ALBERT-LUDWIGS-UNIVERSITÄT

FREIBURG

INSTITUT FÜR INFORMATIK

Department of Computer Networks and

Telematics

Master's Thesis

Implementation of a Massively Parallel Wireless Sensor Network

Atef Abdel-Rahman

Faisal Aslam

Prof. Dr. Schindelhauer

June 30, 2007

Acknowledgment

First, I would like to thank Professor Christian Schindelhauer for giving me the opportunity to work on this interesting project and for the help and guidance that he has extended to me during this work. Further acknowledgement go to my supervisor Faisal Islam for helping me with my questions. In addition I would like to thank all my colleagues in institute für Informatik at the university Freiburg. Thanks to all the people who supported me while to finishing this thesis.

Abstract

In this work wireless sensor networks consisting of a very large number of sensor nodes are considered. Furthermore, it is assumed that the network is to be deployed at a remote region and works for a long duration without any human intervention. During the experiments carried out useful experimental (log) data is collected. This log data has to be communicated to a central computer or base station so that it could be analyzed and used by humans. The collecting data during the experiment saves power and network bandwidth. It is desirable to store data temporarily in the memory of the sensor nodes during the experiment and collect it after the end of experiment or after a fixed time period. Our objective is to show such an offline strategy to get the log data from the sensor nodes perserves energy. This strategy requires little memory, computation and human intervention. Furthermore, it is independent from the type, the number of sensor nodes used as well as the kind of sensing data gathered. The logging application has no dependency on data collected. It should be dynamic and change the provided display depending upon data available. Since a sensor has only small memory, memory reserved for logging. The logging application uses priority based log overwriting when the amount of log data collected exceeds memory reserved for data logging. The time complexity of the used strategy is $O(log(n))$, where n is the number of fixed sized log blocks called entry in the memory. A priority based linked list is used to store the log data. Each element of this linked list contains a FIFO queue having log entries corresponding to priority of linked list elements. A new log element is added at the end of the corresponding FIFO queue with right priority. To dynamically displayed data, the Extensible Markup Language (XML) and Extensible Stylesheet Language (XSL) are used. With XML and XSL dynamic HTML is generated. The XML format definded is independent from the log data collected. XSL makes sure to display the data without consideration its size. In summary, the logging application collects log data independent of the developed sensor hardware and the type of experiments carried out using sensor memory and computational resources efficiently.

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

`Ort, Datum` Unterschrift

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless sensor networks(WSNs) have become the most recent exciting and pervasive technology in telecommunications. Rapid improvement and recent advances in wireless communications, digital electronic technologies like ASIC design, micro-electro-mechanical system(MEMS) and mobile computing enabled sensor nodes to be deployed with the target to access information and measurements of physical phenomena anywhere and anytime. WSNs are capable to revolutionize many areas of our information technology and by this— even our information society. Joining sensors together into wireless sensor networks opens up the door for new branches of research and applications. In general a sensor is a device that responds to a physical stimulus, such as heat, light or pressure. It generates a signal that can be measured or interpreted. A WSN is defined as a network, which consists of individual nodes that are able to interact with the environment by sensing or controlling physical parameters. These nodes have to collaborate to fulfill their task using wireless communication to enable this collaboration. Wired networks need to be maintained, which is accompained by higher costs of ownership. Wiring prevents entities from being mobile and is not suitable for a range of environmental, health, home, commercial and military applications. WSNs suffer from some limitations such as energy consumption, its optimization, the accuracy of the delivered measured data, size and cost of a node as well as the capacity of its onboard energy supply. Many applications do not suffer from these limitations and will become an integral part of our lives.

1.1 Tools Used in this Thesis

The tools used in this work are as follows:

- **Hardware:**
  The ScatterWeb sensor nodes have been developed by the computer sys-

tem and telematics group at Freie Universität Berlin(FU-Berlin) [23]. Chapter 3 shows the components of this sensor nodes in details.

- **Software:**
  The ScatterWeb software of sensor nodes allows to initiate, track and schedule experiments. It allows the reprogramming of the sensor nodes and reading of sensor data which is generated during the experiment.
- **Java Technology**
- **XML Technology**

The software of sensor nodes is separated into two parts: firmware and application [25] [26].

- **The Firmware** provides abstract functions to interface and use the hardware. It contains the main execution loop, interrupts and provides some OS-like concepts. The programm System.c contains main function and the main OS loop. It contains some very fundamental functions which interact with hardware. Data.c consists mostly of the functions that constitute the sensors architecture. It allows building components that efficiently manage data from multiple data sources. Time.c represents an instant in time, typically expressed as a date and a time of day. It contains methods to read, write and convert current time. Net.c implements a network functions which allows to develop applications using network resources. In String.c the usual text manipulation functions can be found.
- **Application** is built on top of the firmware. Function in Process.c allows to initialize and start the application. Event.c contains event related function to different sensors in a sensor node. For example: vibration, movement, button-press, temperature and battery usage. These functions are used to decide what should be done with the events: e.g.: broadcast, log or print.

1.2 Goal of this Thesis

The goal of this work is to develop a system to gather log data from the sensor nodes, process and represent it. First, the system should save file for log data. The log file contains sensor's ID, time, date, name. Second, the system should allows to overwrite older information with new information. Third, the system allows to attach priorities to the log data. Fourth, information is collected after the experiment and saved in well-defined XML foramt. Finally, the system enable to generate dynamically HTML from XML.

Chapter 2

Related Work

2.1 Sensor Network Applications

Application areas of WSN according to [2], [22] and [43] are:

- Disaster relief applications (e.g. forest fire monitioring),
- environmental monitoring (e.g.: high temperature, wind or waterlevel),
- machine surveillance and preventive maintenance (e.g.: to detect vibration patterns indicating need for maintenance),
- precision agriculture (e.g.: for percise irrigation and fertilizing),
- intelligent building (e.g. buildings consuming less energy),
- telematics (e.g. embedded sensors in the streets to gather information about traffic conditions),
- medicine and health care (e.g. for integrated patient monitoring),
- military defense network (e.g. WSNs can be used for detection of biological, nuclear or chemical attacks or to track a passing vehicle in a region.),
- safety (e.g. for workers and engineers working in secure area),
- inventory and logistics management (e.g. attaching a sensor node to items in warehouses to view the exact number and location of the items),
- process control (e.g. controlling product conditions) and
- civilian (e.g. highway traffic monitoring).

Many of the listed applications above need to distinguish between:

- **Source:** A sensor node that collects the data.
- **Sink:** A sensor node, where the data and the information is received.
- **Intermediate node:** A sensor node, which adds additional data processing or simply forwards the data.

The interaction between these types of sensor nodes has to fulfill some purpose such as:

- **Event detection:** A sensor node, which is deployed as a source should report to the sink when required data is detected.
- **Periodic measurements:** A sensor nodes can be tasked to report measured values periodically in a defined interval.
- **Function approximation and edge detection:** Physical value like temperature can be mapped to a region, where every region has a different temperature. These values within WSN can be approximated according to the different regions.
- **Tracking:** When the source of the event is mobile, a WSN can be used to get information like temperature or velocity and report the event source, location or behavior.

2.2 Constraints and Challenges

A sensor is a device that measures or detects a real-world condition, such as vibration, heat or light and converts its measurements from an analog to a digital representation. It can be connected by a wire to a power supply or it has to rely on its on-board batteries. A sensor node is defined as a basic unit in a sensor network, with on-board sensors, processor, memory, wireless modem, and power supply. When it has one sensor on board, it is often simply referred to as sensor, but when it has more than one sensor on board, it is referred to as a sensor node [15]. A sensor network consists of a large number of cooperating small-scale sensor nodes, which are densely deployed either inside their environment or very close to it. Information gathered by a sensor network describes measurements of physical parameters like temperature, humidity and vibration. The most important constraints on sensor nodes are the finite on-board battery power and its limited bandwidth for communication. Each system has its special characteristics and required mechanisms. The major challenges for WSNs are the realization of such characteristics and mechanisms.

2.2.1 Characteristic Requirements

The following characteristics are typical for many applications [19][22][43][59].

- **Type of service:** The WSN is amenable to support a lot of various applications and there is no single set of requirements that can obviously classify all WSNs. A WSN should offer the user meaningful information about the object of interest and therefor the type of the service handeled by a WSN is dependent on the supported application.
- **Fault tolerance:** Since the nodes may be damaged or blocked due to the lack of power, the failure of sensor nodes should not influence the whole task of a WSN. Fault tolerance is the ability to validate sensor networks functionalities without any interruption due to sensor node failures. Referring to [43], the fault tolerance of a sensor node according to [19] is defined using the Poisson distribution to get the probability of not having a failure within the time interval $(0; t)$:

  $$R_k(t) = e^{-\lambda_k} t$$

  where λ_k is the failure rate of sensor node k and t is the time period.
- **Scalability:** Scalability is refered to as the skill to hold required performance regardless of the size of the network. Depending on the application, the number of sensor nodes in a WSN may be hundreds or thousands. The density can range from a few nodes to hundred nodes in a region, which can be less than 10m in diameter. Referring to [43], the density can be calculated according to [52] as

  $$\mu(R) = (n\pi R^2)/A$$

  where $\mu(R)$ is the number of nodes within the transmission radius of each node in region A, n is the number of scattered sensor nodes in region A and R is the radio transmission range.
- **Quality of service(QoS):** In the application communities, QoS usually refers to the quality as perceived by the user or application while in the networking community, QoS is accepted as a measure of the service quality that the network offers to the applications or to the user [11]. Figure 2.1 shows the interaction between the WSN and the application. The level of QoS may be definied by a set of elements such as: delay, bandwidth and packet loss.
- **Wide range of densities:** The node density can be defined as the number of nodes in a unit region. The node density depends on the application in which the sensor nodes are deployed.

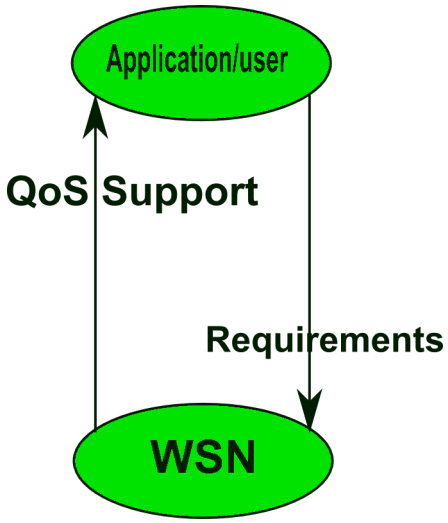

Figure 2.1: Simple QoS Model.

- **Lifetime:** A critical aspect of applications with wireless sensor networks is the lifetime of the network. Battery-powered sensors are usable as long as they can communicate captured data to a processing node. Sensing and communication consumes energy. A good power management and scheduling can effectively extend operational time [50]. To reduce the energy consumption and improve the lifetime of a WSN there are various kinds of methods, which have been developed. Mostly these methods can be categorized into:
  - Scheduling operations, to allow a node to enter low energy sleep states.
  - Choosing routes that consume the lowest amount of energy.
  - Selective use of wireless nodes based on their energy status.
  - Reducing the amount of data.
- **Programmability:** The nodes should process information and also be able to react flexible on changes in their given tasks. Therefor the nodes should be programmable to serve the desired application.
- **Maintainability:** The changes in the environment and in the network require a flexible solution that can adapt itself and maintain the services of a WSN.
- **Production costs:** The cost of a single node is critical to justify the overall cost of the networks. If the cost of the network is more expensive than deploying traditional sensors, the sensor network is not cost-justified.

2.2.2 Required Mechanisms

In order to realize the characteristic requirements of an application new mechanisms for communication in the network, architectures and protocol concepts have to be defined. Typical mechanisms for WSN are:

- **Multi-Hop wireless communication** avoids long distance limitation in the direct communication between a source and a sink, other sensor node can be used. The intermediate nodes are used as relays and can reduce the total power required.
- **Energy-efficient operation** is a key mechanism to support a long lifetime of the network. This can be achieved for example by energy-efficient data transportation between two nodes or by energy-efficient determination of requested information.
- **Auto-configuration** WSN should be able to configure some of its operational parameters autonomously, independent of external configuration. For example, nodes have to determine their geographical positions only using other nodes in the WSN.
- **Collaboration and in-network processing** The sensor nodes have to interact in order to decide, whether an event has happened or to complete the information processing.
- **Data-centric:** In traditional communication networks, the data are transfered between two specific devices and each device is equiped with one or more network address. This approach is named address-centric. It is not suitable for a WSN. In a WSN the answer to a request from an application and the value of the sensor nodes are dominant not the addresses of the sensor nodes that are incorporated to fulfill this task. This approach is called data-centric.
- **Locality:** The need to estimate the position (the spatial coordinates of nodes) is important due to the limited resources of a sensor node. Applications in WSNs (e.g. disaster relief application) might require information about its neighbors to achieve the whole task.

2.3 Hardware of Sensor Nodes

A WSN is a deployment of sensor nodes. There are many factors according to the application, which should be considered in the design of the hardware.

- Low-cost,
- small in size,

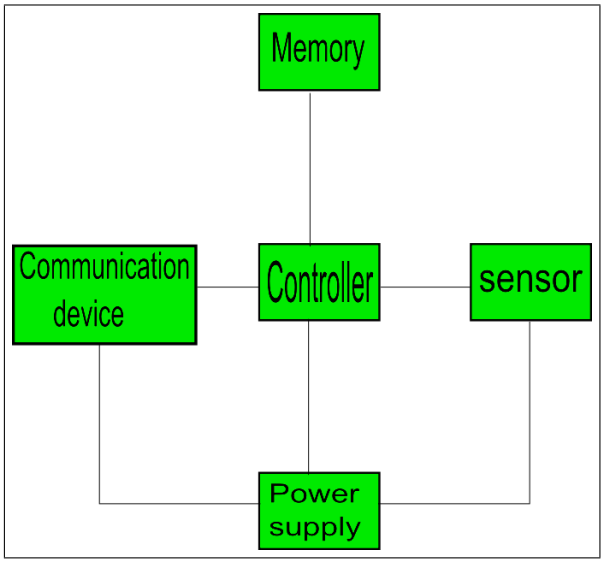

Figure 2.2: Sensor Node Hardware.

- low power consumption,
- communication and computation facilities and
- size of memory.

The trade-off between these factors is an important prerequirement in the implementation and developement of a WSN [29][41]. The five basic components of sensor node are shown in figure 2.2 [22]. Each component from this figure is explained in more details below.

1. **Microcontroller:** A microcontroller is highly integrated chip that contains all components comprising a controller. This includes a CPU, RAM, some form of ROM, I/O ports and timers. Unlike general-purpose computers, which also include all of these components, a microcontroller is designed for a very specific task – to control a particular system. As a result, the parts can be simplified and reduced, which cuts down production costs. Microcontrollers are sometimes called embedded microcontrollers, which just means that they are part of an embedded system. They collect data from the sensors, enable to process the data, decide when and where to send the data, receive data from other sensor nodes and execute programs [22][34].
2. **Memory:** Memory is where data and program code are stored. There are various types of memory:

- **Random access memory(RAM):** It is usually used to store immediate sensor readings from this and other sensor nodes. It loses its content when it loses power. RAM is read and write memory and stores data that continually changes during microcontroller operations. Random access means that each location can be accessed in the same amount of time, independent of its address and without a physical movement of the storage medium or a physical reading head. The most types of RAM are volatile, which mean they lose their contents when power is lost.
- **Read only memory(ROM):** It is used for permanent program information that can never be erased. The semiconductor manufacturer fabricates the memory with the data, which can not be altered later.
- **Erasable Programmable Read-Only Memory(EPROM):** It don not lose data when it loses the power. In other words, it is non-volatile, but program can not write data to it. However, it is possible to put data on it by using a special programming procedure. One can erase it by exposing it to ultraviolet(UV) light for a period of time, then it can be reprogrammed. The UV light has usually a wavelength of 235nm.
- **Electrically Erasable Programmable Read-Only Memory (EEPROM):** It is like EPROM, but has the adventage that it can be erased electrically instead of using UV light.
- **Flash memory:** It is like EEPROM, but it is possible to change a sector of multiple bytes at a time. Programing can be done a bit at time, but erasing is done in a large blocks-flash erase process, from which memory gets its name [35]. Thus individual bits can be read randomly, but writing must be done in a block.

3. **Sensors:** A sensor is a device that detects and converts a natural physical quantity into output that a human can interpret [34]. One classification of sensor is into:
   - **Passive, omnidirectional sensors:** They can measure a physical quantity at the point of the sensor node without actually manipulating the environment. Typical examples for those sensors are thermometer, light, vibration, microphones.
   - **Passive, narrow-beam sensors:** They have a well-defined notion of direction of measurement. Examples of this type are cameras.
   - **Active sensors:** They probe the environment, for example radar.
4. **Communication device:** It is used to exchange data between individual nodes. Usual options for transmission medium involve radio frequencies(RF), optical communication and ultrasound. The RF-based communication is mostly used. It provides long range high bandwidth data at

acceptable error rates. Required energy is reasonable and further a line of sight between transmitter and receiver is not neccessary. A transceiver is a device that combines transmitter and a receiver. The most important tasks of transceivers are:

- Offering the approperiate sevices to MAC layer.
- Providing several channels for MAC protocol.
- Switching to deferent states like active, sleeping or idle to support energy efficiency.
- Matching the application requirements and restrictions.
- Controling the transmission power.
- Allowing various coding scheme to be selected.

The transceiver can distinguish between four states:

(a) Transmit state (the transmit part of transceiver is active).

(b) Receive state (the receive part of transceiver is active).

(c) Idle state (the transceiver is ready to receive but it is not currently receiving anything).

(d) Sleep state (a significant parts of the transceiver are switched off).

5. **Power supply:** There are mainly two sources of power supply:

   - Storing energy using batteries.
   - Scavening energy from the environment with solar cells.

2.4 Sensor Network Scenarios

While the sources in a WSN are sensor nodes, the sinks can be a sensor node or an entity outside the WSN as personal digital assistant (PDA) or a gateway as illustrated in figure 2.3, figure 2.4 and figure 2.5.

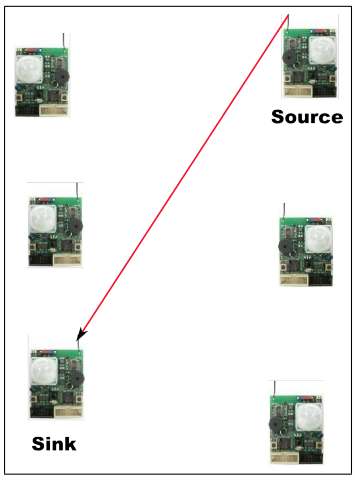

Figure 2.3: Sink as Sensor Node.

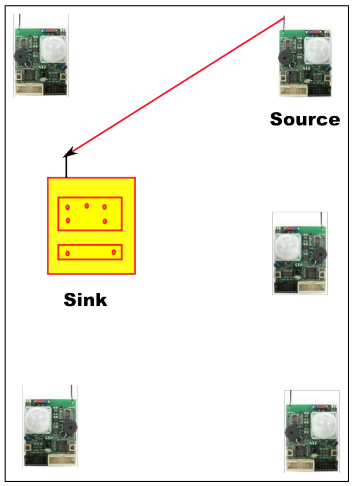

Figure 2.4: Sink as PDA.

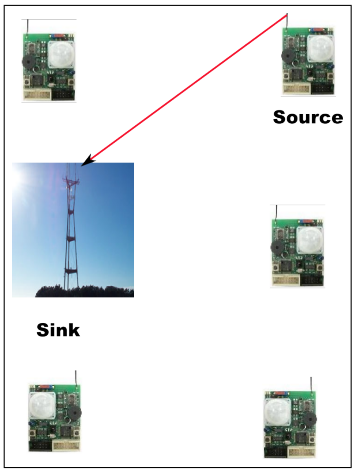

Figure 2.5: Sink as Gateway.

2.5 Single-Hop versus Multi-Hop Networks

Single-Hop networks use a direct communication between the source and the sink. Due to the limited distance using radio transmission, a simple direct communication between the sources and the sinks is not always possible. Hence, communication relays in Multihop networks. A hop represents one portion of the path between source and its destination. In networks data passes from the source to the sink through a number of intermediate devices like routers. Each such device causes data to hop between one point-to-point network connection.

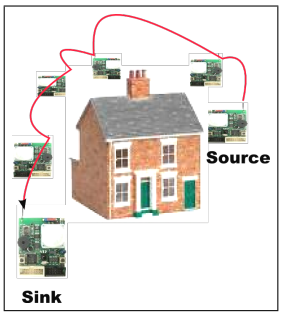

Figure 2.6: Multi-Hop Network with one Source and one Sink.

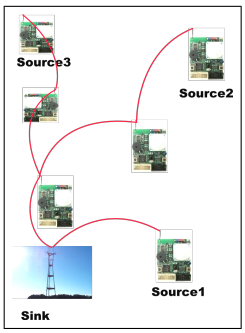

Figure 2.7: Multi-Hop Network with three Sources and one Sink.

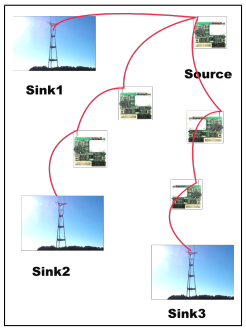

Figure 2.8: Multi-Hop Network with one Source and three Sinks.

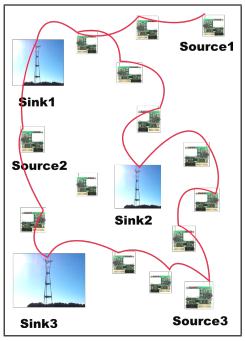

Figure 2.9: Multi-Hop Network with three Sources and three Sinks.

The main benefits of a Multihop networks are increased wireless coverage areas, enhanced performance, reduced energy consumption and enabled automated reorganization of access point distribution [22][62]. In Multi-Hop WSNs the sensor nodes can act like relay nodes without need of intermediate devices. Commonly WSN applications have multiple sources and sinks. Examples of various WSN Multi-Hop networks are shown in the figures 2.6, 2.7, 2.8 and 2.9.

2.6 Mobility in Wireless Sensor Networks

Flexibility and ease of deployment of wireless sensor networks enable WSNs to be used in various locations (e.g. Inventory, tracking, military). WSNs increase their usefullness through the ability to support mobility. The mobility in WSNs provides several new challenges in techniques, protocols and energy support. The mobility in WSNs can be categorized in three types:

- **Node mobility:** The wireless sensor node can change its position during operation time. Specifically when the network is used to monitor a moving object (e.g. a sensor node attached to cattle). Requirements in such networks are auto configurablility and an acceptable energy consumption to maintain a good level of functionality.
- **Sink mobility:** Here the sink is considered as an external part of the network. Having a mobile sink in WSNs offers many adventages as [22][27][4]:
  - Energy consumption is reduced and lifetime of the WSN is increased. When a mobile sink moves near to a node, the data is transmitted over fewer hops and the number of transmitted packets is reduced.
  - WSNs with sink mobility can be better handeled due to the flexibility of the sink's location.
  - A mobile sink enables to monitor an area through fewer sensor nodes and reduces the cost of the network.
  - Sensor nodes can decrease their data transmission range to the lowest range required to reach the mobile infrastructure.
  - A mobile sink can navigate through problematic regions where sensor nodes can not operate or are defect.
  - Decreasing the number of hops, lowers the probability of transmission errors and collisions.
  - Since, the data does not traverse multiple hops, sink mobility improves WSNs security.
- **Event mobility:** In a typical application the source of the event is mobile (e.g. in tracking applications). This type of WSN's mobility needs a sufficient number of sensor nodes. Sensor nodes that are currently not used to detect any approprate event, can be sent to sleep state.

2.7 Protocol Stack

All sensor nodes in the WSN should have capabilities to collect and route data to the sink or to another sensor node. The WSN protocol has to fulfill the following management plans:

- **The power management plan:** To minimize power consumption.
- **The mobility management plan:** To detect and register the movement of nodes. To determine the sensor node's current neighbors and always maintain a data route to the sink.
- **The task management plan:** To plan and schedule the sensing task for a given region.

Layers in a sensor network protocol stack are structured similar to an ISO protocol stack [22][43]. The stack has the following layers:

- Physical layer,
- Data Link layer,
- Network layer,
- Transport layer and
- Application layer.

Physical Layer:

In WSN the layers of the protocol stack can be described further by their function. The physical layer underlies all other communication-related technologies in WSNs and offers sevices, which are needed by the data link layer. The physical layer defines how to encode and modulate a single raw bit for transmission. No packet headers or trailers are handled. The main functions of the physical layer are:

- Bit by Bit transmission .
- Digital modulation(A digital data is mapped to one or a finite number of waveforms of the same finite length).
- Line coding(A digital bit stream is transferred over an analog lowpass channel using a discrete number of signal levels, by modulating a pulse train).
- Signal processing and encryption.
- Start and stop of signaling in asynchronous serial communication.
- Bit synchronization.
- Offering an interface to the transmission medium.

The design of the physical layer should meet the requirements of WSNs. The radio transition must be inexpensive and containable in a small device. Technologies discussed to be used in the physical layer of WSNs [37] are:

- Narrowband technologies.
- Spread spectrum technologies.
- Ultra-Wideband (UWB) technologies.

Data Link Layer:

Data link is definied as connection from one location to another in order to transmit and receive data. It has the resposibility to customize requests from the network layer and issues service requests to the physical layer. The data link layer provides functional and procedural means to multiplex data streams, transfer data between network entities, detect and possibly correct errors that may occur in the physical layer. To view the functions of the data link layer, it may be divided into other subsystems [44]:

- The Media Access Control (MAC) subsystem decides when to transmit and what channel to use.
- The error control subsystem encodes or decodes data based on a specific error detection or correction code.
- The transmit data subsystem transmits data to the physical layer.
- The local address subsystem is responsible for assigning a locally unique address to a node.
- The location subsystem determines or refines a nodes location. This computation can be based on sensor node's neighbors assumed location and the distances between neighbors and itself .
- The process data subsystem processes the data from the physical layer.
- The neighbor list subsystem creates and maintains the neighbor list. The neighbor list has the following information about every neighbor: location, local address and link metric.
- The mobility subsystem supports mobile nodes.
- The link metric subsystem provides a metric for every link. The network layer uses the metric to compute the probability of taking a path. The subsystem also stores channel status (needed by MAC subsystem) and Received Signal Strength (RSSI) measurements (needed by the location subsystem).
- The power control subsystem specifies the transmission power level.

Network Layer:

The network layer responds to custom requests from the transport layer and issues custom requests to the data link layer [43]. Where the data link layer is responsible for node to node packet delivery, the network layer is responsible for source to destination packet delivery. In other words, the Link layer handles how two nodes talk to each other and the network layer is responsible to decide which node to talk to. In the network layer of WSNs there are several factors to

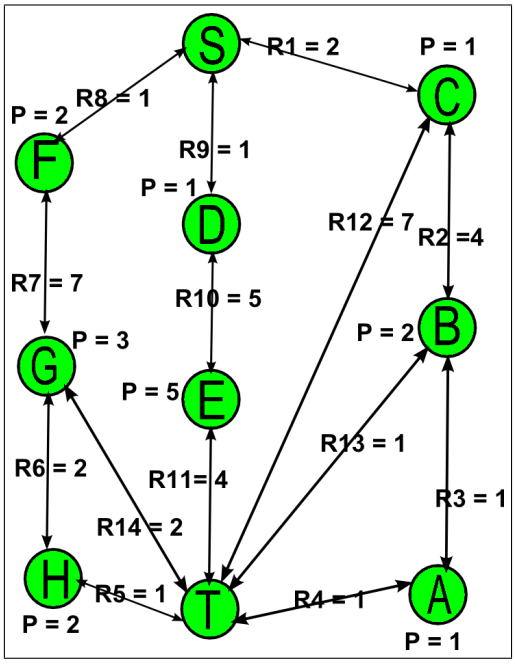

Figure 2.10: The Power Efficiency of the Routes.

consider: Power efficiency, data-centric design and data aggregation. In figure 2.10, node S is the sink node and node T is the source node. The nodes A, B, C, D, E, F, G and H are intermediate nodes. P refers to the energy efficiency based on the available power and R refers to the energy efficient based on the energy required to transmit a data packet through the related link. Table 2.1 shows 6 routes from the source to the sink with their energy efficiency based on the available power and energy needed.

		Sum of energy based on available power	Sum of energy based on energy required
Route 1	S-C-B-T	3	7
Route 2	S-C-B-A-T	4	8
Route 3	S-D-E-T	6	10
Route 4	S-F-G-T	5	10
Route 5	S-F-G-H-T	7	11
Route 6	S-C-T	1	9

Table 2.1: 6 Possible Routes to Communicate with the Sink.

There are different approaches to select energy efficient routes:

1. **Maximum available power route:** The selected route is the one with the maximum sum value of the available power to transmit data packets from the source node to the sink node. Although in table 2.1, route 5 has the maximum sum value 7 but it is not power efficient because route 4 is included in it. After eliminating route 5 the prefered one is route 3.
2. **Minimum energy route:** The selected route is the one that needs minimum energy to transmit the data packets from the source node to the sink node. In table 2.1 this is route 1.
3. **Minimum hop route:** The selected route is the one with the minimum number of hops from the source node to the sink node. In table 2.1 this is route 6.

Transport Layer:

The transport layer responds to custom requests from the application layer and issues custom requests to the network layer. The transport layer gets its importance when the system needs to communicate with the outside world. It handles the segmentation of large packets into smaller packets. The WSN is not based on global addressing. Attribute-based naming is used to indicate the destinations of data packets.

Application Layer:

A Sensor Management Protocol(SMP) at the application layer is used to make the hardware and software of lower layers transparent to the Sensor Network Management Applications. The system administrators and programers interact with WSN using SMP. Again the lack of global identification in sensor networks must be taken into consideration. SMP provides rules to enable interaction between applications and the sensor networks for:

- Data aggregation, attribute-based naming and clustering.
- Data exchange related to the location finding algorithms.
- Time synchronization.
- Movement of sensor nodes.
- Turning nodes on or off.
- Querying WSN configuration status, reconfiguring the WSN.
- Authentication, key distribution and security.

2.8 Medium Access Control (MAC)

Medium Access Control (MAC) is the first protocol layer upon the physical layer. One of the main functions of the MAC protocol is to avoid collisions. The collision may happen when sensor nodes intend to transmit data, control or manage packets to another sensor node at the same time point. The MAC protocol solves the question which sensor node should access the communication channel in the next time point. As in all traditional networks MAC is an essential technique to enable the successful operation of the network. In contrast to traditional networks, the MAC protocol for WSNs should take into account the nature of data transmission and application requirements. The goal of the MAC protocol in traditional network such as cellular networks is to provide a high QoS and bandwidth efficiency. Power conservation may be a secondary goal in such systems. Power efficiency plays an important factor in WSNs infrastructure. In tradtional networks, where wire is deployed, the sent data is usually forwarded to every participant, in WSNs the radio signal propagates into all directions and the received power decreases with distance between transmitting and receving sensor nodes. This nature arises a problem in WSNs called the hidden-terminal problem. As illustrated in figure 2.11 sensor node A can communicate directly with sensor node B and likewise sensor node B can communicate directly with sensor node C. But C can not reach A and A can not reach C. A and C are not within the same sending range. If A tries to send a packet to B, C can not hear this communication because A's packet can not reach C. If C tries to send a packet to B, A can not hear this communication because C's packet can not reach A. If A sends a packet to B and C has no sense of this communication and sends at the same time point a packet to B, both A's packet and C's packet will collide at B. Only B can detect that something is wrong.

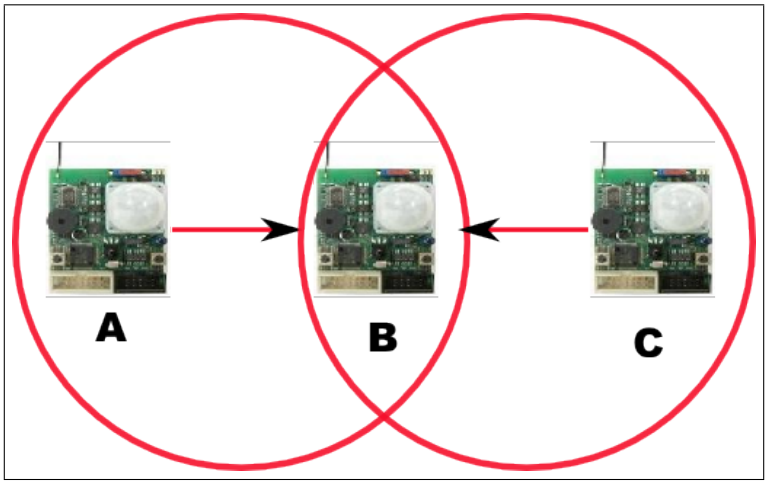

Figure 2.11: Hidden-Terminal Problem.

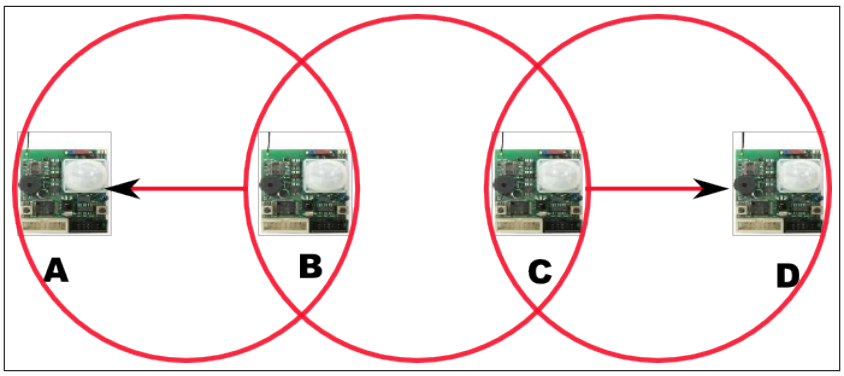

Figure 2.12: Exposed-Terminal Problem.

Another problem in WSNs is called the exposed terminal problem. As illustrated in figure 2.12.
A and B are within the same sending range. A and B can reach each other.
C, D are within the same sending range. C and D can reach each other. B and C are within the same sending range and can reach each other, too. If B sends a packet to A, C can hear this ongoing communication. If C would like to send at the same time point a packet to D and listens to this ongoing communication between A and B, then C knows that the channel is busy and should wait until the channel is free. Although C's transmission to D might not cause a great problem, C has prefered to wait. Both hidden terminal and exposed terminal problem are dealt by the Multiple Access with Collision Avoidance(MACA) protocol or RTS/CTS protocol where:
RTS means Ready To Send and
CTS means Clear To Send,
as illustrated in figure 2.13. If A wants to send a packet to B, it sends first a RTS packet to B, which includes the address and the number of bytes to send. When B recevies C's RTS packet, it sends an CTS packet to A, which includes a copy of the number of bytes that will be sent by C. When A receives B's CTS packet, it starts the transmission of the data packet. At the end of the transmission B will send an acknowledge packet ACK to A, announcing that the transmission is successfull. Lack of acknowlegment will be interpreted as collision and a possible retransmission will be done. All other sensor nodes hearing either RTS, CTS or ACK will be silent for a short moment through setting an internal timer called Network Allocation Vector(NAV). If any sensor node in the sending range hears CTS, it can conclude that it is in the sending range but not in the receiving range. It can make a transaction with another sensor node.

There are still other problems:

- If sensor nodes A and C send RTS packets to B simultanously, the RTS packets are lost.
- As illustarted in figure 2.13, A sends a RTS packet to B and B answers

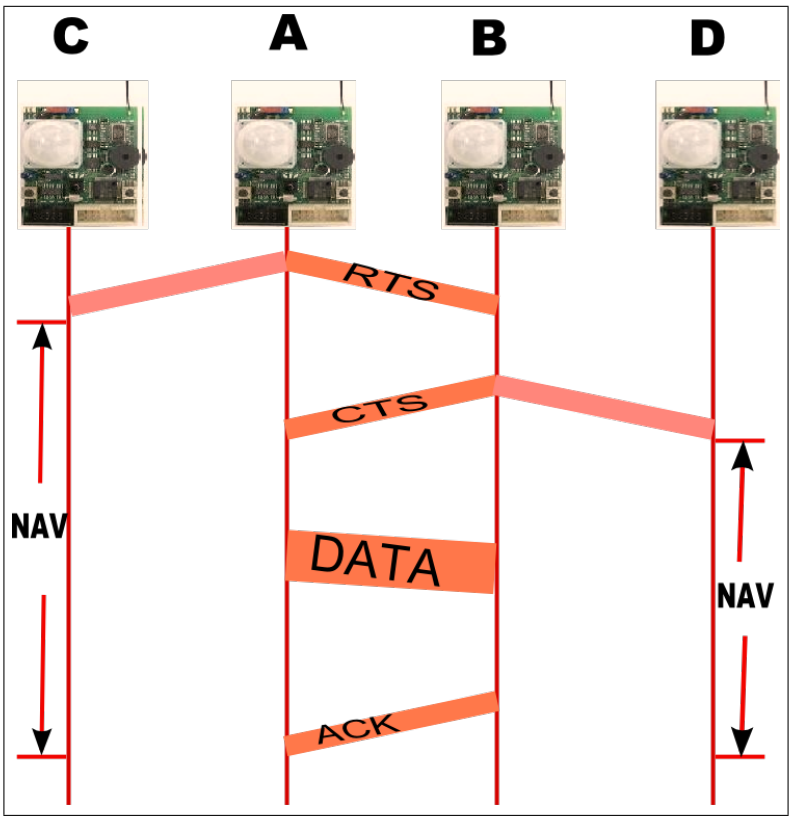

Figure 2.13: RTS/CTS.

with CTS packet to A. C hears this B's CTS packet at the same time point as C receives a RTS packet from D. Both packets B's CTS and D's RTS packet will collide at C. Therefor C can not set its NAV variable accordingly and D's RTS packet fails. D will retransmit the RTS packet again to C. In this case C answers with a CTS packet to D. One solution for this problem is to make CTS packets take more time than RTS packets.

The most important requirements of MAC protocol designed for WSN are energy efficiency and scalability. Major reasons of useless consumption of energy [22] [64] are:

- Collisions: When a destination sensor node receives a corrupted packet, it consumes useless energy as the transmitted packet has to be discarded. The retransmission consumes another part of energy. In general collisions should be avoided.
- Overhearing: When a destination sensor node senses packet transmissions,

which are sent to another destination sensor node, it consumes useless energy.

- Protocol overhead: For packet-headers, -trailers and other protocol information data has to be sent and received by each packet according to the used protocol. Since this is not the orginal data, this consumes additional energy such as RTS and CTS packets in MACA.
- Idle listening: A sensor node that is listening to receive a packet, which is not sent, uses energy. In this case the data rate is very low through the idle state.

2.8.1 Sensor Node's Wakeup and Duty Cycle

As shown in figure 2.14, the sensor node left its sleep state to transmit or receive packets. Most of the time, the sensor node is in this state. A sensor node goes in sleep state and wakes up itself to listen if any other node wants to send a packet to it. Any sensor node attending to send a packet to this node should have knowledge about the listen period. The wakeup period consists of both the sleep period and the listen period.

$$Wakeup\ period = listen\ period + sleep\ period.$$

The duty cycle is the ratio of the listen period length to the wakeup period length.

$$Duty\ cycle = \frac{listen\ period}{Wakeup\ period}.$$

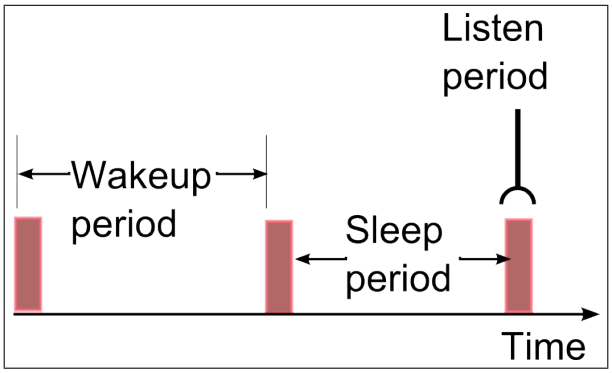

Figure 2.14: Periodic Wakeup and Sleep.

If the duty cycle is small compared to the overall time interval, the transceiver is in sleep state most of the time. This means that the idle listening period is small and energy can be preserved. The traffic concentrates in the listen period and at certain time period a node may send more data than other sensor nodes in the WSN.

2.9 Sensor-MAC(S-MAC) Protocol

The primary goals of S-MAC protocol are reduced energy consumption, self configuration, scalability and collision avoidance [28][64]. Two interfering sensor nodes do not transmit data at the same time point. One fundamental consideration is the obsevation that the sensor node is for a long time in an idle state. If no sensing event occures, it is not necessary to keep the sensor node listening all the time in the idle state. By this the sensor node can save energy. Every sensor node can go to sleep for some period of time and wake up to see if any other sensor node wishes to make conversation with it. In sleep state the sensor node turns off its radio and sets a timer to awake itself later.

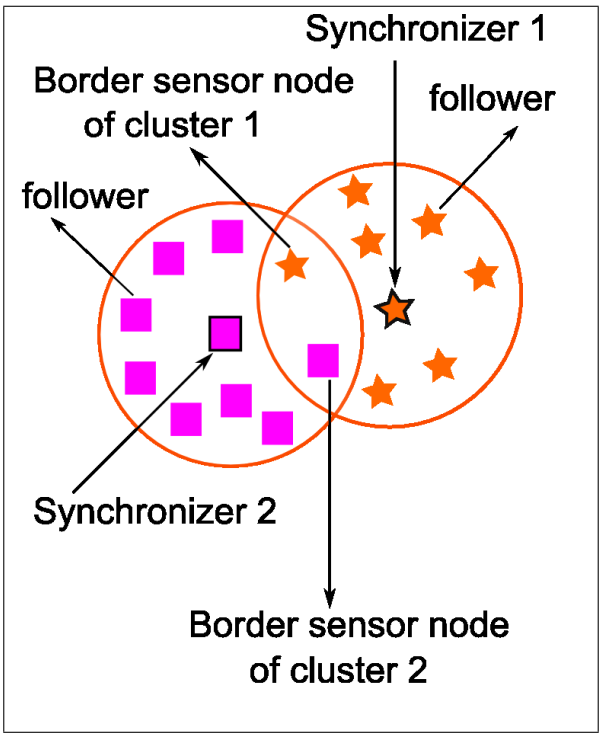

Figure 2.15: Two Groups in WSN with Deferent Synchronizer, Follower and Border Sensor Nodes.

Every sensor node must choose a schedule for its next sleep time point and broadcast this schedule to all its neighbors. The sensor node retains a schedule table to store the schedule of all its neighbors. After all sensor nodes are deployed in the WSN a random timer will be started by every of these sensor nodes. The sensor node which listen first to discover a neighbor and does not hear any schedule from another sensor node, choses a random time to go to sleep and broadcasts this schedule in a message called SYNC. The SYNC includes the address of the sender and the sender's sleep time in T_1 seconds. The sensor node which has first chosen its schedule independently is called the synchronizer. In figure 2.15 two synchronizers are shown. All the other sensor nodes, which received the synchronizer's schedule are called followers. A follower must set its schedule according to the synchronizer's schedule. It waits for a random delay time T_2 seconds to go to sleep in $T_1 - T_2$ seconds. It transmits its schedule to all its neighbors. If a follower hears a different schedule than the schedule of its synchronizer, it must follow both schedules and wake up according to both schedules. If two different sensor nodes have defined their schedule independently because they can not hear each other, then there are two different synchronizers. Each synchronizer and its followers constitute a group or cluster. The border sensor nodes, which lay in both sending ranges of the two synchronizers adopt both schedules. Border sensor nodes are the only sensor nodes that are able to transmit a packet from one cluster to the other cluster. However the border sensor nodes have a small time to sleep and consume more energy than their neighbors. When the border sensor nodes consume all their energy, both clusters will be isolated. When sensor node has sent a packet to another sensor node with RTS-CTS-DATA-ACK mechanisums, uninvolved sensor nodes may hear RTS or CTS of this transmission. These uninvolved sensor nodes can go to sleep and turn off their radio transceiver. Each RTS and CTS packets include the number or length of the bytes which will be sent. The uninvolved sensor nodes use this length to set up their NAV variable. The NAV variable is decremented every time unit by one through the timer-intrrupt until it reaches zero. Every sensor node checks its NAV variable to know, if there is transmission ongoing. When the value of the NAV is greater than zero then the transmission is still ongoing or the channel is busy. So the sensor node should check its NAV before it decides to transmit. This process is called virtual carrier sense.

2.10 Routing Protocols

Routing is the process of moving a packet of data from source to destination. The packet may need to traverse many sensor nodes to reach its destination network. Routing is the complex process to determine which sensor nodes will move the packet to its destination.

2.10.1 Characteristics of Routing in WSN

According to [41] the characteristics of routing in WSN are:

- The tradional IP-based protocols may not be applied to WSN due to the large number of sensor nodes. In WSN, the overhead of ID maintainance is very high as the number and position of sensor nodes can not be predetermined.
- In WSN the gathering of data is more important than knowing the IDs of sensor nodes, which sent the data.
- Several applications of WSN require the flow of sensed data from multiple sources and regions to a particular sink.
- Significant need of network management.
- In WSNs multiple sensor nodes may generate the same data. Hence, there is a high probability that data has significant redundancy.
- The design requirements of WSNs change according to the application.

Routing protocols should take these characteristics into consideration. Routing protocols in WSN can be catagorized into:

- **Data-centric protocols:** They are query-based and depend on the naming of the desired data. The data is requested based on certain attributes. This attribute based address consists of a set of attribute-value pair queries.
- **Hierarchical protocols:** They aim at clustering the sensor nodes so that cluster heads can do some aggregation and reduce data in order to save energy.
- **Location-based protocols:** They utilize the position information to relay the data to the desired regions rather than the whole network.

One important task at the network layer is to establish reliable relaying of data from a source node, bypassing intermediate sensor nodes, to reach the sink node. Intermediate sensor nodes should decide to which neigbors the incoming packet must be forwarded. Forwarding passes packets from one sensor node to another. Routing selects the most reliable sensor node to forward the data.

There are some techniques for forwarding such as flooding, where each sensor node sends the incoming packet to all its neighbors. In this way the same packet may be sent several times to the same sensor node.

This is not energy efficient. As illustrated in figure 2.16 this forwarding technique suffers from implosion. In a WSN consisting of N sensor nodes a sensor node may receive N copies of the same packet. N is the number of the

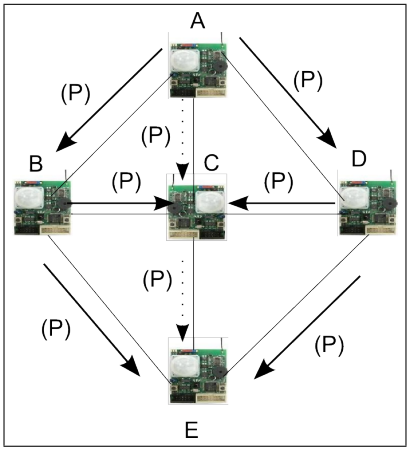

Figure 2.16: The Implosion Problem.

same neighbors of both the sending sensor node and the receiving sensor node. Another drawback of flooding technique is overlap, where some sensor nodes may sense the same region and send the same data to the same neighboring sensor nodes. The flooding technique consumes a large amount of energy, too. Another technique for forwarding is gossiping, where each sensor node sends the incoming packet to a randomly selected neighbors but not to all the neighbors. The gossiping technique can avoid the implosion problem but it takes a long time to transmit a packet.

2.10.2 Sensor Protocol for Information via Negotiation(SPIN)

In SPIN protocol there are three types of message to transmit a packet from a sending sensor node to a receiving sensor node. These three types are:

- ADV message,
- REQ message and
- DATA message.

As illustrated in figure 2.17, 2.18 and 2.19. When a sensor node tries to send data to another sensor node, it sends an advertisement message(ADV) containing meta-data first. The sensor node, which is interested in this packet sends a request message(REQ) to the sending sensor node. After this, the sending sensor node sends the data. Although the SPIN approach can avoid the problem of the flooding technique, it suffers from finding an intermediate sensor node to forward the packet to the sink when this intermediate sensor node is not interested in the data and the sink is far away from the source sensor node.

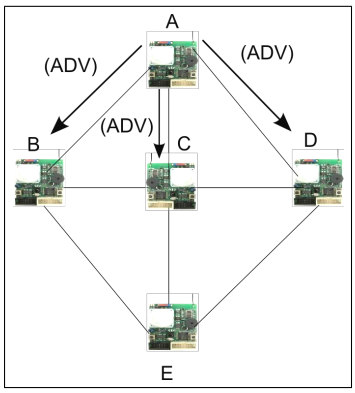

Figure 2.17: Sensor Node A starts its Transmation by Sending an Advertising Message.

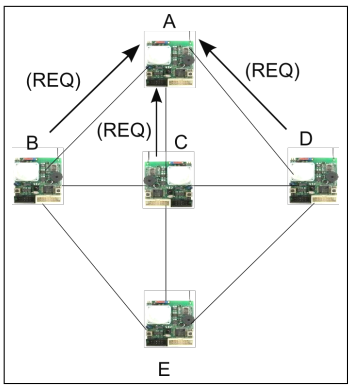

Figure 2.18: Sensor Nodes B, C, D answer to Sensor Node A by Sending the Request Messages.

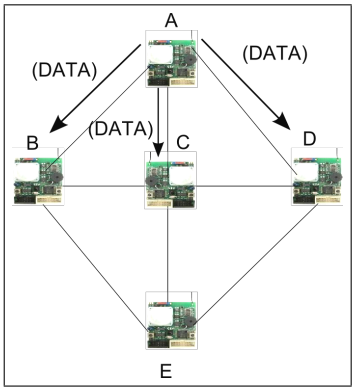

Figure 2.19: Sensor Node A sends the DATA Message after Receiving the REQ Message.

2.11 Time Synchronization

Time synchronization in networks is the process of adjusting computer clocks of different devices [16] [22][33]. A sensor node has usually an oscillator and a counter register. The oscillator produces a repetitive electronic signal, often a sine wave or a square wave. This clock is the source for the clock in the sensor node. The frequency at which an oscillator works is usually determined by a quartz crystal. Oscillators have often a slight random deviation from their nominal frequency, called drift. The counter register will be incremented after a certain number of oscillator pulses. If two events happen between two increments of the counter register, both have the same timestamp. The software

of the sensor node has only access to the value of the counter register. Let $H_i(t)$ be the hardware clock of a node i at real time t, a local software clock $L_i(t)$ can be approximated as follows:

$$L_i(t) = \theta_i \cdot H_i(t) + \phi_i.$$

Where
θ_i is called the drift rate of sensor node i and
ϕ_i is called the phase shift of the sensor node i.
Clock adjustment is done by adjusting the coefficients of θ_i and ϕ_i.
The synchronization can be either external or internal:

- **External Synchronization** is the synchronization of all clocks in the network to a time supplied from outside the network [40]. Given a network with n sensor nodes, the sensor nodes (1, 2,.....,n) are said to be accurate at time t within a bound δ, if $|L_i(t) - t| < \delta \quad \forall\ i \in [1, 2, ..., n]$. The source of the common system time can be a sensor node in the network.
- **Internal synchronization** is the synchronization of all clocks in the network, without a predetermined master time. The sensor nodes (1, 2,.....,n) are said to agree on the time with a bound of δ, if $|L_i(t) - L_j(t)| < \delta \quad \forall\ i, j \in [1, 2, ..., n]$.

In a WSN, the sensor nodes are switched on at random times. Hence the initial phases ϕ_i of the sensor node i is assumed to be randomly distributed.
The Oscillator frequency is time variable. If two sensor nodes have the same type of oscillator and have been started at the same time, the difference $|L_i(t) - L_j(t)|$ might increase overtime. therefor a time synchronization algorithm should resynchronize once in a few minutes to keep track of changing frequencies.

2.11.1 Error Source in Network Time Synchronization

Errors in network time synchronization are due to:

- **Send time:** The time, the sender sensor node takes to construct a message.
- **Access time:** The delay time before the actual transmission of the message. This delay time may depend on the MAC scheme used or on the waiting time when the channel is idle.
- **Propagation Time:** The time spent to propagate the message between the network interface of the sender and the receiver sensor node.
- **Receive time:** The time, the network interface of the receiver sensor node takes to receive the message and transfer it to the host.

2.11.2 Lightweight Time Synchronization Protocol (LTS)

Referring to [30] according to [13] the time synchronization process can be divided into three main components:

- **The resynchronization event detection:** It indentifies the time at which the sensor nodes have to resynchronize their clock. One technique is to relay on a specific sensor node to send an initiating message to every sensor nodes in the system after kR time. Where R is the duration of a single synchronization round and $k > 1$ is a real number to prevent overlap between the rounds.
- **The remote clock estimation:** It determines the local time of another sensor node in the system. One technique to do this is time transmission when the time of a remote clock is sent in a message. This adds unwanted communication overhead, which is avoided in LTS.
- **The clock correction:** It is used to update the local time of a sensor node when a resynchronization event has occurred.

In the Lightweight time Synchronization protocol(LTS) two major steps is carried out:

- A pair-wise synchronization protocol synchronizes two neigboring sensor nodes [60].
- A spanning tree is constructed from the reference sensor node to all other sensor nodes.

Pair-Wise Synchronization

The following scenario describes a basic scheme for sensor node i to synchronize its clock to the clock of sensor node j:

- After the resynchronization is triggered at sensor node i, sensor node i formats a synchronization request packet.
- Sensor node i forms a timestamp to the synchronization request packet at time t_1 with time $L_i(t_1)$.
- Sensor node i hands the synchronization request packet over to its operating system and the protocol stack.
- When sensor node i is sending the first bit at time t_2, sensor node j receives the last bit of the packet at time t_3 due the variability of the medium access delay time. $t_3 = t_2 + \tau + t_p$, where τ is the propagation delay and t_p is the packet transmission time.

- The packet arrival is signaled to the operating system of sensor node j at some time later t_4 due to the interrrupt latency.
- The sensor node j forms a timestamp to the packet at time t_5 with $L_j(t_5)$.
- Sensor node j forms its answer packet and timestamps it at time t_6 with $L_j(t_6)$.
- Sensor node j hands the answer packet over to its operating system and protocol stack. The answer packet includes the timestamps: $L_i(t_1)$, $L_j(t_5)$ and $L_j(t_6)$.
- Sensor node i receives the packet reception interrupt at time t_7.
- The time t_7 can be calculated as the sum of t_6 + operating system overhead + medium access delay + propagation delay + packet transmission time + interrupt latency.
- Sensor node i timestamps the packet at time t_8 with $L_i(t_8)$.

Assuming that there is no drift between the clocks in the time between t_1 and t_8. Sensor node i estimates the value of

$$\mathbf{O} = \Delta(t_1) = L_i(t_1) - L_j(t_1).$$

Sensor node i can estimate the value of

$$\mathbf{O} = \Delta(t_1).$$

by estimating the value of $\mathbf{O} = \Delta(t_5) = L_i(t_5) - L_j(t_5)$. Values of $L_j(t_5)$ and $L_j(t_6)$ are known by sensor node i from the answer packet of sensor node j. Sensor node i can compute the time between t_5 and t_6 from the difference $L_j(t_6) - L_j(t_5)$. As shown in figure 2.21 the value of t_5 lies in the interval

$$I = [L_i(t_1) + \tau + t_p, L_i(t_8) - \tau - t_p - (L_j(t_6) - L_j(t_5))].$$

In the following it is assumed, that the following times are the same in both directions:

- Operating system time.
- Networking stack time.
- Interrrupt latency time.
- Medium access delay time.

Sensor node i can conclude that the sensor node has created its timestamp $L_j(t_5)$ at time

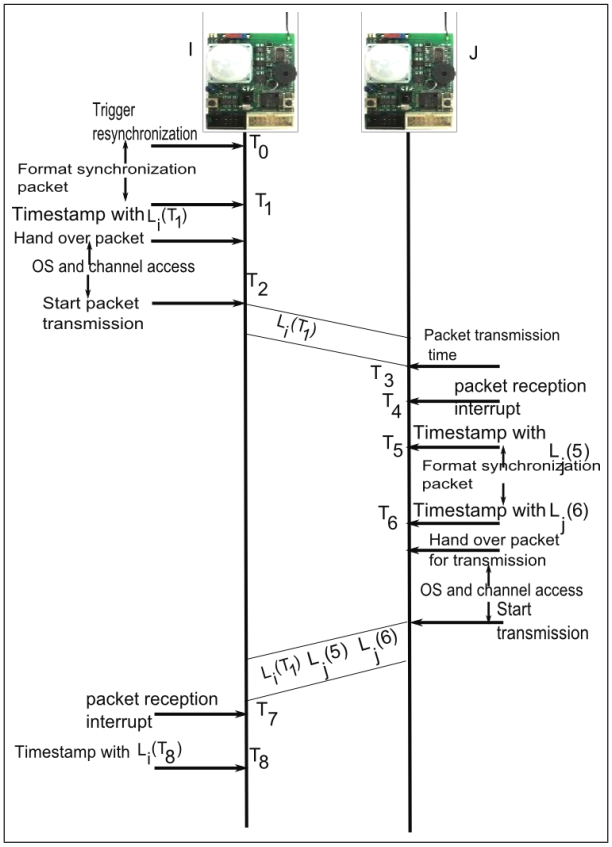

Figure 2.20: Pair-Wise Synchronization.

$$\mathbf{L_i(t_5)} = \frac{\mathbf{L_i(t_1)}+\tau+\mathbf{t_p}+\mathbf{L_i(t_8)}-\tau-\mathbf{t_p}-(\mathbf{L_j(t_6)}-\mathbf{L_j(t_5)})}{\mathbf{2}}$$

$$= \frac{\mathbf{L_i(t_1)}+\tau+\mathbf{t_p}+\mathbf{L_i(t_8)}-\tau-\mathbf{t_p}-\mathbf{L_j(t_6)}+\mathbf{L_j(t_5)}}{\mathbf{2}}$$

$$= \frac{\mathbf{L_i(t_1)}+\mathbf{L_i(t_8)}-\mathbf{L_j(t_6)}+\mathbf{L_j(t_5)}}{\mathbf{2}}$$

Hence,

$$\mathbf{O} = \Delta(t_5) = L_i(t_5) - L_j(t_5) = \frac{L_i(t_1)+L_i(t_8)-L_j(t_6)+L_j(t_5)}{2} - L_j(t_5)$$

$$= \frac{L_i(t_1)+L_i(t_8)-L_j(t_6)-L_j(t_5)}{2}.$$

Sensor node i adds the offset **O** to its local clock to synchronize to sensor node j's local time. Sensor node j needs to know the value of **O**, too. therefor sensor

node i may send a third packet to the sensor node j. The whole synchronization is completed with three packets.

Multi-Hop Synchronization

LTS can be extended to a Multi-Hop sensor network. Important considerations for Multi-Hop synchronization are:

- **Global reference:** There is at least one sensor node that has access to a global time reference.
- **Selective synchronization:** The algorithm can synchronize only sensor nodes, which are broadcasting time-sensitive data.
- **Resynchronization rate:** One-time synchronization is not useful because drift varies overtime. According to [30] referring to [13] a sensor node's clock $H(t)$ is defined to be ρ-bounded, where $\rho > 0$ and for all real time t:

$$\frac{\mathbf{1}}{(\mathbf{1}+\rho)} \leq \frac{\mathbf{dH(t)}}{\mathbf{dt}} \leq \mathbf{1} + \rho.$$

- **Error estimation and limitation:** The synchronization algorithm should keep track of the accuracy, performance and errors caused by a clock drift among sensor nodes. If a sensor node's clock has drifted, the resynchronization scheme should be called.
- **Robustness:** The algorithm should be robust to sensor node failure.
- **Mobility:** The algorithm should perform synchronization for both stationary and mobile sensor nodes.

Chapter 3

Sensor Platform

In this work ScatterWeb Embedded Sensor Board(ESB) nodes as shown in figure 3.1 are used. These sensor nodes have been developed by the computer system and telematics group at the Freie Universität Berlin(FU-Berlin) [23]. Other platforms on the market are the Mica-2 nodes and BTnodes. Mica-2 nodes have been developed at the University of california at berkely in collaboration with Intel [65]. BTnodes have been developed at the ETH Zürich [3].

3.1 Embedded Sensor Board (ESB)

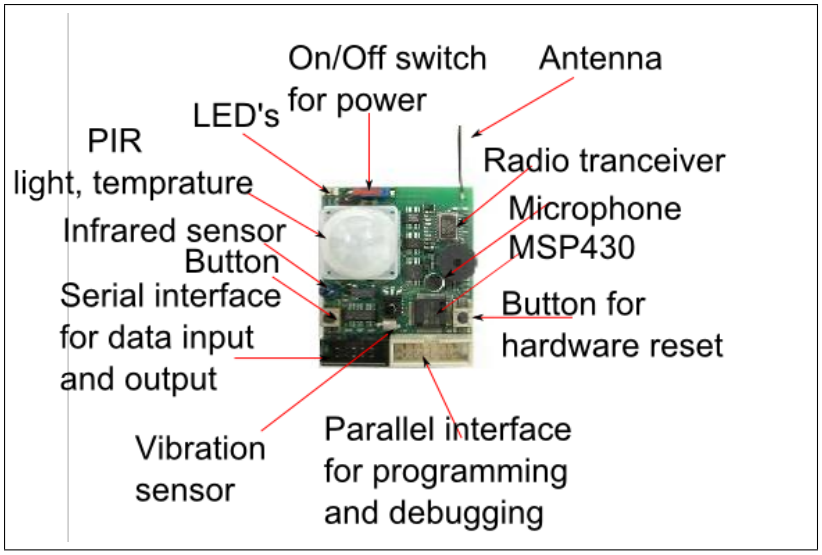

Figure 3.1: ESB Platform.

ESB nodes consist of a micro controller unit(MCU), various sensors and equipements:

- **Passive Infrared Sensor(PIR):** A PIR is a passive electronic device which can measure infrared light radiating from objects in the field of view and detect motion by sensing the infrared fluctuations. This allows to monitor rooms [8] [23] [56]. A PIR sensor is usually used in remodeling the PIR-based motion detection, mounted on printed circut board to interpret the signals. It consists of a solid-state pyroelectric chip, which generates an electric charge when exposed to infrared radiation [5] [63]. Plastic covering the sensor is used to prevent dust and insects obscuring the sensor's field of view. A PIR sensor can only detect motion 20 feet away. It needs approximately 10-60 seconds to calibrate. During this time, one must ensure that there is no motion in the sensors' visual range. The term 'passive' means that the PIR does not emit any type of energy but merely sits 'passive' accepting infrared energy through the front of the sensor, known as the sensor face.
- **Infrared Sensor:** It translates the light intensity directly into a proportional frequency to enable the measurement of light. The device responds over the infrared light range of 800 nm to 1100 nm [23]. The microcontroller has a direct interface to this sensor.
- **Temperature sensor (integrated with Real-time clock):**
  The DS1629 2-Wire digital thermometer integrates a real time clock and a temperature monitor. Communication to the DS1629 is accomplished via a 2-wire interface. The sensor enables accurate time/temperature measurements in battery-powered applications. The open-drain alarm output of the DS1629 can be used as the oscillator input for a microcontroller. It is possible to wake-up the microcontroller from power-down mode.
- **Vibration sensor:** The ESB is equipped with a vibration sensor. This can be used to monitor persons walking through a room. The vibration sensor can trigger and wake-up the microcontroller from power-down mode due to shake the ESB.
- **Microphone:** There is a microhpne equipped on the ESB, which can be used for voice monitoring.
- **Beeper:** The Beeper can be used to signal an event or a required situation.
- **LED:** There are three LEDs on the ESB colored red, green and yellow.
- **RS232:** The ESB contains a standard RS232 serial interface that enables connection with a PC or notebook to establish connection to the wide area wireless network.
- **JTAG:** Figure 3.2 shows a JTAG. The JTAG interface has 2x7 pins and allows flashing and real-time debugging of programs with no need for an external power supply [55]. The JTAG interface has the following properties:

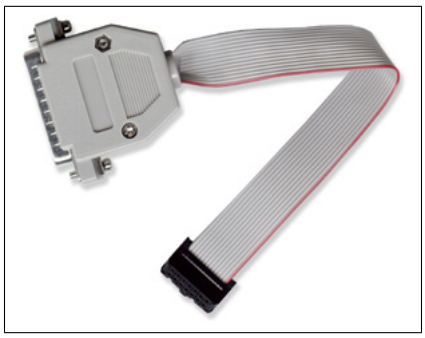

Figure 3.2: JTAG.

1. Programs all MSP430Fxxx flash microcontrollers.
2. No need for external power supply.
3. Works with free GCC C compiler.
4. Compatible with IAR Kickstart software for programming, debugging, real time emulation, step by step program execution. IAR Kickstart enables to write code in assembler with unlimited code size. For C code the memory limit is 2Kbyte.

All sensor nodes may form an ad-hoc network with some nodes acting as data sources, some as relays, and some as data sinks collecting the data. Nodes may act in all three roles at the same time. The ESBs form a sound basis for research in sensor networks. University level courses in CS/EE are based on this hardware. It allows to develop prototyp of sensor network applications for real-world deployment. Typical communication scenarios of sensor networks based on the ESBs are [18]:

- ESBs communicate via the serial port with a standard PC/PDA for application development.
- ESBs communicate with mobile phones/GSM modules to connect to wide-area mobile phone networks. This enables remote configuration of ESBs via short messages (SMS) as well as receiving sensor data on mobile phones world-wide.
- ESBs communicate via their radio interface with other ESBs, ECRs, or eGates to form a truly embedded, highly flexible sensor network solution.

The main features of the ESB are summarized in table 3.1.

Feature	
EEPROM	64 kbyte
Microcontroller	MSP430F149
ESB up and running with all sensors	12 mA
ESB transmitting data	8 mA
ESB deep-sleep (clock only)	8 μA
Flash	60 kbyte

Table 3.1: Features of ESB.

3.2 Microcontroller Unit (MCU)

The Texas Instrument micro controller MSP430F149 is well suited to battery driven applications in ESBs. It has very low power consumption and can be switched into power-down mode [9][17]. The major componenets of the MSP430 family shown in figure 3.3 are:

- 60 kbyte Flash memory.
- An AD converter with 8 external input(port 6) and 4 internal inputs (on chip temperature, Vcc, VeREF+, VeREF-and) with conversation speed of 200000 conversions per second.
- 16 8-bit register in setting for conversation.

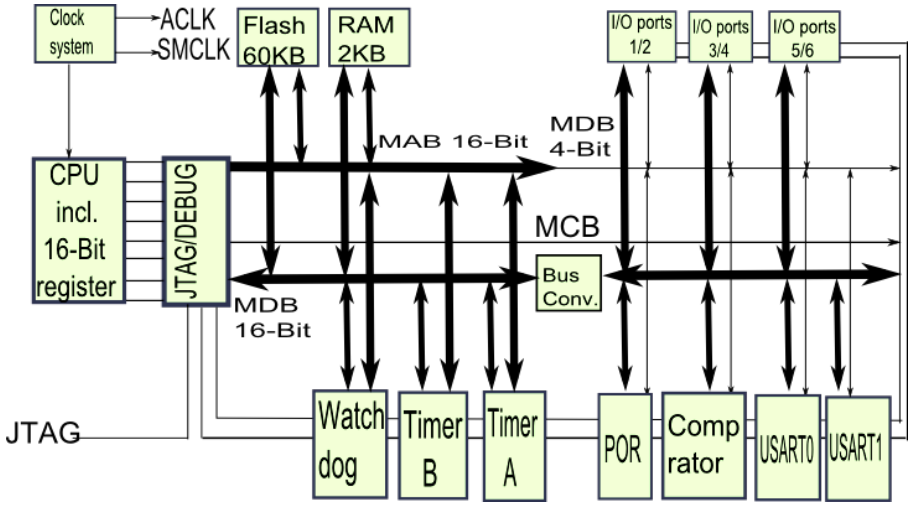

Figure 3.3: MSP430F149 Block Diagram.

- 16 12-bit register to store the result of conversation.
- 2 timers, timer-A and timer-B, are based on 16 bit counters and supporting 3 different modes of operation with different clock sources. The timers comprises Capture-/Compare-Registers, which have special meaning depending on their current mode (capture or compare).
- Timer-A consisits of 4 Capture-/Compare-Registers(CCR0-CCR3).
- Timer-B consists of 7 Capture-/Compare-Registers.
- The Watchdog consists of a timer and a control register. It offers the possibility to reset the processor after a predefined interval. A watchdog is reset periodically in the main loop of a program or an OS. As soon as the program or OS crashes or is locked in a loop, the watchdog triggers a reset. The 16 bit watchdog control register can be written. If the higher 8 bits do not equal 0x5A, the watchdog "password" is wrong and a reset is triggered.

3.2.1 MSP430F149 Memory Map

The MSP430F149 microcontroller has 60kbyte of flash program memory, 256 bytes of flash information memory and 2kbyte of RAM. Its memory map is shown in figure 3.4:

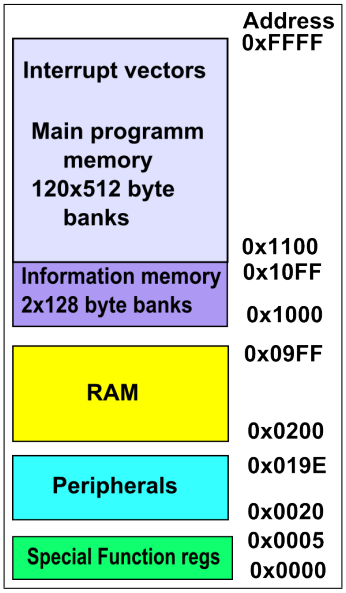

Figure 3.4: Memory Map.

3.2.2 Interrupts

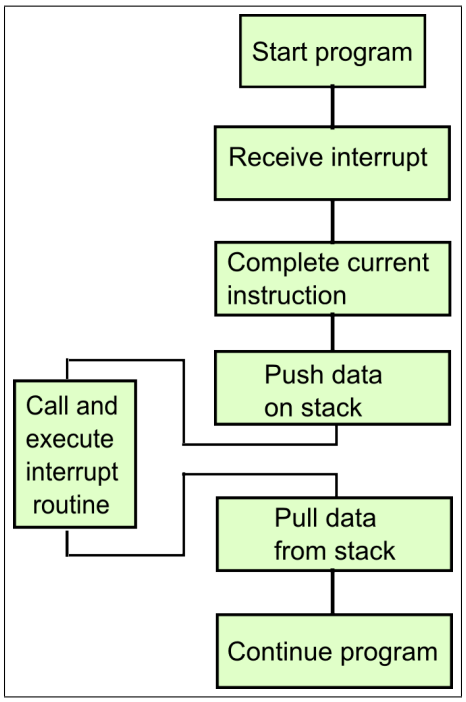

Figure 3.5: Interrupts.

An interrupt is a signal that stops the microcontroller from what it is doing to allow something else to happen. The signal is used to control the digital processor at unexpected events [21][34][46][57]. When an interrupt occurs the main program halts, while another routine is carried out. When this routine finishes, the processor goes back to the main routine again. The interrupt can be seen as an event from hardware that triggers the processor to jump from its current program counter to a special point in the code as illustrated in figure 3.5. A possible reason for an interrupt is when the processor is following a program and new input signals are needed. If these signal inputs are available, the input circuits announces this by interrupt to signal the processor that inputs are present. This situation begins an interrupt to the processor. The processor halts on the interrupt and stops to input the required data. After the data is proccessed, the processor continues its task from the place it was interrputed. When an interrupt occurs, the following tasks will be done [6]:

- The program counter as it is after the above instruction is pushed onto the stack.
- The status register is pushed onto the stack.

- If many interrupts occure during the last instruction, the highest priority interrupt is selected.
- Single source interrupts have their interrupt request flags reset automatically. Multiple source interrupt flags do not. The interrupt service routine(ISR) can determine by the flags what has to be done next.
- The status register with the exception of the SCG0 bit is cleared. The SCG0 bit is left unchanged. Clearing the status register leads to terminate any low-power mode. Further interrupts are disabled, since the general interrupt enable bit (GIE) is cleared.
- The counter of the interrupt vector is loaded into the program counter. An interrupt vector is the memory address of an interrupt handler.

MSP has some different kinds of events, which can trigger interrupts. For each of these events the processor sends the execution to an unique, specific point in memory. The interrupts in MSP can be classified into maskable and non-maskable:

- **Maskable Interrupts:** They can be turned off by the programer. The trigger event of the maskable interrupt is usually not important. therefor the programer can decide if the event should cause the program to jump. Generally, the maskable interrupts can be allowed to occur or prevented to occur by software. The processor is able to mask, or temporarily ignore, any interrupt if it is needed.
- **Non-Maskable Interrupts (NMI):** They can not be handeled by software. They are used to report hardware errors that are non-recoverable such as access violation in the flash memory. NMI can be used for serious conditions that demand the immediate attention of the processor. The NMI cannot be ignored by the system unless it is shut off specifically.

The interrupt priority determines which of the interrupt is served first. When two interrupts happen at the same time, the higher priority event is handled first. The interrupt service routine(ISR) is the function to be called when an interrupt happens. It is called when the GIE is set. The interrupt handling routine terminates with the instruction RETI (They return from an interrupt service routine). A special interrupt of MSP430 is the reset interrupt forcing the processor to jump to the beginning of the memory. The interrupt vectors and the power-up starting address are located in the address range 0xFFFF - 0xFFE0. The highest priority interrupt vector starts at address 0xFFFF and the lowest priority interrupt vector is at address 0xFFF0. Tabel 3.2 shows the interrupts and their priorities.

System interrupt	Intrrupt source	Address	Priority
Reset	External rest, Power up Watchdog timer reset, Invalid flash memory activation	0xFFFE	15
Non-maskable	NMI, Oscillator fault Flash memory access violation	0xFFFC	14
Maskable	Timer_B7	0xFFFA	13
Maskable	Timer_B7	0xFFF8	12
Maskable	Comparator_A	0xFFF6	11
Maskable	Watchdog Timer	0xFFF4	10
Maskable	USART0 Receive	0xFFF2	9
Maskable	USART0 Transmit	0xFFF0	8
Maskable	ADC12	0xFFEE	7
Maskable	Timer_A3	0xFFEC	6
Maskable	Timer_A3	0xFFEA	5
Maskable	I/O Port P1	0xFFE8	4
Maskable	USART1 Receive	0xFFE6	3
Maskable	USART1 Transmit	0xFFE4	2
Maskable	I/O Port P2	0xFFE2	1
Maskable	Basic Timer1	0xFFE0	0

Table 3.2: Interrupts and their Priorities

3.2.3 Watchdog Timer

The watchdog timer is a computer hardware timing device that performs a controlled system restart after a certain period of time. It can be used to automatically fix software problems and reset the processor if software problems occur. The watchdog timer is based on a counter that counts down from its intial value to zero. If the defined time interval expires, a system reset is generated. The intention is to bring the system back from the hung state to the normal

How to select timer mode:

```
/* WDT is clocked by fACLK (assumed 32Khz) */
WDTCL=WDT_ ADLY _ 250; // WDT 250MS/4 INTERVAL TIMER
IE1 | =WDTIE; // ENABLE WDT INTERRUPT
```

How to stop watchdog timer:

```
WDTCTL=WDTPW + WDTHOLD ; // stop watchdog timer
```

Assembly programming:

```
WDT_ key .equ 05A00h ; Key to access WDT
WDTStop mov # (WDT_ Key+80h),& WDTCTL ; Hold Watchdog
WDT250 mov # (WDT_ Key+1Dh),& WDTCTL ; WDT, 250ms Interval
```

Table 3.3: Watchdog Timer Programming

state. If the watchdog function is not needed in an application, the module can work as an interval timer, to generate an interrupt after the selected time interval. Special properties of the watchdog timer are:

- There are eight software time intervals to select from.
- There are two operating modes, watchdog mode and timer mode.
- The access to the watchdog timer can only be through a password.
- The expiration of the time interval in watchdog mode generates a system reset.
- The expiration of the time interval in timer mode generates an interrupt request.

Table 3.3 explain how to use watchdog to select a timer mode and how to stop watchdog.

3.3 Embedded Chip Radio (ECR)

In contrast to ESB nodes, ECR nodes shown in figure 3.6 come only with a vibration sensor and many I/O ports.

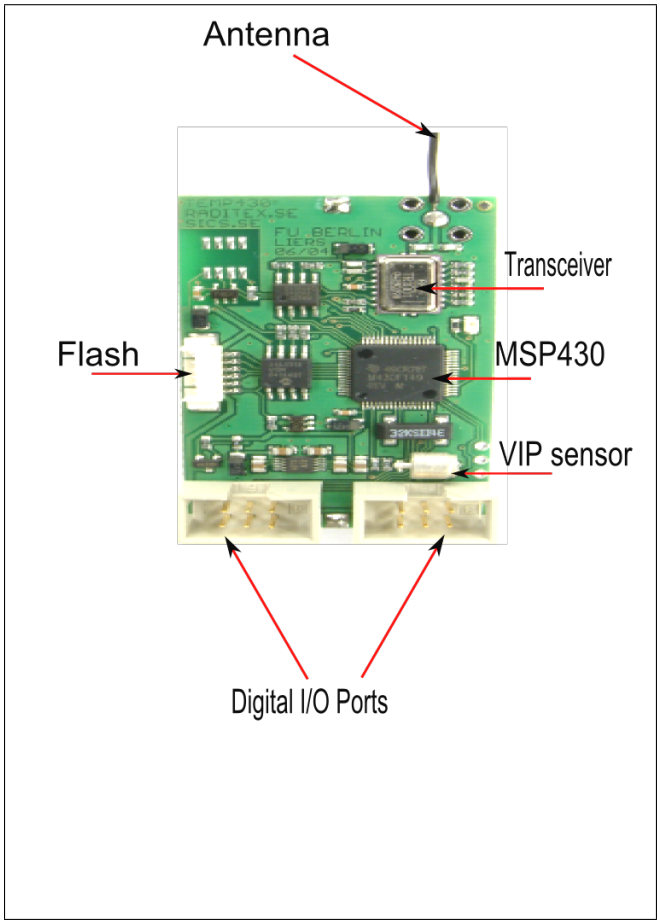

Figure 3.6: ECR Platform.

3.4 eGate/USB Platform

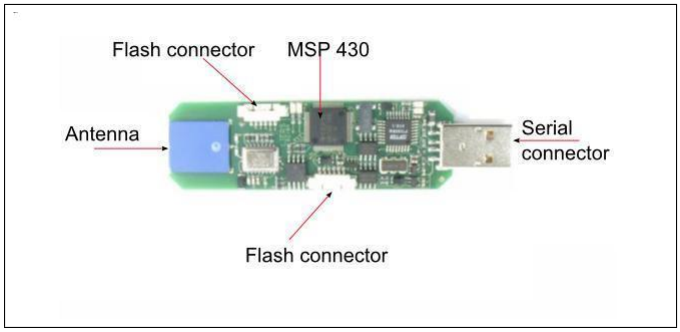

Figure 3.7: eGate/USB Platform.

The eGate/USB platform is a simple USB stick shown in figure 3.7. Attaching the eGate/USB to a PC grants access to the sensor network. It allows debugging and collecting sensor data.

Chapter 4

Data Logging

WSNs introduce several constraints: small memory, limited power supply and computational constraints. In WSNs several distributed sensor nodes are deployed to observe, understand and analyse the physical data. The amount of data, that the sensor nodes sense is growing with the number of sensor nodes in a WSN. therefor the data must be logged while taking into account the infeasibility of a fully centralized data collection. Processing the overall data might not be necessary for specific area of interest. The software of the USB sensor nodes allows a user to initiate, track and schedule experiments. It enables the reprogramming of the sensor nodes and logs the data generated during the experiment.

4.1 Types of Logging

There are two approaches of data logging:

1. **Online Data Logging:** During the experiments the log data is collected by many sensors and sent to the main computer as illustrated in figure 4.1. For example, consider a WSN consisting of hundereds or thousands of sensors. Each sensor senses temperature periodically. During the experiments it keeps sending information to a main computer. A problem in this approach is the huge traffic overhead leading to unwanted energy consumption. This extra traffic creates extra collision and increases packet loss. therefor an offline approach defined below is considered in this project.

2. **Offline Data logging:** In the offline approach each sensor temporary saves the log data locally. After the end of the experiment or after a fixed period of time it sends the log collected to the main computer as illustrated in figure 4.2. Consider the same application described above consisting of a huge number of sensors collecting temperature measurements periodically.

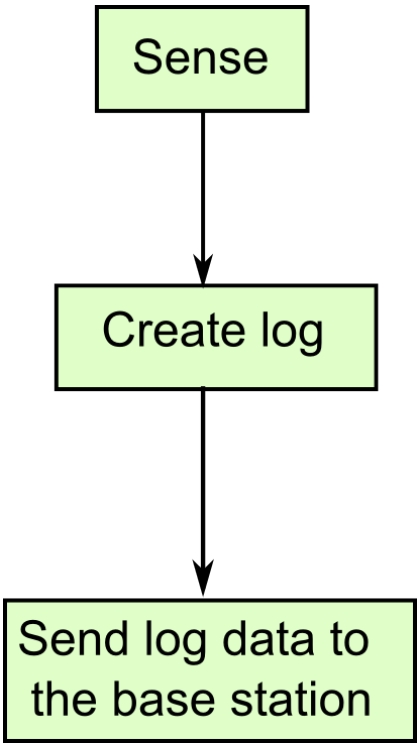

Figure 4.1: Online Data Logging.

In the offline approach each sensor will keep temperature readings locally. After a long time period (e.g. at the end of a day) it will send the data to main computer. This approach is selected in this project instead of the online approach to reduce traffic overhead, collisions and save sensor node's energy.

The sensor nodes collect a great amount of data, only some of it may be useful. therefor data gathered by sensor nodes needs to be controlled and managed to extract useful information based for decision-making. In the offline approach data is sent to the main computer by a sensor. This can be depend on two things.

- **Time period:** After a fixed period of time, for example at each end of the day, a sensor sends data which is collected since previous data despatch is sent to main computer.
- **Trigger event:** In this case, a sensor sends data to the main computer after receiving a trigger event, for example upon receiving a request from the main computer. This trigger event may also indicate what kind of log

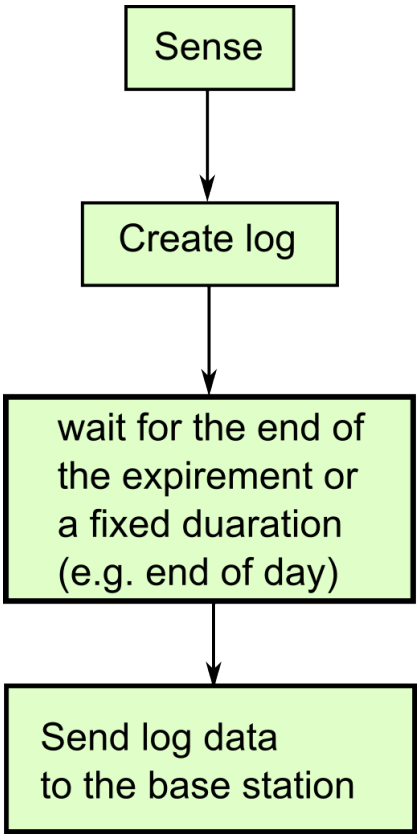

Figure 4.2: Offline Data Logging.

is required and the time interval of the log data needed by the base station. This approach reduces traffic compared to the time period approach.

The data logging could be wireless or wired.

4.1.1 Wired Logging

In case of wired logging each sensor is manually connected with the base station and the base station retrieves log data from the sensors. Advantages of this approach are reliability and energy conservation during experiments. However in a sensor network consisting of a large number of sensor nodes this wired logging is not feasible. Furthermore the approach of trigger events to collect log data can not be wireless. In this case a trigger event could be simply pressing a button on a sensor or a base station sending a request to a sensor attached to it. Wired logging is not feasible when sensors can not be attached with the base computer and only provides a wireless interface.

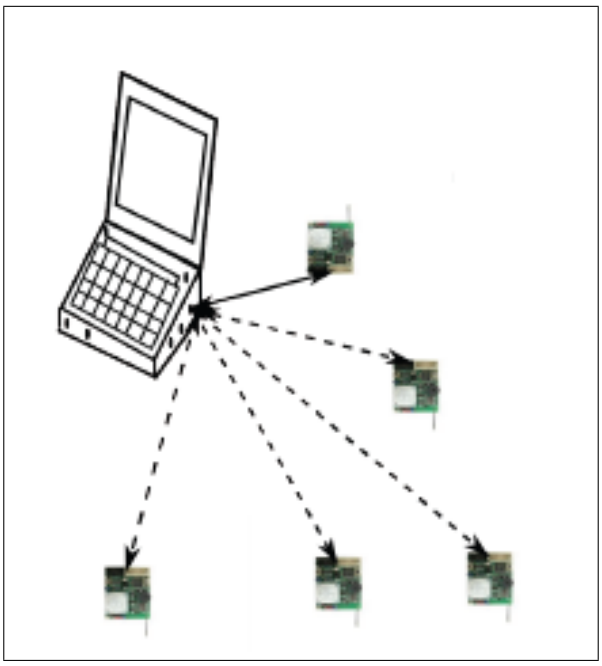

Figure 4.3: Wired Logging.

4.1.2 Wireless Logging

In this case trigger events and log data is sent wireless. First a base station sends trigger events to sensors in its surrounding. Upon receiving a trigger event a sensor could propagate this trigger event to further sensors and send locally saved logs to the base station through multiple hops. Advantages of this approach are feasibility for larger sensor network and less manual efforts involved.
In applications of WSNs the data is the focus of interest, not the source of the data. Typical queries in such aplication are:

- Can you tell me if anyone is going in this room?
- Can you tell me, how many people are going in that room?
- What is the current average humidity in room Y?
- In which room is the temperaure greater than 25 °C?
- Is the light of room y turned on?
- What is the maximum temperature in room Z?
- Tell me in which room is Mr. Hans?

In such applications only the information is of interest, not the source of this information, not the identity of the sensor node and it does not matter how many events occured. Hence, interaction in WSN is data-centric. In this approach the application depends only on the reported data, not on the identity of the sensor nodes that report the data. In this suitation multiple sensor nodes can report the same event. In the first query the task is to announce the presence of an object. Hence, the sensor nodes in the WSN that can sense the objects announce the Interest. The last query's task is to know the location of an object. The sensor nodes should know their location and be able to maintain it. There are some differents between the traditional network and data-centric network. In data-centric networking:

1. The sensor nodes, which report events do not care about the identity of the source node.
2. The sensor nodes are interested only in the offered information and do not care about the source of the information.
3. The sensor nodes, which report events, do not care about the number of the events.
4. The source node can be switched off to save energy once the data transmission is made.
5. The networks comprise of many sensor nodes. Each node can be both a provider and a consumer of information.
6. The sensor nodes, which report events, do not care about the number of the sink nodes.
7. The sink node can be switched off to save energy, untill the data is available.
8. The data should be sent and made available in an asynchronous approach.

4.2 Publish/Subscribe Model

To implement a data-centric network the publish/subscribe model as illustrated in figure 4.4 is used. In this model any sensor node interested in a given type of data subscribes to its publication by a subscribe action. All sensor nodes are connected to a software bus, where data is published by the publisher by a publish action. When the data is published, all subscriber of this type of data are notified about the availability of this data. The publish/subscribe model has three major properties in the relationship between publisher and subscriber of the information:

- **Decoupling in time:** Publication and notification of data can happen at different times. The software bus provides an intermediate storage.

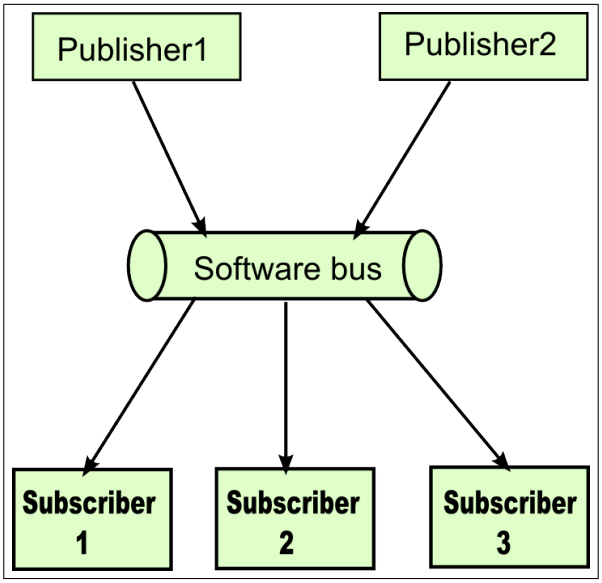

Figure 4.4: Publish/Subscribe System.

- **Decoupling in space:** Publishers and subscribers do not need to know each other. therefor they can ignore their mutual identities.
- **Decoupling in flows:** Interactions with the software bus can happen asynchronously. therefor the operation of the software bus can happen without blocking.

4.2.1 Content-based Naming and Addressing

The question when publishing data or when subscribing to it is:
How to refer to this data?
Refering to data is done via a content-based adressing scheme of the sensor nodes. In WSNs the sensor nodes are not independent. They collaborate together to solve a given task and offer the user an interface to the physical world providing the appropriate data. The user for WSN wants to know data about some physical phenomene and not about individual sensor nodes. therefor the user should be allowed to name the interested data but not the sensor nodes that produce this data. In a WSN, names refer to things such as data, sensor nodes or transaction, addresses provide the needed information to find these things.

Chapter 5

Displaying and Processing Log Data

Each sensor saves data in its memory. This format is not intended for presentation to the end user. The base station collects all data from different sensors and processes the data to make it user friendly. Data representation at the base station should be independend of:

1. The number of sensors,
2. the number of events sensed,
3. the type of sensors and
4. the type of events.

The ScatterWeb command "rld 03" is used to display data. In figure 5.1 and 5.2 the log data from two ESBs is shown. The ID of the first ESB is 4 and the ID of the second is 16. There are other termial commands such as:

- **"dea":** It is used to erase the EEPROM. The whole code of this command can be found in the file ScatterWeb.IO.c.
- **"RST":** It is used to reset the system. The code of this command can be found in ScatterWeb.Messaging.c.
- **"rid":** It is used to read the ID of the sensor node. This command can be found in ScatterWeb.Configuration.c.
- **"sid x":** It is used to set the ID of the sensor node to value **x**. The command can be found in ScatterWeb.Configuration.c.
- **"mem":** It is used to read to display free stack bytes. The command is in ScatterWeb.Messaging.c.

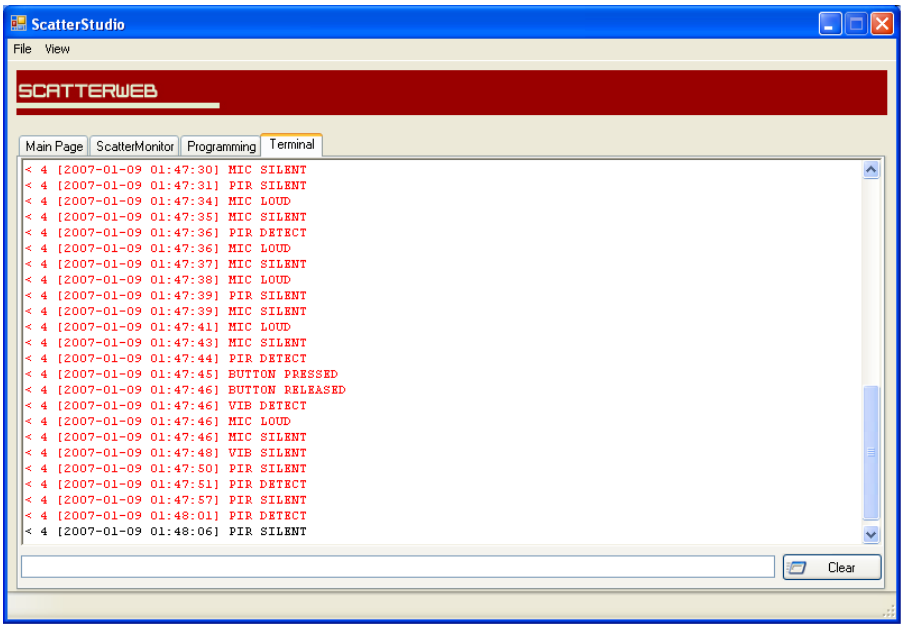

Figure 5.1: Log Data from Sensor Node 4

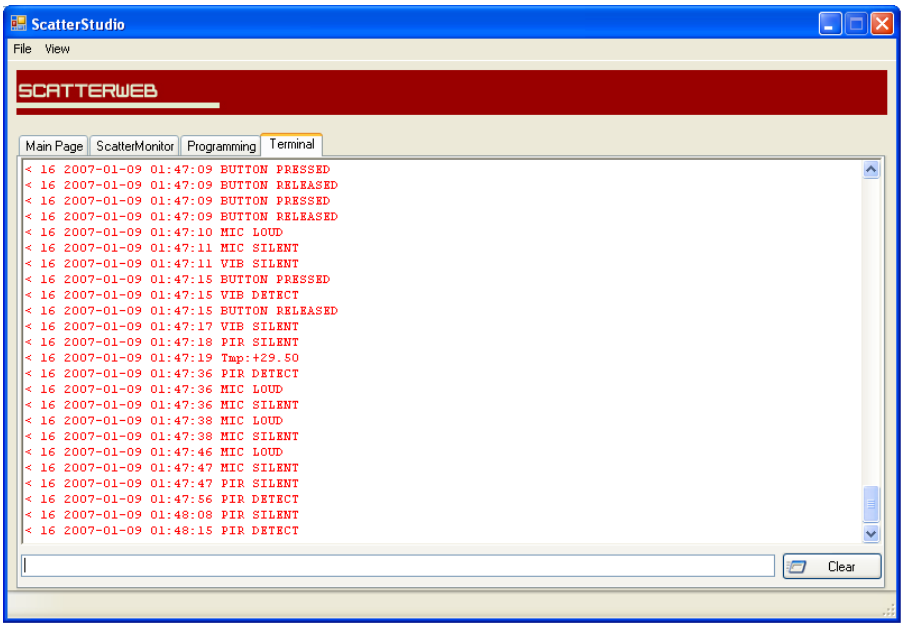

Figure 5.2: Log Data from Sensor Node 16

5.1 How to Write a Command to the Sensor

The ScatterWeb software is divided into two parts [24]: The system and the application. The system handles all interrupts, packet sending, packet receiving and access to the hardware. It initializes the hardware and continuously checks whether sensor events were noticed, whether data arrived at the serial port and data in the outgoing buffers has to be handled. The interaction between the system and the applications is realized by callback functions. The application resembles a userlevel application in full-featured operating systems. It is responsible for registering callback functions with the system when the application is interested in sensor events or packets are arriving in the network. It registers own handler functions as callbacks in the system during the initialization phase:

```
void Process_init() {
  System_registerCallback(C_RADIO, Process_radioHandler);
  Event_init();
  Comm_log(LOW, "| Application %s initialized\r\n\r\n",
                  imageData.versionName);
  System_registerCallback(C_PERIODIC, periodic_function);
}
```

The **periodic$_{\mathbf{f}}$unction**() function has to be implemented in the application and is linked with the system after compilation.

```
void periodic_function() {
    Data$_redOn();
    Data_beeperOn();
    System_wait(50000);
    Data_beeperOff();
    Data_redOff();$
}
```

This code example above switches the red light on and lets the sensor beep for 50000 milliseconds. After this time the code stops the beeper and the red light off. The command "**rLi**" has to be added using this macro:

```
COMMAND(rLi,0){
periodic_function();
}
```

5.2 Sending and Receiving Information in a Simple Alarm Application

When anyone is going, the sensor switches the red, green and yellow lights on. In addition, the sensor beeps to alarm. With lights and beeper on, alarm continues for one minute. In the example movement is handled by the **handleEvent** function. The **handleEvent** function is implemented in ScatterWeb.Event.c. In the example above it looks like:

```
void handleEvent(sdata_t* data) {
    if(data->sensor==SENSOR_MOVEMENT && data->value==1){
       Comm_print("\%s","Alarm,
          someone is going now in the room" );
       Data_redOn();
       Data_greenOn();
       Data_yellowOn();
       Data_beeperOn();
       System_wait(60000);
       Data_beeperOff();
       Data_redOff();
       Data_yellowOff();
       Data_greenOff();
       sendConnectivityTestPacket();
   }
}
```

Parameter of this function is a structure called **sdata_t**. This structure is defined in ScatterWeb.data.h.

```
typedef struct {
  UINT16 id;
  UINT8 sensor;
  time_t timeStamp;
  UINT16 value;
} sdata_t;
```

UNIT8, UNIT16 and UNIT32 are elementary data types defined in ScatterWeb.System.h.

```
typedef unsigned char     UINT8;
typedef unsigned int      UINT16;
typedef unsigned long     UINT32;
```

The structure **time_t** is defined in ScatterWeb.time.h.

```
typedef struct {
  UINT32 secs;
  UINT16 millis;
} time_t;
```

The **handleEvent** function calls another function.
The function **sendConnectivityTestPacket**() is implemented in the same file ScatterWeb.Event.c.

```
void  sendConnectivityTestPacket(){
         struct ConnectivityTestPacketStruct ConnectivityTestPacket;
         ConnectivityTestPacket.type=0;
         ConnectivityTestPacket.src=Configuration.id;
         packet_t p;
         p.to = BROADCAST;
         p.type = CONNECTIVITY_TEST_PACKET;
         p.header=0;
         p.header_length = 0;
         p.data = (UINT8*)&ConnectivityTestPacket;
         p.data_length = sizeof(ConnectivityTestPacket);
         Net_send(&p, NULL);
    }
```

The structure **ConnectivityTestPacketStruct** is implemented in ScatterWeb.Event.h as.

```
struct ConnectivityTestPacketStruct{
  char type;
  unsigned int src;
 };
```

The structure **packet_t** is defined in ScatterWeb.Net.h as:

```
typedef struct {
  UINT16 to;
  UINT16 from;
  UINT8 type;
  UINT8 num;
  UINT8* header;
  UINT16 header_length;
  UINT8* data;
  UINT16 data_length;
} packet_t;
```

The Bitmasks **BROADCAST** and **CONNECTIVITY_TEST_PACKET** are defined in ScatterWeb.net.h.

```
#define BROADCAST                       (0xFFFF)
#define CONNECTIVITY_TEST_PACKET        (0x0c)
```

or any other unused Bitmaskocation. The **Net_send** function will be used to send the packet. This function is defined in ScatterWeb.net.c. It should be checked if the packet **CONNECTIVITY_TEST_PACKET** arrived to take the appropriate actions. Checking the packet type will be implemented in ScatterWeb.Process.c with the case-statement in the **Process_radioHandler**.

```
case CONNECTIVITY_TEST_PACKET:
      Comm_print("%s","Connectivity Test packed received");
      handle_connectivity_test_packet((unsigned int*)&rxPacket.from,
      rxPacket.data);
      break;
```

The appropriate actions are defined in the **handle_connectivity_test_packet** function.

```
int handle_connectivity_test_packet(unsigned int* from, char* payload){
struct ConnectivityTestPacketStruct* ConnectivityTestPacket;
  ConnectivityTestPacket = (struct ConnectivityTestPacketStruct*)payload;
  Comm_print("%s","Connection test Packet from: ");
  Comm_print("%i",*from);
  Comm_print(",from field in payload:");
  Comm_print("%i\r\n", ConnectivityTestPacket->src);
  Data_redOn();
  Data_greenOn();
  Data_beeperOn();
  System_wait(60000);
  Data_redOff();
  Data_greenOff();
  Data_beeperOff();
  return 1;
}
```

The *rxpacket* is defined as a structure in *Scatterweb.net.c* as:

```
volatile packet_t rxPacket;
```

While the structure **packet_t** structure is defined in ScatterWeb.net.h.

```
typedef struct {
  UINT16 to;
  UINT16 from;
  UINT8 type;
  UINT8 num;
```

```
  UINT8* header;
  UINT16 header_length;
  UINT8* data;
  UINT16 data_length;
} packet_t;
```

5.3 Getting Data from the Sensors with Java Serialization

The Java Communications API is located in the *javax.comm* package. It enables to write Java software accessing communication ports in a platform-independent way [31][48]. The JavaComm API is not part of the standard Java 2 distribution. The implementation of the API has to be downloaded separately. The first three things to program the serial lines in the Java communication API are typically to the following:

- Enumerate all serial ports (port identifiers) available to the Java communication API.
- Select the desired port identifier from the available ones.
- Acquire the port via the port identifier.

CommPortIdentifier is the central class to control access to communication ports. It includes methods to:

- Determine the communication ports made available by the driver.
- Open communication ports for I/O operations.
- Determine port ownership.
- Resolve port ownership contention.
- Manage events that indicate changes in port ownership status.

After defining port configuration variables, the **CommunicateWithSerialPort** program constructs a $javax.comm.CommPortIdentifier$ object to handle the serial communications port.
The **CommunicateWithSerialPort** program calls the open method on that object to return an object of the $javax.comm.SerialPort$ class. This object describes the low-level interface to the $COM1$ serial port, assumed to be connected to the ESB. After this the program calls several methods on the $SerialPort$ object to configure the serial port. The program uses the I/O package *java.io* to write to and read from the serial port [20]. It calls a static method to return a $PrintStream$ object bound to the serial port. This object is passed to the

constructor of *PrintStream*. *write*() methods on the *PrintStream* object are used to write a command to the serial port and acquire the log data. An object of the *BufferedReader* class is used to read from the serial port. In details, communication with the serial port is set up in the following steps

- **Define variable for serial port configuration and output:**
  The configuration of the serial port is made with the parameters of the method *SetSerialPortParams*. The first parameter defines the baud rate to be 115200. The second parameter defines number of data bits to be 8. The third defines the number of stop bits to be 1. The fourth parameter defines parity to be off.

```
public void initSerialPort(SerialPort port) throws Exception {
        port.setSerialPortParams(115200, SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
    }
```

- **Create a CommPortIdentifier object:**
  The *javax.comm.CommPortIdentifier* class has static methods that return an instance of the class. The *CommunicateWithSerialPort* uses the *getPortIdentifier* to return a *CommPortIdentifier* object for port *COM*1.

```
CommPortIdentifier comId = connect.getPortId("COM1");
```

- **Open the serial port:**
  The *CommunicateWithSerialPort* program opens the serial port by calling the *open*() method on the *CommPortIdentifier* object **portId**. The *open*() call returns a *SerialPort* object, assigned to **port**. The first argument in the *open*() method is the name for the port, and the second argument is the number of milliseconds to wait for the port to open the connection.

```
 public SerialPort OpenPortCom(CommPortIdentifier portId) {
        SerialPort port = null;
        try {
            port = (SerialPort) portId.open( applicationName,
               // Name of the application asking for the port
                    10000 // Wait max. 10 sec. to acquire port
                  );
        } catch (PortInUseException e) {
            System.err.println("Port already in use: " + e);
            System.exit(1);
        }
        return port;
    }
```

- **Set up a PrintStream object to write to the ESB:**
  The *CommunicateWithSerialPort* program calls the constructor to create and open a *PrintStream* object. The constructor call passes the *OutputStream* object, returned by a call to the *getOutputStream*() method and assign the *PrintStream* object to the instance **os**.

  ```
  PrintStream os = new PrintStream(port.getOutputStream(), true);
  ```

- **Write command to the serial port and close print stream:**
  The *CommunicateWithSerialPort* program writes a string command to the serial port, by calling the *write*() method of the instance **os**. the command "**rld03**" is used to get log data from ESB.

  ```
  os.print("rld 03");
  ```

  The *CommunicateWithSerialPort* program then calls *close*() to close *os*.

  ```
  os.close();
  ```

- **Open an input stream:**
  The *CommunicateWithSerialPort* program reads the log data expected from the ESB in response to the given command through opening an input stream. The program opens an input stream by calling the static method, *getInputStream*(), to obtain an instance InputStream from the serial port.

- **Create an input stream reader for the serial port:**
  The **CommunicateWithSerialPort** program calls the constructor of *InputStreamReader*, with the *InputStream* object and assigns the new object to an instance of *BufferedReader* **is**.

  ```
  BufferedReader is = new BufferedReader(new InputStreamReader
                      (port.getInputStream()));
  ```

- **read data from serial port and close the buffer:**
  The *CommunicateWithSerialPort* program reads from the serial port, by calling the read method of the *BufferedReader* object **is** byte by byte.

  ```
  try {
        char c =(char) is.read();
        while (c != '\0' || c != '\n' || c != -1 ) {
          out.write(c);
          System.out.println(c);
          c = (char)is.read();
         }
       }catch (IOException ex) {
          ex.printStackTrace();
        }
  ```

After reading the data, the program calls *close*() method on **is** to close the $BufferedReader$ object **is**.

```
is.close();
```

- **Close the serial port:**
  The program closes the serial port, by calling *close*() on the $SerialPort$ object.

```
port.close();
```

5.4 XML Representation

In this work the base station converts data from the sensors into Extensible Markup Language (XML). While HTML was designed to display data and to focus on how data looks on the screen, XML was designed to described data in details [10][38][66]. A XML document must adhere to the following rules to be well-formed:

- XML document must have only one root element.
- Every start-tag must have a matching end-tag.
- XML tags can not overlap.
- XML tags are case-sensitive.
- XML attribute values must always be quoted.
- XML will keep white space in the text.
- XML element names must obey XML naming conventions.

A sample xml file for this project is look like that:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="dip.xsl"?>
<Sensor sID="1747"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="dip.xsd">
<Event eID="0">
<Time>
  <time>16:32:25</time>
  <date>2001-03-29</date>
</Time>
<EventName>PIR</EventName>
<state>DETECT</state>
```

```
</Event>
<Event eID="0">
 <Time>
  <time>16:32:28</time>
  <date>2001-03-29</date>
 </Time>
 <EventName>Tmp</EventName>
 <state>+23.00</state>
</Event>
<Event eID="0">
 <Time>
  <time>16:32:30</time>
  <date>2001-03-29</date>
 </Time>
 <EventName>PIR</EventName>
 <state>SILENT</state>
</Event>
</Sensor>
```

An XML Schema is required to describe the structure of the XML document above. The schema has to define the following:

- The elements and attribute that can appear in a document.
- The child elements for a particular element.
- The number and the order of the child elements.
- Whether an element is empty or can include text.
- Define the data types for elements and attributes.

The XML Schema is independent of the number of sensors and the kind of sensed events. If the number of sensors increase or their capability to sense the environment change, no further change are required to the XML Schema. The XML Schema for log data looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3schools.com"
  xmlns="http://www.w3schools.com"
  elementFormDefault="qualified">
<xs:element name="Sensor">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Event" maxoccurs="unbounded">
         <xs:complexType>
            <xs:sequence>
              <xs:element name="Time" maxoccurs="unbounded">
```

```
                <xs:complexType>
                  <xs:sequence>
                   <xs:element name="time" type="xs:time"/>
                   <xs:element name="date" type="xs:date"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="EventName" type="xs:string"/>
              <xs:element name="State" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

5.5 Creating Dynamic HTML

To present the end user a log data, easy to understand and user friendly XML is converted dynamically to HTML. The dynamic generation of HTML from XML is done using Extensible Stylesheet Language (XSL) as shown in figure 5.3. It is used to define style and data represented in XML. If the data changes or increases, the XSL document remains same. XML does not have a fixed tag set like HTML and does not have semantics by itself [32] [53]. The XML document does not say anything about the way in which a particular piece of information should be presented. There is no formatting information included. For example, the Sensor element in XML document is not understood. It can mean any thing. XSL describes how the XML document should be displayed. The goal of XSL is to develop a stylesheet language that creates the desired output. XSL consists of three parts:

- XSLT: XSL Transformations(XSLT) is a language for transforming XML documents into another. A transformation expressed in XSLT describes rules to transform a source XML tree into a result tree.
- XPath: A language to navigate through elements and attributes in XML documents.
- XSL-FO: A language for formating XML documents.

Since a XSL stylesheet is a XML file itself, the file begins with an xml declaration. The xsl:stylesheet element indicates that this document is a stylesheet. The template has also been wrapped with xsl:template match="/" to indicate that this is a template that corresponds to the root (/) of the XML source document.

The root element is the Sensor in the XML document shown in the beginning of this chapter. The XSL document transform the log data into HTML table looks like:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
      <head>
        <title>Log Data</title>
      </head>
      <body>
        <h3>Log Data</h3>
        <table border="6"  width="750">
         <tr bgcolor="#9ACD32">
           <th>Event Name</th>
           <th>Event Time</th>
           <th>Event Date</th>
           <th>State</th>
         </tr>
         <xsl:for-each select="Sensor/Event">
          <tr>
          <td bgcolor="#C6DEFF"><xsl:value-of select="EventName"/></td>
          <td bgcolor="#00FFFF"><xsl:value-of select="Time/time"/></td>
          <td bgcolor="#FAF8CC"><xsl:value-of select="Time/date"/></td>
          <td bgcolor="#ff00ff"><xsl:value-of select="state"/></td>
          </tr>
         </xsl:for-each>
        </table>
      </body>
    </html>
```

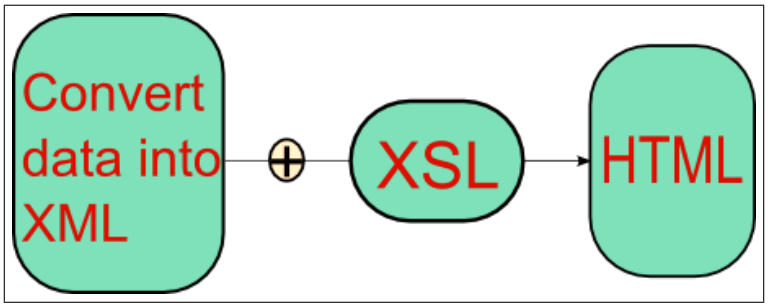

Figure 5.3: Creating HTML.

```
</xsl:template>
</xsl:stylesheet>
```

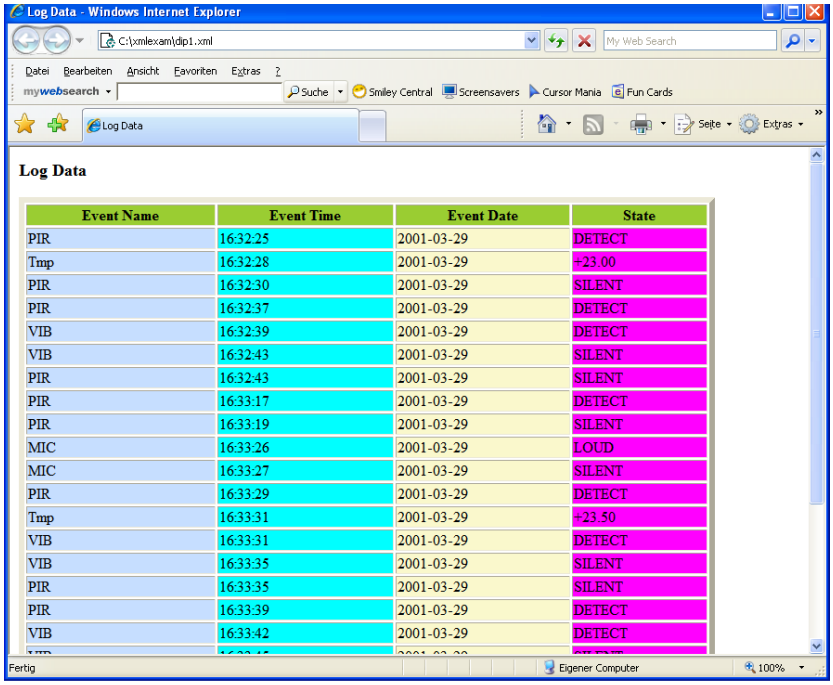

Figure 5.4: Presenting XML Log Data in Table

Separating stylesheet (XSL) from data (XML) offers the benefit to present the data in different ways. A XSL stylesheet basically consists of a XSL transformation templates. Each template matches some set of elements in the source tree and describes the contribution that the matched elements make in the result tree. A template is instantiated for a particular source element to create part of the result tree. A template can contain elements that specify literal result element structure. When a template is instantiated, each instruction is executed and replaced by the result tree fragment that it creates. Instructions can select and process descendant source elements. Processing a descendant element creates a result tree fragment by finding the applicable template rule and instantiating its template. The elements are only processed when they have been selected by the execution of an instruction. The result tree is constructed by finding the template rule for the root node and instantiating its template. The **xsl:for-each** is an iterator which processes each of the selected nodes using the template that it contains. The **xsl:value-of** element inserts the string value of an expression into the result tree. Figure 5.4 illustrate the result presentation

from the HTML document generated. The combination of XML and XSL generate dynamic and data independent HTML. The whole process is illustrated in figure 5.5. To convert log data from sensor nodes into XML file, the following three classes are implemented:

1. **GenXMLFromtext**
2. **Sensor**
3. **Event**

The main class is **GenXMLFromText**. It has five methods: The constructor **GenXMLFromFile**(), **parseText**(), **WriteInFile**(), *toString*() and **main**(). This class aggregates a *HashMap* keeping the **sensors** instance.

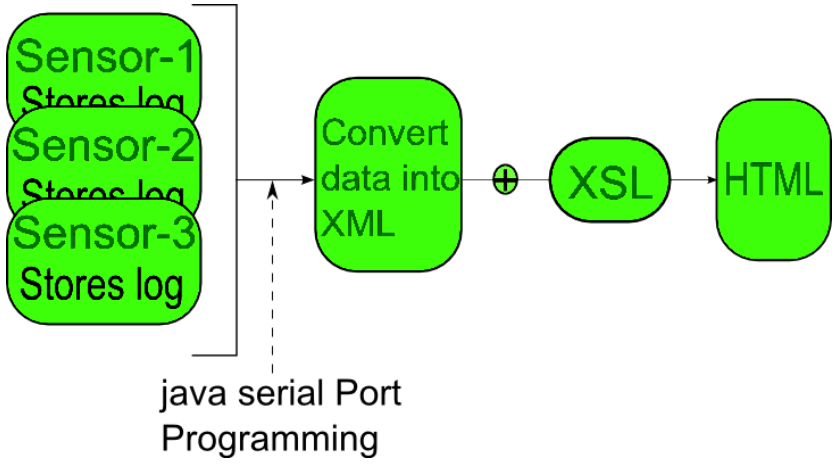

Figure 5.5: Creating HTML from Log Data

The constructor **GenXMLFromFile**() takes three parameters: The path of the schema, the log data file generated from the sensor node and the name of the output XML file. It adds the path of the schema to be linked to the XML file. The **parseText**() method opens the log data file and reads each line. It uses the Java *StringTokenizer* class to break a string into tokens according to the data of the event (time, date, name, data). According to the kind of the data in the token, *parseText*() calls a particular method from the class **Sensor** or the calss **Event** to handle this token. The *parseText*() method looks like:

```
public void parseText() throws Exception {
   RandomAccessFile file = new RandomAccessFile(LogDataFile, "r");
   String line = "";
   StringTokenizer token = null;
   String tokenNext = "";
```

```
  Event evt = null;
  while (file.getFilePointer() < file.length()) {
    line = file.readLine();
    if (line.length() == 0) {
      continue;
    }
    token = new StringTokenizer(line, " ");
    System.out.println(line);
    Sensor sensor = null;
    int size = token.countTokens();
    for (int loop = 0; loop < size; loop++) {
      tokenNext = token.nextToken();
      switch (loop) {
        case 0:
          sensor = (Sensor) sensors.get(tokenNext);
          evt = new Event();
          if (sensor == null) {
            sensor = new Sensor(tokenNext);
            sensors.put(tokenNext, sensor);
          }
          sensor.addEvent(evt);
          break;
        case 1:
          tokenNext = tokenNext.substring(1);
          evt.date = tokenNext;
          break;
        case 2:
          tokenNext = tokenNext.substring(0, tokenNext.length()-1);
          evt.time = tokenNext;
          break;
        case 3:
          evt.name = tokenNext;
          break;
        case 4:
          evt.data = tokenNext;
          break;
        case 5:
          evt.eventId = Integer.parseInt(tokenNext);
        break;
      }
    }
  }
}
```

5.6 Remote Log Display

As explained above for the offline log a trigger event could be used. Sensors which receive this event send their logs towards the base station and propagate this event to other sensors. This event could be generated by an operator on the base station or in regular intervals. In case there is no operator near base station and log needs to be displayed at a remote computer, a new mechanism is required. This mechanism is outlined below.

- Communication with the base station is done using a cellular-phone or internet.
- Upon receiving a message, the base station generates a trigger event.
- Sensors receiving the trigger event send their logs towards the base station and propagate trigger events to their neighbors.
- The base station generates XML and converts it into HTML.
- HTML is viewed by users, who do not need to be near the base station, accessing the web-server running on the base station.

The procedure described above enables end users to view the log data at different location only in need of an internet connection. This eliminates the need of being present near the sensor network.

Chapter 6

Memory Management in the Application

Sensors are inexpensive devices with tiny memory. Proper memory management is necessary for an application. In the last chapter it is explained, how to display and processing the log data. In this chapter memory management techniques are presented. Log data contains entries of the structure:

```
struct {
    int logType;      //log type are defined in a separate file for
                      //example 49 means temperature
    Time time;       //time at which log is generated
    Data date;       //date at which log is generated
    vector data;     //data associated with log.
} logEntry;
```

6.1 Block Reservation

In case a new sensing event occurs and no space is left to log data, some previous log entries have to be overwritten. In the following text two log strategies are explained: FIFO and Priority based overwriting.

6.2 FIFO Log Overwriting

In the FIFO (First In First Out) log overwriting scheme, the oldest log entry is overwritten by an new log entry [7] [54]. This scheme has two advantages: It is easy to implement and needs little processing. It can be implemented using a list only keeping track of two pointers. One pointer points to the head of the

list and the other points to the bottom. When the bottom pointer reaches the reserved memory limit for logging, it is set to the head pointer. New log entries overwrite the oldest log entries. The scheme is illustrated in figures 6.1, 6.2 and 6.3.

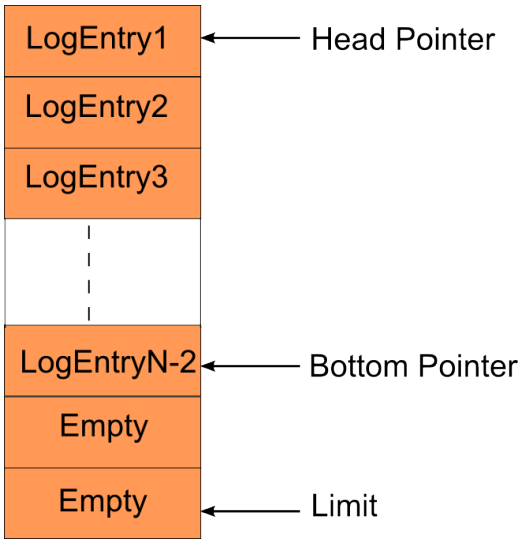

Figure 6.1: Memory Block with n-2 Log Entries

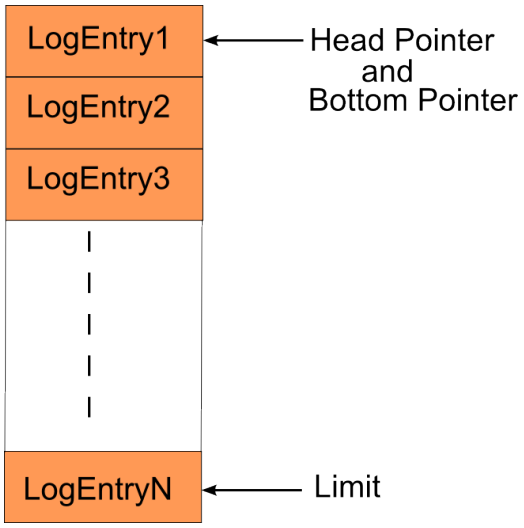

Figure 6.2: Memory Block with N Log Entries

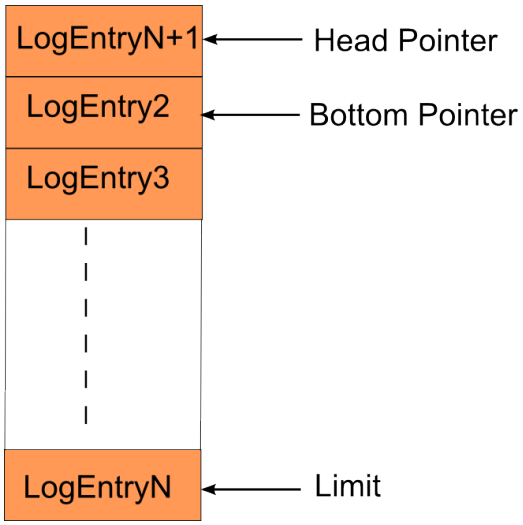

Figure 6.3: Memory Block with N+1 Log Entries

A further advantage of this scheme is that no traversing of the list is required. Overwriting is done in O(1) time. The log data entries are stored in the reserved memory block with n entries. The log data entries are stored according to their arrival time.

6.3 Priority Based Log Overwriting

Concernning the possibility that some log entries are more important than others, FIFO-Log overwriting might not always be the best strategy. The FIFO-Log overwriting scheme does not care about the importance of log entries in the application. Log entries are always replaced, when a new log entry arrives. If the oldest log entry has higher priority than the new log entry, it is still replaced. Priority based log overwriting scheme comprises an alternative strategy in this situation. When memory block reservation has reached its limit, the new log entry is written by overwriting the oldest log entry with lesser priority. This log overwriting scheme is illustrated in figure 6.4 and 6.5. When environmental variables are collected by sensors distributed over the area, reading of these sensors must be transmitted to a base station for further processing. We assume wireless communication and only a limited number of sensors in the range of the base station. The size of the sensor node's memory is only 64 kbyte. Events occured are stored in the sensor node memory. When memory is full and a new event ocurres, an old log entry has to be overwitten by the new data. The oldest log entry is choosen to be overwritten by the new log entry. In figure 6.4, log entry number 1 is the oldest. The head pointer points to the oldest log entry. Depending on the priority of the new event, the oldest log entry is overwritten

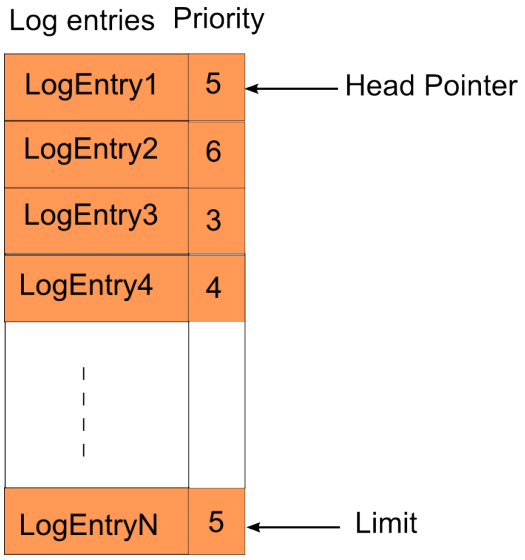

Figure 6.4: Memory Block with N Log Entries.

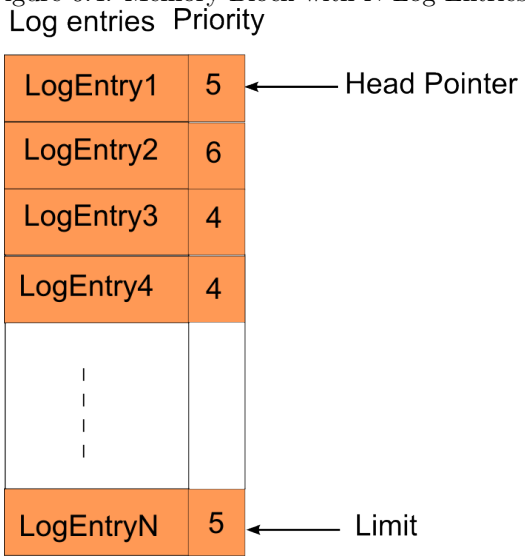

Figure 6.5: Memory Block after Overwriting wit log entry 4.

if its priority is smaller. Another example is illustrated in figure 6.5. The new event has priority 4, log entry number 3 is overwritten with the new log entry. In a worst case, an entry with smaller priority is stored in the last position of the memory. Since this entry is the only, which has a smaller priority (2) than

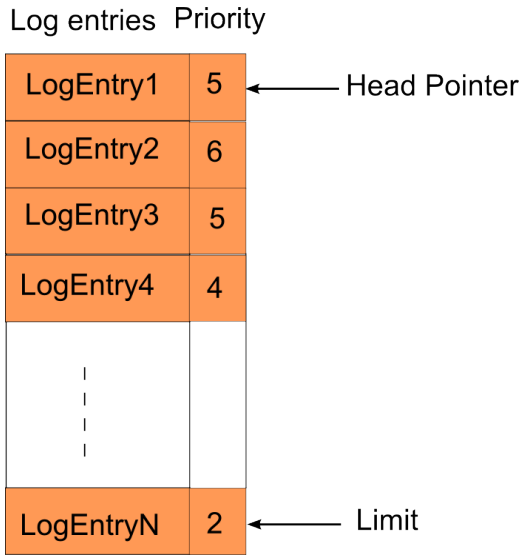

Figure 6.6: Memory Block in the Worst Case.

the new entry's priority (3) as illustrated in figure 6.6. In this case the whole memory is searched without finding a place for the new entry in O(n) time.

6.3.1 Implementation of Priority Based Log Overwriting

The time complexity of priority based log overwiting can be reduced to O($log(n)$). Two implementations, a priority queue based and a priority linked list based are explained below.

Queue Based Implementation

A priority queue is an abstract data type (ADT). Each element has an associated priority. The dequeue operation always removes the lowest (or highest) priority item remaining in the queue [7] [12] [14] [47] [49] [58]. Data structure supporting these operations is called a min(max) priority queue. Priority queues can be implemented by a simple unordered linked list. Log entries are compared by attributes called keys. The Keys are assigned to each log entry in the log data. Keys are used to identify or weight entries. Keys assigned to an log entry must not necessarily be unique. A comparsion rule ($\leq$) is needed for priority queue that will never contradict itself. therefor the comparison rule must define a total relation. A relation is total if for every pair of keys, it satisfies the following properties:

- **Reflexive:** $k \leq k$.

- **Antisymmetric:** if $k_1 \leq k_2$ and $k_2 \leq k_1$, then $k_1 = k_2$.
- **Transitive:** if $k_1 \leq k_2$ and $k_2 \leq k_3$, then $k_1 \leq k_3$.

A priority queue supports the following operations:

1. **size()**, which returns the number of entries in the queue.
2. **Insert(k, e)**, which inserts an entry e with key k in the priority queue.
3. **min()**, which returns an entry with smallest key.
4. **deleteMin()**, which finds, returns and removes the minimum entry in the priority queue.
5. **isEmpty()**, which tests whether the queue is empty.

Event	Priority
MOVEMENT	6
VIBRATION	5
BUTTON PRESSED	4
MICROPHONE LOUD	3
SENSOR LIGHT	2
SENSOR TEMPERATURE	1

Table 6.1: Events and their Properties.

Operation	Priority Queue
insert(3, BUTTON PRESSED)	{(3, BUTTON PRESSED)}
insert(5,VIBRATION)	{ (3, BUTTON PRESSED), (5, VIBRATION)}
insert(1, SENSOR TEMPERATURE)	{(1, SENSOR TEMPERATURE), (3, BUTTON PRESSED), (5, VIBRATION)}
deleteMin()	{(3, BUTTON PRESSED), (5, VIBRATION)}
deleteMin()	{(5, VIBRATION)}

Table 6.2: Operations and their Effects in a Priority Queue.

Note that one type of log entry could occure many times. In this case, the priority number and the time constitute the whole priority of the log entry. For log entries with the same priority, a second comparsion on the time stamp is carried out.
Table 6.1 shows events coded in log entries and their priorties. Table 6.2 shows in a series of operations. A binary heap could be used to implement the priority queue. A binary heap has the following structure properties:

- A heap is a complete filled binary tree with the exception of the bottom level. The bottom level is filled from left to right.
- A binary tree has between 2^h and $2^{h+1} - 1$ nodes, where h is the height of the tree.
- The height h of a complete binary tree is $\lfloor logN \rfloor$.
- A complete binary tree can be represented in an array and no links are needed.
- For any element in array position i, the left child element is in array position 2i and the right child is in array position 2i+1.

Figure 6.7 shows a binary tree and its array implementation.

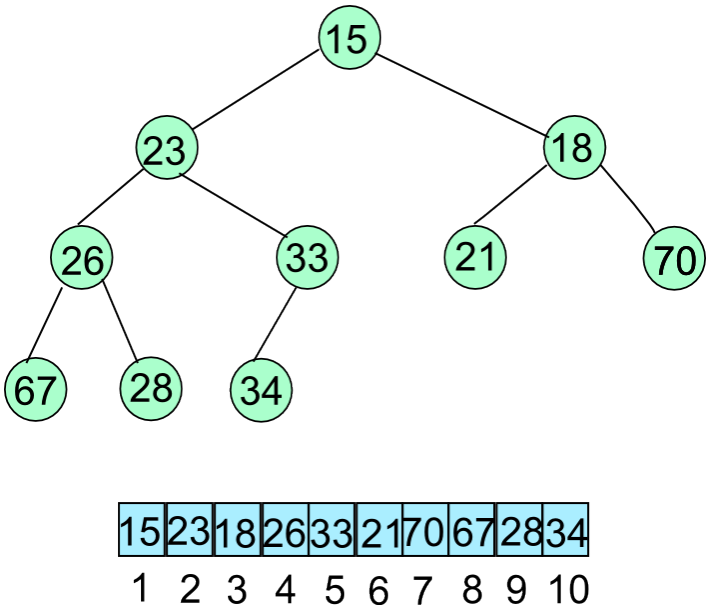

Figure 6.7: A Binary Tree and its Array Implementation.

A binary heap allows operations to be performed quickly. When searching minimal entries, it makes sense that the smallest entry should be at the root. The binary tree has the following order properties:

- There is a root node, which does not have parents.
- Every node except the root node, has a key greater than (or equal to) the key in its parent.
- The minimal entry can be found at the root node in constant time.

Insert operation:
Consider a binary heap with N entries. After insertion of a new entry, it will have N+1 entries. To insert an entry into the binary heap, a node is added so that the resulting tree is a compelete binary tree with N+1 nodes. If the new entry can be placed in the new node without violating heap order, then there is nothing left to do. Otherwise the entry's key is greater than its parent's key. In this case, keys are swapped between the two nodes (i.e. the entry and its parent). This process is repeated from the bottom toward the root until the order properties are perserved. Figure 6.8 illustrates the initial heep. Figure 6.9 and 6.10 illustrate the procedure to place a new entry **e** with key 6 into the heap.

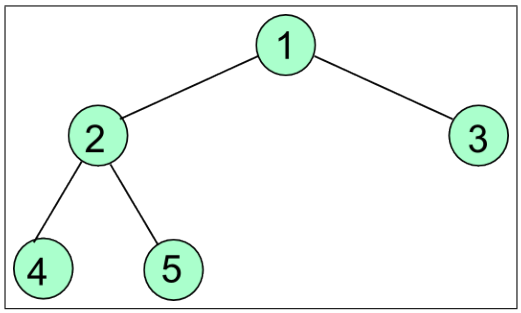

Figure 6.8: Initial Heap before Insertion

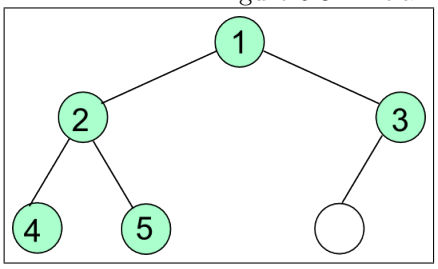

Figure 6.9: Finding Place e=6

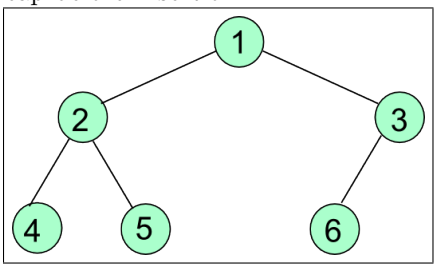

Figure 6.10: Inserting e=6

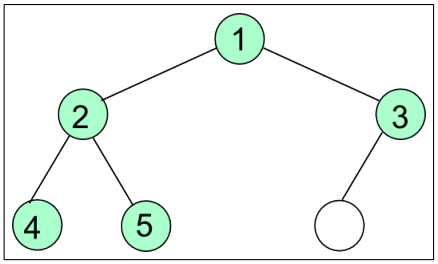

Figure 6.11: Finding Place for e=2

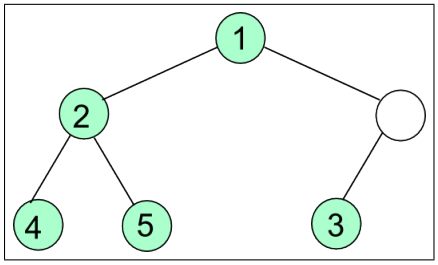

Figure 6.12: Finding Place for e=2

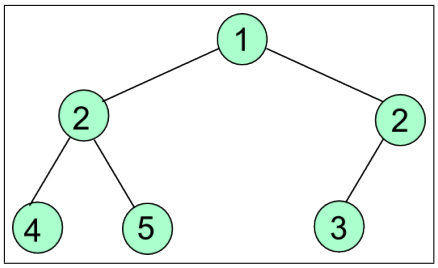

Figure 6.13: Inserting e=2

In figure 6.11, 6.12 and 6.13 a new entry **e** with key 2 is inserted into the initial heap from figure 6.8. Figure 6.14, 6.15 and 6.16 illustrate how to place a new entry **e** with key 1 into the initial heap from figure 6.8. It is supposed that a new entry occures after the first entry is placed in the initial heap.

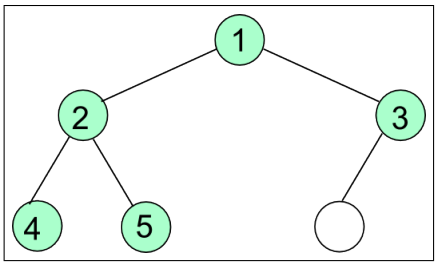

Figure 6.14: Finding Place for e=1

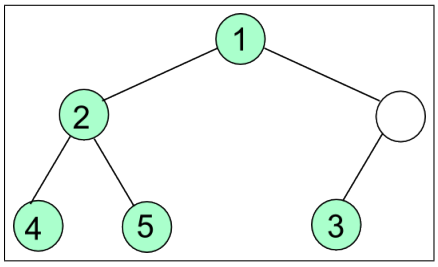

Figure 6.15: Swapping Keys

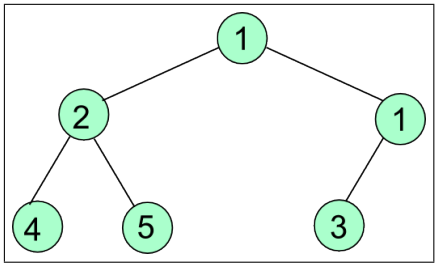

Figure 6.16: Inserting e=1

Delete operation:
The entry to be deleted (i.e. the minimum key in the binary heap) is removed from the root node. When the minimum is removed, a hole is created at the root. The heap's size becomes now one smaller and the last entry **e** in the heap must move somewhere in the heap to satisfy the heap properties. Since the binary tree must be restructured to become a complete binary tree with N-1 entries. The entry in the last node is deleted and placed into the root. If this entry has a key less than or equal to either of the root's two childern, nothing is left to do. If this entry has a key greater than or equal to either of the root's two childern, the heap properties are broken. In this case, we slide the smaller of the root's childern into the hole, thus pushing the hole down one level. This step is repeated until the new entry can be placed into the hole. Figure 6.17 shows an initial heap prior to deleteMin. Figure 6.18. 6.19, 6.20 and 6.21 show the whole deletion process step by step. In the worst case, the deletion algorithm moves down the heap from the root to its leafs spending O(1) time at each level. For a heap with n elements, this takes O($log(n)$) time. In figure 6.16 a heap prior to the deleteMin is shown. After the entry 1 is removed, entry 5 must be moved to the right place in the heap. The entry 5 can not be placed in the hole, sliding the hole down one level. This operation is repeated again and since entry 5 is larger than entry 2, entry 2 is placed into the hole and this creates a new hole one level deeper. Then entry 4 is placed into the hole and a new hole is created

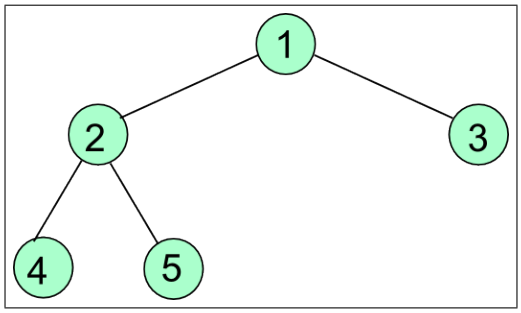

Figure 6.17: Heap before Deletion

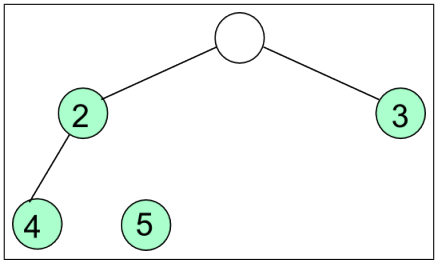

Figure 6.18: Deletion in the Heap

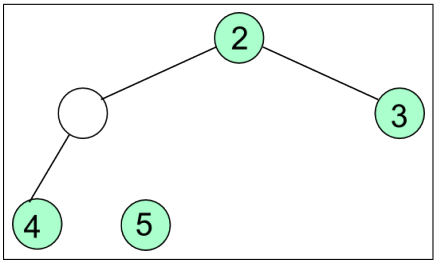

Figure 6.19: First Swapping Keys

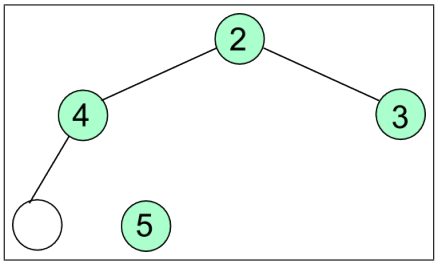

Figure 6.20: Second Swapping Keys

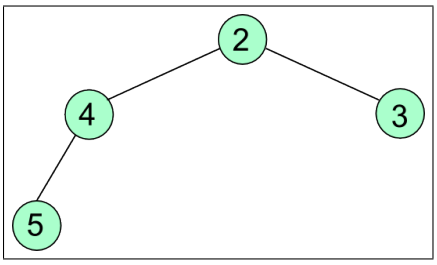

Figure 6.21: Final Heap

in the bottom level. Since once again, entry 5 it too large. Finally, entry 5 is placed in the hole at the bottom level. This general strategy is known as a percolate down.

6.4 Binary Tree and Linked list based implementation

A binary tree and a linked list based implementation improves the worst case time complexity of queue-based implementation. A linked list is a data structure in which the objects are arranged in a linear order [7] [54]. It is a collection of nodes gathered in a linear order. The linked list is an alternative to the array when a collection of objects is stored. Where in array the linear order is determined by the index of the array, the linear order in a linked list is determined by a pointer in each object. Thus, the linked list is implemented using pointer. Figure 6.22 shows a linked list whose elements are integer. Each element in the list is an object with fields for the key and the pointer to the next object. The *next* field of the tail is NIL. Linked list provides a simple, flexible representation for dynamic sets. A linked list supports the following operations:

- $Search(S, k)$, which returns a pointer x to an element in a given set S such that $key[x] = k$, or NIL if no such element belongs to S.

- $INSERT(S, x)$, which augments the set S with the element pointed to by x.
- $DELETE(S, x)$, which removes the pointer x to an element from S. This operation uses a pointer to an element x, not a key value.
- $MINIMUNM(S)$, which returns the element of S with the smallest key.
- $MAXIMUM(S)$, which returns the element of S with the largest key.
- $SUCCESSOR(S, x)$, which returns the next element in S greater to x.
- $PREDECESSOR(S, x)$, which returns the next element in S smaller to x.

Note that the operations $SUCCESSOR(S, x)$ and $PREDECESSOR(S, x)$ are queries that can be often extended to sets with non-distinct keys. The search operation in linked list takes $\Theta(n)$ time, where n is the number of object in the list.

6.4.1 Binary Search Trees

A binary search tree is a tree data structure that can be used to implement both dictionary and priority queue. It supports all the operations of linked list but takes only $\Theta(log(n))$ time for the $DELETE$ and $SEARCH$ operations [12] [54]. A key in a binary search tree T are always stored in such a way that satisfies the binary search tree property:
Let v be a node in a binary search tree T. If w is a node in the left subtree of v, then $key[w] \leq key[v]$. If w is a node in the right subtree of v, then $key[w] \geq key[v]$.

A binary search tree can be represented by a linked data structure in which each node is an object. The property allows to print out all the key in a binary search tree in sorted order by an algorithm called $INORDER - TREE - WALK$ as illustrated in table 6.3. This algorithm takes $\Theta(n)$ to walk an n-node binary search tree. If INORDER-TREE-WALK algorithm is used with the binary tree in figure 6.23, it prints the keys in the order 2, 3, 3 , 4, 5, 6. Note that when a child or the parent is missing, the appropriate field contains the value NIL. Each node in a binary tree contians the following fields:

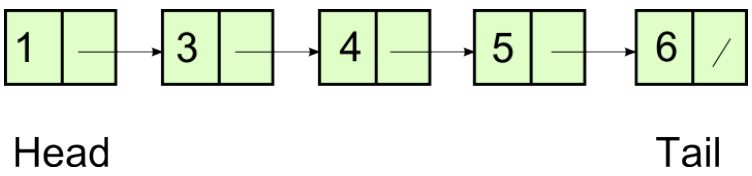

Figure 6.22: Linked List Example

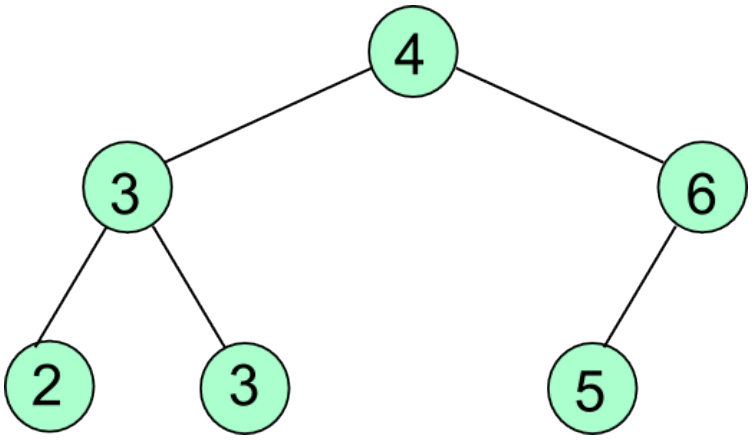

Figure 6.23: Binary Tree Example

```
INORDER-TREE-WALK(root[T])
  x ← root[T]
 if x ≠ NIL
     then INORDER-TREE-WALK(left[x])
          print key[x]
          INORDER-TREE-WALK(right[x])
```

Table 6.3: Inorder Tree Walk Algorithm.

- $key[x]$, which points to the key of the node x.
- $left[x]$, which points to the left child of the node x.
- $right[x]$, which points to the right child of the node x.
- $p[x]$, which points to the parent of the node x.

Search in Binary Search Trees

To search for a key stored in the binary tree, the binary tree is seen as a decision tree. Given a pointer to the root of the tree and a key k, the TREE-SEARCH algorithm shown in table 6.4, returns a pointer to a node with key k if it exists. Otherwise it returns NIL. The TREE-SEARCH algorithm begins its search at the root of the tree and traces a path downward in the tree. It takes O(h) time, where h is the height of the tree. Using the TREE-SEARCH algorithm on the binary tree depicted in figure 6.23 to search for the key 2, the path $4 \rightarrow 3 \rightarrow 2$ is followed from the root.

```
TREE-SEARCH(x, k)

 if x = NIL or k = key[x]
     then return x
 if k < key[x]
     then return TREE-SEARCH(left[x], k)
     else return TREE-SEARCH(right[x], k)
```

Table 6.4: Search Algorithm in Binary Trees

Insertion in Binary Search Trees

The TREE-INSERT(T, z) algorithm, as shown in table 6.5, is used to insert a new value v into a binary search tree T. This algorithm is passed a node z for which $key[z] = v$, $left[z] = NIL$, and $right[z] = NIL$. It begins at the root of the binary tree and traces a path downward. The algorithm takes O(h) time, where h is the height of the binary tree. If the TREE-INSERT algorithm is applied to insert a node z in the binary tree in figure 6.23 with $key[z] = 1$, the resulting tree is shown in figure 6.24.

```
TREE-INSERT(T, z)

  y ← NIL
  x ← root[T]
 while x ≠ NIL
     do y ← x
         if key[z] < key[x]
            then x ← left[x]
            else x ← right[x]
 p[z] ← y
 if y = NIL
     then root[T] ← z
     else if key[z] < key[y]
         then left[y] ← z
         else right[y] ← z
```

Table 6.5: Insert Algorithm in Binary Trees

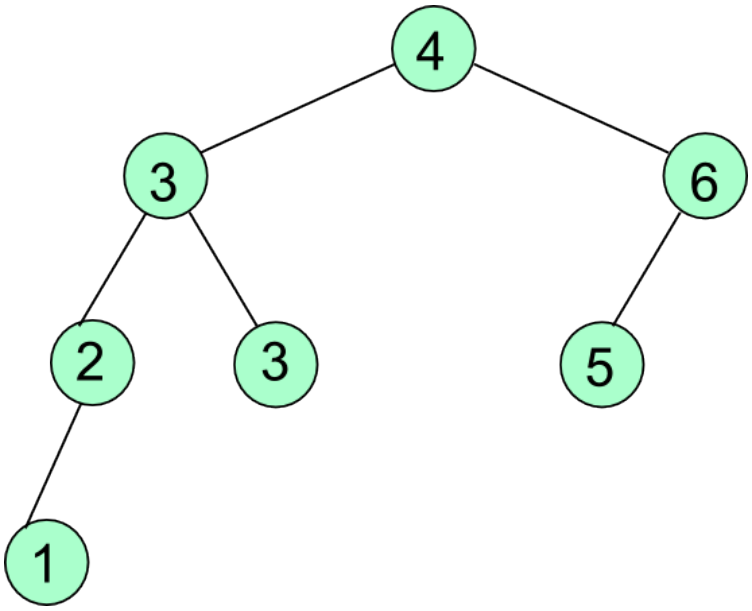

Figure 6.24: Inserting Key 1 into the Binary Tree shown in figure 6.23.

Deletion from Binary Search Trees

The TREE-DELETE(T, z) algorithm is shown in table 6.6 It calls another algorithm called TREE-SUCCESSOR(x). This algorithm is shown in table 6.7. The TREE-SUCCESSOR(x) algorithm finds the successor of the node x in the sorted order determined by an order tree walk. If TREE-SUCCESSOR algorithm is applied to the key 3 in the binary tree from figure 6.23, it returns the root node with key 4. If it is applied to the key 1 in the binary tree from figure 6.24, it returns the node with key 2.

1. If the node z has no children, it just be removed from the binary tree. For example, if the node with key 2 in figure 6.23 is removed, the result binary tree will be as in figure 6.25.
2. If node z has only one child, it will be spliced out. A link between its parent and its child is made. For example, if the node with key 6 in figure 6.23 is removed, the result binary tree will be as in figure 6.26.
3. If node z has two children, its successor will be spliced out, which has at most one child. A link between its parent and its successor is made. For example, if the root node with key 4 in figure 6.23 is removed, the result binary tree will be as in figure 6.27.

The TREE-DELETE(T, z) algorithm takes O(h) time, where h is the height of the binary tree.

```
TREE-DELETE(T, z)

 if left[z] = NIL   or   right[z] = NIL
     then y ← z
     else y ← TREE − SUCCESSOR(z)
 if left[y] ≠ NIL
     then x ← left[y]
     else x ← right[y]
 if x ≠ NIL
     then p[x] ← p[y]
 if p[y] = NIL
     then root[T] ← x
     else if y = left[p[y]]
          then left[p[y]] ← x
          else if right[p[y]] ← x
 if y ≠ z
     then key[z] ← key[y]
 return y
```

Table 6.6: Delete Algorithm in Binary Trees

```
TREE-SUCCESSOR(x)

  if right[x] ≠ NIL
     then return TREE-MINIMUM(right[x])
   y ← p[x]
  while y ≠ NIL and x = right[y]
     do x ← y
        y ← p[y]
  return y
```

```
TREE-MINIMUM(x)

  while left[x] ≠ NIL
     do x ← left[x]
  return x
```

Table 6.7: Successor Algorithm in Binary Trees

In the case where a small number of priotity and log data, a linked list can be used to store the priority of the event and a queue can be used to store the

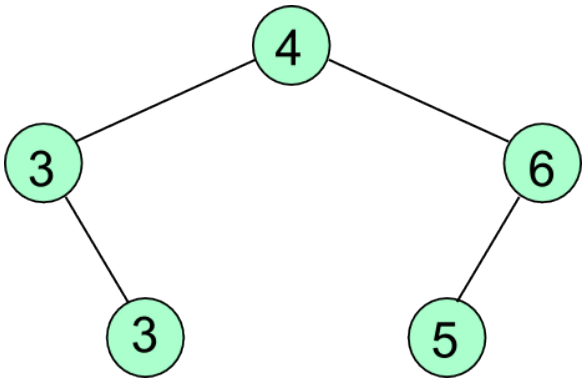

Figure 6.25: Deleting Key 2 from the Binary Tree shown in figure 6.23.

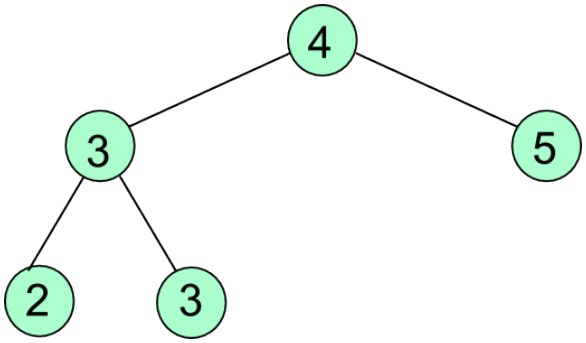

Figure 6.26: Deleting Key 6 from the Binary Tree shown in figure 6.23.

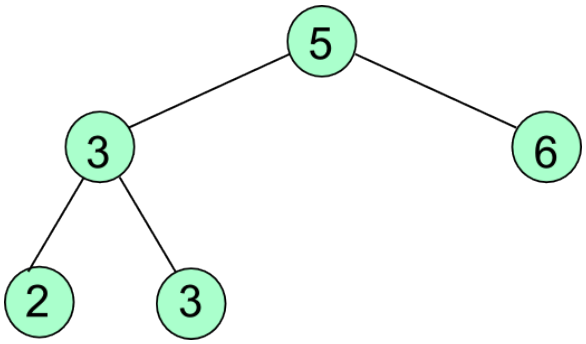

Figure 6.27: Deleting the Root Key 4 from the Binary Tree shown in figure 6.23.

log entries. A linked list and a queue are used in this case due to their simple implementation. Each element of the linked list as illustrated in figure 6.28 has

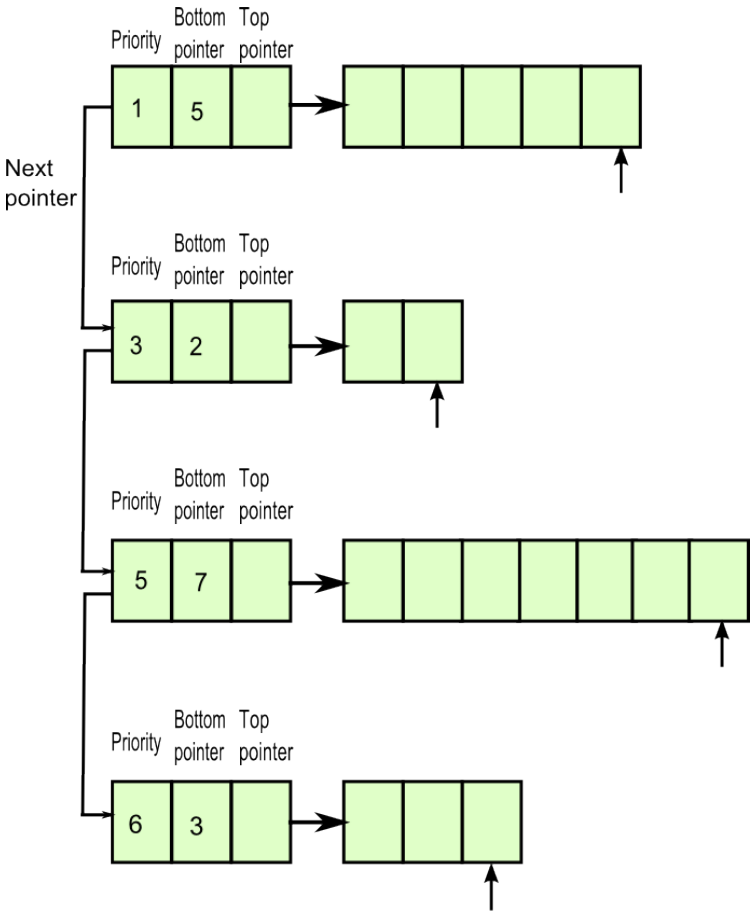

Figure 6.28: Using Linked Lists and Queues to store Log Data

three pointers:

1. *Next* pointer, which points to the next element in the linked list.
2. *Bottom* pointer, which points to the tail of the queue.
3. *Top* pointer, which points to head of the queue.

To insert a new log entry in the queue, priority search algorithm is made first in the linked list. This algorithm to search for the aprpriate priority of the new log entry. The new log entry is then inserted in the queue and the bottom pointer is increased by 1 to points to the new position of the tail of queue. If the appropriate priority of the new log entry is not found in the linked list, a new element is inserted in the linked list. This new priority element in the linked list points to a new queue, where the new log entry will be inserted.

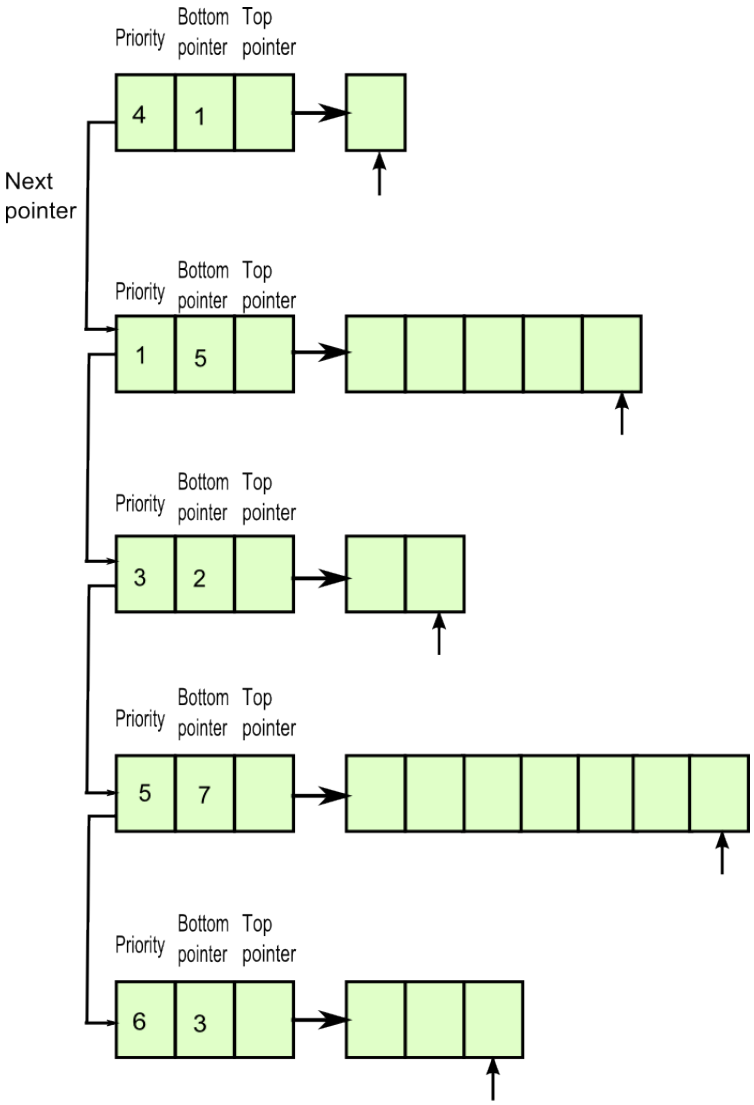

Figure 6.29: Inserting new Log Data with Priority in the Structure shown in Figure 6.28.

This case is illustrated in figure 6.29. If a queue is full, the new log entry overwrite the first element in the queue, where the top pointer points to.
Figure 6.30 shows the case, where a new log entry is inserted in the structure from figure 6.28. The new log entry has priority 5 and there is a free place in the queue belongs to this priority. In practice, the number of priorities is small and the number of log entries is a huge. therefor, the priority is implementd

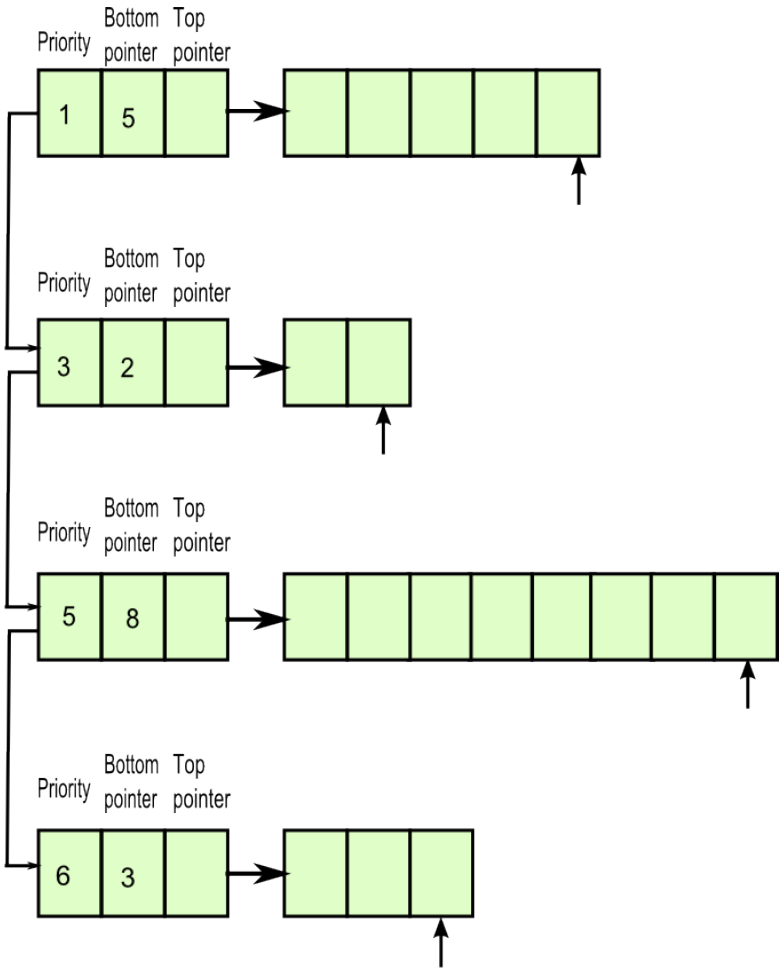

Figure 6.30: Inserting new Log Data in free Space in the Queue.

using linked list like before, but the log entries is implemented using binary search tree as illustrated in figure 6.31. This makes the search operation take only O($log(n)$).

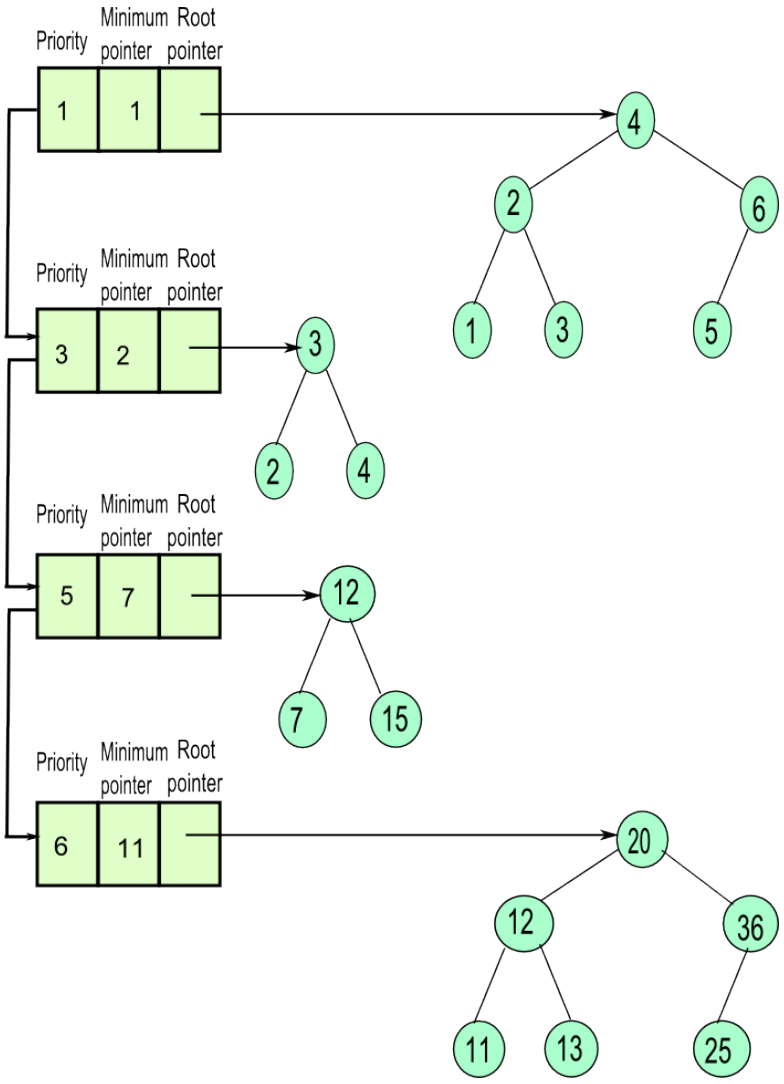

Figure 6.31: Using Linked Lists and Binary Search Trees to Store the Log Data.

Chapter 7

Conclusion

The fundamental focus of this thesis has been on analysis, design and implentation of data-logger using offline strategy. The system should be flexible enough to include existing access solutions and extensible enough to work with future ones. To reach this goal the system is perused and evaluated in order to determine the important features required in its design. Although it is difficult to perform a full evaluation of such a design, a prototype implementation can be used to show the feasibility of the solution. Thus when the evaluation was finished and the design was complete a prototype was implemented to test the design. The implemented system represents the log data and independent of the number and type of sensor nodes as well as the number and type of events. The work has been achieved in an experimental fashion. Many initial ideas were discussed to give an understanding of its integerity. Those ideas that were found to be very important were implemented and tested to see if they should be investigated further. Successfully implemented ideas were then refined and further discussed. The developed system able to extract the data from sensor nodes, process and represent the log data using Java and XML technologies. The XML representation has been presented and tested. Aspects such as event priority, time complexity and independency have been considered. The system is dynamic and change the provided display depending on the collected data. The need for mobility and data management in WSNs makes it necessary to develop algorithms, protocols and data structures which can combine together to offer flexible QoS and a user friendly interface. In this sence a new worst case optimal solution is implemented for the problem of combining log data with priorities in a given sensor network. The implemented system gives the possibility of comparing different constructions methods for WSNs. The data structure used is similar to ScatterWeb, it adds time, identification and priority information to the sensor readings. The XML representation can be accessed by other applications independent of their home-platform and their programming language. Polling the information from the sensor nodes relaxes the sensor node memory size limitation at the price of adding a little bit latency. However, since

no complicated processing is done in each step of processing chain in parallel, the latency could even be smaller than human reaction time. Incapsulation action behind a facade makes it possible to interface to application via XML like-query and additional command. In WSNs, the mobility of requester for information and log data from sensor network is a big issue. Based on this implementation different WSN applications can be realized:

- Implementation and testing of WSN protocols for future real world applications.
- Organizing the data in a database with a query language similar to SQL.
- Building an asynchronize service to reconfigure the sensor nodes in real time.
- Embedding the XML representation in a meta-XML language to extend the implemented application.

Further enhancement
For the system developed in this project there are further enhancemnets.

- During the expriments, a lower range of the ESB's transceiver have been observed to deliver stable wireless connections inside building like the test-lab.
- Adding security to the system architecture, without security the system can become a point of attack (e.g.:cryptography and key management).

Bibliography

[1] A. Liers, H. Ritter, J. Schiller, Data Gathering and Routing in the ScatterWeb Sensor Network, Freie Universität Berlin Institute of Computer Systems and Telematics, Technical Report, August 2004.

[2] Bharath Sundararaman, Ugo Buy, and Ajay D. Kshemkalyani, Clock Synchronization for Wireless Sensor Networks: A Survey, Department of Computer Science, University of Illinois at Chicago, 2005.

[3] http://www.btnode.ethz.ch/Documentation/Tutorials,ETH Zurich, 2007.

[4] C. Schindelhauer. Mobility in wireless networks. In 32nd Annual Conference on Current Trends in Theory and Practice of Informatics, Czech Republic, January 2006.

[5] C. F. Tsai and M. S. Younga, Pyroelectric infrared sensor-based thermometer for monitoring indoor objects, Department of Electrical Engineering, National Cheng-Kung University, Tainan, 701 Taiwan, Republic of China, 2003.

[6] http://cnx.org, 2006,Naren Anand,Connexions.

[7] Cormen Thomas H.,Ronald L. Rivest, Charles E. Leiserson, Introduction to Algorithms, McGraw-Hill Companies (Juli 1990), ISBN-13: 978-0070131439.

[8] http://cva.ap.buffalo.edu/courses/f06/arc598/node/122,2006,Physical computing.

[9] http://www.datasheetcatalog.com/datasheets_pdf/M/S/P/4/MSP430F149.shtml, 2006.

[10] David Hunter, Kurt Cagle, Chris Dix, Beginning XML, Wiley & Sons;2nd (2004), ISBN-10: 1861005598.

[11] Dazhi Chen and Pramod K. Varshney, QoS Support in Wireless Sensor Networks: A Survey, Dazhi Chen and Pramod K. Varshney, Department of EECS, Syracuse University Syracuse, NY, U.S.A 13244, 2004.

[12] Dinesh P. Mehta, Sartaj Sahni, Handbook of Data Structures and Applications, Chapman & Hall/CRC (October 28, 2004), ISBN-13: 978-1584884354.

[13] E. Anceaume and I. Puaut , A Taxonomy of Clock Synchronization Algorithms, Research report IRISA, NoPI1103, July 1997.

[14] Ellis Horowitz, Sanguthevar Rajasekaran, Computer Algorithms, W.H.Freeman & Co Ltd; Auflage: 2Rev Ed (August 1997), ISBN-13: 978-0716783169.

[15] Feng Zhao,Leonidas Guibas, Wireless Sensor Networks: An Information Processing Approach, Morgan Kaufmann (2004), SBN-10: 1558609148.

[16] Fikret Sivrikaya, Blent Yener, Time Synchronization in Sensor Networks: A Survey, Department of Computer Science Rensselaer Polytechnic Institute, NY., 2004.

[17] http://focus.ti.com/lit/ug/slau056f/slau056f.pdf,User Guide,MSP430x4xx Family.

[18] http://www.fu-berlin.de/forschung/transfer/messen/04HMISchiller.pdf, ScatterWeb.

[19] G. Hoblos, M. Staroswiecki, A. Aitouche, Optimal design of fault tolerant sensor networks, IEEE International Conference on Control Applications, Anchorage, AK, September 2000.

[20] Herbert Schildt, Java 2: The Complete Reference, McGraw-Hill Osborne Media; 5th edition, ISBN−10: 0070495432, 2004.

[21] http://www.hobbyprojects.com/pic_tutorials/tutorial11.html Electronic Circuits and tutorials.

[22] Holger Karl, Andreas Willig, Protocols and Architectures for Wireless Sensor Networks., Wiley (June 24, 2005), ISBN-13: 978-0470095102.

[23] http://www.inf.fu-berlin.de/inst/ag−tech/scatterweb_net/datasheets/ TSL245.pdf

[24] www.inf.fu-berlin.de/inst/ag−tech/scatterweb_net/documentation/ First_Steps.pdf.

[25] http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb_net/documentation/ ScatterViewer_UserGuide.pdf

[26] http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb_net/documentation/ User_Guide.pdf

[27] Ioannis Chatzigiannakis, Athanasios Kinalis, Sotiris Nikoletseas, Sink Mobility Protocols for Data Collection in Wireless Sensor Networks, Computer Technology Institute (CTI) and University of Patras, 2006.

[28] Ilker Demirkol, Cem Ersoy, and Fatih Alagz, MAC Protocols for Wireless Sensor Networks:a Survey, 2006.

[29] Jamal N. Al-Karaki, Ahmed E. Kamal, Routing Techniques in Wireless Sensor Networks:A Survey,Dept. of Electrical and Computer Engineering Iowa State University, Ames, Iowa 50011, 2004.

[30] Jana van Greunen, Jan Rabaey, Lightweight Time Synchronization for Sensor Networks, University of California, Berkeley, 2003.

[31] http://www.javaworld.com/javaworld/jw-05-1998/jw-05-javadev.html.

[32] Jeni Tennison, Beginning XSLT, WROX Press Ltd 2002, ISBN-10: 1861005946.

[33] Jeremy Elson, Time Synchronization Services for Wireless Sensor Networks, Department of Computer Science University of California, Los Angeles Los Angeles, CA, 90095, 2001.

[34] Jerry Luecke, Analog and Digital Circuits for Electronic Control System Applications: Using the TI MSP430 Microcontroller, Newnes,ISBN-10: 0750678100,2004

[35] John P. Hayes, Computer Architecture and Organization, McGraw-Hill Companies; 3rd edition (December 1, 1997),ISBN-10: 0070273553.

[36] Juan Alonso, Adam Dunkels and Thiemo Voigt, Bounds on the Energy Consumption of Routings in Wireless Sensor Networks, SICS, Swedish Institute of Computer Science, Stockholm Sweden Dept. of Comp. Sci. and Engineering, Malardalen University, Sweden, 2005.

[37] K. Daniel Wong, Physical Layer Considerations for Wireless Sensor Networks, Department of Information Technology Malaysia University of Science and Technology Petalhg Jaya, Selangor, Malaysia, 2004.

[38] Kal Ahmed, Sudhir Ancha, Andrei Cioroianu, Jay Cousins, Jeremy Crosbie, John Davies, Kyle Gabhart, Steve Gould, Ramnivas Laddad, Sing Li, Brendan Macmillan, Daniel Rivers-Moore, Judy Skubal , Karli Watson, Scott Williams,Professional Java XML, WROX Press Ltd (3. Mai 2001), ISBN-10: 186100401X.

[39] Kay Rmer, Oliver Kasten, Friedemann Mattern, Middleware Challenges for Wireless sensor networks, Department of computer science, ETH Zurich, Switzerland, 2002.

[40] Kay Rmer, Philipp Blum, Lennart Meier, Time Synchronization and Calibration in Wireless Sensor Networks, ETH Zurich, Switzerland

[41] Kemal Akkaya and Mohamed Younis, A Survey on Routing Protocols for Wireless Sensor Networks, Department of Computer Science and Electrical Engineering University of Maryland, Baltimore County, 2003.

[42] Kyle Jamieson, Hari Balakrishnan, A MAC Protocol for Event-Driven Wireless Sensor Networks, MIT Laboratory for Computer Science, Massachusetts Institute of Technology Cambridge, MA 02139, 2006.

[43] Lan F. Akyildz, Wilian su, Yogesh Sankarasubramaniam, Erdal Cayiric, A Survey on sensor networks, george Institute of technology, 2002.

[44] Lizhi Charlie Zhong, Jan Rabaey, Chunlong Guo, Rahul Shah, Data Link Layer Design For Wireless Sensor Networks, Berkeley Wireless Research Center Department of EECS University of California at Berkeley, 2001.

[45] Loannis Chatzigiannakis, Athanasios Kinalis, Sotiris Nikoletseas, Sink Mobility Protocols for Data Collection in Wireless Sensor Networks, Computer Technology Institute (CTI) and University of Patras, 2006.

[46] Lutz Bierl, Das grosse MSP430 Praxisbuch, Franzis,2004.

[47] Mark A. Weiss, Data Structures and Algorithm Analysis in Java, ddison-Wesley Longman, Amsterdam; Auflage: US Ed (Dez. 2002), ISBN-13: 978-0201357547.

[48] http://www.math.carleton.ca/∼help/matlab/MathWorks_R13Doc/techdoc/matlab_external/ch_jav38.html.

[49] Michael T. Goodrich, Roberto Tamassia, Data Structures and Algorithms in Java, Wiley & Sons; Auflage: 2nd (September 2005), ISBN-13: 978-0471383673.

[50] Mihaela Cardei, Ding-Zhu Du, Improving Wireless Sensor Network Lifetime through Power Aware Organization, 2005.

[51] Mihail L. Sichitiu and Chanchai Veerarittiphan, Simple, Accurate Time Synchronization for Wireless Sensor Networks, Electrical and Computer Engineering Department North Carolina State University Raleigh, NC 27695-7911, 2003.

[52] N. Bulusu, D. Estrin, L. Girod, J. Heidemann, Scalable coordination for wireless sensor networks: self-configuring localization systems, International Symposium on Communication Theory and Applications (ISCTA 2001), Ambleside,UK, July 2001.

[53] http://nwalsh.com/docs/tutorials/xsl.

[54] Ottmann Thomas, Peter Widmayer, Algorithmen und Datenstrukturen, Spektrum Akademischer Verlag; Auflage: 4. A. (Januar 2002), ISBN-13: 978-3827410290.

[55] http://www.olimex.com/dev/pdf/msp430-jtag-d.pdf.

[56] http://www.parallax.com/dl/docs/prod/audiovis/PIRSensor-V1.2.pdf.

[57] Peter Spasov, Microcontroller technology the 68HC11,Printic Hall,1999.

[58] Randy Brown,Calendar Queues: A Fast O(1) Priority Queue Implementation for the Simulation Event Set Problem, Communications of the ACM, Volume 31, Pages: 1220 - 1227, Issue 10 (October 1988).

[59] S. Cho, A. Chandrakasan, Energy-efficient protocols for low duty cycle wireless microsensor, Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, Maui, HI Vol. 2 (2000).

[60] Saurabh Ganeriwal, Ram Kumar, Sachin Adlakha and Mani Srivastava, "Network-wide Time Synchronization in Sensor Networks," Technical Report UCLA, April 2002.

[61] http://www.scatterweb.net.

[62] Seungjoon Lee,Suman Banerjee,Bobby Bhattacharjee, The Case for a Multi-Hop Wireless Local Area Network,University of Maryland College Park, MD 20742 USA, 2003.

[63] http://www.st.com/stonline/books/pdf/docs/10305.pdf,PIR (PASSIVE INFRARED) DETECTOR USING ST7FLITE05/09/SUPERLITE.

[64] Wei Ye, John Heidemann, Deborah Estrin, An Energy-Efficient MAC Protocol for Wireless Sensor Networks, 2001.

[65] http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf.

[66] http://www.w3schools.com/xml/xml_whatis.asp.