihop positioning problem is (predominantly) based upon connectivity information and considers the position determination as a feasibility problem. Assume that the positions of n anchors are known and the positions of m nodes is to be determined, that connectivity between any two nodes is only possible if nodes are at most R distance units apart, and that the connectivity between any two nodes is also known. The fact that two nodes are connected introduces a constraint to the feasibility problem – for two connected nodes, it is impossible to choose positions that would place them further than R away. For a single node, multiple such constraints can exist that have to be satisfied concurrently – akin to the overlapping circles from above, which restrict the possible positions of a node.

On the basis of this formulation, the feasibility problem as a semidefinite program (a generalization of linear programs). This problem can be solved, but only centrally, requiring all connectivity information at one point. The main observation here is that in this formulation, the fact that two nodes are not connected does not provide any additional information – it is impossible to write down a constraint that two nodes are at least a given distance away from each other in a semidefinite program; nodes cannot be "pushed apart". As an example consequence, a linear chain of nodes with only one anchor in it cannot be distinguished from a situation where all nodes are clustered around the anchor. This implies that anchor nodes should preferably be placed at the borders of the network, to impose as many "pull apart" constraints as possible. Such controlled placement considerably reduces the average positioning error compared to random anchor placements.

It also discuss variations and extensions of this basic idea, in particular, using estimates of the actual distance instead of only the upper bound derived from connectivity; angular information instead of ranges (inspired by directed, optical communication); and computing bounds on the position error by solving multiple feasibility problems. Because of its essentially centralized character, however, this concept is only of limited applicability to WSNs.

Multidimensional scaling

The same basic problem of range-free, connectivity-based locationing is solved using the mathematical formalism of Multi-Dimensional Scaling (MDS). On the basis of connectivity between nodes, an all-pair shortest path algorithm roughly estimates positions of nodes. This initial estimate is improved by MDS, and if nodes with absolute position information are available, the resulting coordinates are properly normalized.

The details of this mathematical technique are somewhat involved. The main advantage to this approach is that it is fairly stable with respect to anchor placement, achieving good results even if only few anchors are available or placed, for example, inside the network. It show, in addition, that MDS is also suitable for anisotropic networks (networks where the distance between neighbors is not uniform over the extent of the network).

Multihop range estimation

The basic multilateration approach requires a node to have range estimates to at least three anchors to allow it to estimate its own position. consider the problem when anchors are not able to provide such range estimates to all nodes in the network, but only to their direct neighbors (because of, for example, limits on the transmission power). The idea is to use indirect range estimation by multihop communication to be able to reuse the well-known multilateration algorithm.

To compute range estimates between a node and a far-off anchor via multiple intermediate hops, describe three different possibilities. All of them are based on flooding the network with information, independently starting from each anchor, similar to the operation of a distance vector (DV) routing protocol. The simplest possibility is the "DV-Hop" method. The idea is to count the number of hops (along the shortest path) between any two anchors and to use it to estimate the average length of a single hop by dividing the sum of the distances to other anchors by the sum of the hop counts. Every anchor computes this estimated hop length and propagates it into the network. A node with unknown position can then use this estimated hop length (and the known number of hops to other anchors) to compute a multihop range estimate and perform multilateration. Note that this is, in fact, a range-free approach as there is no need to estimate internode distances. When range estimates between neighboring nodes are available, they can be directly used in the same framework, resulting in the "DV-Distance" method.

In presence of range estimates and a sufficient number of neighbors, a node can actually try to compute its true Euclidean distance to a faraway anchor. Figure 9 illustrates the idea: Assuming that the distances AB, AC, BC, XB, XC are all known, it is possible to compute the unknown distance XA (actually, there are two solutions, one where X is on the other side of the line BC – node X can potentially distinguish these two solutions based on local information). This way, actual positions can be forwarded between nodes.

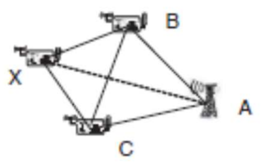


Figure 9.9 Euclidean distance estimation in the absence of direct connectivity [597]

The obtainable accuracy here depends on the ratio of anchors relative to the total number of nodes. The "Euclidean" method increases accuracy as the number of anchors goes up; the "distance vector"-like methods are better suited for a low ratio of anchors. As one would expect, the distance vector methods perform less well in anisotropic networks than in uniformly distributed networks; the Euclidean method, on the other hand, is not very sensitive to this effect.

Iterative and collaborative multilateration

The previous approach tried to estimate distances between nodes with unknown position and the anchors in order to apply multilateration with the anchors themselves. An alternative approach is to use normal nodes, once they have estimated their positions, just like anchor nodes in a multilateration algorithm. Figure 10 shows an example: Nodes A, B, and C are unaware of their position. Node A can triangulate its own position using three anchors. Once node A has a position estimate, node B can use it and two anchors for its own estimate, in turn providing node C with the missing information for its own triangulation.

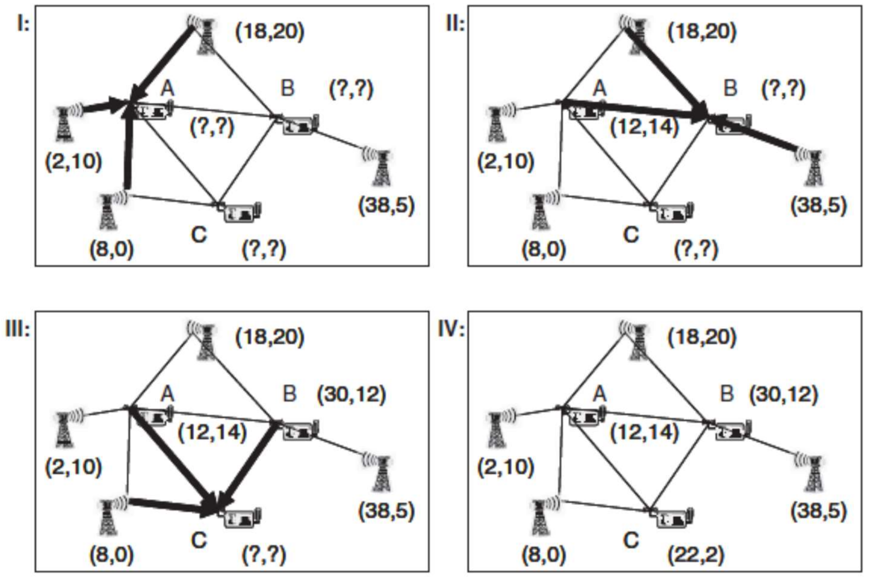


Figure 9.10 Iterative multilateration: Nodes A, B, and C can determine their positions in several iterations (bold arrows indicate the links whose range estimates are used in a given iteration)

A centralized implementation is fairly trivial, typically starting with the as-yet-undetermined node that has the most connections to anchors/nodes with already-determined position estimates and iteratively computing more position estimates. In a distributed implementation, nodes can compute a position estimate once at least three neighbors can provide position information, resulting in an initial estimate of a node's position. When more information becomes available – for example, because more neighbors have estimated their own position – it is possible to use it to improve the position estimate and propagate an updated estimate to a node's neighbors. The hope is that this algorithm will converge to the correct set of positions for all nodes. It should be pointed out that the initial position estimates for such an iterative refinement can also be computed by other means, for example, the DV-hop or DV-distance approaches.

The average position error after such an iterative refinement depends on the accuracy of the range estimation, the initial position estimate, the average number of neighbors, and on the number of anchors. Also, it is not guaranteed that the refinement algorithm converges at all; there can be situations where the position error increases the longer the algorithm runs. In fact, the straightforward refinement algorithm does not result in acceptable performance. This is mainly due to error propagation through the entire network. It add confidence weights to all position estimates and to solve a modified weighted optimization problem, resulting in the convergence of almost all scenarios.

Figure 11 illustrates two problematic cases. The scenario on the left side is still fully determined as sufficient information is available to solve the equation system for the two nodes with unknown position. For the right scenario, there are two solutions (X and X) for position of node X, which cannot be distinguished, but the position of the other unknown node can still be determined.

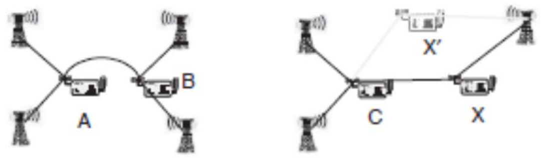


Figure 9.11 Problematic scenarios for iterative multilateration (adapted from [725])

Another approach these problematic cases by defining “participating nodes” – nodes that have at least three anchors or other participating nodes as neighbors, making nodes A and B in Figure 11 participating nodes. For such participating nodes, positioning can be solved.

The crucial observation is that a node, in order to determine its position, needs at least three independent references to anchor nodes – the paths to the anchors have to be edge-disjoint. Such nodes are called sound. In Figure 11, nodes A, B, and C are all sound. Soundness can be detected during the initial position estimation, for example, by recording over which neighbor the shortest path to a given anchor extends. If three or more such paths are detected, the node declares itself sound and enters the refinement phase. Node X from Figure 11, for example, will not be able to declare itself sound. As a consequence, the “soundness” procedure will be able to locate more nodes than the participating node concept.

4. SENSOR TASKING AND CONTROL

Because of Limited battery power and Limited bandwidth careful tasking and the control id needed. Information collected from the sensors.

- All information aggregation is needed.
- Selective information aggregation is needed.

Which sensor nodes to activate and what information to transmit is a critical issue. Classical algorithms are not suitable for WSN :

- Sense values are not known.
- Cost of sensing may vary with the data.

Designing strategy for Sensor Tasking and Ctrl:

1) What are the important object in the environment to be sensed?

2) What parameters of these object are relevant?

3) What relations among these objects are critical to whatever high level information we need to know?

4) Which is the best sensor to acquire a particular parameter?

5) How many sensing and the communication operations will be needed to accomplish the task?

6) How coordinated do the world models of the different sensor need to be?

7) At what level do we communicate information in a spectrum from a signal to symbol?

Roles of Sensor nodes and utilities: A sensor may take on a particular role depending on the application task requirement and resource availability such as node power levels.

Example:

- Nodes, denoted by SR, may participate in both sensing and routing.
- Nodes, denoted by S, may perform sensing only and transmit their data to other nodes.
- Nodes, denoted by R, may decide to act only as routing nodes, especially if their energy reserved is limited. Nodes, denoted by I, may be in idle or sleep mode, to preserve energy.

CONCLUSION:

Topology control – namely, power control, backbones, and clustering – is a powerful means to change the appearance and properties of a network for other protocol layers: MAC layers see reduced contention, routing protocols work on a different graph, changes in neighborhood relationships can be hidden. Judicious use of topology control can significantly improve operational aspects of a network, such as lifetime. However, determining an optimal topology is usually prohibitively.

In all wireless networks, the major problem for synchronization protocols is the variance in the send time, access time, propagation time, and the receive time. Elimination or the ability to accurately predict any of these greatly increases the effectiveness of the synchronization protocol. Discussion on RBS, TPSN, and FTSP was provided with each protocol's advantages. These protocols were designed with performance in mind and did not take into account for security. It was shown that synchronization attacks on all these protocols were possible. Authentication, redundancy, and the refusal to pass on corrupt timing information were the countermeasure discussed.

Determining positions – and, to a lesser degree, also locations – in a wireless sensor network is burdened with considerable overhead and the danger of inaccuracies and imprecision. A nonnegligible amount of anchors is necessary for global coordinate systems, and the time and message overhead necessary to compute positions if no direct communication between anchors and nodes is available should not be underestimated. Nonetheless, it is possible to derive out of erroneous measurements an often satisfactory degree of position estimates.

We have developed a number of important ideas for allocation the sensing, processing, communication, and other application tasks. Some key themes emerge from these discussions: Central to these ideas is the notion of information utility, and the associated costs of acquiring the information. The information utility measures can take on many different forms. However, inappropriate uses of information utility may consume intolerable amounts of resources and diminish the benefit. We must rethink the role of routing: Routing in a sensor network often does not just perform a pure information transport. It must be co-optimized with the information aggregation or dissemination. A group of sensors collectively support a set of tasks. The challenge is to efficiently create, maintain, and migrate groups.

Notes on
Sensor Network Platforms and Tools
Unit V

Dr. G. Senthil Kumar,
Associate Professor,
Dept. of ECE, SCSVMV,
email: gsk_ece@kanchiuniv.ac.in

==

OBJECTIVES:

A real world sensor network application most likely has to incorporate all these elements, subject to energy, bandwidth, computation, storage, and real-time constraints. An end user may view a sensor network as a pool of data and interact with the network via queries. At the same time be structured enough to al low efficient execution on the distributed platform. An application developer must provide end users of a sensor network with the capabilities of data acquisition, processing, and storage. Unlike general distributed or database. Systems, collaborative signal and information processing (CSIP) software comprises reactive, concurrent, distributed programs.

CONTENTS:

INTRODUCTION

A real-world sensor network application is likely to incorporate all the functionalities like sensing and estimation, networking, infrastructure services, sensor tasking, data storage and query. This makes sensor network application development quite different from traditional distributed system development or database programming. With ad hoc deployment and frequently changing network topology, a sensor network application can hardly assume an always-on infrastructure that provides reliable services such as optimal routing, global directories, or service discovery.

There are two types of programming for sensor networks, those carried out by end users and those performed by application developers. An end user may view a sensor network as a pool of data and interact with the network via queries. Just as with query languages for database systems like SQL, a good sensor network programming language should be expressive enough to encode application logic at a high level of abstraction, and at the same time be structured enough to allow efficient execution on the distributed platform. On the other hand, an application developer must provide end users a sensor network with the capabilities of data acquisition, processing, and storage. Unlike general distributed or database systems, collaborative signal and information processing (CSIP) software comprise reactive, concurrent, distributed programs running on ad hoc resource- constrained, unreliable computation and communication platforms. For example, signals are noisy, events can happen at the same time, communication and computation take time, communications may be unreliable, battery life is limited, and so on.

In previous chapters, we discussed various aspects of sensor networks, including sensing and estimation, networking, infrastructure services, sensor tasking, and data storage and query. A real-world sensor network application most likely has to incorporate all these elements, subject to energy, bandwidth, computation, storage, and real-time constraints. This makes sensor network application development quite different from traditional distributed system development or database programming. With ad hoc deployment and frequently changing network topology, a sensor network application can hardly assume an always-on infrastructure that provides reliable services such as optimal routing, etc.

1. SENSOR NODE HARDWARE

Sensor node hardware can be grouped into three categories, each of which entails a different trade-offs in the design choices.

• Augmented general-purpose computers: Examples include low-power PCs, embedded PCs (e.g. PC104), custom-designed PCs, (e.g. Sensoria WINS NG nodes), and various personal digital assistants (PDA). These nodes typically run –ff-the-shelf operating systems such as WinCE, Linux, or real-time operating systems and use standard wireless communication protocols such as IEEE 802.11, Bluetooth, Zigbee etc. Because of their relatively higher processing capability, they can accommodate wide variety of sensors, ranging from simple microphones to more sophisticated video cameras.

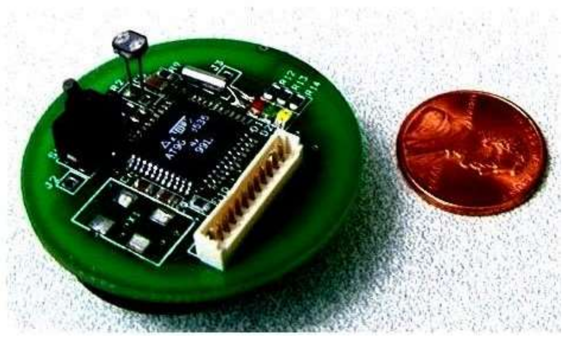
• Dedicated embedded sensor nodes: Examples include the Berkeley mote family, the UCLA Medusa family, Ember nodes and MIT MicroAMP. These platforms typically use commercial off-the-shelf (COTS) chip sets with emphasis on small form factor, low power processing and communication, and simple sensor interfaces. Because of their COTS CPU, these platforms typically support at least one programming language, such as C. However, in order to keep the program footprint small to accommodate their small memory size, programmers of these platforms are given full access to hardware but rarely any operating system support. A classical example is the TinyOS platform and its companion programming language, nesC.

• System on-chip (SoC) nodes: Examples of SoC hardware include smart dust, the BWRC picoradio node, and the PASTA node. Designers of these platforms try to push the hardware limits by fundamentally rethinking the hardware architecture trade-offs for a sensor node at the chip design level. The goal is to find new ways of integrating CMOS, MEMS, and RF technologies to build extremely low power and small footprint sensor nodes that still provide certain sensing, computation, and communication capabilities. Among these hardware platforms, the Berkeley motes, due to their small form factor, open source software development, and commercial availability, have gained wide popularity in the sensor network research.

BERKELEY MOTES

The Berkeley motes are a family of embedded sensor nodes sharing roughly the same architecture. Let us take the MICA mote as an example. The MICA motes have a two-CPU design. The main microcontroller (MCU), an Atmel ATmega103L, takes care of regular processing. A separate and much less capable coprocessor is only active when the MCU is being reprogrammed. The ATmega103L MCU has integrated 512 KB flash memory and 4 KB of data memory.

Berkeley Motes



Given these small memory sizes, writing software for motes is challenging. Ideally, programmers should be relieved from optimizing code at assembly level to keep code footprint small. However, high-level support and software services are not free. Being able to mix and match only necessary software components to support a particular application is essential to achieving a small footprint. A detailed discussion of the software architecture for motes.Berkeley Mote with processing board, sensing board and AA battery pack. The mote was essentially a small form factor computer with self-contained processing, sensing and power resources. TinyOS provides a set of software components that allows applications to interact with the processor, network transceiver and the sensors.

In addition to the memory inside the MCU, a MICA mote also has a separate 512 KB flash memory unit that can hold data. Since the connection between the MCU and this external memory is via a low-speed serial peripheral interface (SPI) protocol, the external memory is more suited for storing data for later batch processing than for storing programs. The RF communication on MICA motes uses the TR1000 chip set (from RF Monolithics, Inc.) operating at 916 MHz band. With hardware accelerators, it can achieve a maximum of 50 kbps raw data rate. MICA motes implement a 40 kbps transmission rate.

2. PROGRAMMING CHALLENGES

Traditional programming technologies rely on operating systems to provide abstraction for processing, I/O, networking, and user interaction hardware. When applying such a model to programming networked embedded systems, such as sensor networks, the application programmers need to explicitly deal with message passing, event synchronization, interrupt handling, and sensor reading. As a result, an application is typically implemented as a finite state machine (FSM) that covers all extreme cases: unreliable communication channels, long delays, irregular arrival of messages, simultaneous events etc.

For resource-constrained embedded systems with real-time requirements, several mechanisms are used in embedded operating systems to reduce code size, improve response time, and reduce energy consumption. Microkernel technologies modularize the operating system so that only the necessary parts are deployed with the application. Real-time scheduling allocates resources to more urgent tasks so that they can be finished early. Event-driven execution allows the system to fall into low-power sleep mode when no interesting events need to be processed. At the extreme, embedded operating systems tend to expose more hardware controls to the programmers, who now have to directly face device drivers and scheduling algorithms, and optimize code at the assembly level. Although these techniques may work well for small, stand-alone embedded systems, they do not scale up for the programming of sensor networks for two reasons:

• Sensor networks are large-scale distributed systems, where global properties are derivable from program execution in a massive number of distributed nodes. Distributed algorithms themselves are hard to implement, especially when infrastructure support is limited due to the ad hoc formation of the system and constrained power, memory, and bandwidth resources.

• As sensor nodes deeply embed into the physical world, a sensor network should be able to respond to multiple concurrent stimuli at the speed of changes of the physical phenomena of interest.

There no single universal design methodology for all applications. Depending on the specific tasks of a sensor network and the way the sensor nodes are organized, certain methodologies and platforms may be better choices than others. For example, if the network is used for monitoring a small set of phenomena and the sensor nodes are organized in a simple star topology, then a client-server software model would be sufficient. If the network is used for monitoring a large area from a single access point (i.e., the base station), and if user queries can be decoupled into aggregations of sensor readings from a subset of nodes, then a tree structure that is rooted at the base station is a better choice. However, if the phenomena to be monitored are moving targets, as in the target tracking, then neither the simple client-server model nor the tree organization is optimal. More sophisticated design and methodologies and platforms are required.

Design issues:

- Fault –tolerant Communication: Due to the deployment of sensor nodes in an uncontrolled or harsh environment, it is not uncommon for the sensor nodes to become faulty and unreliable.
- Low latency: The events which the framework deals with are urgent which should be recognized immediately by the operator. Therefore, the framework has to detect and notify the events quickly as soon as possible.
- Scalability: A system, whose performance improves after adding hardware, proportionally to the capacity added, is said to be a scalable system. The number of sensor nodes deployed in the sensing area may be in the order of hundreds or thousands, or more.
- Transmission Media: In a multi-hop sensor network, communicating nodes are linked by a wireless medium. The traditional problems associated with a wireless channel (e.g., fading, high error rate) may also affect the operation of the sensor network.
- Coverage Problems: One fundamental problem in wireless sensor networks is the coverage problem, which reflects the quality of service that can be provided by a particular sensor network. The coverage problem is defined from several points of view due to a variety of sensor networks and a wide-range of their applications.

Topology Issues

- Geographic Routing: Geographic routing is a routing principle that relies on geographic position information. It is mainly proposed for wireless networks and based on the idea that the source sends a message to the geographic location of the destination instead of using the network address.
- Sensor Holes: A routing hole consists of a region in the sensor network, where either node are not available or the available nodes cannot participate in the actual routing of the data due to various possible reasons. The task of identifying holes is especially challenging since typical wireless sensor networks consist of lightweight, low-capability nodes that are unaware of their geographic location.

- Coverage Topology: Coverage problem reflects how well an area is monitored or tracked by sensors. The coverage and connectivity problems in sensor networks have received considerable attention in the research community in recent years. This problem can be formulated as a decision problem, whose goal is to determine whether every point in the service area of the sensor network is covered by at least k sensors, where k is a given parameter.

Other Issues

- The major issues that affect the design and performance of a wireless sensor network are as follows:
- Hardware and Operating System for WSN
- Wireless Radio Communication Characteristics
- Medium Access Schemes
- Deployment
- Localization
- Synchronization
- Calibration
- Network Layer
- Transport Layer
- Data Aggregation and Data Dissemination
- Database Centric and Querying
- Architecture
- Programming Models for Sensor Networks
- Middleware

Wireless Sensor Networks (WSNs) consist of small nodes with sensing, computation, and wireless communications capabilities. Many routing, power management, and data dissemination protocols have been specifically designed for WSNs where energy awareness is an essential design issue. As wireless sensor networks are still a young research field, much activity is still ongoing to solve many open issues. As some of the underlying hardware problems, especially with respect to the energy supply and miniaturization, are not yet completely solved, wireless sensor networks are having certain short comings, which are to be solved.

3. NODE LEVEL SOFTWARE PLATFORMS

Most design methodologies for sensor network software are node-centric, where programmers think in terms of how a node should behave in the environment. A node- level platform can be node-centric operating system, which provides hardware and networking abstractions of a sensor node to programmers, or it can be a language platform, which provides a library of components to programmers.

A typical operating system abstracts the hardware platform by providing a set of services for applications, including file management, memory allocation, task scheduling, peripheral device drivers, and networking. For embedded systems, due to their highly specialized applications and limited resources, their operating systems make different trade-offs when providing these services. For example, if there is no file management requirement, then a file system is obviously not needed. If there is no dynamic memory allocation, then memory management can be simplified. If prioritization among tasks is critical, then a more elaborate priority scheduling mechanism may be added.

Operating System: TinyOS

Tiny OS aims at supporting sensor network applications on resource-constrained hardware platforms, such as the Berkeley motes.

To ensure that an application code has an extremely small foot-print, TinyOS chooses to have no file system, supports only static memory allocation, implements a simple task model, and provides minimal device and networking abstractions. Furthermore, it takes a language-based application development approach so that only the necessary parts of the operating system are compiled with the application. To a certain extent, each TinyOS application is built into the operating system.

Like many operating systems, TinyOS organizes components into layers. Intuitively, the lower a layer is, the ‘closer’ it is to the hardware; the higher a layer is, the closer it is to the application. In addition to the layers, TinyOS has a unique component architecture and provides as a library a set of system software components. A components specification is independent of the components implementation. Although most components encapsulate software functionalities, some are just thin wrappers around hardware. An application, typically developed in the nesC language, wires these components together with other application-specific components.

A program executed in TinyOS has two contexts, tasks and events, which provide two sources of concurrency. Tasks are created (also called posted) by components to a task scheduler. The default implementation of the TinyOS scheduler maintains a task queue and invokes tasks according to the order in which they were posted. Thus tasks are deferred computation mechanisms. Tasks always run to completion without pre-empting or being pre-empted by other tasks. Thus tasks are non-pre-emptive. The scheduler invokes a new task from the task queue only when the current task has completed. When no tasks are available in the task queue, the scheduler puts the CPU into the sleep mode to save energy.

The ultimate sources of triggered execution are events from hardware: clock, digital inputs, or other kinds of interrupts. The execution of an interrupt handler is called an event context. The processing of events also runs to completion, but it pre-empts tasks and can be pre-empted by other events. Because there is no pre-emption mechanism among tasks and because events always pre-empt tasks, programmers are required to chop their code, especially the code in the event contexts, into small execution pieces, so that it will not block other tasks for too long.

Another trade-off between non-pre-emptive task execution and program reactiveness is the design of split-phase operations in TinyOS. Similar to the notion of asynchronous method calls in distributed computing, a split-phase operation separates the initiation of a method call from the return of the call. A call to split-phase operation returns immediately, without actually performing the body of the operation. The true execution of the operation is scheduled later; when the execution of the body finishes, the operation notifies the original caller through a separate method call.

In TinyOS, resource contention is typically handled through explicit rejection of concurrent requests. All split-phase operations return Boolean values indicating whether a request to perform the operation is accepted.

In summary, many design decisions in TinyOS are made to ensure that it is extremely lightweight. Using a component architecture that contains all variables inside the components and disallowing dynamic memory allocation reduces the memory management overhead and makes the data memory usage statically analysable. The simple concurrency model allows high concurrency with low thread maintenance overhead. However, the advantage of being lightweight is not without cost. Many hardware idiosyncrasies and complexities of concurrency management are left for the application programmers to handle. Several tools have been developed to give programmers language-level support for improving programming productivity and code robustness.

Imperative Language: nesC

nesC is an extension of C to support and reflect the design of TinyOS. It provides a set of language constructs and restrictions to implement TinyOS components and applications.

A component in nesC has an interface specification and an implementation. To reflect the layered structure of TinyOS, interfaces of a nesC component are classified as provides or uses interfaces. A provides interface is a set of method calls exposed to the upper layers, while a uses interface is a set of method calls hiding the lower layer components. Methods in the interfaces can be grouped and named.

Although they have the same method call semantics, nesC distinguishes the directions of the interface calls between layers as event calls and command calls. An event call is a method call from a lower layer component to a higher layer component, while a command is the opposite.

The separation of interface type definitions from how they are used in the components promotes the reusability of standard interfaces. A component can provide and use the same interface type, so that it can act as a filter interposed between a client and a service. A component may even use or provide the same interface multiple times.

Component Implementation

There are two types of components in nesC, depending on how they are implemented: modules and configurations. Modules are implemented by application code (written in a C-like syntax). Configurations are implemented by connecting interfaces of existing components. nesC also supports the creation of several instances of a component by declaring abstract components with optional parameters. Abstract components are created at compile time in configuration. As TinyOS does not support dynamic memory allocation, all components are statically constructed at compile time. A complete application is always a configuration rather than a module. An application must contain the main module, which links the code to the scheduler at run time. The main has single *StdControl* interface, which is the ultimate source of initialization of all components.

Dataflow-Style Language: *TinyGALS*

Dataflow languages are intuitive for expressing computation on interrelated data units by specifying data dependencies among them. A dataflow diagram has a set of processing units called actors. Actors have ports to receive and produce data, and the directional connections among ports are FIFO queues that mediate the flow of data. Actors in dataflow languages intrinsically capture concurrency in a system, and the FIFO queues give a structured way of decoupling their executions. The execution of an actor is triggered when there are enough input data at the input ports.

Asynchronous event-driven execution can be viewed as a special case of dataflow models, where each actor is triggered by every incoming event. The globally asynchronous and locally synchronous (GALS) mechanism is a way of building event- triggered concurrent execution from thread-unsafe components. TinyGALS is such as language for TinyOS.

One of the key factors that affect component reusability in embedded software is the component composability, especially concurrent composability. In general, when developing a component, a programmer may not anticipate all possible scenarios in which the component may be used. Implementing all access to variables as atomic blocks, incurs too much overhead. At the other extreme, making all variable access unprotected is easy for coding but certainly introduces bugs in concurrent composition. TinyGALS addresses concurrency concerns at the system level, rather than at component level as in nesC. Reactions to concurrent events are managed by a dataflow-style FIFO queue communication.

TinyGALS Programming Model

TinyGALS supports all TinyOS components, including its interfaces and module implementations. All method calls in a component interface are synchronous method calls-that is, the thread of control enters immediately into the callee component from the caller component. An application in TinyGALS is built in two steps:

(1) Constructing asynchronous actors from synchronous components, and

(2) Constructing an application by connecting the asynchronous components through FIFO queues.

An actor in TinyGALS has a set of input ports, a set of output ports, and a set of connected TinyOS components. An actor is constructed by connecting synchronous method calls among TinyOS components.

At the application level, the asynchronous communication of actors is mediated using FIFO queues. Each connection can be parameterized by a queue size. In the current implementation of TinyGALS, events are discarded when the queue is full. However, other mechanisms such as discarding the oldest event can be used.

NODE LEVEL SIMULATORS

Node-level design methodologies are usually associated with simulators that simulate the behaviour of a sensor network on a per-node basis. Using simulation, designers can quickly study the performance (in terms of timing, power, bandwidth, and scalability) of potential algorithms without implementing them on actual hardware and dealing with the vagaries of actual physical phenomena. A node-level simulator typically has the following components:

• ***Sensor node model:*** A node in a simulator acts as a software execution platform, a sensor host, as well as a communication terminal. In order for designers to focus on the application-level code, a node model typically provides or simulates a communication protocol stack, sensor behaviours (e.g., sensing noise), and operating system services. If the nodes are mobile, then the positions and motion properties of the nodes need to be modelled. If energy characteristics are part of the design considerations, then the power consumption of the nodes needs to be modelled.

• ***Communication model:*** Depending on the details of modelling, communication may be captured at different layers. The most elaborate simulators model the communication media at the physical layer, simulating the RF propagation delay and collision of simultaneous transmissions. Alternately, the communication may be simulated at the MAC layer or network layer, using, for example, stochastic processes to represent low-level behaviours.

• ***Physical environment model:*** A key element of the environment within a sensor network operates is the physical phenomenon of interest. The environment can also be simulated at various levels of details. For example, a moving object in the physical world may be abstracted into a point signal source. The motion of the point signal source may be modelled by differential equations or interpolated from a trajectory profile. If the sensor network is passive- that is, it does not impact the behaviour of the environment-then the environment can be simulated separately or can even be stored in data files for sensor nodes to read in. If, in addition to sensing, the network also performs actions that influence the behaviour of the environment, then a more tightly integrated simulation mechanism is required.

• ***Statistics and visualization:*** The simulation results need to be collected for analysis. Since the goal of a simulation is typically to derive global properties from the execution of individual nodes, visualizing global behaviours is extremely important. An ideal visualization tool should allow users to easily observe on demand the spatial distribution and mobility of the nodes, the connectivity among nodes, link qualities, end-to-end communication routes and delays, phenomena and their spatio-temporal dynamics, sensor readings on each node, sensor nodes states, and node lifetime parameters (e.g., battery power).

A sensor network simulator simulates the behaviour of a subset of the sensor nodes with respect to time. Depending on how the time is advanced in the simulation, there are two types of execution models: cycle-driven simulation and discrete-event simulation. A cycle-driven (CD) simulation discretizes the continuous notion of real time into (typically regularly spaced) ticks and simulates the system behaviour at these ticks. At each tick, the physical phenomena are first simulated, and then all nodes are checked to see if they have anything to sense, process, or communicate. Sensing and computation are assumed to be finished before the next tick. Sending a packet is also assumed to be completed by then. However, the packet will not be available for the destination node until next tick. This split-phase communication is a key mechanism to reduce cyclic dependencies that may occur in cycle-driven simulations. Most CD simulators do not allow interdependencies within a single tick.

Unlike cycle-driven simulators, a discrete-vent (DE) simulator assumes that the time is continuous and an event may occur at any time. As event is 2-tuple with a value and a time stamp indicating when the event is supposed to be handled. Components in a DE simulation react to input events and produce output events. In node-level simulators, a component can be a sensor node, and the events can be communication packets; or a component can be software module within and the events can be message passing’s among these nodes. Typically, components are causal, in the sense that if an output event is computed from an input event, then the time stamp of the output should not be earlier than that of the input event. Non-causal components require the simulators to be able to roll back in time, and worse, they may not define a deterministic behaviour of a system. A DE simulator typically requires a global event queue. All events passing between nodes or modules are put in the event queue and sorted according to their chronological order. At each iteration of the simulation, the simulator removes the first event (the one with earliest time stamp) from the queue and triggers the component that reacts to that event.

In terms of timing behaviour, a DE simulator is more accurate than a CD simulator, and as a consequence, DE simulators run slower. The overhead of ordering all events and computation, in addition to the values and time stamps of events, usually dominates the computation time. At an early stage of a design when only the asymptotic behaviours rather than timing properties are of concern, CD simulations usually require less complex components and give faster simulations. This is partly because of the approximate timing behaviours, which make simulation results less comparable from application to application, there is no general CD simulator that fits all sensor network simulation tasks. Many of the simulators are developed for particular applications and exploit application- specific assumptions to gain efficiency.

DE simulations are sometimes considered as good as actual implementations, because of their continuous notion of time and discrete notion of events. There are several open- source or commercial simulators available. One class of these simulators comprises extensions of classical network simulators, such as NS-2, J-Sim (previously known as JavaSim), and GloMoSim/ Qualnet. The focus of these simulators is on network modelling, protocol stacks, and simulation performance. Another class of simulators, sometimes called software-in-the-loop simulators, incorporate the actual node software into the simulation. For this reason, they are typically attached to particular hardware platforms and are less portable. Example include TOSSIM for Berkeley motes and Em* for Linux-based nodes such as Sensoria WINS NG platforms.

The NS-2 Simulator and its Sensor Network Extensions

The simulator NS-2 is an open-source network simulator that was originally designed for wired, IP networks. Extensions have been made to simulate wireless/mobile networks (e.g. 802.11 MAC and TDMA MAC) and more recently sensor networks. While the original NS-2 only supports logical addresses for each node, the wireless/mobile extension of it, introduces the notion of node locations and a simple wireless channel model. This is not a trivial extension, since once the nodes move, the simulator needs to check for each physical layer event whether the destination node is within the communication range. For a large network, this significantly slows down the simulation speed.

There are two widely known efforts to extend NS-2 for simulating sensor networks: SensorSim form UCLA and the NRL sensor network extension from the Navy Research Laboratory. SensorSim also supports hybrid simulation, where some real sensor nodes, running real applications, can be executed together with a simulation. The NRL sensor network extension provides a flexible way of modelling physical phenomena in a discrete event simulator. Physical phenomena are modelled as network nodes which communicate with real nodes through physical layers.

The main functionality of NS-2 is implemented in C++, while the dynamics of the simulation (e.g., time-dependent application characteristics) is controlled by TCL scripts. Basic components in NS-2 are the layers in the protocol stack. They implement the handlers interface, indicating that they handle events. Events are communication packets that are passed between consecutive layers within one node, or between the same layers across nodes.

The key advantage of NS-2 is its rich libraries of protocols for nearly all network layers and for many routing mechanisms. These protocols are modelled in fair detail, so that they closely resemble the actual protocol implementations. Examples include the following:

• TCP: reno, tahoe, vegas, and SACK implementations.

• MAC: 802.3, 802.11, and TDMA.

• Ad-hoc routing: Destination sequenced distance vector (DSDV) routing, dynamic source routing (DSR), ad hoc on-demand distance vector (AOPDV) routing, and temporarily ordered routing algorithm (TORA).

• Sensor network routing: Directed diffusion, geographical routing (GEAR) and geographical adaptive fidelity (GAF) routing.

The Simulator TOSSIM

TOSSIM is a dedicated simulator for TinyOS applications running on one or more Berkeley motes. The key design decisions on building TOSSIM were to make it scalable to a network of potentially thousands of nodes, and to be able to use the actual software code in the simulation. To achieve these goals, TOSSIM takes a cross-compilation approach that compiles the nesC source code into components in the simulation. The event-driven execution model of TinyOS greatly simplifies the design of TOSSIM. By replacing a few low-level components such as the A/D conversion (ADC), the system clock, and the radio front end, TOSSIM translates hardware interrupts into discrete-event simulator events. The simulator event queue delivers the interrupts that drive the execution of a node. The upper-layer TinyOS code runs unchanged.

TOSSIM uses a simple but powerful abstraction to model a wireless network. A network is a directed graph, where each vertex is a sensor node and each directed edge has a bit- error rate. Each node has a private piece of state representing what it hears on the radio channel. By setting connections among the vertices in the graph and a bit-error rate on each connection, wireless channel characteristics, such as imperfect channels, hidden terminal problems, and asymmetric links can be easily modelled. Wireless transmissions are simulated at the bit level. If a bit error occurs, the simulator flips the bit.

TOSSIM has a visualization package called TinyViz, which is a Java application that can connect to TOSSIM simulations. TinyViz also provides mechanisms to control a running simulation by, for example, modifying ADC readings, changing channel properties, and injecting packets. TinyViz is designed as a communication service that interacts with the TOSSIM event queue. The exact visual interface takes the form of plug-ins that can interpret TOSSIM events. Beside the default visual interfaces, users can add application- specific ones easily.

EmStar

The introduction of EmStar and the comparison with other simulation tools will be discussed in this subsection. EmStar is an emulator specifically designed for WSN built in C, and it was first developed by University of California, Los Angeles. EmStar is a trace-driven emulator [Girod04] running in real-time. People can run this emulator on Linux operating system. This emulator supports to develop WSN application on better hardware sensors. Besides libraries, tools and services, an extension of Linux microkernel is included in EmStar emulator.

OMNeT++

The introduction of OMNeT++ and the comparison with other simulation tools will be discussed in this subsection. OMNeT++ is a discrete event network simulator built in C++. OMNeT++ provides both a non-commercial license, used at academic institutions or non-profit research organizations, and a commercial license, used at "for-profit" environments. This simulator supports module programming model. Users can run OMNeT++ simulator on Linux Operating Systems, Unix-like system and Windows. OMNeT++ is a popular non-specific network simulator, which can be used in both wire and wireless area. Most of frameworks and simulation models in OMNeT++ are open sources.

J-Sim

J-Sim is a discrete event network simulator built in Java. This simulator provides GUI library, which facilities users to model or compile the Mathematical Modeling Language, a “text-based language” written to J-Sim models.

4. STATE CENTRIC PROGRAMMING

Many sensor network applications, such as target tracking, are not simply generic distributed programs over an ad hoc network of energy-constrained nodes. Deeply rooted in these applications is the notion of states of physical phenomena and models of their evolution over space and time. Some of these states may be represented on a small number of nodes and evolve over time, as in the target tracking problem, while others may be represented over a large and spatially distributed number of nodes, as in tracking a temperature contour.

A distinctive property of physical states, such as location, shape, and motion of objects, is their continuity in space and time. Their sensing and control is typically done through sequential state updates. System theories, the basis for most signal and information processing algorithms, provide abstractions for state updates, such as:

$$xk+1 = f(xk, uk)$$

$$yk = g(xk, uk)$$

where x is the state of a system, u is the system input, y is the output and k is an integer update index over space and/or time, f is the state update function, and g is the output or observation function. This formulation is broad enough to capture a wide variety of algorithms in sensor fusion, signal processing, and control (e.g., Kalman filtering, Bayesian estimation, system identification, feedback control laws, and finite-state automata).

However, in distributed real-time embedded systems such as sensor networks, the formulation is not as clean as represented in the above equations. The relationships among subsystems can be highly complex and dynamic over space and time. The following issues (which are not explicitly tackled in the above equations) must be properly addressed during the design to ensure the correctness and efficiency of the system.

• Where are the state variables stored?

• Where do the inputs come from?

• Where do the outputs go?

• Where are the functions f and g evaluated?

• How long does the acquisition of input take?

• Are the inputs in uk collected synchronously?

• Do the inputs arrive in the correct order through communication?

• What is the time duration between indices *k* and *k +1*? Is it a constant?

These issues, addressing where and when, rather than how, to perform sensing, computation, and communication, play a central role in the overall system performance. However, these ‘non-functional” aspects of computation, related to concurrency, responsiveness, networking, and resource management, are not well supported by traditional programming models and languages. State-centric programming aims at providing design methodologies and frameworks that give meaningful abstractions for these issues, so that system designers can continue to write algorithms on top of an intuitive understanding of where and when the operations are performed.

A collaborative group is such an abstraction. A collaborative group is a set of entities that contribute to a state update. These entities can be physical sensor nodes, or they can be more abstract system components such as virtual sensors or mobile agents hopping among sensors. These are all referred to as agents.

Intuitively, a collaboration groups provides two abstractions: its scope to encapsulate network topologies and its structure to encapsulate communication protocols. The scope of a group defines the membership of the nodes with respect to the group. A software agent that hops among the sensor nodes to track a target is a virtual node, while a real node is physical sensor. Limiting the scope of a group to a subset of the entire space of all agents improves scalability. Grouping nodes according to some physical attributes rather than node addresses is an important and distinguishing characteristic of sensor networks.

The structure of a group defines the “roles” each member plays in the group, and thus the flow of data. Are all members in the group equal peers? Is there a “leader” member in the group that consumes data? Do members in the group form a tree with parent and children relations? For example, a group may have a leader node that collects certain sensor readings from all followers. By mapping the leader and the followers onto concrete sensor nodes, one can effectively define the flow of data from the hosts of followers to the host of the leader. The notion of roles also shields programmers from addressing individual nodes either by name or address. Furthermore, having multiple members with the same role provides some degree of redundancy and improves robustness of the application in the presence of node and link failures.

PIECES: A State –Centric Design Framework

PIECES (Programming and Interaction Environment for Collaborative Embedded Systems) is a software framework that implements the methodology of state-centric programming over collaboration groups to support the modelling, simulation, and design of sensor network applications. It is implemented in a mixed Java-Matlab environment.

PIECES comprises principals and port agents. A principal is the key component for maintaining a piece of state. Typically, a principal maintains state corresponding to certain aspects of the physical phenomenon of interest. The role of a principal is to update its state from time to time, a computation corresponding to evaluation function *f*. A principal also accepts other principals' queries of certain views on its own state, a computation corresponding to evaluating function *g*.

To update its portion of the state, a principal may gather information from other principals. To achieve this, a principal creates port agents and attaches them onto itself and onto the other principals. A port agent may be an input, an output, or both. An output port agent is also called an observer, sine it computes outputs based on the host principal's state and sends them to their agents. Observers may be active and passive. An active observer pushes data autonomously to its destination (s0, while a passive observer sends data only when a consumer requests for it. A principal typically attaches a set of observers to other principals and creates a local input port agent to receive the information collected by the remote agents. Thus port agents capture communication patterns among principals.

The execution of principals and port agents can be either time-driven or event-driven, where events may include physical events that are pushed to them (i.e., data-driven) or query events from other principals or agents (i.e., demand-driven). Principals maintain state, reflecting the physical phenomena. These states can be updated, rather than rediscovered, because the underlying physical states are typically continuous in time. How often the principal states need to be updated depends on the dynamics of the phenomena or physical events. The executions of observers, however, reflect the demands of the outputs. If an output is not currently needed, there is no need to compute it. The notion of "state" effectively separates these two execution flows.

To ensure consistency of state update over a distributed computational platform, PIECES requires that a piece of state, say x|s, can only be maintained by exactly one principal. Note that this does not prevent other principals from having local caches of x|s for efficiency and performance reasons; nor does it prevent the other principals from locally updating the values of cached x|s. However, there is only one master copy for x|s. All local updates should be treated as “suggestion” to the master copy, and only the principal that owns x|s has the final word on its values. This asymmetric access of variables simplifies the way shared variables are managed.

Most sensor network interfaces only provide primitives to discover communication peers and to send and receive messages in a best-effort manner. Reliable communications are implemented at the application level and must be backed up by fault-tolerant plans to cope with communication failures. As a result, applications are typically constructed as parallel finite-state machines running on each individual node to anticipate every possible combination of concurrent sensing and communication events. The system’s global behaviours are the result of these local FSMs’ interactions. On the other hand, declarative interfaces such as SQL for databases provide a hardware independent abstraction. However, CSIP application developers must still write collaborative processing programs to support the high-level declarative queries.

Thus, state-centric, agent-based design methodology is described a to mediate between a system developer's mental model of physical phenomena and the distributed execution of DSAN applications. Building on the ideas of data-centric networking, sensor databases, and proximity-based group formation, we introduce the notion of collaboration groups, which abstracts common patterns in application-specific communication and resource allocation. An application developer specifies computations as the creation, aggregation, and transformation of states, which naturally map to the vocabulary used by signal processing and control engineers. More specifically, programmers write applications as algorithms for state update and retrieval, with input supplied by dynamically created collaboration groups. As a result, programs written in the state-centric framework are more invariant to system configuration changes, making the resulting software more modular and portable across multiple platforms. Using a distributed tracking application with sensor networks, we'll demonstrate how state-centric programming can raise the abstraction level for application developers.

CONCLUSION:

Most current sensor-networking deployments include square-inch-size generic sensor devices that represent an interconnected mesh tied to the Internet through one or more gateway-class devices. More advanced networks include high-bandwidth sensor nodes capable of dealing with complex data streams, including voice and video. Alternatively, they may include tiny special-purpose sensor nodes that are just millimeters on a side and weigh only a few grams each. While the capabilities, cost, and size of each class of device will change with technological advances, these four fundamental classes of device will likely remain for the foreseeable future.

Integral to the performance of sensor-network nodes is the software supporting them. Sensor-network applications require precision control over the underlying hardware in order to meet the strict power limitations they must satisfy. Current development efforts focus on two software platforms for use in wireless sensor networks. TinyOS provides the precise, efficient, low-level control demanded by both general and special-purpose networking nodes. In contrast, special kernel modifications have been added to Linux to enable it to support gateway and high-bandwidth class device operation. Combined with the hardware platforms, TinyOS and embedded Linux are together being shaped into a powerful toolbox for building wireless sensor-network applications