Diplomarbeit

Manfred Wermke

Entwicklung eines kostengünstigen
Sensornetzwerkes für Monitoring Aufgaben

Fakultät Technik und Informatik *Faculty of Engineering and Computer Science*
Studiendepartment Informatik Department of Computer Science

Manfred Wermke

Entwicklung eines kostengünstigen Sensornetzwerkes für Monitoring Aufgaben

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Softwaretechnik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Gunter Klemke
Zweitgutachter : Prof. Dr. rer. nat. Kai von Luck

Abgegeben am 18. August 2005

Manfred Wermke

Thema der Diplomarbeit

Entwicklung eines kostengünstigen Sensornetzwerkes für Monitoring Aufgaben

Stichworte

Sensornetzwerk, Protokollentwicklung, ZigBee, Multi-Hop-Netze, Gebäudeüberwachung, Netzwerktechnik, verteilte Systeme

Kurzzusammenfassung

Diese Arbeit umfasst die Entwicklung eines Sensornetzwerkes auf ZigBee Basis. Prototypisch werden existierende ZigBee-Module verwendet. Das Sensornetzwerk soll kostengünstig allgemeine Überwachungsaufgaben durchführen können.
Das exemplarisch gewählte Szenario ist eine Hausüberwachung im Falle längerer Abwesenheit.

Ende des Textes

Manfred Wermke

Title of the paper

Development of a cost-efficient sensor network for monitoring tasks

Keywords

wireless sensor network, development of protocols, ZigBee, multi-hop-networks, monitoring of buildings, network engineering, distributed systems

Abstract

Inside this report the development of a wireless sensor network based on ZigBee. Prototypic will be used existing ZigBee-modules. The wireless sensor network, with the task of general monitoring, includes the aspect of low costs. The choosen scenario is monitoring of building in longer absence.

Einführung

Die Anregung zur Entwicklung dieser Diplomarbeit kam aus meinem privaten Umfeld. Ich wurde immer wieder von Freunden gefragt, ob ich, während sie längere Zeit im Ausland sind, auf ihr Haus aufpassen würde. Dies bedeutete immer einen zusätzlichen Zeitaufwand und auch Fahrtkosten für mich.

Da entstand die Idee, dies von einem automatischen System übernehmen zu lassen. Aber es stellten sich dabei viele Probleme und Fragen.

Was soll alles überwacht werden?

Wie oft soll kontrolliert werden?

Wie ist der technische Aufwand minimierbar?

Wie ist dies möglichst kostengünstig zu erledigen?

Eine Variante wäre einen mobilen Agenten, einen Robot zu benutzen. Allerdings ist zu beachten, wie viele Robots benötigt werden, wie sie navigiert werden, wie groß die Akkulaufzeit ist und wie hoch die Wahrscheinlichkeit eines Ausfalls ist. Die nächste Frage die auftaucht, wie und wohin überträgt der Robot seine Daten. Diese technischen Voraussetzungen ließen sich nur durch einen enormen Hard- und Softwareaufwand bewältigen, verbunden mit hohen Kosten, was diese Möglichkeit nicht effizient und sinnvoll erscheinen lässt.

Zurück zur Ausgangssituation. Es ist notwendig verschiedene Punkte im ganzen Haus zu überwachen, wobei aber immer unterschiedliche Daten erfasst werden müssen. Das Naheliegendste, um dies zu gewährleisten, sind verschiedene Sensoren zu installieren. Eine ständige Überwachung der Sensoren wäre natürlich optimal. Zu große Intervalle bergen immer die Gefahr das wichtige Daten verloren gehen, also Ereignisse innerhalb des Intervalls.

Der technische Aufwand ist nicht unerheblich und zu unterschätzen. Jeder Sensor müsste verkabelt werden und die Kabel durch das ganze Haus verlegt werden. Eine Übertragung per Funk wäre wünschenswert, es wäre komfortabler und lange nicht so aufwendig, wie eine Verkabelung. Dieses Material würde bei der Hardware sogar eingespart werden.

Für die Aufgabe des Monitoring müsste also ein System entwickelt werden, was aus mehreren Sensoren besteht und die anfallenden Daten selbstständig per Funk austauscht, sammelt und überträgt. Die sogenannten Sensornetzwerke stellen hier die ideale Lösung dar. Die letzte offene Frage hier sind die entstehenden Kosten. Derzeitige handelsübliche Sensorknoten sind einfach noch zu teuer für den privaten Einsatz. Da selbst in einem kleinen Haus eine erhebliche Anzahl dieser Module benötigt werden.

An diesem Punkt knüpft nun die Diplomarbeit an „Entwicklung eines kostengünstigen Sensornetzwerkes für Monitoring Aufgaben“.

1. Aufgabenstellung: Szenarioentwicklung

Die Aufgabe dieser Diplomarbeit besteht in der Entwicklung eines Sensornetzwerkes, zur Überwachung eines Gebäudes oder einer Wohnung. Das gesamte System sollte möglichst kostengünstig sein. Das Sensor- / Funknetzwerk muss sich einfach auf die jeweiligen Bedürfnisse anpassen lassen.
Es wird davon ausgegangen, das die Art der Sensoren und deren Wertebereich nicht bekannt sind. Eine Netzstromversorgung ist gegeben.

In der Gesamtheit betrachtet lässt sich die Aufgabe in zwei Teilbereiche aufsplittern, in Hardware und Software.

Die Hardwareentwicklung bearbeitet Christian Fischer in seiner Bachelorarbeit. Im Folgenden wird das zu entwickelnde Hardwaremodul als CFMW1 bezeichnet. Im Kapitel 3 und dessen Unterpunkten wird noch etwas genauer auf diese Entwicklung und die Randparameter der Hardware eingegangen.

In meiner Diplomarbeit habe ich die Softwareentwicklung in folgende Bereiche unterteilt: die Aspekte der Protokoll- und Netzbildung, sowie die Beschreibung und Implementierung einer funktionsfähigen Software zum Betrieb der Sensorknoten. Die graphische Darstellung der Daten auf einem Computer wird in dieser Arbeit nicht bearbeitet. Im Kapitel 7.3. werden jedoch einige Ideen dazu vorgestellt.

Ausgewähltes mögliches Szenario:

Die Basis bildet ein Einfamilienhaus, ausgestattet mit einer Internetverbindung und Telefonanschluß.

Dieses Gebäude besteht aus 4 Zimmern und einem Wintergarten, mit automatischer Bewässerungsanlage. Es gibt ein Aquarium, ein Wasserbett und diverse elektrische Haushaltsgeräte, wie z. B. Waschmaschine, Wäschetrockner, Geschirrspüler, Kühlschrank, Gefrierschrank etc..

Überwacht werden müssen alle kritischen Bereiche, inklusive der Eingangstüren und der Fenster. Zu den kritischen Bereichen zählen natürlich vor allen Dingen das Aquarium, das Wasserbett der Kühlschrank und der Gefrierschrank, da diese technischen Geräte ständig in Betrieb sind. Zur optimalen Sicherheit überwacht man aber auch die anderen Komponenten, da z.B. das Ausschalten der Geräte vergessen werden kann oder deren Zuflüsse, wie bei der Waschmaschine der Wasserzulauf.

Tabelle der zu erfassenden Daten und deren zeitliche Überprüfung:

Ort	Art der Daten	Abfrage pro 10 min	Sensortyp
Flur/Korridor			
Eingangstür	offen / geschlossen	5	Digital
Raum	Temperatur	1	Analog
Wohnzimmer			
Eingangstür	offen / geschlossen	5	Digital
Raum	Temperatur	1	Analog
Aquarium	Nässe	10	Digital
Glasscheiben	Glasbruch	2	Digital
Küche			
Eingangstür	offen / geschlossen	5	Digital
Raum	Temperatur	1	Analog
Waschmaschine	Nässe	10	Digital
Tiefkühltruhe	Temperatur	2	Analog
Schlafzimmer			
Eingangstür	offen / geschlossen	5	Digital
Raum	Temperatur	1	Analog
Wasserbett	Nässe	10	Digital
Kinderzimmer			
Eingangstür	offen / geschlossen	5	Digital
Raum	Temperatur	1	Analog
Glasscheiben	Glasbruch	2	Digital
Bad			
Eingangstür	offen / geschlossen	5	Digital
Raum	Temperatur	1	Analog
Boden	Nässe	10	Digital
Wintergarten			
Eingangstür	offen / geschlossen	5	Digital
Raum	Temperatur	1	Analog
Boden	Feuchtigkeit	1	Analog
Glasscheiben	Glasbruch	2	Digital

Aus dieser Bedarfsanalyse ergeben sich 24 benötigte Sensoren.

Mit einem Knoten ließen sich mehrere Sensoren abfragen, aber in diesem Beispiel gehen wir davon aus, für jeden Sensor einen eigenen Knoten zu benutzen. Weiterhin setzen wir voraus, dass alle Abfragen mit geringfügigem Zeitversatz erfolgen, so erhält man ein Datenvolumen von 71 Nachrichten pro 10 Minuten.

2. Sensornetzwerke

Dieses Kapitel gewährt einen Einblick in den Bereich Sensornetzwerke.

- Was versteht man unter Sensornetzwerken?
- Was sind die besonderen Eigenschaften von Sensornetzwerken?
- Welche Möglichkeiten bieten Sensornetzwerke? Wie sehen die möglichen Anwendungsfelder für diese Technik aus?
- Wie ist der derzeitige Stand in der Entwicklung?
- Was bringt die Zukunft, was kann noch möglich sein?

2.1. Was sind Sensornetzwerke?

Allgemein betrachtet besteht ein Sensornetzwerk aus einer Menge intelligenter Sensoren, die Daten aufnehmen, verarbeiten und untereinander austauschen.

Aber was ist ein intelligenter Sensor? Was heißt Daten verarbeiten und austauschen?

Ein Sensornetzwerk baut sich aus einer Anzahl von Einzelknoten auf. Diese bestehen grundlegend aus drei Komponenten:

- die Sensoren zur Erfassung der gewünschten Daten
- einem kleinen Computer zur Verarbeitung der Sensordaten
- einer Funktechnik zum drahtlosen Austausch der Daten

Mit diesen Komponenten lassen sich jetzt Daten sammeln, verarbeiten und ausgeben. Unbeantwortet ist die Frage der Netzwerkbildung.

Üblicherweise erfolgt die Vernetzung der einzelnen Sensorknoten nach dem Prinzip der *Ad Hoc Netzwerke*. *Ad Hoc Netze* entstehen aus einer sogenannten spontanen Vernetzung. Bekannt aus Bereichen wie UMTS, W-Lan, Bluetooth etc..
Das Eintreten in solche Netzwerke bzw. das Verlassen ist sehr einfach, da es keine feste Infrastruktur gibt. Vielmehr wird die Infrastruktur durch die einzelnen Kommunikationsteilnehmer selbst gebildet.

Durch die Netzwerkbildung können neue Informationen gewonnen werden, z.B. Informationen über den räumlichen Verlauf von Ereignissen und das sogar über große Entfernungen.

2.2. Eigenschaften von Sensornetzwerken

Das Besondere an diesen Modulen ist die kleine Bauform und der geringe Energieverbrauch. Eigenschaften die diese Module in die Lage versetzen, lange Zeit unabhängig von einer festen Stromversorgung, ihre Daten zu sammeln und zu übermitteln.

Zur Lösung der gesetzten Aufgabenstellung könnte man auch viele andere Techniken anwenden. Sensornetzwerke bieten jedoch einige ganz entscheidende Vorteile.

- Sensornetzwerke arbeiten autonom und sind somit unabhängig von einem Steuersystem. Die Netze werden zwar mit einem Rechner gekoppelt, aber dies dient lediglich zur Darstellung der gewonnenen und ausgewerteten Daten bzw. der Gewinnung neuer Daten aus der Menge der einzelnen Informationen.
- Solche Netzwerke sind sehr unanfällig gegenüber dem Ausfall einzelner Knoten, dies ist im Falle einer Überwachungsaufgabe in unwegsamen Bereichen besonders wichtig.
- Es ist üblicherweise keinerlei Installationsaufwand notwendig.

Zusammengefasst kann man sagen, Sensornetzwerke bestechen durch ihre geringe Baugröße, sparsamen Energieverbrauch und den relativ günstigen Stückpreis. Bezüglich der Installierung und ihrer Anwendung sei noch erwähnt, dass die Vernetzung spontan erfolgt, üblicherweise durch Multi-Hop-Netze und das System selbstorganisierend arbeitet.

Die Vorteile von Sensornetzwerken sind offensichtlich, jedoch muss man, bei einer objektiven Entscheidungsfindung über einen Einsatz, die Eigenschaften von Sensornetzwerken in seiner Gesamtheit kritisch betrachten.

Nach meiner Feststellung richten sich die Nachteile und Risiken von Sensornetzwerken sehr nach dem jeweiligen Anwendungsfall. Das heißt man sollte sich über einige Punkte klar werden, bevor man für einen bestimmten Anwendungszweck ein Sensornetzwerk mit Funktechnik oder doch fest verdrahtete Sensoren mit entsprechender Basisstation verwendet.

Folgende Faktoren könnten sich kritisch auf eine Anwendung auswirken:

- Energieverbrauch: Funktechnik an sich hat einen hohen Energieverbrauch, sowohl für den Empfang von Daten, als auch für das Senden. Der Verbrauch ist natürlich auch abhängig von der benötigten Reichweite. Das heißt bei einer langfristigen Beobachtung muss eine kontinuierliche Stromversorgung sichergestellt sein.
- Adressierung der Module: Zur Adressierung stehen verschiedene Verfahren zur Verfügung. Zum Einen eine feste Adressierung, die jedoch bei einer großen Anzahl von Kommunikationspartnern entsprechend aufwendig werden kann. Eine automatische Adressierung kann jedoch den Nachteil in sich bergen, das man nicht weiß, welcher Sensor wo zu lokalisieren ist (entsprechend: wurde Alarm in Zone 1 oder in Zone 2 ausgelöst). Dieses Problem würde sich durch GPS etc. umgehen lassen.
- Datensicherheit: Diesen Aspekt sollte man nicht unbeachtet lassen, denn wie überall wo Informationen gesammelt werden und aus der Kombination dieser Daten neue Daten entstehen, besteht immer die Gefahr des gläsernen Menschen. Besonders, solange die Sensoren überall einsetzbar sind.

Dies betrachte ich aber alles als Risiken, die sicherlich in der Zukunft durch eine ständige Weiterentwicklung der Systeme gelöst werden. Für die Adressierung, die Ortsbestimmung und für das Routing in solchen Netzen existieren schon eine Anzahl von Algorithmen als Lösungsansätze.

2.3. Anwendungsmöglichkeiten dieser Technik

Natürlich kommt schnell die Frage auf, wo liegen die Anwendungsbereiche und Möglichkeiten dieser Netzwerke.

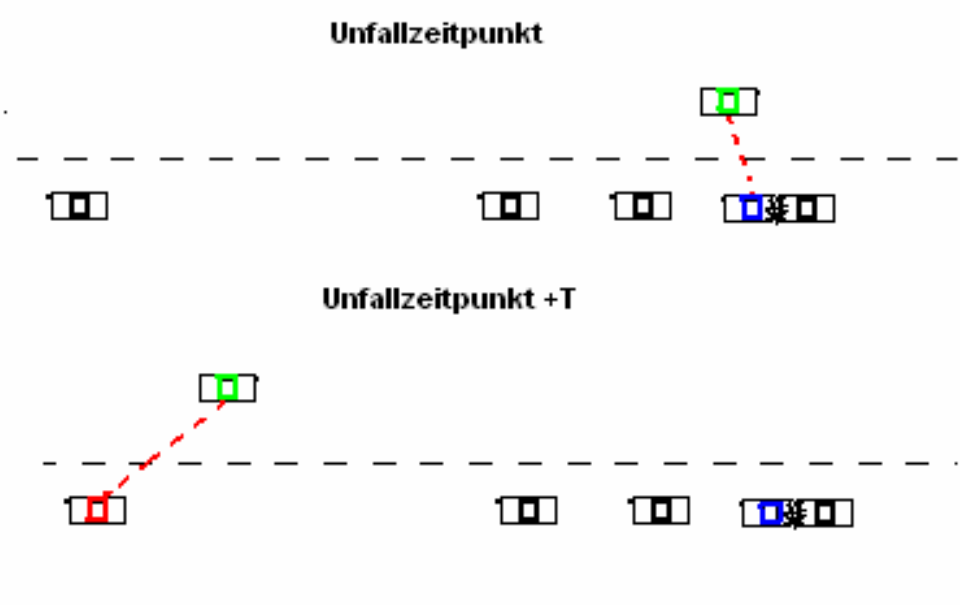
Einsatzmöglichkeiten dieser Netzwerke spiegeln sich in vielen Szenarien wider. Die Art der Datenerfassung ist prinzipiell überall dort einsetzbar, wo keine feste Infrastruktur besteht, zum einen was die Energieversorgung angeht und zum anderen was die Verkabelung für den Datenaustausch betrifft.

Dies ist gerade bei mobilen Systemen von Vorteil oder Systeme die nur für eine gewisse Zeit betrachtet werden. Im Folgenden einige Anwendungsbeispiele und die Funktion der Sensornetzwerke in diesen Bereichen:

- Einbringung von Sensorknoten zur Feuchtigkeitsmessung in einen Deich. Dadurch misst man den Fortschritt der Durchfeuchtung und warnt gegebenenfalls vor einem Deichbruch (siehe Quellenangabe 12).
- Ausstattung von Fahrzeugen mit Sensorknoten. Man gewinnt Informationen über den Verkehrsfluss, die Verkehrsdichte, der Straßenbeschaffenheit und einiges mehr. (siehe Quellenangaben 25; 27)
- Militärische Nutzung: Abwurf von Sensorknoten über feindliches Land zur Erkennung von Truppenbewegungen, in Stärke und Richtung der Aufmärsche.
- Je nach Grad der Miniaturisierung könnten solche Systeme auch zur Bioüberwachung in Körpern von Menschen oder Tieren eingesetzt werden. Als Schlagwort sei hier "*Smart Dust*"[1] genannt.
- Bei einer Kombination der Sensoren mit Aktoren, ist eine autonome Steuerung ganzer Systeme denkbar, z.B. bei Gewächshäusern etc.

[1] Bei *Smart Dust* handelt es sich um eine „System on Chip Variante" der Sensorknoten. Diese zeichnen sich durch ihre extremkleine Ausführung aus. Wie der Name schon verlauten lässt, ist hier das Ziel der Entwicklung „Staubkorngröße". (siehe Quellenangabe 16)

Beispiel einer Unfallwarnung:



Das blaue Fahrzeug hat einen Unfall auf der Autobahn und warnt per Funktechnik automatisch alle nachfolgenden Fahrzeuge, dabei auch die auf der Gegenspur.

Das grüne Fahrzeug bewegt sich unvermittelt weiter und kann alle entgegen kommenden Fahrzeuge (rot) warnen, dass diese auf eine Unfallstelle zufahren. So ein System ist natürlich gerade bei Unfällen in Kurven interessant, aber auch bei langen Staus auf der Autobahn, so würden sicherlich eine Vielzahl von Massenkarambolagen verhindert werden.

Weiterhin könnte man sich eine intelligente Ampelschaltung vorstellen, weil jedes Fahrzeug seine Bewegungsdaten weitergibt und ein Verkehrsleitsystem auf Grund dieser Informationen den Verkehrsfluss optimieren könnte.
Für einen solchen Prozess an jeder Ampel Induktionsschleifen zu installieren wäre im Verhältnis mit unglaublichen Kosten und Bauarbeiten verbunden und doch könnte nicht die selbe Effizienz erreicht werden. Da es immer nur eine punktuelle Situationsdarstellung wäre.

2.4. Techniken von Sensornetzwerken und Stand der Entwicklungen

Es gibt verschiedene Grundsysteme von Sensornetzwerken. Diese unterscheiden sich jedoch kaum in ihrer Funktionsweise, als vielmehr in der benutzten Hardware und in der Art ihrer Anwendung.

So bestehen natürlich ganz andere Randparameter für ein Sensorsystem das zur Erforschung des Verhaltens eines ganzen Vogelschwarms eingesetzt wird, im Gegensatz zu einem Netzwerk, was den deutschen Straßenzustand mit Hilfe mobiler Agenten (Autos) überwachen soll.

Zur Beobachtung des Vogelschwarms ist eine möglichst kleine Bauform und ein geringer Stromverbrauch unumgänglich. Im anderen Fall, Autos als Agenten zur Überwachung einzusetzen, ist das Sensornetzwerk gezwungen eine schnelle Datenübertragung zu liefern und große Datenmengen zeitnah und zuverlässig zu verarbeiten.

Hier einige Typen von Sensorknoten:

MICA2 /MICA

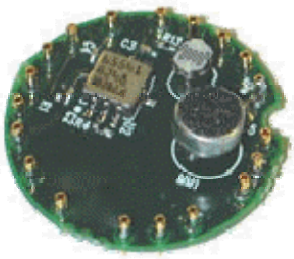
Technische Daten:
Abmessungen: 57mm(l) x 31mm(b) bzw. 25mm(D)
Prozessortyp: Atmega
Prozessorspeed: 4 Mhz
Flash: 128 KBytes;
Radio Frequency: 916 Mhz
Stromverbrauch:
- Power Down Modus < 25 µA
- Empfangen mit ca. 15 – 20 mA
- Senden mit ca. 20 – 25 mA

Betriebssystem: TinyOS

(siehe Quellenangaben 7; 28)

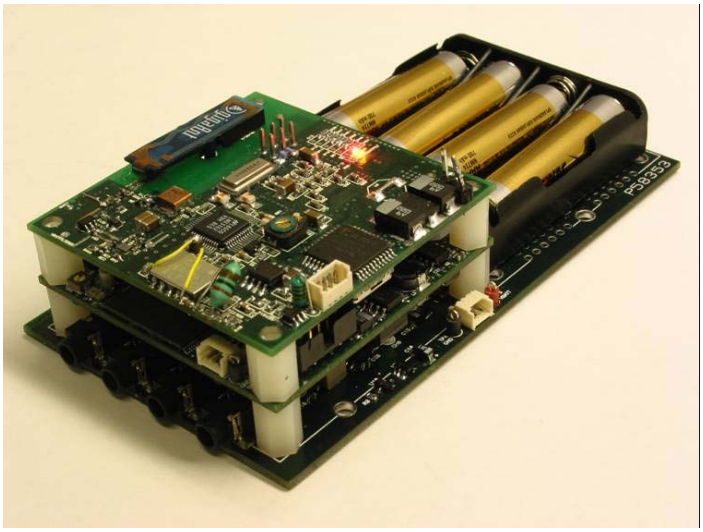
MTS510 MICA2DOT Sensor-Board
Eine weitere miniaturisierte Variante des MICA 2 Boards. Die Firma “crossbow“ liefert auch für diese Bauform eine ständig wachsende Zahl von Sensorboards, die einfach aufgesteckt werden können.

(siehe Quellenangabe 8)



(siehe Quellenangabe 9)

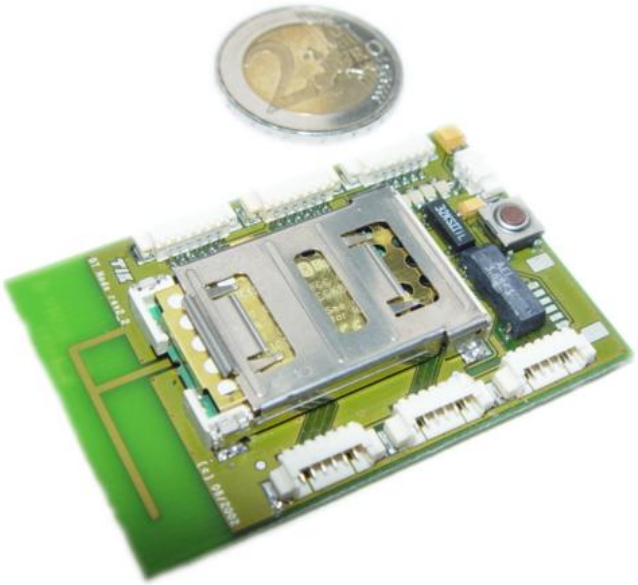
Am MIT (Massachusetts Institute of Technology) wurde dieser μamps-Sensorknoten entwickelt.
μ-Adaptive Multi-domain Power aware Sensor.



(siehe Quellenangabe 10)

BTnode features at a glance

- Mikrokontroller: Atmel ATmega 128L (8 MHz 8 MIPS)
- Memories: 64KByte RAM, 128KByte FLASH ROM, 4KByte EEPROM
- Bluetooth radio, Ericsson rok 101 007.
- External Interfaces (ISP, UART, SPI, I2C, GPIO, ADC,...)
- 4 LEDs



(siehe Quellenangabe 11)

Es befinden sich noch viele weitere verfügbare Sensorknoten auf dem Markt, deren Darstellung im Einzelnen für diese Arbeit aber nicht relevant sind. Die Zukunft wird zeigen, welche Systeme auf dem Markt bestehen werden.

2.5. Laufende Forschung und Entwicklung im Bereich Sensornetzwerke

Der Bereich Sensornetzwerke hat sich zwar erst in den letzten Jahren zu einem Interessenschwerpunkt im Bereich der Informatik entwickelt, allerdings findet man mittlerweile kaum eine Hochschule, die nicht entsprechende Projektgruppen zur Thematik der Sensornetzwerke beschäftigt. Das sowohl ob es die Hardwareentwicklung betrifft oder die Software der Knoten.

Wer nun denkt, Sensornetzwerke seien nur ein reines Forschungsprojekt an den Universitäten, dem sollte gesagt sein, das diese Technik längst die Labore und Universitäten verlassen hat und im praktischen Einsatz ist bzw. in vielen Bereichen kurz vor der Markteinführung steht.

Das Fraunhofer Institut arbeitet an miniaturisierten Sensoren zur Vitalüberwachung, um ein drahtloses Untersuchen des Patienten zu Hause zu realisieren. Weitere Ausführungen darüber sind unter der angegebenen Quelle unter dem Projektnamen Body Area Network zu finden.
(siehe Quellenangaben 16; 31)

Die Firma FMN vertreibt komplette Systeme zum Aufbau von Sensornetzwerken. Zuvor war die Firma nur im Bereich der Telekommunikation erfolgreich tätig, hat aber die Möglichkeiten von Sensornetzwerken erkannt und das Geschäftsfeld erweitert.
(siehe Quellenangabe 17)

In Bezug auf das Beispiel der Unfallwarnung, sei an dieser Stelle erwähnt, dass die Automobilindustrie schon seit einiger Zeit an einem Projekt zur Einbindung von Automobilen in einem Netzwerk arbeitet. Angestrebtres Ziel dieses Projektes ist die Unfallwarnung und die multimediale Unterhaltung / Informationsbeschaffung.
(Timo Kosch, BMW Group Forschung)

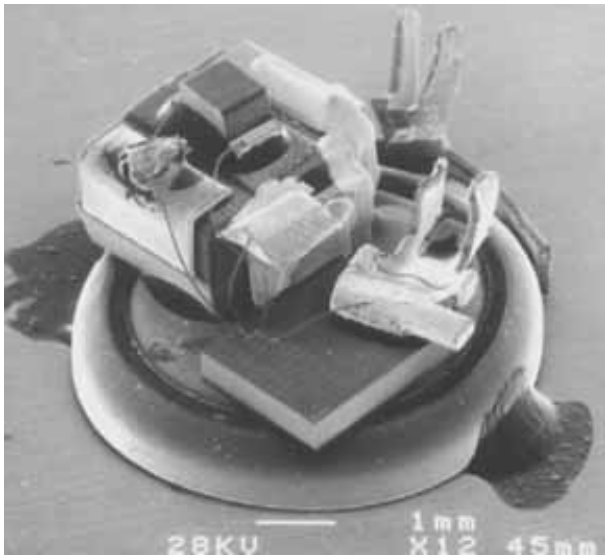
Besonders zu erwähnen sei hierbei noch mal der MICA2 /MICA dot und TinyOS (Näheres zu TinyOS im Kapitel 3.3.1.).
Gerade im Umfeld dieser Hard- und Software hat sich ein hartnäckiger Kern von Entwicklern und Usern gebildet.
Weiterführende Links zu dieser Thematik:
(siehe Quellenangabe 14; 15)

2.6. Zukunftsperspektiven Sensornetzwerke

Das Gebiet der Sensornetzwerke ist in seiner Weiterführung und Entwicklung längst nicht abgeschlossen. Täglich ergeben sich neue Anwendungsbereiche für diese Technik. Bedingt ist dies nicht zuletzt durch die Weiterentwicklung der Hardware. Die Sensorknotenmodule werden immer kompakter und verursachen dadurch auch weniger Produktionskosten.

Ein interessanter Bereich der weiteren Vernetzung ist z.B. auch der Gedanke, dass mittlerweile in fast jedem Gerät schon ein Mikrokontroller eingebaut und mit Sensoren ausgestattet ist. Ob es sich nun um eine Waschmaschine handelt oder um einen Toaster. Erweitert man diese Technik noch um eine drahtlose Kommunikationsschnittstelle, könnte man auch die Informationen vernetzen.

Ein weiteres, sehr interessantes Projekt ist, die schon zuvor erwähnte "*Smart Dust*" Idee, also die extreme Miniaturisierung der Sensorknoten. Das untenstehende Bild zeigt die bestehenden Möglichkeiten einer solchen Miniaturisierung (beachte Größenangaben).



Flashy Dust von Bryan Atwood,
Mote mit Lichtsensor und Einweg-Kommunikationstool.
(siehe Quellenangaben 18; 33)

3. Konzeptionelle Erstellung eines Sensornetzwerkes

Dieses Kapitel soll Klarheit schaffen, was bei der Erstellung eines Sensornetzwerkes zu bedenken ist. Sowohl, was die verwendete Hardware betrifft, als auch die Software.

Die grundlegenden Randbedingungen sind durch die Aufgabenstellung gegeben und werden jetzt konkretisiert. Auf die Entwicklung des Hardwaremoduls CFMW1 wird hier auch kurz eingegangen, da die Hardware eine entscheidende Rolle für die Möglichkeiten des Systems spielt.

3.1. Hardware

Da die einzelnen Sensorknoten kostengünstig sein sollen, ist dies schon bei der Auswahl der Hardware zu bedenken. Da keine große Rechenleistung notwendig ist, ist ein kleiner Mikrokontroller vollkommen ausreichend.

Durch einen einfachen Aufbau der Module ist ein recht niedriger Preis pro Modul zu erreichen, der mit Sicherheit unter dem Preis bestehender Systeme liegt.
Was die Kosten für andere Module in die Höhe schnellen lässt, ist z.B. die Ausstattung der Hardware. Da die Module schon verschiedene Sensoren besitzen. Nachteilig wirkt sich dabei aber aus, dass diese meist nicht für individuelle Zwecke genutzt werden können, weil sie sehr allgemein gehalten sind. Das spiegelt sich auch in den Kommunikationsschnittstellen wider, ob nun USB oder RS 232. Jedoch ist diese Vielfalt meistens zu groß, da man nur ein System nutzt.

Eine Reduktion der Ausstattung, das heißt die Besinnung auf das Wesentliche, spart hier enorme Kosten ein. Es erscheint mir sinnvoller, das der Anwender sich die Sensoren, die er wirklich nutzen möchte, selbst auf die Module stecken kann. Kosten verursachen dann nur die Sensoren, die auch wirklich genutzt werden, denn nur weil man ein paar Eier benötigt kauft man sich ja nicht gleich einen ganzen Bauernhof.

Ziel bei der Hardwareentwicklung ist ein kleiner kostengünstiger und modular aufgebauter Sensornetzwerkknoten.

3.1.1. Prozessor / Mikrokontroller

Benötigt wird ein recht kleiner Mikrokontroller, dessen Funktionen darin bestehen, Sensorwerte aufzunehmen und diese zu verarbeiten. Der Atmega 128 (siehe Quellenangabe 5) ist hierfür bestens geeignet, da er die entsprechende Rechenleistung besitzt und dazu nicht kostenintensiv ist.

Des weiteren bietet er die notwendigen Ein- und Ausgänge. Dieser Mikrokontroller lässt sich einfach programmieren. Er findet häufig Verwendung, was bedeutet, es wurden genügend Erfahrungen mit der Programmierung gemacht und es existieren ausgereifte Programmbibliotheken.

3.1.2. Funktechnik

Was die Funktechnik betrifft, so gibt es verschiedene Möglichkeiten, z.B. Bluetooth, WLAN etc.. Jedoch sind diese Techniken relativ teuer. Eine rudimentäre serielle Übertragung ist auch möglich, allerdings ist der damit verbundene Aufwand die Software zu erstellen immens groß.

Eine alternative Funktechnik dazu, die einfach in der Handhabung, standardisiert und kompakt gebaut ist, ist das sogenannte ZigBee. Dieses wurde speziell für Heimvernetzungen entwickelt. Besonders ist dabei der CC2420 Chip von Chipcon zu erwähnen, da es sich dabei um einen Kontroller handelt, der sämtliche Funktionalitäten übernimmt und mit einem Einzelstückpreis von ca. 5,00 € sehr günstig ist (näheres siehe Einführung ZigBee Kapitel 4).

Der ZigBee Standard ist relativ neu. Er verinnerlicht viele, für die Erstellung eines Sensorknotens, notwendige Funktionalitäten.
Durch den CC2420 Chip ist der gesamte MAC Zugriff auf der Hardware-Ebene geregelt, sodass man eine rationale Basis zum Erstellen der Applikationen vorfindet.

3.1.3. Sensorenschnittstellen

Für die Aufgabe des Monitoring sind in den meisten Fällen viele verschiedene Sensoren notwendig.
Es ist unmöglich alle denkbaren Sensoren auf dem Board anzubringen. Daher ist die Entscheidung für die Definition einer analogen und einer digitalen Schnittstelle unabdingbar. So das über diese Schnittstellen verschiedene Sensoren angesteuert werden können.

Die Sensoren werden zusätzlich auf das Modul aufgebaut, sodass ein einzelner Knoten mit verschiedenen Arten von Sensoren bestückt werden kann, je nach Anwendungsbereich.

Es ist möglich Akustik, Temperatur, Feuchtigkeit etc. zu überwachen ohne ständig die Module auszutauschen.

Denkbar wäre noch viel mehr, zumal der Atmega 128 über genügend Eingänge verfügt. Selbst analoge / digitale Ausgänge wären möglich, nur sind diese für das Monitoring nicht erforderlich und werden dementsprechend eingespart, um so die Kosten zu minimieren.

3.1.4. Kommunikationsschnittstellen

Eine Kommunikationsschnittstelle ist natürlich die Funkanbindung, die bidirektional ist.
Des weiteren benötigen die Module noch eine Schnittstelle zu einem Computer, hier ist das kostengünstigste System die RS 232 Schnittstelle.

Eine alternative Kommunikationsschnittstelle kann ISP sein, um das Modul zu programmieren. Außerdem kann so dem Benutzer die Möglichkeit gegeben werden, eigene Software zu entwickeln bzw. die Bestehende zu erweitern.

3.1.5. Unterschiede in der Ausstattung Basismodul und Sensorknoten

Generell existieren zwei Varianten zur Nutzung des Moduls

1. Nutzung als reinen Sensorknoten, der die Daten aufnimmt, verarbeitet und weiterleitet.

2. Nutzung als Basisstation, die sämtliche Daten an einen Rechner weitergibt, der dann diese graphisch darstellt und dem Benutzer des Netzes zur Verfügung stellt. (Mit Basismodul ist hier nicht ein Modul gemeint an dem sich alle Sensoren anmelden.)

Die Frage, ob das Basismodul und das Sensormodul sich im Hardwareaufbau unterscheiden sollten oder nicht, ist von verschiedenen Faktoren abhängig.

Der Vorteil, dass das Basismodul und die einzelnen Knoten baugleich sind, liegt natürlich darin, dass nur ein Hardwareaufbau notwendig wäre. Außerdem würde die Möglichkeit bestehen, beim Ausfall des Basismoduls einen Sensorknoten als Ersatz zu benutzen.
Die Sensordaten können nicht nur von einer Schnittstelle aus dem Netz empfangen werden, sondern man kann an jedem Sensorknoten einen Rechner anschließen und die Daten dort aufnehmen. Dies bedeutet, das eine Doppelauswertung möglich ist, ohne die Daten wieder vom Rechner per Kabel an eine Nebenstelle zu senden (solange diese in der Reichweite des Netzwerkes liegt).

Der Nachteil ist jedoch der Kostenfaktor, denn wenn alle Module gleich sind, muss jeder einzelne Knoten eine weitere Kommunikationsschnittstelle (RS 232) haben. Darüber hinaus ergibt sich eine größere Bauform, durch die höhere Anzahl an Bauteilen. Das Thema Hardware wird jedoch ausführlicher in der Arbeit CFMW1 behandelt. Bei meiner Ausarbeitung bin ich von identischen Modulen ausgegangen, auf Grund der sich daraus ergebenen Vorteile.

3.2. Resümee

Da das CFMW1 Modul zum Zeitpunkt der Erstellung dieser Arbeit noch nicht verfügbar war, musste ein äquivalentes Entwicklungsboard gefunden werden, welches vom Aufbau der Hardware dem CFMW1 am Besten entspricht. Hier bot sich als Lösung das sogenannte “CC2420dbk“ Board von Chipcon an.

(siehe Quellenangabe 20)

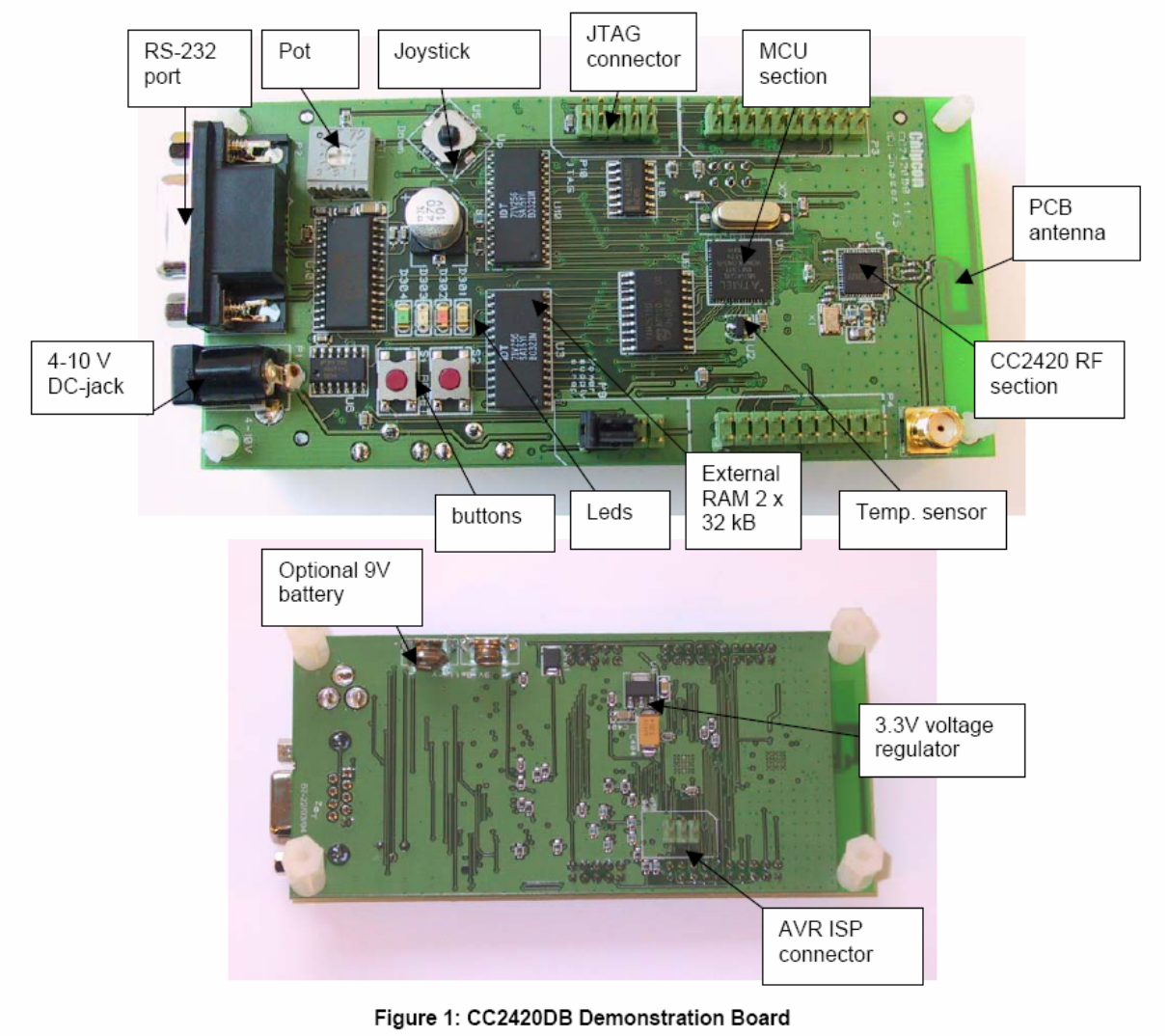


Figure 1: CC2420DB Demonstration Board

(siehe Quellenangabe 19)

Dies ist im Wesentlichem identisch im Aufbau mit dem CFMW1 Board.

Technische Daten:

- Atmel Mega128 Mikrokontroller
- CC2420 ZigBee Funkchip
- RS 232 Schnittstelle
- zwei Analogsensoren (Temperaturmesser, Potentiometer)
- ein Joystick als Digitalsensor
- ISP Schnittstelle

Die weitere Ausstattung, wie der zusätzliche Ram-Speicher, Taster etc., werden an dieser Stelle vernachlässigt, weil diese aller Wahrscheinlichkeit nach, auf dem CFMW1 Board auch nicht zur Verfügung stehen werden.

3.3. Software

Die laufende Software auf den Sensorknoten ist entscheidend für die Anwendung der Module, deren Funktionalität und Anwendungsbereiche.

3.3.1. Bestehende Systeme

Es gibt viele verschiedene Betriebssysteme für Sensorknoten.
Im Großen und Ganzen ist aber das *TinyOS* (*TinyOS* Mikrothreading Operating System), das Gebräuchlichste der angebotenen Systeme. Im Folgenden eine kurze Darstellung was sich hinter *TinyOS* verbirgt.

TinyOS ist eine effiziente und modulare Softwareplattform für Mote-Module. Dabei handelt es sich um eine Entwicklung der UC Berkeley mit Open Source Code und wird aktiv durch eine große Gemeinde von Anwendern unterstützt.

Dieses System arbeitet mit einzelnen Modulen, die für bestimmte Funktionalitäten eingebunden werden (Routing, Sensorabfragen, Timer, usw.).
TinyOS unterstützt mittlerweile eine recht große Anzahl von Hardwaremodulen, so z.B. Mica Mica2, telos, mica2dot usw..

TinyOS ist ein eventbasiertes Betriebssystem, das mit verschiedenen Tasks arbeitet, die untereinander mit Events kommunizieren. Unterbrochen werden können diese Tasks nur durch Systemereignisse.

Prinzipiell besticht *TinyOS* durch seinen modularen Aufbau, sodass man für die verschiedenen Funktionalitäten einfach die entsprechenden Module einbinden kann.

Component Name	Code Size (bytes)	Data Size (bytes)
Multihop router	88	0
AM_dispatch	40	0
AM_temperature	78	32
AM_light	146	8
AM	356	40
Packet	334	40
RADIO_byte	810	8
RFM	310	1
Photo	84	1
Temperature	64	1
UART	196	1
UART_packet	314	40
I2C_bus	198	8
Procesor_init	172	30
TinyOS scheduler	178	16
C runtime	82	0
Total	3450	226

(siehe Quellenangabe 29)

Diese Tabelle veranschaulicht anhand der Code-Größen, wie klein und kompakt das Betriebssystem ist.
Nun liegt natürlich die Überlegung nahe, ob man nicht eines der bestehenden Betriebssysteme auf die CFMW1 Hardware Module konvertieren sollte.

Doch es herrschen ausreichend Gründe vor, die gegen eine Konvertierung sprechen.

- *TinyOS* stellt nur eine sogenannte Entwicklungsumgebung für die eigene Software dar. Es stehen einige Funktionalitäten zur Verfügung, die als reine Hardwareabstraktion verstanden werden können.
- *TinyOS* ist auf Grund der vollkommen anderen Funkhardware nicht so einfach und problemlos auf das CFMW1 Modul zu konvertieren.
- Die Entwicklung des eigenen Netzes hat einen eng begrenzten Zeitrahmen, daher ist der zeitliche Aufwand um eine bessere Entwicklungsumgebung zu erhalten nicht gerechtfertig. Die eigentliche Software für die Knoten müsste trotzdem noch erstellt werden.

Für die Benutzung der CFMW1 Module in weiteren Anwendungsbereichen, wäre eine Umgebung ähnlich dem *TinyOS* jedoch wünschenswert. Dies sollte aber eher Gegenstand einer eigenständigen Arbeit sein.

3.3.2. Anforderungen an die eigene Software

Vor der Erstellung einer eigenen Software auf den Sensorknoten, ist es wichtig für sich zu definieren, was das System leisten muss.

Welche Aufgaben sollen erfüllt werden.

Was ist unbedingt notwendig für die bestehende Aufgabe.

Die Hauptaufgabe besteht in der Überwachung eines Hauses. Die zentralen Fragen bei der Erstellung der Software sind somit:

- Was sind die Aufgaben?
- Wie oft sollen die Daten erfasst werden?
- Was soll mit den Daten passieren?
- Wie entsteht die Vernetzung?
- Wie und in welcher Form werden Daten übermittelt?
- Wie werden Daten ausgewertet?
- Wie groß soll sich das Netz ausdehnen?
- Welche Datenmengen werden auftreten?

4. Einführung in ZigBee

(Das folgende Kapitel 4 und die darin enthaltenen Grafiken basieren auf den Quellen 1; 26)

ZigBee ist ein offener Funknetz-Standard für Sensor- und Aktor-Netzwerke, der gerade in den Markt eintritt. Bestimmende Eigenschaften von ZigBee sind die Zuverlässigkeit des Systems, der geringe Leistungsverlust und die niedrigen Kosten.

Die Entwicklung von ZigBee basiert auf dem Standard 802.15.4. Eine Arbeitsgruppe von Philips Electronics, die sich Ende 1998 gründete, lieferten als Ergebnis einen Entwurf einer günstigen, störungsunempfindlichen Funklösung ab. Die ersten Präsentationen der Ergebnisse im Oktober 1999 sind geprägt vom Begriff „Protocol for Universal Radio Links“ (PURL).

Die ZigBee Alliance bildete sich als Kooperation von verschiedenen Industrieunternehmen Ende des Jahres 2002.
Während Vorschläge zur Bitübertragungs- und Sicherungsschicht aus PURL 2001 der neuen IEEE 802.15.4 Arbeitsgruppe offeriert wurde, machte es sich die ZigBee Alliance zum Ziel, die oberen aufgesetzten Protokollschichten bis hin zum eigentlichen Applikationsteil zu entwickeln und zu definieren.

Der wichtigste Baustein in der Entwicklung von ZigBee stellt die Verwendung des 802.15.4 Standards dar. Dessen Bearbeitung heute weitestgehend abgeschlossen ist. Es wurde eine stabile Grundlage für die Halbleiterhersteller zur weiteren Entwicklung geschaffen. Heutzutage verfügbare Modelle findet man von Atmel, Chipcon, Motorola und ZMD.

Jedoch eine brauchbare Entwicklung der höheren Layer war noch nicht fertiggestellt. Ember, Helicomm und figure8wireless versuchten sich an einer ersten Implementierung der zur Zeit vorhandenen Spezifikation der ZigBee Protokolle.

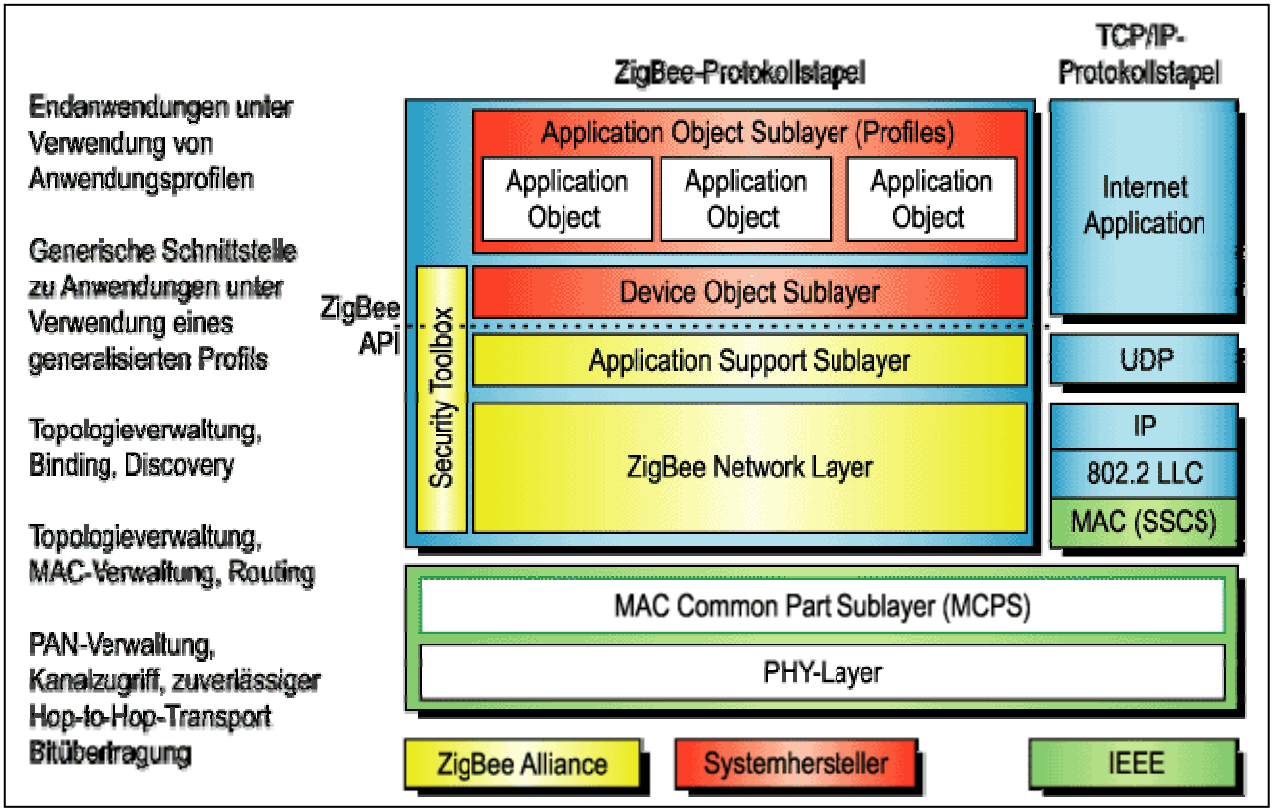


Bild 1: Der 802.15.4-Standard bildet eine Implementierung der Bitübertragungs- und der Sicherungsschicht (unten). Auf diesen setzt ZigBee auf. Der ZigBee Network Layer (NWK) übernimmt Aufgaben wie Verwaltung der ZigBee-bezogenen Netzwerk-Topologie, des Routing und des Sicherheitsmanagements. Das ZigBee Application Programming Interface stellt die Schnittstelle zu den Anwendungen dar. Diese werden in sog. ZigBee-Profilen beschrieben. Hierbei kann es sich um standardisierte Profile (wie Light Sensor oder Light Controller für einfache Schaltanwendungen) oder um proprietäre Profile handeln.

Bild 1 stellt den Protokollstack des IEEE 802.15.4 und dem darauf aufbauenden ZigBee dar.
Neben der Benutzung von ZigBee Protokollen, ist es ebenso möglich die Module über Internetprotokolle anzusprechen. Dazu ist lediglich ein „Service Specific Convergence Sublayer“ SSCS notwendig, der die 802.15.4 Mac und die IP miteinander verkoppelt. Der Funknetzstandard 802.15.4 ist ein Entwurf eines einfachen und kostengünstigen Standards, dessen Systembeschreibung dennoch aber mehr als 600 Seiten umfasst. Kritiker befürchten das die beeindruckende Vielfalt von Betriebsoptionen, also gerade diese flexible Anpassung an die vielen verschiedenen Gegebenheiten der Interoperabilität, in der Praxis im Weg stehen könnte.

Ein Vorteil des ZigBee-Standards. Mit seinem Zugriff auf das die Mac-Schnittstelle wird die Anpassung über die LLC-Schicht gespart.

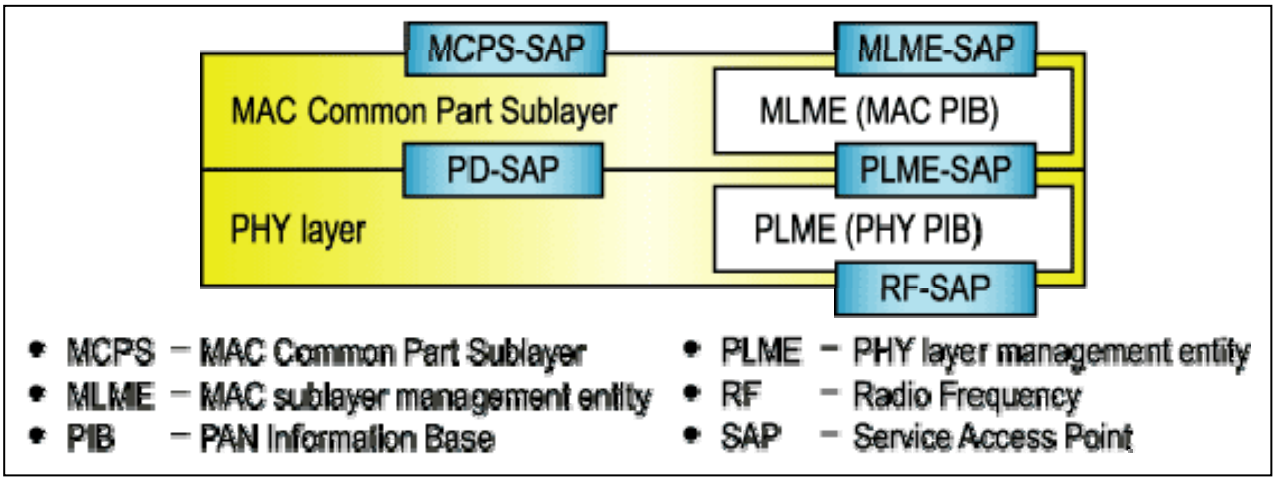


Bild 2: Grundaufbau 802.15.4
MAC und PHY sind in jeweils zwei Bestandteile aufgeteilt. Die Behandlung des Datenverkehrs übernimmt die eigentliche Protokolleinheit (Entity). Zusätzlich existieren sog. Management Entities, die die Verwaltung des Netzwerks übernehmen. Diese Aufteilung ist auch schon von anderen Funkstandards, wie dem 802.11, verwendet worden.

Durch drahtlose Netze erfolgt eine Aufteilung in Datenübertragungs- und Managementteilschichten. Diese Aufteilung stellt eine sinnvolle und notwendige Erweiterung des Standards dar, denn bei verkabelten Systemen ist die Bedeutung der Management-Funktionalitäten viel geringer als bei drahtlosen. Die Konfigurationsparameter und die bekannten Informationen über vorhandene Topologien werden in der PAN Information Base in der Managementteilschicht gespeichert.

4.1. Netzwerkstruktur des ZigBee-Standards

Die *Reduced Function Devices (RFD)* und die *Full Function Devices (FFD)* sind die beiden Geräteklassen die man beim 802.15.4 Standard unterscheidet.
Zu beachten hierbei ist eine Kommunikation zwischen *FFD* und *RFD* ist möglich genau wie die Kommunikation zwischen zwei *FFD´s. RFD´s* können untereinander nicht direkt kommunizieren.

Folglich lassen sich *RFD´s* einfach und ohne hohe Kosten einsetzen, wobei auch eine Softwareprogrammierung auf einem 8-Bit-Mikrokontroler möglich wird.

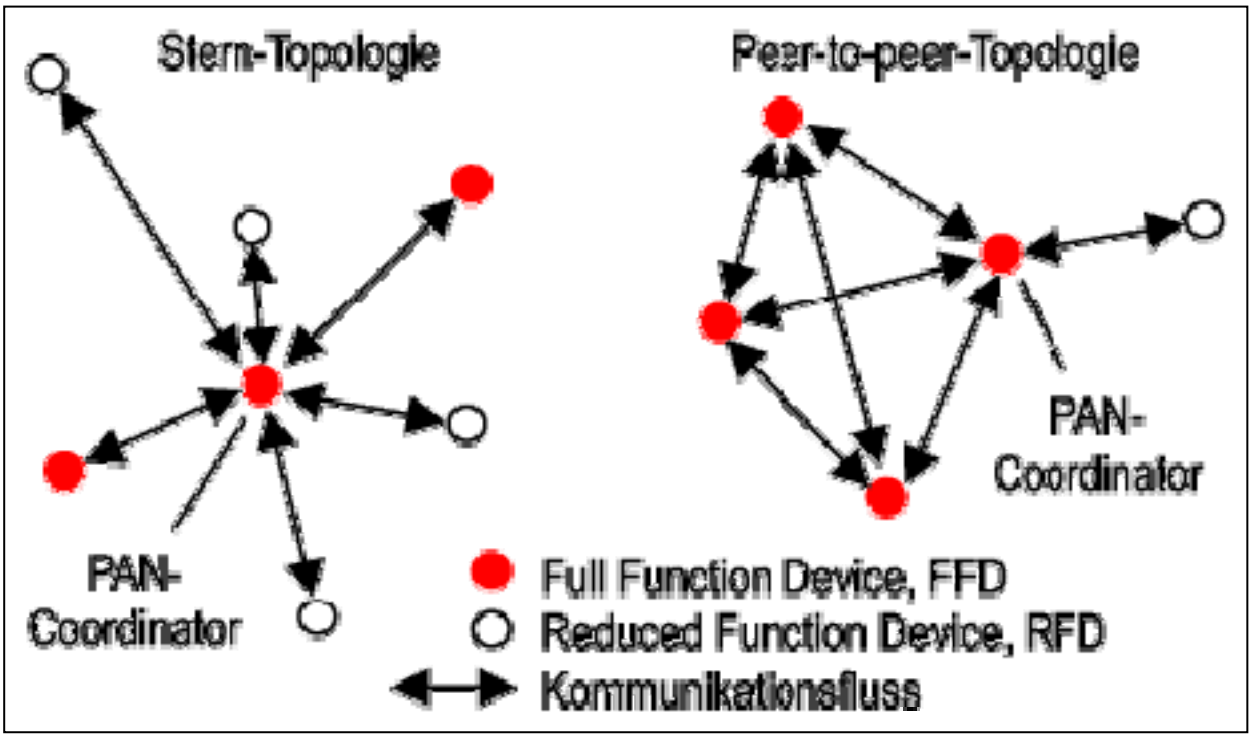


Bild 3 Topologien des 802.15.4

Die im oberen Bild 3 gezeigten Topologien, Sterntopologie und peer-to-peer-Topologie, lassen sich durch die beiden Geräteklassen *FFD* und *RFD* bilden. Dies gilt als Basis zum Anlegen von flexiblen vermaschten Netzwerken mit hoher Leistung. Da diese Protokolle zur Vermaschung nicht im Standard enthalten sind, wird folgende Vorgehensweise vorgeschlagen. Die Benutzung von Cluster-Trees, wobei einzelne Netz-Cluster zusammengefasst werden. Durch Protokolle zur Vermaschung verlässt man den 802 Standard, da diese im OSI Referenzmodell in einer höheren Ebene (Layer 3) liegen und geht somit über zum aufbauenden ZigBee-Standard, der noch definiert werden muss.

Von großer Bedeutung ist im 802.15.4 Standard die Funktion eines Koordinators. Dieser Koordinator ist verantwortlich für die Synchronisierung des Netzwerkes, dies geschieht durch versenden von Broadcastnachrichten (*beacons*), daher darf es auch nur genau einen geben.
Es sind jedoch alternative PAN Koordinatoren denkbar, die im Falle des Ausfalls des PAN Koordinators deren Aufgabe übernehmen.
Somit kann ein *FFD* mehrere Aufgaben haben. Ein *FFD* kann als Gerät, als PAN Koordinator oder als Koordinator zwischen mehreren Netzen betrieben werden.

Der Datentransfer kann durch drei Möglichkeiten erfolgen

- Datenübertragung von Device zum Koordinator
  (möglich bei Stern- & peer-to-peer-Topologie)
- Datenübertragung vom Koordinator zum Device
  (möglich bei Stern- & peer-to-peer-Topologie)
- Datenaustausch zwischen zwei Stationen im Netzwerk
  (möglich bei peer-to-peer-Topologie)

4.2. Adressierung im ZigBee-Standard

Zur Adressierung werden die 64 Bit strukturierten Extended Adresses des IEEE-Standards benutzt. Durch die Möglichkeit die vollständigen Adressen zu übernehmen, können sehr große Netzwerke aufgebaut werden.

Im Fall, dass eine große Anzahl von Devices angemeldet sind, erscheint dies sinnvoll und stellt darüber hinaus einen wesentlichen Vorteil zum Bluetooth-Netz dar, das nur sieben Stationen beinhalten kann, die sich auch immer wieder neu anmelden müssen. Folglich werden so der Energieverbrauch und die Verzögerungszeiten reduziert.

In einem PAN können aber auch 16 Bit Adressen verwendet werden, die jede Station beim Anmelden vom PAN-Koordinator zugewiesen bekommt, dies spart ohnehin die Übertragung der langen 64 Bit Adressen ein.

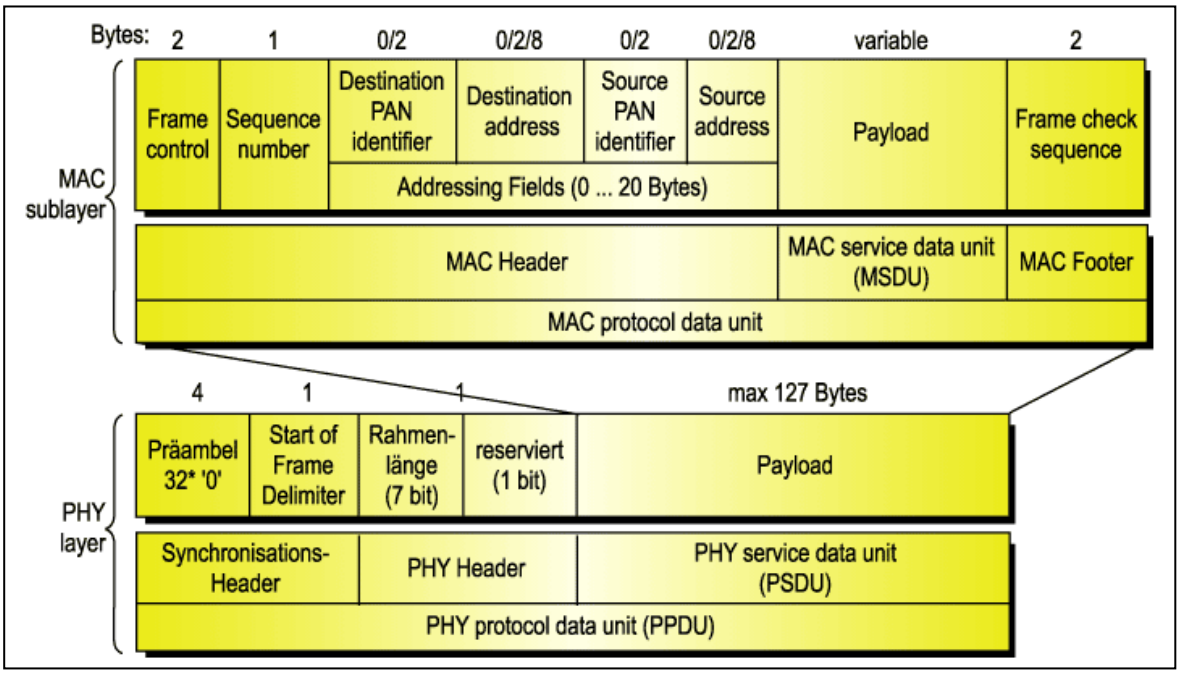


Bild 4 Die Struktur des 802.15.4-Rahmens entspricht weitestgehend der des 802.11-Standards.

- Auf MAC-Ebene stehen vier verschiedene Rahmenarten zur Verfügung: *beacon*, Data, ACK, MAC command.
- Auch die PHY-Ebene ergänzt den zu übertragenen Rahmen um layerbezogene Steuerinformationen.
- Diese werden durch einen Synchronisations-Header ergänzt, der – wie auch bei den drahtgebundenen Systemen – die Synchronisation des Empfängertakts auf den Sendetakt ermöglicht.

Im 802.15.4 –Rahmen ist die maximale Größe auf Mac-Ebene auf 127 Byte begrenzt. ZigBee schießt darüber hinaus und macht es möglich noch größere Pakete fragmentiert zu versenden. Die Rückmeldung über den erfolgreichen und fehlerfreien Empfang eines Datenpakets an den Sender, wird mit Hilfe eines Acknowledgement-Rahmens gewährleistet.

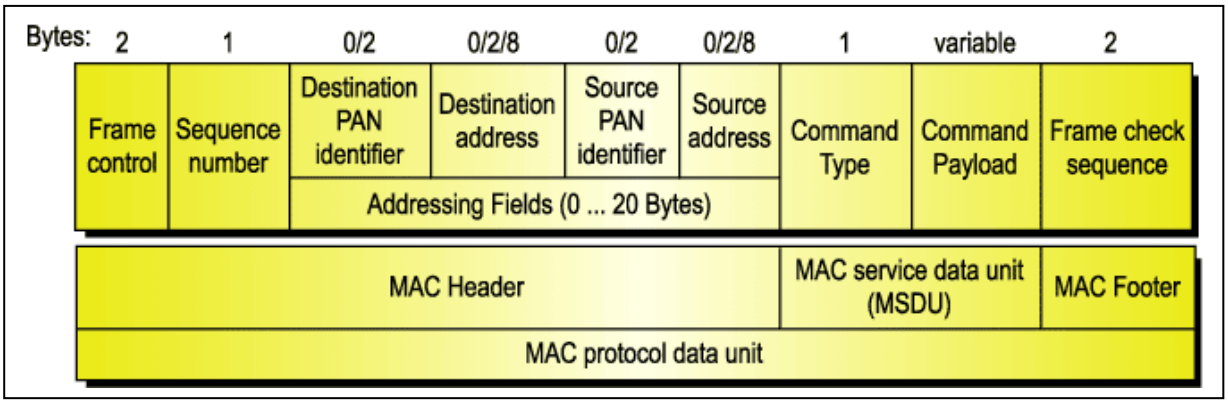


Bild 5 Mit Hilfe der Command-Rahmen kann beispielsweise eine zentralisierte Verwaltung eines Netzwerks realisiert werden.

Die abgebildeten Command-Rahmen stellen den Mechanismus zur Steuerung und Konfiguration von Knoten dar. Die sogenannten *beacon*-Rahmen, im unten angeführten Bild, ermöglichen eine Erweiterung der Netzfunktionalitäten.

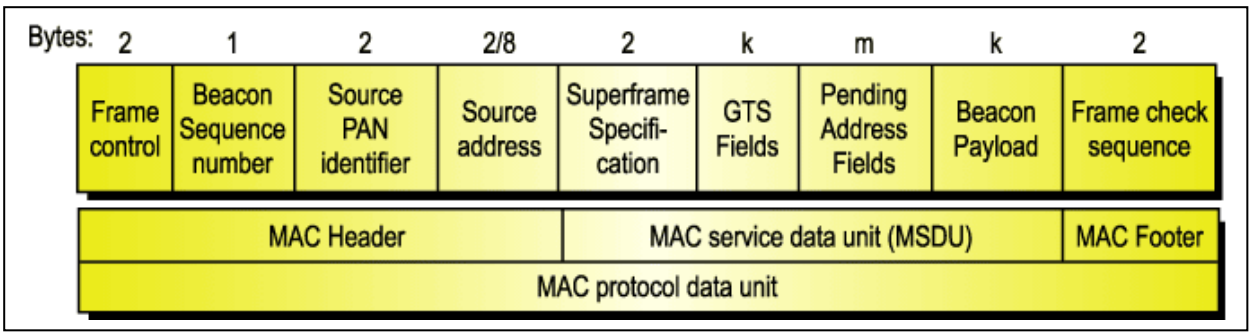


Bild 6 Die *Beacon*-Rahmen erlauben die Synchronisation der Netzteilnehmer – auch in Bezug auf ihre Power-down-Zeiten.

Durch das CSMA/CA-Verfahren erfolgt der Kanalzugriff. Jede Station überprüft innerhalb einer zufälligen Zeitspanne eventuelle Aktivitäten anderer Stationen und beginnt dann mit der Übertragung. Vorherrschende Problematik hier, es kann von der sendenden Station nicht sichergestellt werden, das die Daten ungestört beim Empfänger auflaufen.
Die Empfangsbestätigung (Acknowledgement) durch die Empfängerstation auf der Sicherungsschicht stellt eine Hop-to-Hop-Verlässlichkeit dar. Es kann aber auf Grund der Near-Far-Problematik, senderseitig nicht sichergestellt werden, dass die gesendeten Pakete ungestört ankommen. Es ist eben nur ein Versuch der Kollisionsvermeidung, jedoch keine Sicherstellung.

Beim 802.15.4 Standard ist die Versendung einer Empfangsbestätigung optional. Dies ist durchaus ein Vorteil, denn auf Basis von Sensornetzwerken erscheint der Versand einer Empfangsbestätigung wenig sinnvoll. Bei den regelmäßig versendeten Sensordaten sind immer nur die letzten Messwerte relevant, ein Nachsenden älterer Daten würde nur unnötig den Netzdurchlauf belasten.

Des weiteren kann dies auch deaktiviert werden, indem man durch die unten dargestellte Superframe-Struktur (Bild 7) Zeitspannen reserviert, für kritische Daten.

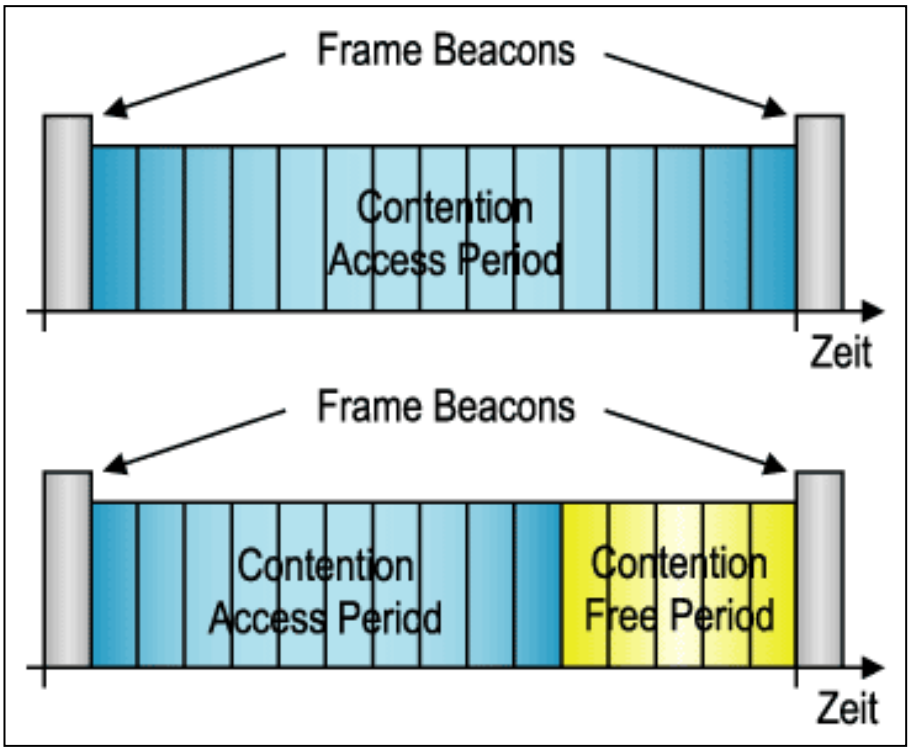


Bild 7 (oben: normal; unten: garantierte Zeitschlitze).

Die Zuteilung der Zeitspannen ist Aufgabe des PAN-Koordinators, aber nur möglich in *beaconenabled* Netzen.

4.3. Schnittstellen im ZigBee-Standard

Neben der Übertragung von Nutzdaten muss die Bitübertragungsschicht einige Funktionen mehr erfüllen. Da mit deren Hilfe die Möglichkeit besteht adaptive Netze zu realisieren, wird diesen eine große Bedeutung beigemessen:

- Aktivierung und Deaktivierung des Steuermoduls
- Energiemessungen in ausgewählten Kanälen
- Informationen über die Qualität der Luftschnittstelle für empfangene Rahmen
- Signalisierung freier Kanäle

Verschiedene Varianten werden von Funkschnittstellen geboten. Das global verfügbare 2,4-GHz-ISM-Band ist für 16 Kanäle mit einer maximalen Datenrate von 250 kbit/s geschaffen, um auch bei hoher Auslastung des 2,4-GHz-Bandes den Ablauf sicherzustellen, sind weitere Frequenzbänder definiert.

- Für Europa der Kanal bei 868,3 MHz mit 20 kbit/s
- Für USA/Pazifik zehn Kanäle zwischen 902 und 928 MHz im 2 MHz Abstand und 40 kbit/s

	Kanalparameter			**Datenparameter**			**Spreizparameter**	
Bereich	**Frequenz-band**	**Region**	**Kanal-nummer**	**Daten-rate**	**Symbol-rate**	**Symbole**	**Chiprate**	**Modulation**
868/915 MHz	868 – 868,6 MHz	Europa	0	20 kbit/s	20 kBaud	binär	300 kchips/s	BPSK
	902 – 928 MHz	Amerika	1 – 10	40 kbit/s	40 kBaud	binär	600 kchips/s	BPSK
2,4 GHz	2400 – 2483,5 MHz	weltweit	11 – 26	250 kbit/s	62,5 kBaud	16fach (orthogonal)	2000 kchips/s	O-QPSK

Tabelle der Kanal-/Datenparameter in den unterschiedlichen Regionen

Hier nur eine kurze Darstellung der Modulationsverfahren:

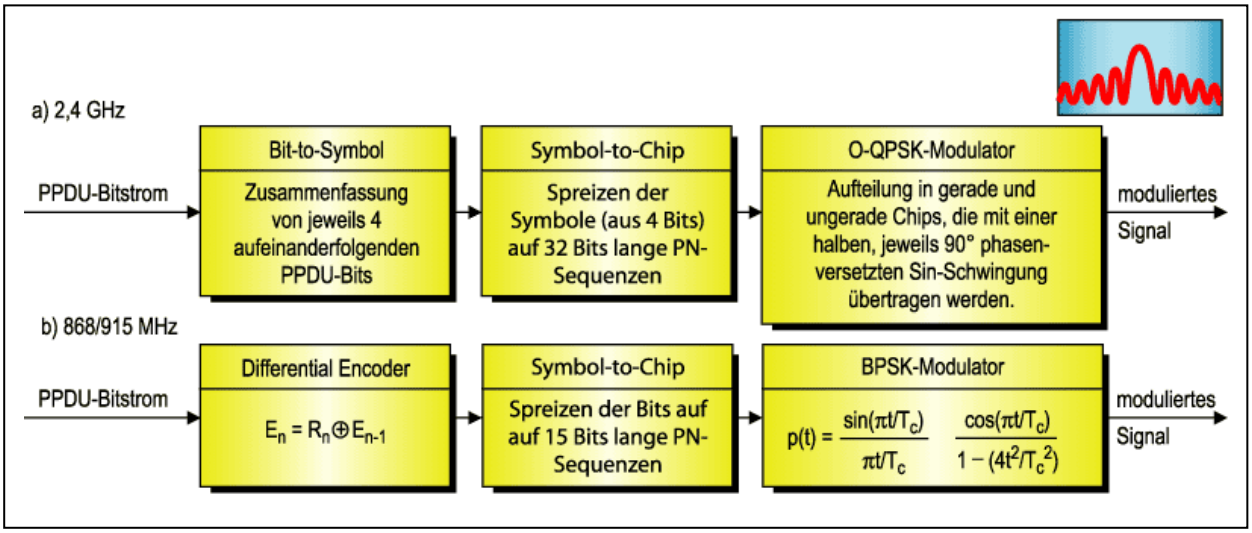


Bild 9 Die Verarbeitungsschritte bei den Modulationsverfahren des 802.15.4 in den verschiedenen Frequenzbereichen; a) 2,4 GHz; b) 868/915 MHz.

Die Störempfindlichkeit gegen schmalbandige Störer, kann durch dieses vereinfachte Verfahren, reduziert werden. Anstatt wie bei Bluetooth, das Frequenzverfahren FHSS, wurde hier das Direct-Sequence-Spread-Spectrum-Verfahren (DSSS) vorrangig eingesetzt. Aus dem einfachen Grund, das beim FHSS-Verfahren die Synchronisation der kommunizierenden Geräte notwendig ist, um den Frequenzsprung gleichzeitig durchführen zu können. Dies ist aber schwer realisierbar, wenn Geräte lange Power-down-Phasen aufweisen.

Während sich Chipcon und Motorola auf den global verfügbaren und leistungsstarken 2,4-GHz-Bereich konzentrieren, implementieren Atmel und ZMD den Sub-GHz-PHY.

4.4. ZigBee in der Praxis

Der ZigBee-Standard beschreibt das Protokoll für eine verbindungslose Ende-zu-Ende-Datenübertragung. Die Implementierung dieser Protokolle wird durch die Netz-Infrastruktur verwaltet und ist daher nur in ZigBee-Koordinatoren existent. Folgende Aufgaben sind dabei zu erfüllen:

- konfigurieren und administrieren der Netz-Topologie
- konfigurieren und administrieren der logischen Verbindungen zwischen ZigBee-Geräten, mit Hilfe von Pairing-Tables
- Transfer von Daten zwischen 802.15.4-Mac-Layer und der Anwendungsschicht
- Datenrouting von Gerät zu Gerät oder zur Anwendungsschicht, abhängig von der aktuellen Pairing-Table

Das Datenrouting soll unabhängig vom Aufbau des Netzes über einen oder mehrere Knoten erfolgen können. Ebenso sollen auch vermaschte Netze unterstützt werden. Dabei handelt es sich um sogenannte Multi-Hop-Netzwerke. Die Knoten erfüllen Weiterleitungs- und Routingaufgaben, sie sind nicht nur Endsysteme, sondern auch Zwischensysteme. Die transparente Pflege der Routing-Tabellen werden durch dynamische Routing-Protokolle gestützt.

Im ZigBee-Standard ist die Verwirklichung der Anwendungsschicht nicht unmittelbar definiert, sie werden in den Applikationsprofilen beschrieben. Darin beinhaltet sind für spezifische Anwendungen, der Aufbau der Geräte und deren Verhalten und Interaktionen. Geplant ist hier eine Gliederung nach Anwendungsgebieten, gegenwärtig erarbeiten ZigBee-Gremien Profile zur Gebäudeautomatisation, Haustechnik, Industriesteuerung, Fernsteuerung und Consumer-Elektronik. Darüber hinaus werden zukünftig noch weitere Profile erwartet, auch proprietäre sollen möglich sein.

Der ZigBee-Standard bezeichnet und beschreibt sowohl die Schnittstelle zur Anwendungsschicht als auch ein übergreifendes Gerätemodell und einen grundlegenden Befehlssatz. Die erarbeiteten Profile sind nicht Bestandteil des Standards. Im sogenannten General Operational Framework (GOF) wird die Schnittstelle dargestellt, es enthält zurzeit folgende Faktoren:

- Adressmodi
- Objektmodell
- Gerätebeschreibung (Typ; Energiequelle; Stromsparmodi; Funktion als Koordinator)
- Befehlssatz (Set/Get; Method; Event; Transparent und die dazugehörigen Antworten)
- Datenformate (linear; exponentiell, relative Zeit oder String)
- Fehlersignalisierung
- Fehlerbehandlung auf Anwenderebene

Über den Standard 802.15.4 hinaus stellt ZigBee Mechanismen zu sicheren Verteilung der Schlüssel im Netzwerk als auch zu sicheren Fern-Administration des Netzwerks zur Verfügung, mit der sogenannten „ZigBee security toolbox“.

Diese Standards stellen einen rationalen Kompromiss für kostengünstige Lösungen dar. Zum heutigen Zeitpunkt besteht bereits die Möglichkeit sinnvolle Entwicklungen zu starten, wie man an dieser Diplomarbeit sieht.

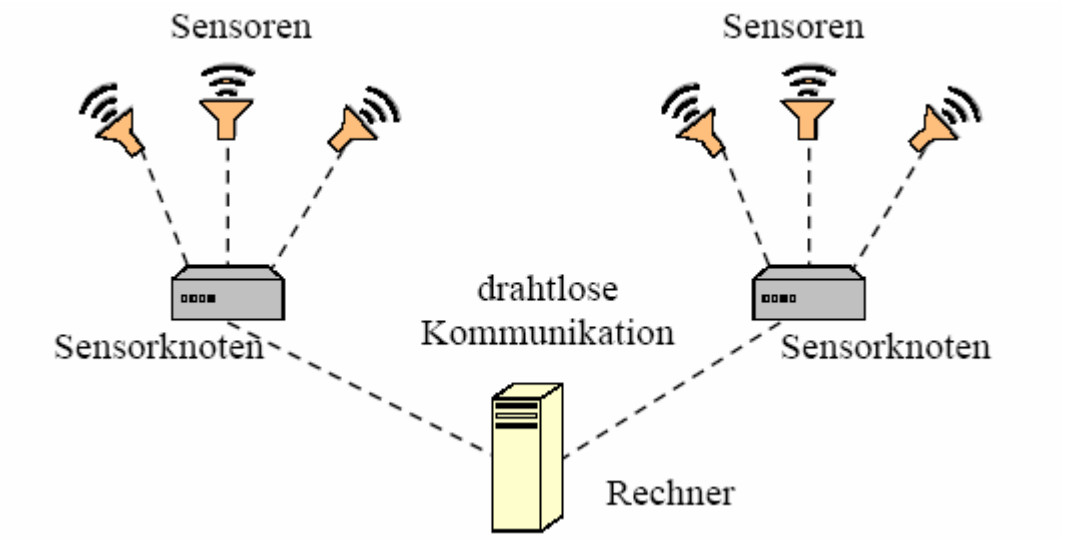
5. Entwicklung eines Protokolls auf ZigBee - Basis

Als Basisfunksystem ist nun ZigBee festgesetzt, jedoch nicht wie man seine Applikation über dieses Funksystem darstellt. Da Leider nicht alle ZigBee-Funktionalitäten durch den CC2420 Chip und den dazu gehörigen Programmlibraries gegeben sind, ist man gezwungen einiges selbst zu erstellen und zu programmieren.

5.1. Art der Netzwerkbildung / Netztopologie

Es existieren verschiedene Möglichkeiten von Netzwerk-Topologien.

Sterntopologie:



(siehe Quellenangabe 2)

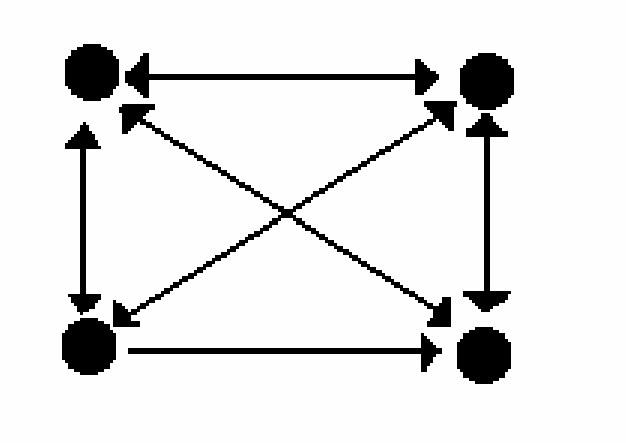
Bei dieser Variante melden sich alle Knoten bei einer Basisstation an. Dieses Verfahren wird z.B. im Bereich des WLAN benutzt. Der Nachteil dieser Variante ist jedoch schon aus der Graphik zu ersehen, die maximale Ausdehnung des Netzes entspricht gerade mal dem doppelten Funkradius eines Moduls. In eng begrenzten Flächen wäre dieses Verfahren einsetzbar, und auch nur dann, wenn es keine störenden Einflüsse gibt. Die Vorteile liegen zum Einen in einer zentralen Netzwerkadministration, einem einfachen Ein- und Austritt ins Netzwerk und einer zentralen Vergabe von Adressen.

Wenn man die Idee einer Sterntopologie weiter denkt, also mehrere solcher Netze verbindet, könnte man so das Problem der Begrenzung lösen.

Verbindet man also mehrere Rechner, die jeweils ein eigenes Netz besitzen, diese Verbindung ist natürlich auch per Funk möglich, so ist die Gesamtnetzgröße unbegrenzt. Die Verwaltung des Gesamtnetzes ist natürlich nun dezentral, aber auch nicht sehr aufwendig.

Aber es entstehen Probleme im Bereich der Ausfallsicherheit, denn fällt eine Verbindung zwischen zwei Rechnern aus, so kann es sein das große Bereiche des Netzes nicht mehr erreichbar sind.

Eine Topologie, die eine sehr große Ausfallsicherheit bietet und auch eine prinzipiell unbegrenzte räumliche Netzausdehnung ist ein peer to peer Netzwerk.



Bei dieser Art der Vernetzung kann jeder Knoten mit allen Knoten in seiner Reichweite kommunizieren.

Dieses Modell wirft dennoch wieder neue Fragestellungen auf.

Wie und auf welchem Wege sind einzelne Knoten zu erreichen?

Wie werden die Adressen der einzelnen Knoten vergeben?

Wie erfolgt die Adressierung?
(Das Thema Adressierung wird im Kapitel 5.2. behandelt)

Das Routing in einem solchen Netz ist nicht ganz so einfach. Doch es lässt sich durch verschiedene Mechanismen handhaben. Die verschiedenen Routingalgorithmen alle darzustellen wäre jedoch übertrieben und würden den Rahmen dieser Arbeit sprengen. Dazu findet man ausreichend Ausarbeitungen in der entsprechenden Fachliteratur.

Eine sehr einfache Variante, um Informationen in einem solchen Netz zu verbreiten, ist das Broadcastverfahren. Dabei sendet jeder Sensorknoten seine Informationen an alle erreichbaren anderen Knoten. Jeder Knoten sendet alle erhaltenen Pakete einfach per Broadcast weiter, das heißt jeder Sensorknoten arbeitet wie ein Repeater. Ein herkömmlicher Repeater arbeitet zwar auf einer anderen Schicht des ISO/OSI Referenzmoduls, aber die Funktionsweise ist die gleiche.

So pflanzen sich sämtliche Informationen im Netz automatisch fort. Die dadurch auftauchenden Probleme wie Flooding und Kollisionen, werden in dem Kapitel 5.4. dieser Arbeit genauer erläutert.

Für meine Ausarbeitung halte ich das Broadcastverfahren für das geeigneteste, da dadurch kein Verwaltungsaufwand für Routingtabellen oder deren Austausch bzw. Aktualisierung entsteht.

Zwar ist dieses System in Bezug auf den Energieverbrauch der einzelnen Knoten nicht ideal, da das Senden und Empfangen von Daten einen hohen Verbrauch fordert ,aber für den Anwendungsfall der Gebäude- und Systemüberwachung kann man von einer ständigen Stromversorgung ausgehen, die auch keiner unnötigen Verkabelung bedarf.

An dieser Stelle ist noch kurz anzumerken, das es Möglichkeiten der Energieersparnis und der Routingoptimierung. Dies kann durch eine Koordination per *beacon* realisiert werden, um ein Netz oder einzelne Knoten für eine gewisse Zeit “schlafen“ zu legen. Routingalgorithmen wie link-state, Distance-Vector, bgp etc. sind auch bekannt und sorgen für eine Reduzierung des Datentransfers im Netz.

5.2. Adressierung

Adressierung ist eine der wichtigsten Faktoren im Zusammenhang mit dem Netz, denn erst durch die Festlegung einer Adresse wird ein Knoten eindeutig bestimmbar.

Die grundlegende Form der Adressierung ist durch das ZigBee vorgegeben. Man könnte zwar noch eine Protokoll- und Adressierungsschicht darüber legen(IP, etc.), dies wäre jedoch unsinnig, da alles benötigte bereits zu Verfügung gestellt wird. Die Frage ist vielmehr, wie werden die einzelnen Adressen festgelegt und wie werden die Adressen an das Modul übergeben.

Zu bedenken sind dabei auch die Punkte Benutzerfreundlichkeit, Ortsbestimmung der Module und weitere.

- Festadressierung meint, das jeder Knoten durch die aufgespielte Software eine feste Adresse bekommt, wobei ZigBee eine 2 Byte Adresse und eine 2 Byte Netzadresse hat. Es gibt zwar noch eine eigene Mac-Adresse, diese ist aber mit Hilfe der verfügbaren Library, nicht auszulesen. Die 2 Byte ZigBee-Adresse, ist zur Adressierung vollkommen ausreichend, erst recht wenn man bedenkt das man verschiedene Einzelnetze auch noch mit eigenen Netzadressen versehen kann.

- Mit Funkübertragung ist gemeint, das jedes Modul separat per Funk vom Basismodul initialisiert wird, notfalls durch das bestehende Funknetz, also einzelnes Anschalten der Module und anschließende Eingabe einer Adresse über das Basissystem.

- Die Eingabe per RS 232 würde bedeuten, jedes Modul wird an einen Rechner per Kabel angeschlossen und mit einer Adresse versehen.

- Bei der Autoattribierung handelt es sich um einen Algorithmus, bei dem sich die Netzteilnehmer selbständig adressieren und prüfen ob diese Adresse schon vergeben ist.

	Benutzer-freundlichkeit	Software-aufwand	Ortsbestimmung der Module	Erweiter-barkeit der Software	Summe
Fest-adressierung	1	4	3	3	11
Funk-übertragung	2	1	2	1	6
Eingabe per RS232	4	1	1	1	7
Autoadress-attribierung	1	5	6	2	14

Bewertung nach dem Schulnotensystem 1 = sehr gut. ....6 unzureichend. Widerspiegelung persönlicher Empfindungen und Erfahrungen.

Aus dieser Tabelle ist ersichtlich, das der einfachste Weg der Adressierung eine Adressierung per Funk ist. Das System dazu könnte folgendermaßen aussehen. Per Firmware bekommt jedes Modul vom Start an eine Defaultadresse, das Basissystem ist im Initialisierungsmodus und weist dem Knoten per Funk eine neue Adresse zu. Das Basissystem kann dieses durch eine geeignete Rückmeldung oder durch eine Abfrage bestätigt bekommen.
Diese Initialisierung ist natürlich auch durch ein bestehendes Netz möglich, da die Defaultadresse sonst nicht vergeben ist und das Netz die Informationen über die Adressänderung weiterleitet.

5.3. Datentypen

Nachdem nun der Aspekt der Informationsverbreitung und der Adressierung der Module im Netz geregelt ist, stellt sich die Frage welche Informationen / Daten übermittelt werden sollen.

Aus der Aufgabenstellung und der Art der Adressierung, ergeben sich verschiedenste Arten von Datensätzen die anfallen können und die sauber differenziert werden müssen.

Zum Einen müssen die Sensordaten übermittelt werden, dabei muss natürlich unterschieden werden von welchem Sensor sie kommen, nicht was für ein Sensortyp es ist.

Die Information, um was für einen Sensor es sich handelt, und wie die einzelnen Werte genau zu interpretieren sind, ist natürlich Sache des Basissystems. Aber ob die Daten vom Analog- bzw. Digitalsensor kommen ist sehr wohl von Interesse.

Für die Sensorenabfrage sind jedoch noch weitere Informationen notwendig, denn es ist nicht bekannt was für Sensoren auf dem Board installiert werden, also ob wirklich zwei Sensoren aufgebracht werden oder doch nur einer. So muss die Abfrage der Sensoren auch Ein- und Auszuschalten sein.

Des weiteren muss noch klar definiert werden, wie oft die Sensoren abgefragt werden sollen (pollingtime). Sensorenwerte die sich schnell und oft ändern müssen in kürzeren Intervallen abgefragt werden, als solche die sich langsam und selten ändern.

Außerdem sollte die Möglichkeit bestehen eine aktuelle Abfrage aller Sensoren zu erhalten, also unabhängig vom Zeitintervall der Abfrage, eine Momentaufnahme des zu überwachenden Systems.

Wie sich aus der Form der Adressierung ergibt, muss es auch Daten geben, die den Knoten adressieren und eine neue Adresse festlegen.

Als Letztes wäre eine Information über die Netzsituation des Knotens wünschenswert. Der Sinn dieser Information wird genauer im Kapitel 6.2.7. erläutert.

Eine gute Möglichkeit alle diese Informationen sauber zu unterscheiden und entsprechend einfach auszuwerten, ohne erst alle Daten zu lesen, ist die Einführung eines *T.O.S.-Feldes* (Type Of Service). Dadurch lässt sich das Ganze auch beliebig erweitern und notfalls auch für andere Datenübertragungen nutzen. So werden nicht nur Sensorwerte übermittelt, sondern auch z.B. Steuerdaten, can Nachrichten, etc..

Aus den vorigen Überlegungen hat sich die folgende Liste von *T.O.S. identifeiern* ergeben.

Typ Of Service 1 Byte	Weitere Daten Data array	Daten array Größe in Byte
“D” – Normale Sensordaten	“A” +Wert(8 Bit) oder “D“ +Wert (8 Bit) --- A bezeichnet den Analogsensor D bezeichnet den Digitalsensor A und D sind vom Typ Char	1+1 =2
“1“ – Adressänderung	Adresse des zu ändernden Knotens + neue Adresse --- Kein Antwortpaket!	2+2 =4
“2“ – Status des Knotens abfragen	Adresse des Knotens der seine Status ausgeben soll --- Antwortpaket hat *TOS* 3 /4	2
“3“ - Statusantwort des Knotens	Analogsensor “0“oder “1“+polling time + Digitalsensor “0“ oder “1“ pollingtime + wie viele Knoten (1 Byte) Knotenadressen 1-3 --- Es wird ausgegeben ob Analog, bzw. digital Sensor aktiv sind oder nicht, welche Abfrageintervalle gelten (pollingtime), wie viele Nachbarknoten es gibt, und deren Adressen, bzw. die ersten 3 Adressen	(1+1)+(1+1) +1+(max 6)=11
“4“ – Weitere Statusantworten des Knotens	Weitere Netzwerkknotenadressen Maximal pro Paket 6 Adressen	2 bis maximal 12
“5“ – Sensor Einstellungen ändern	Adresse des zu ändernden Knotens “A“ +(“1“ oder “0“) + pollingtime A (Char) + D(“1“ oder “0“) + Polling time D (Char)	2+(1+1+1)+ (1+1+1) =8
“6“ – aktuelle Sensorwerte ausgeben	Adresse des Knotens der seine Sensorwerte ausgeben soll	2

Es reichen also 12 Byte als reines *Datenarray* vollkommen aus.
Hier sollte noch erwähnt sein, das alle Zeichen als Char versendet werden, das heißt auch die Zeitwerte für die Abfrageintervalle der Sensoren. Es könnte zwar eine gewaltige Datenmenge eingespart werden, indem man solche Darstellungen binär codiert, insbesondere beim *T.O.S.-feld*. Aber es gibt Gründe für die Darstellung als

Char. Die Daten aus dem Netz sollen auf einer Basisstation enden, die diese Daten aufnimmt und graphisch darstellt. Die Kommunikation des Basismoduls mit einem Rechner basiert aber auf dem Standard der RS 232 Schnittstelle und diese arbeitet üblicherweise mit 8 Bit. Man könnte die Daten zwar umwandeln und auch per 8 Bit Frame übertragen, nur würde dies ein großer zusätzlicher Programmieraufwand bedeuten.
Er müsste auf jedem einzelnen Modul sein und dies führt zu einer stärkeren Belastung der CPU. Außerdem ist die Lesbarkeit eines Programms das Daten nur per Char verarbeitet um einiges einfacher, was für alle, die die originale Software abändern wollen von Vorteil ist.
Die Auswertung auf dem Rechner (mit Basisstation) wird somit auch einfacher.

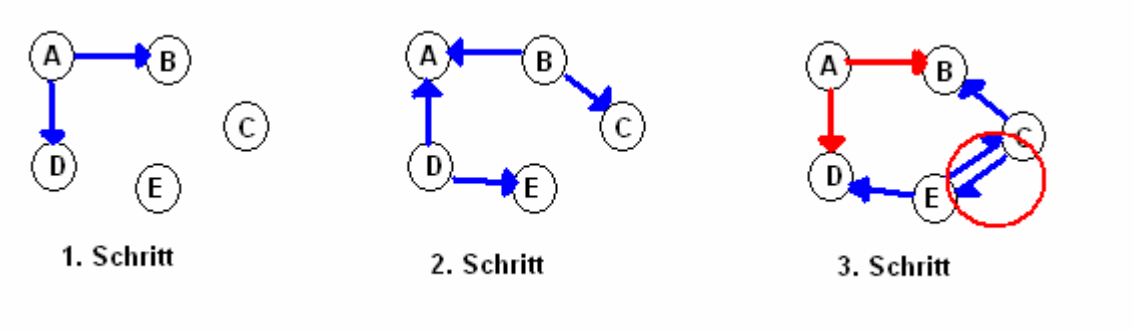
5.4. Flooding vermeiden

5.4.1. Was ist flooding

Da es sich bei diesem Netz um ein Broadcastnetz handelt und nicht um eines das durch Routing gesteuert wird, muss unbedingt darauf geachtet werden, das die Datenpakete nicht unbegrenzt im Netz kreisen. Ein geroutetes Netz hätte natürlich auch gewisse Vorteile, wie z.B. einer Verminderung des Datentransfers innerhalb des Netzes, da geringere Datenmengen und geringerer Transfer auch geringeren Energieverbrauch bedeutet.

Da für die Aufgabe nicht auf den Stromverbrauch geachtet wird, wird auch kein Routing benötigt. Das Routing würde die Software unnötig komplizierter machen und entsprechend mehr Leistung vom Mikrokontroller verlangen.

Das Problem Flooding und Kollision stellt die folgende Graphik dar:



1. Schritt

Knoten A sendet Daten an alle erreichbaren Knoten.

2. Schritt

Die Knoten A und D haben die Nachricht erhalten und leiten diese an alle erreichbaren Knoten weiter. Knoten A erhält sowohl von B und D ein Datenpaket, falls diese Datenpakete gleichzeitig ankommen, kann es zu einer Zerstörung der Daten kommen (was nicht schlimm wäre, da das in diesem Fall rücklaufende Daten sind, also Daten die Knoten A selber gesendet hat). Andernfalls dürften natürlich keine Daten zerstört werden, aber dafür sorgt schon das ZigBee-System (Dies ist nur ein Versuch der Kollisionsvermeidung, es wird durch Acknowledgepakete unterstützt, für die Praxis ist es jedoch vollkommen ausreichend.). Durch minimalen Zeitversatz kann es aber auch sein, das der Knoten zwei Datensätze in sehr kurzer Abfolge erhält. Da der Empfang der Daten interrupt gesteuert wird, müssen diese gepuffert werden, um sie überhaupt verarbeiten zu können.

3. Schritt

Hier treten die ersten Probleme auf. Knoten E und C haben die Nachricht erhalten und wollen diese weiterleiten (roter Kreis). Im Falle das dies zur selben Zeit geschieht, tritt normalerweise eine sogenannte Kollision auf und die Daten werden zerstört. Eine Kollisionsüberprüfung ist jedoch im ZigBee-System schon enthalten. Das zweite Problem im 3. Schritt ist jedoch größer. Knoten A bekommt Daten und versucht diese automatisch an die erreichbaren Nachbarknoten weiterzuleiten. Das bedeutet, dass das ursprüngliche Datenpaket ein zweites Mal losgeschickt wird, was zur Folge hätte, das diese Information ewig im Netz kreist und das gesamte Netzwerk flutet (Flooding).

5.4.2. Fazit

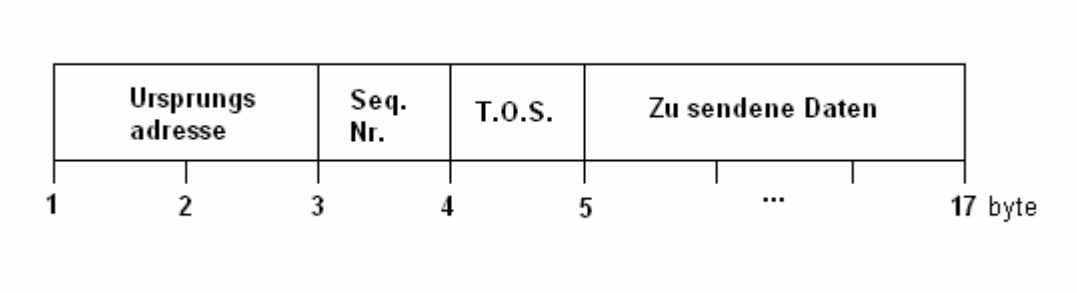
Aus dieser Überlegung ergeben sich zwei Probleme, die in der Software und im Protokoll behandelt werden müssen:

1. schnelle Paketabfolge, es darf möglichst kein Paket verloren gehen, wie dies am besten funktioniert und wo das Problem genau liegt, wird im Kapitel 6.2.1. behandelt (*Nachrichtenqueue*).

2. Differenzierung ob das Paket schon einmal gesendet wurde oder nicht, eine sehr simple Lösung wird im Kapitel 6.2.4. (*Messagebuffer*) beschrieben

Es wird also zu jedem einzelnen Datenpaket die ursprüngliche Absenderadresse benötigt, da sonst durch das *Repeating* eventuell nicht klar ist, von welchem Knoten die Daten kommen.

Um ein *Flooding* zu verhindern sollte jeder Knoten wissen, welche Nachricht er schon weitergeleitet hat bzw. welche Daten er geschickt hat, um diese nur einmal weiterzuschicken. Dies ist am Einfachsten zu lösen, indem jeder Knoten eine eigene Sequenznummer pflegt, sodass alle anderen Knoten auf Grund der Kombination aus Sequenznummer und ursprünglicher Absenderadresse ermitteln können, ob sie diese Nachricht schon erhalten haben und kennen, oder nicht.
Also sieht der Frame für das Datenprotokoll des Sensornetzwerkes folgendermaßen aus:



Bei diesem Frame ist unbedingt zu beachten, das in den 17 Byte eine abschließende Nullterminierung enthalten sein muss. Das bedeutet das pro Datenpaket maximal 12 Byte übertragen werden können. Dies ist ausreichend, da es selten Fälle gibt, wo große Datenmengen sinnvoll sind. Selbst wenn große Datenmengen notwendig werden, so kann man diese fragmentieren und beim Empfänger wieder zusammensetzen.

Der Vorteil der Nullterminierung ist, das die Daten mit einfachen Stringbefehlen bearbeitet werden können. Dieser Frame ist natürlich komplett im Payload des ZigBee Protokolls untergebracht.(siehe Bild 4 Kapitel 4.2.)

5.5. Ausblicke und weitere Möglichkeiten

Derzeit ist das Protokoll nur für die Übertragung der Sensordaten und der einfachen Initialisierung der Module gedacht. Es ließe sich aber ohne großen Aufwand auf vielerlei andere Anwendungen auf ZigBee-Basis erweitern, z.B. für mobile Roboter und deren Übertragung der Nachrichten. Das System, das in diesem Kapitel beschrieben und deren programmtechnische Umsetzung im Kapitel 6 näher erläutert ist, ist prinzipiell für alle peer to peer Kommunikationen mit ZigBee nutzbar. Es müssen nur neue *T.O.S.-felder* definiert werden.
Ob es sich nun um die Kommunikation zweier Rechner handelt oder der Übertragung von can Nachrichten, die Art der Daten ist nicht relevant.

Es gibt aber auch Vieles, was an diesem System als solches noch erweitert werden könnte. Als Beispiel sei einmal die Adressierung angeführt. Man müsste sich bei der Adressierung noch gedanklich damit auseinander setzen, was passieren würde, wenn alle Knoten einem plötzlichen Stromausfall ausgesetzt sind, würden sie nach dem Neustart ihre Defaultadresse benutzen und neu initialisiert werden oder schreibt man die Adressen in einen Festwertspeicher, beides hätte Vor- und Nachteile. Würde man die Adresse in den Festwertspeicher schreiben, so müsste man sich ein weiteres *T.O.S.-feld* definieren, auf das jeder Knoten mit Ausgabe seiner Adresse reagiert.

Eine weitere sinnreiche Erweiterung wäre die dynamische Vergabe von Netzwerkadressen, also auch eine Weiterleitung von Daten aus anderen Netzen oder das Wechseln eines Knotens von einem in ein anderes Netz. Derzeit ist noch keine Änderung der Netzadresse implementiert.

6. Softwareentwicklung auf den Knoten und dem Basissystem

Dieses Kapitel stellt die Gedankengänge und aufkommenden Fragen bei der Entwicklung der Software dar, darunter auch einige Lösungsansätze. Der gesamte Source-Code befindet sich im Anhang der Arbeit. Die grundsätzlichen Funktionsweisen und Prinzipien sind jedoch gut aus den Darstellungen und den einzelnen Beschreibungen ersichtlich.

Folgende Anforderung an die Software habe ich mir gestellt:

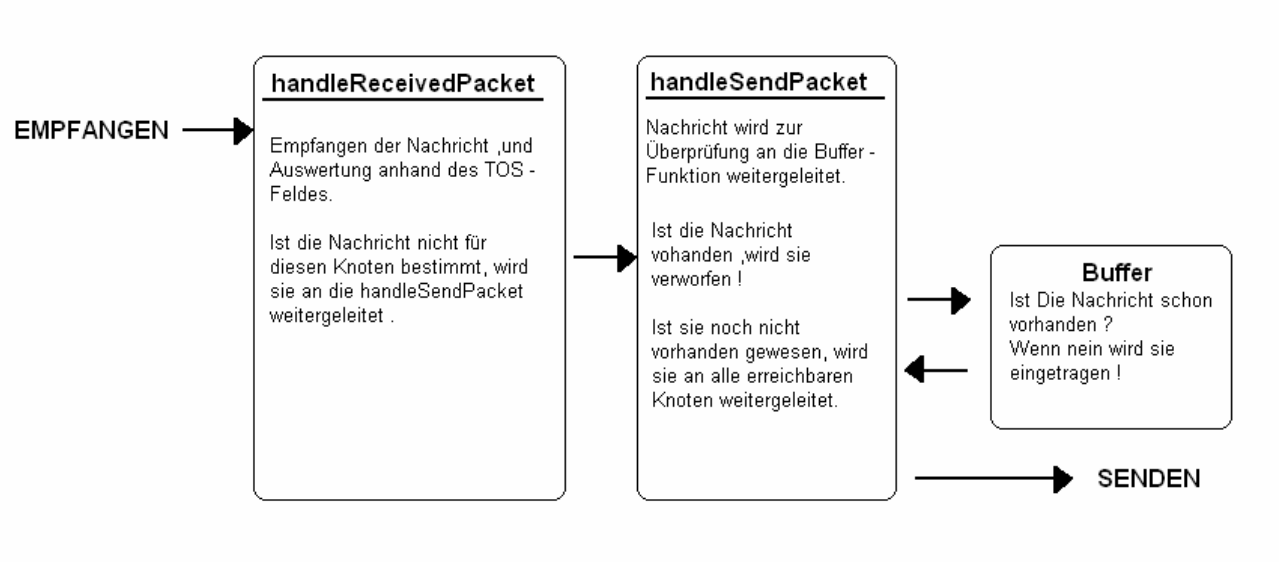
Auf den Sensorknoten und den Servermodulen wird eine identische Software installiert.

Alle Funktionalitäten sollen sowohl per Funk als auch per RS 232 einstellbar sein.
Aktivierung / Deaktivierung einzelner Sensoren sollte möglich sein.
Die Abfrageintervalle der einzelnen Sensoren müssen individuell einstellbar sein.
Es muss die Möglichkeit einer Adressänderung bestehen.
Die Statusabfrage muss jederzeit gewährleistet sein.

6.1. Programmstruktur

Als Basis des Systems ist natürlich die Erhaltung der Netzstruktur bzw. der Aufbau des Netzes festzulegen. Das heißt das Weiterleiten der Daten und damit die Integration in das Netz.

Der erste Schritt ist der Empfang einer Nachricht. Beim Empfang wird durch den Knoten überprüft, ob er diese Nachricht schon einmal erhalten hat und er sie kennt. Ist dies nicht der Fall, wird die Nachricht in den Messagebuffer eingetragen und dann weitergesendet.



Dieses Bild stellt eine einfache Funktionsweise zum Erhalt des Netzes und zur Verbreitung der Daten im Netz dar (Weiterleitung). Zusätzlich ist durch die Bufferfunktion eine Floodingvermeidung integriert.

Da der Empfang von Daten per Funk interrupt-gesteuert ist, muss noch eine Funktion vorgeschaltet werden, welche die Daten puffert (*Queue*). Die genaue Funktionsweise der einzelnen Teilsegmente sind weiter unten in diesem Kapitel beschrieben.

6.2. Darstellung verschiedener Funktionalitäten

Hier werden nun einige grundlegende Funktionen der Software etwas genauer beleuchtet, um das Gesamtsystem verständlicher zu machen.

6.2.1. Nachrichtenqueue

Da das von mir geschriebene Programm keinerlei definierten Scheduler enthält und ich die Headerfunktionalitäten des CC2420 zur Erstellung der Software benutze, gibt es noch ein weiteres Problem.

Die Daten werden vom ZigBee empfangen und dadurch wird beim Atmel ein Interrupt ausgelöst. Die entsprechende ISR ist zwar vorhanden, übergibt die Daten allerdings in ein C-struct an die eigene Software.

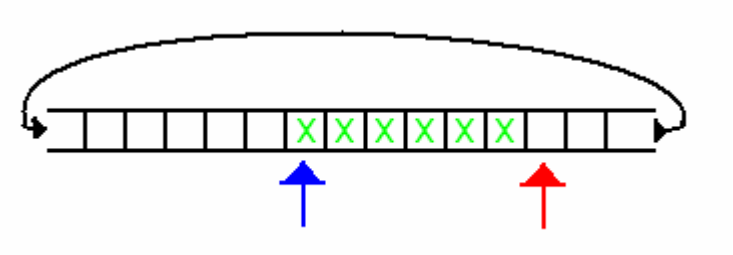
Daher muss die Verarbeitung der Daten extrem schnell verlaufen, ob es sich nur um eine einfache Weiterleitung handelt oder um Änderungen der Einstellungen des Knotens. In allen Fällen sind jedoch einige Routinen zu durchlaufen. Es kann also sehr schnell passieren, das zwei schnell aufeinanderfolgende Nachrichten zwar vom ZigBee-Modul erkannt und weitergereicht werden, diese jedoch nicht vom Atmel verarbeitet werden können. Also ist auch hier ein entsprechender Buffer notwendig. Der Ablauf sieht nun folgendermaßen aus:

Eine Nachricht trifft beim ZigBee-Modul ein, ein IRQ wird ausgelöst und die entsprechende ISR ruft das folgende Programmsegment auf.

```
BASIC_RF_RX_INFO* basicRfReceivePacket(BASIC_RF_RX_INFO *pRRI)◁
    basicRfReceiveOff();   ← 1.
    pRRISave = pRRI;
    queue(pRRI);           ← 2.
    basicRfReceiveOn();    ← 3.
```

1. im Programm wird der Empfang von Nachrichten als Erstes ausgeschaltet
2. nun können die Daten im Ringpuffer gespeichert werden
3. der Nachrichtenempfang wird wieder aktiviert und das normale Programm läuft weiter

Die Frage ist, wie ist am sinnvollsten die Speicherung der Nachrichten zu bewerkstelligen.
Und wie sind diese Daten auszulesen.
Es muss also zwei unabhängige Zugriffszeiger geben, einmal eine zum reinschreiben und ein zweiter Zeiger zum auslesen.



Blau: lesender Zeiger Rot: schreibender Zeiger

Das Einschreiben in den Speicher ist nicht sehr aufwendig. Es werden aber nicht nur die Daten aus dem Payload gespeichert, sondern auch die Adresse des Knotens der gerade diese Nachricht gesendet hat. Diese Daten lassen sich aus dem ZigBee-Header entnehmen. Wofür diese zusätzlichen Daten benötigt werden, wird im Kapitel 6.2.7. genauer beschrieben.

Da die Daten in den Puffer geschrieben werden, durch den Programmsprung aus einer ISR, stellt sich natürlich sofort die Frage wann und wie können die Daten wieder ausgelesen werden.

Die Lösung sieht folgendermaßen aus.
Das *Main*programm läuft in einer Endlosschleife, bei jedem Schleifendurchlauf wird einmal die Auswertungsfunktion aufgerufen, die in der *Nachrichtenqueue* schaut, ob Daten da sind, diese dann auswertet oder im Falle das die *Queue* leer ist sofort zurückspringt.

Dasselbe Verfahren wird auch benutzt zwischen den Abfragen der Sensoren, Genaueres dazu im Bereich Sensorabfrage.

6.2.2. handleReceivedPacket

Dies ist sozusagen die Hauptfunktion der ganzen Softwareentwicklung.

Diese Funktion verarbeitet alle eingehenden Nachrichten und wird zyklisch von der *Main*funktion (Kapitel 6.2.6) aufgerufen.
Die Hauptaufgabe besteht darin, die eingehenden Pakete zu analysieren und anhand des *T.O.S.-feldes* verschiedene Funktionalitäten auszuführen.

Es wird aus jedem Datenpaket die Absenderadresse in einer extra Variablen gespeichert, um diese beim Weitersenden, der *handleReceivedPacket* Funktion zu übergeben. Dasselbe passiert mit der Sequenznummer und dem *T.O.S.-feld*.

Anhand des *T.O.S.-feldes* wird nun weiter verzweigt, im Falle von *T.O.S*. D 3,4 werden die Daten einfach versucht weiterzuleiten, weil diese nur Informationen für das Basissystem beinhalten. (siehe Tabelle in Kapitel 5.3.)

Bei den anderen Paketen muss erst überprüft werden, ob die ersten 2 Byte im *Datenarray* der eigenen Adresse entsprechen, wenn dies nicht der Fall ist, kann das Datenpaket umkopiert werden und weiter gesendet werden.

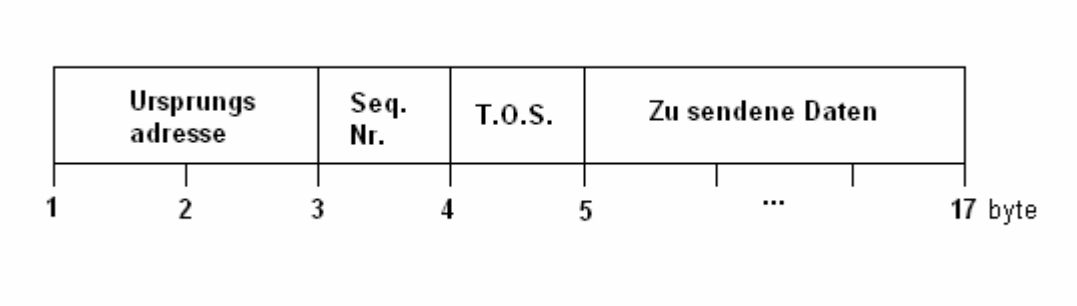
Entsprechen die ersten beiden Byte des *Array* jedoch der eigenen Adresse, so ist das Datenpaket für diesen Knoten bestimmt und wird ausgewertet. Die Auswertung ist prinzipiell nur eine Wertänderung von Variablen, die das System steuern.

6.2.3. handleSendPacket

Dieses Programm fasst die zu sendenden Datenpakete zusammen und ruft die Funktion *Buffer* auf. Genaueres zum *Buffer* findet sich im nächsten Abschnitt 6.2.4..

Dieser Funktion werden die Headerinformationen übergeben, also Adresse, Sequenznummer und T.O.S.-feld. Die Pflege der Sequenznummer des Knotens, wird von dieser Funktion übernommen. Zusätzlich natürlich noch die zu übertragenen Daten.

Hier noch einmal der Aufbau meines Framesets:



Nach der Überprüfung durch die Funktion *Buffer*, wird falls diese Nachricht dem Knoten noch nicht bekannt ist der Datenframe an das ZigBee-System übergeben und somit versendet.

Ist dem Knoten die Nachricht schon bekannt, so beendet sich die Funktion ohne das etwas gesendet wird.

6.2.4. Buffer

Diese Funktion ist notwendig, um ein *Flooding* des Netzes zu vermeiden. Im Grundlegenden besteht sie aus einem *Array*, in dem die bereits erhaltenen Nachrichten gespeichert werden.

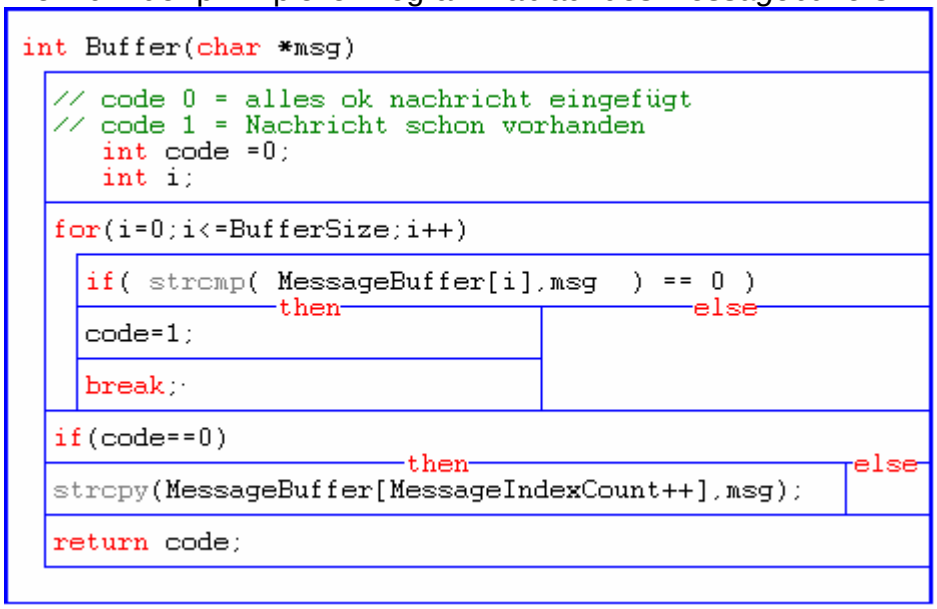
Das einfache Abspeichern aller erhaltenen Nachrichten ist aus folgenden Gründen nicht sinnvoll.

- Speicherbedarf
  der Atmega 128 hat ursprünglich nur einen sehr geringen RAM-Speicher (4k Byte), mit dessen Verwendung man sorgsam haushalten muss, normalerweise würde ein zusätzlicher RAM-Speicher auf dem Knotenmodul sinnreich sein, aber unter der Prämisse der Kostenreduktion ist dieser nicht vorhanden

- Laufzeiten
  wie im Kapitel „Was ist *Flooding*“ schon erwähnt wurde, muss das gesamte *Messagearray* durchsucht werden, ob diese Nachricht schon vorhanden ist, um ein unnötiges Weiterleiten zu verhindern

Das Verfahren zur Unterscheidung, ob eine Nachricht schon gesendet wurde oder nicht, ist kurz im Kapitel *Flooding* schon erwähnt. Wenn jeder Knoten eine eigene Sequenznummer vergibt ist aus der Kombination der Absenderadresse und der Sequenznummer ersichtlich ob es eine alte oder eine neue Nachricht ist. Das bedeutet, das nicht die ganze Nachricht gespeichert werden muss, sondern nur die ersten drei Byte des ZigBee Payloads. Denn dort sind die wichtigsten Headerinformationen meines Datenframe.

Hier kurz der prinzipielle Programmablauf des *Messagebuffers*.



Bei dieser Struktur werden alle Nachrichtenheader, im *Array* gespeichert.

Man könnte zwar alle Nachrichtenheader speichern, da das *Array* als Ringpuffer angelegt ist, aber wenn ein Knoten sehr schnell Daten erfasst und sendet würde dies dazu führen, das Nachrichten von selten sendenden Knoten überschrieben werden.

Ein generelles Verfahren wäre z.B. alle Nachrichten die älter als eine gewisse Zeit sind zu löschen. Dies schließt sich jedoch aus, da jeder Knoten sonst eine genaue Zeit haben müsste, also müssten alle Knoten synchronisiert werden. Ein Gedanke ist, einfach für jede Quelladresse nur ein Speicherplatz im *Array* zu benutzen.

Da es sich bei dem ganzen Netz allerdings um ein Broadcastnetz handelt sind die genauen Wege der Daten durch das Netz nicht bestimmbar, also auch nicht die Laufzeiten. Daraus folgt, das die Nachrichten nicht unbedingt in der selben Folge ankommen. Es muss also Sorge getragen werden, das eine Nachricht mit niedrigerer Sequenznummer, die nach einer Nachricht mit hoher Sequenznummer ankommt, nicht automatisch verworfen wird, denn es könnte sein, das diese Nachricht noch nicht weitergeleitet wurde. Es ist sinnvoll im Buffer nicht nur einen Speicherplatz zu haben, sondern mindestens 2 pro Knoten im Netz.

Dies ist durch eine einfache Abfrage in der for-Schleife, die das ganze Array durchläuft zu realisieren. Es muss nur geprüft werden, ob es schon einen Eintrag gibt der die selbe Adresse hat und eine Sequenznummer, die um 2 differiert, wenn ja werden die Paketdaten des aktuellen Paketes an dieser Stelle eingetragen und die Funktion verlassen. Da sich durch das Szenario eine recht geringe Anzahl von Knoten ergibt, werden nur im *Array* Speicherplätze reserviert. Da es als Ringpuffer arbeitet, funktioniert auch die Pufferung der Daten, dies sollte bis zu 128 Knoten gewährleistet sein.

6.2.5. Sensorenabfrage

Die Sensoren sollten nicht kontinuierlich abgefragt werden, denn dies macht in den meisten Fällen keinen Sinn. Gerade im Bereich der Überwachung kann man sich dies schnell veranschaulichen. Als Beispiel eine Hausüberwachung, in diesem Falle ist eine kontinuierliche Überwachung der Temperatur (jede Millisekunde) nicht notwendig. Bei einer Anlagenüberwachung und deren Steuerung ist ein möglichst kurzes Abfrageintervall jedoch wünschenswert. Daher wird für jeden Sensor ein eigenes Abfrageintervall bestimmt.

Für welche mögliche Größe man sich beim Intervall entscheidet ist sicher eine Frage die sehr abhängig von der individuellen Aufgabe ist. In meiner Software sind Intervalle von 1 bis 9 Sekunden möglich, was wohl für die meisten Monitoring Aufgaben vollkommen ausreichend ist. Dies ist aber auch, ohne großen Aufwand, auf Minuten zu ändern, auch zweistellige Zeitintervalle sind möglich und bedürfen nur kleiner Änderungen im Programm.

Ein weiterer Punkt ist natürlich, welche Sensoren sind angeschlossen, also nicht was für ein Sensor genau, sondern ist es ein Analogsensor, ein Digitalsensor oder vielleicht sogar beides. Die Abfrage der Sensoren muss also ein- und ausschaltbar sein.

Welcher Sensor aktiviert ist und welche Abfrageintervalle diese Sensoren haben, ist auch in globalen Variablen gespeichert. Der Grund dafür liegt darin, das diese Informationen an verschiedenen Stellen im Programm benötigt werden, und nicht direkt übergeben werden können ohne Dutzende unnötige Programmaufrufe zu starten.

Eine strukturelle Darstellung der Sensorabfrageprogramme ist nicht sinnvoll, weil diese Abfragen prinzipiell nur sequenziell sind, das heißt es wird ein Register ausgelesen und dies mit den Headerdaten an die Send Funktion übergeben. Die Zeitsteuerung der Sensorabfrage geschieht in der *Main*funktion.

6.2.6. Main

Die *Main* steuert den gesamten Ablauf und das Zusammenspiel der Komponenten.

So werden als Erstes sämtliche Initialisierungen durchlaufen, z.B. das Anlegen des *Messagebuffers*, die Aktivierung des ZigBee-Moduls, die Initialisierung der RS 232 Schnittstelle etc..

Das es, wie schon erwähnt, keine Scheduler gibt muss es also auch hier einen gewissen Taskswitch geben bzw. einen zeitgesteuerten Aufruf von Programmen.

Eine einfache “do--- while (true)“ Schleife reicht hier leider nicht aus, da die Sensoren in gewissen Zeitabständen abgefragt werden sollen. Diese Zeitabstände sind auch noch unterschiedlich. Es wird also so etwas wie eine Timerfunktion benötigt, am Einfachsten ist dies durch eine Schleife zu realisieren, die einen “nop“ – Assemblerbefehl ausführt, denn die Taktrate ist genau bekannt und der “nop“- Befehl wird in einem Prozessorzyklus abgearbeitet, also gibt es zur Zeitsteuerung eine Funktion die ca. 1 Millisekunde dauert.

Da jedoch immer noch die Frage besteht, wie sind die eingegangenen Nachrichten aus der *Queue* auszulesen und dann entsprechend auszuwerten, bietet sich hier die Möglichkeit beim Durchlauf der Schleife, einfach einen Funktionsaufruf für das *handleReceivedPacket* einzubauen. Durch das ständige Programmswitchen (Funktionsaufrufe) läuft die Schleife nicht mehr in 1 Millisekunde durch, sondern in ca. 2 Millisekunden.

Eine genaue Zeitberechnung des Schleifendurchlaufes macht keinen Sinn, denn wenn Datenpakete eingehen verursacht die *handleReceivedPacket*-Funktion Prozessorlast.
Das System als solches ist zwar ausreichend und funktioniert in der Praxis mit gutem Ergebnis, es sind jedoch besondere Szenarien denkbar, wo es zu Problemen kommen kann. Beispielsweise es kommen in sehr kurzer Zeit viele Datenpakete an, sodass *handleReceivedPacket* sehr oft durchgearbeitet werden muss. Also würde sich die Gesamtdurchlaufzeit der Zeitfunktion extrem verlängern.

So ist es am Besten mit einem Durchschnittswert von 2,5 Millisekunden zu rechnen. Eine ganz genaue Zeitberechnung wäre sicher denkbar, mit einem Timer, entweder mit dem aus dem Atmega 128 oder einem externen Zeitsignal, zusätzlich müsste aber dann auch ein Scheduler geschrieben werden etc..

Dieser Aufwand ist aber unnötig für diese Aufgabe, somit habe ich eine einfache Variante gewählt, die leichter zu verstehen und zu gebrauchen ist.

Bei jedem Durchlauf der *Main*schleife (mit Timer) werden für die Sensoren entsprechende Counter hochgezählt und bei einer Übereinstimmung mit den Abfragezeiten für den Sensor, wird die entsprechende Sensorabfrage gestartet und das Datenpaket losgeschickt.

Zusätzlich zu dieser absolut notwendigen Funktionalität, enthält die *Main* noch eine Switchcaseanweisung, die durch Eingaben über die RS 232 Schnittstelle gesteuert wird. Der Grund dafür ist, das die Software auf dem Basisknoten und die auf dem Sensorknoten identisch sind, also jeder Sensorknoten auch als aktive Basisstation genutzt werden kann ohne eine Softwareänderung.

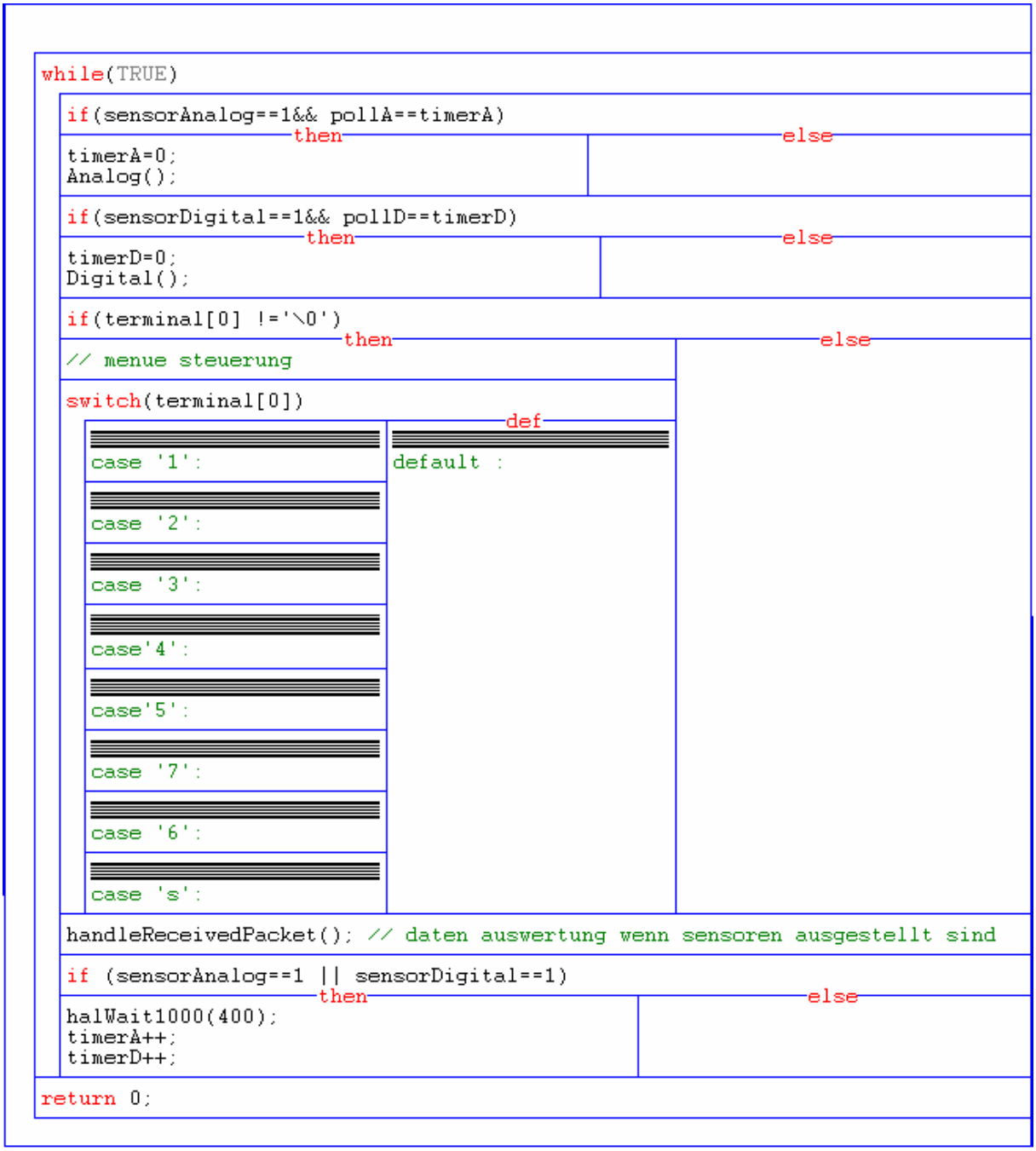
Die gesamte Steuerung des Netzes und des Knotens selbst, lässt sich über die RS 232 Schnittstelle steuern, außerdem kann man die Ausgabe sämtlicher Pakete die im Netz unterwegs sind ein- und ausschalten.

Das man die Ausgabe auf jedem Knoten aktivieren kann, hat bei einigen Anwendungsfällen Vorteile, falls man z.B. die Ausgabe der Daten nicht nur an einer Basisstation benötigt ,sondern an anderer Stelle, die weiter entfernt ist auch gerne wissen möchte was im Netz passiert.

Mögliche Eingaben über RS 232

Eingabe	Funktion	Weitere Eingabe	Beispiel	Ergebnis
1	Eigene Adresse ändern	2 Char neue Adresse	“1AA“	Die Adresse des Knotens wird auf AA geändert
2	Adresse eines Netzknotens ändern	2 Char alte Adresse + 2 Char neue Adresse	“2XXAA“	Die Adresse des Knotens XX wird auf AA geändert
3	Sensoreneinstellungen eines anderen Knotens ändern	2 Char Adresse + 1Char Analog Sensor(1=an,0=aus)+Abfrage Intervall (1- 9) Sek. + 1Char Digital Sensor(1=an,0=aus)+abfrage Intervall (1- 9) Sek	“3XX1900“	Beim Knoten XX wird der Analogsensor aktiviert, und nun alle 9 Sekunden abgefragt, der Digitalsensor wird deaktiviert
4	Status eines Knotens Abrufen	2 Charadresse	„4XX“	Der Knoten sendet alle Informationen, also Sensoreinstellungen, und benachbarte Knoten
5	Sensoren eines Knotens abfragen	2 Charadresse	“5XX“	Der Knoten XX wird nun den aktuellen Sensorwert der bei ihm aktievierten Sensoren ausgeben
6	Konsolenausgabe aktivieren	-----	------	Nun werden alle empfangen Datenpakete auf der RS232 Schnittstelle ausgegeben ,eine nochmalige Eingabe von “6“ schaltet die Ausgabe wieder aus
7	Eigene Sensoreinstellungen ändern	1Char Analogsensor(1=an,0=aus)+ Abfrageintervall (1- 9) Sek.+ 1Char Digitalsensor(1=an,0=aus)+ Abfrageintervall (1- 9) Sek.	“70019“	Beim aktuellen Knoten wird der Analogsensor Deaktiviert und der Analogsensor aktiviert, und nun alle 9 Sekunden abgefragt
8	selbst den Status ausgeben	---	---	Der Knoten sendet alle Informationen, also Sensoreinstellungen, und benachbarte Knoten, auf der RS232 Schnittstelle

Hier ein kurzes reduziertes Struktogramm der *Main*funktion



Hieraus wird das Zusammenspiel der lokalen Eingabe, der Zeitsteuerung und der Zeitsteuerung für die Sensoren ersichtlich. Zu bemerken sei, das *handleReceivedPacket* bei jedem Durchlauf von *halWait1000* einmal aufgerufen wird. Hier in der *Main* wird das Programm normal 400 mal pro *Main*durchlauf aufgerufen im Falle das mindestens 1 Sensor aktiviert ist. Wenn keine Sensoren aktiviert sind, wird die *handleReceivedPacket*-funktion bei jedem Durchlauf aufgerufen. Dieser Schleifendurchlauf ist aber extrem schnell.

Die Struktur der einzelnen Switchcaseanweisung ist prinzipiell identisch. Hier nur ganz kurz das Prinzip dargestellt:

Es wird Zeichen für Zeichen vom Terminal gelesen, diese werden entweder sofort in Variablen geschrieben, bei Änderungen am lokalen Knoten oder sie werden direkt in ein Array geschrieben. Dies wird vor dem Verlassen der Caseanweisung gesendet, deswegen entspricht die Eingabesyntax der Framesyntax für das Zusenden des Datenarray.

Eine wichtige Einschränkung sieht man auch sofort. Während man sich im Switchcase befindet ,wird kein Timer gezählt, also auch keine Sensorenabfrage gestartet, daher sollte man sich überlegen, ob der Basisknoten auch mit Sensoren ausgestattet werden sollte. Eine Korrektheit der Eingaben wird nicht überprüft, da dies unnötige CPU-Last verursachen würde. Die graphische Oberfläche (Kapitel 6.1.3) sollte selbständig dafür Sorge tragen, das nur korrekte Steuerdaten an das Modul gesendet werden.

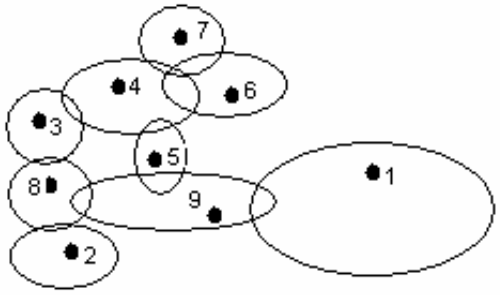
6.2.7. Netzarray

Netzarray ist ein Feature was sich dazu nutzen lässt die Struktur des Netzes zu ermitteln. Prinzipiell handelt es sich dabei um eine lokale Routingtabelle, in der einfach alle Adressen der direkten Nachbarknoten eingetragen werden. So dass das der Knoten erkennt, welche „Nachbarn“ sich in seiner Funkreichweite befinden.

Die Nachbarknoten werden jedoch nicht statisch eingetragen, sondern mit einer TTL (time to life) versehen. Da jeder Nachbarknoten aufgrund der Weiterleitung auf jedes Datenpaket reagiert und eine Antwort schickt, wird für jeden Knoten ein *Array*-Platz reserviert und ein Counter. Dieser Counter wird beim Eintreffen eines Datenpakets auf Null zurück gesetzt. Alle anderen Counter werden erhöht.

Falls ein Counter einen bestimmten Schwellwert erreicht hat, wird dieser aus dem Array gelöscht, weil man davon ausgehen kann, das dieser Knoten nicht mehr aktiv ist. Die Adressdaten (Adresse des letzten Knotens) werden direkt aus dem ZigBee-Header gelesen.

Der Nutzen dieser Information liegt darin, dass man die Struktur des Netzwerkes ablesen kann, wenn man weiß welcher Knoten mit welchen anderen Knoten in Kontakt steht.

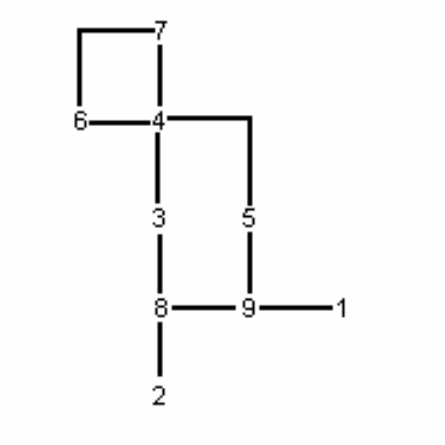


Beispiel Netz mit 9 Knoten und verschiedenen Funkreichweiten

Durch die Informationen von jedem Knoten, mit welchem Nachbarknoten er in einer Funkreichweite liegt, lässt sich eine Tabelle bzw. Matrix aufbauen. Bei diesem Beispiel sieht das folgendermaßen aus

Knoten	**Nachbarknoten**								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1									**X**
2								**X**	
3				**X**				**X**	
4			**X**		**X**	**X**	**X**		
5				**X**					**X**
6				**X**			**X**		
7				**X**		**X**			
8		**X**	**X**						**X**
9	**X**				**X**			**X**	

Daraus ergibt sich für dieses kleine Beispielnetz die folgende Struktur:



Im Falle eines statischen Netzwerkes ist der Ort/Platz eines jeden Knotens bekannt. Jedoch nicht seine Funkreichweite, da die abhängig von verschiedenen Störfaktoren ist (Wände, Funkstörungen, etc.).
Normalerweise ist die genaue Netzstruktur auch nicht relevant, das ist schließlich die Besonderheit von spontanen Funknetzwerken. Bei einem Störfall sieht es jedoch etwas anders aus.

Beispiel:
Nimmt man an, das von Knoten 4;6 und 7 keine Informationen mehr empfangen werden und diese Knoten auch nicht selbst erreichbar sind. Ohne das man die Netzstruktur kennt müsste man alle drei Knoten überprüfen und diese könnten theoretisch aufgrund verschiedener Funkreichweiten im Netz an ganz unterschiedlicher Stelle sein.
Kennt man jedoch die Struktur des Netzes, liegt in diesem Falle der Verdacht sehr nahe, dass nur Knoten 4 defekt ist und gegebenenfalls ausgetauscht werden muss. Dies ist aus der Struktur ersichtlich. Erstens ist die Wahrscheinlichkeit das nur ein Knoten ausfällt größer, als das gleichzeitig drei ausfallen und Knoten 4 ist für Knoten 6 und 7 die einzige Verbindung zum Rest des Netzes.

7. Weitere Ausblicke

Dieses Kapitel soll die möglichen Erweiterungen der Software, sowie weitere Gedanken zu dieser Arbeit widerspiegeln.

7.1. Anwendung

Weitere Anwendungsgebiete sind neben dem dargestellten Szenario denkbar. Selbst wenn man von der derzeitigen Konzeption der Module ausgeht. Ob es nun um Monitoring im Bereich der Gebäudeüberwachung geht, wie zum Beispiel als Alarmanlage, Umweltkontrolle oder auch Monitoring zur Systemüberwachung von Produktionsanlagen. Steuerung von Umweltsystemen etc..

Grundsätzlich müsste diese Software, die auf und für das CC2420DBK Board von Chipcon entwickelt wurde, noch auf das CFMW1 Modul konvertiert werden. Die Konvertierung besteht hauptsächlich in der Abfrage der Sensorenwerte. Da die Fertigstellung der neu entwickelten Module erst nach Abschluss dieser Arbeit zu erwarten ist, konnte die Software nicht darauf portiert werden.

Bei der Portierung ist aber mit keinen größeren Problemen zu rechnen, eventuell sind aber neue Funktionalitäten denkbar.

7.2. Sicherheit

Netzsicherheit
Dies ist natürlich heutzutage ein ganz beachtenswerter Punkt. Wie kann man sicherstellen, das niemand Unbefugtes die Daten nutzt oder schlimmer noch wie kann man sicherstellen, das diese Daten nicht manipuliert werden. Dafür bietet das ZigBee eine recht einfache Möglichkeit, nämlich die Vergabe von Netzschlüsseln. So werden jegliche Daten die übertragen werden verschlüsselt.

Ein Vorteil dieser verwendeten ZigBee Funktechnologie ist, dass diese Verschlüsselung auf Hardware-Ebene ausgeführt wird, sodass die Entschlüsselung nicht durch den Atmega 128 Mikrokontroller gemacht werden muss.

Integrierbar heißt in diesem Zusammenhang, falls entsprechende Libraryfunktionalitäten für den CC2420 zur Verfügung stehen, könnte es leicht ohne großen Programmieraufwand bei der Initialisierung des ZigBee-Moduls in der Software mit eingebunden werden. Derzeit stehen die Verschlüsselungsfunktionen des ZigBee-Moduls noch nicht durch Libraries zur Verfügung.

Aber Netzsicherheit kann noch mehr bedeuten, denn bei der Vernetzung und Sammlung von unterschiedlichen Daten, gibt es immer Querinformationen die entstehen. Dies ist nicht ein besonderes Problem bei dieser Diplomarbeit, sondern vielmehr ein generelles der Datensammlung. Wenn man sich an das Beispiel der Unfallwarnung aus Kapitel 2.3. erinnert, ist es schön zu wissen, das die Fahrzeuge gebremst werden. Als Nebeneffekt eines solchen Netzwerkes, weiß man zu fast jeder Zeit, wo sich ein anderes Auto befindet. Also wenn man die Adresse eines Autos kennt, kann man dieses durch das ganze Netz verfolgen.

Dieser Gedanke klingt erst einmal gut, hinsichtlich von Autodiebstählen etc.. Aber wie ist es mit der Sicherheit, z.B. von Politikern oder hochrangigen Persönlichkeiten. Deren Position wäre prinzipiell aus dem Netz erfragbar.

Wenn man nun diese Sensoren im Auto dazu benutzt eine intelligente Verkehrsführung bzw. bedarfsgerechte Ampelschaltungen zu konzipieren, müsste man die Geschwindigkeit der Fahrzeuge ermitteln. Hier würden sich wahrscheinlich die Großzahl der Autofahrer in Ihrer Privatsphäre und Persönlichkeitsrecht beschnitten fühlen, denn wer will schon gern, das dass Verkehrsamt und Polizei die Geschwindigkeit eines jeden Fahrzeugs kontrolliert und präzise abrufen kann.

Es ist also nicht nur interessant wie die Daten verschlüsselt werden, sondern auch wer und wie Zugang zu den Daten bekommt.

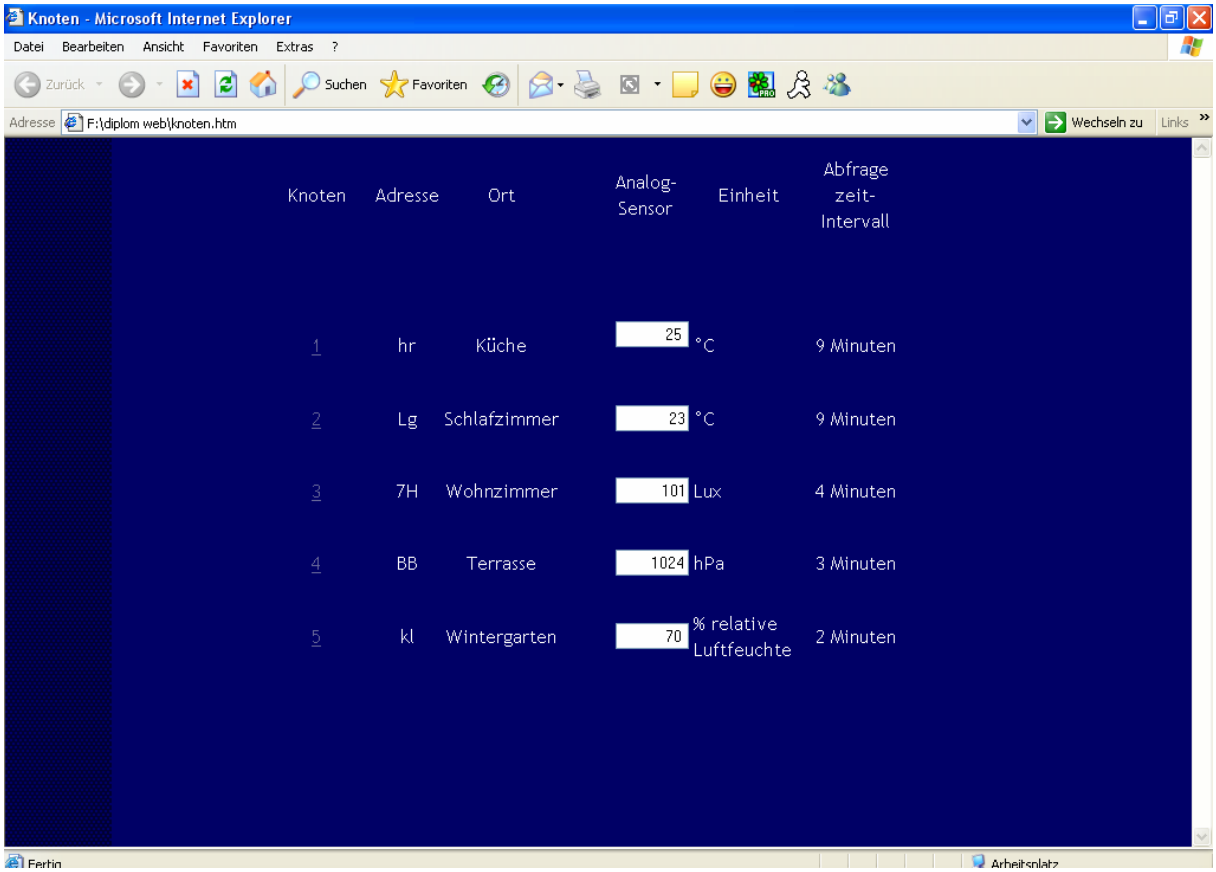
7.3. Oberfläche

Für den Fall einer Raumüberwachung wäre ein System, bestehend aus einem Sensornetz und einem Server denkbar (Rechner mit Internetanbindung, Webserversystem und Telefonverbindung).
Die Sensorknoten werden an strategisch ausgewählten Standorten in Umfeldern, die überwacht werden sollen, installiert und mit entsprechend benötigten Sensoren ausgestattet. Ganz typisch ist hier der Feuchtigkeitssensor neben der Waschmaschine zu nennen. Das Basismodul wird per RS 232 Schnittstelle mit dem Server verbunden.

Das Softwaresystem auf dem Server sollte nun zwei grundlegende Systeme beinhalten.

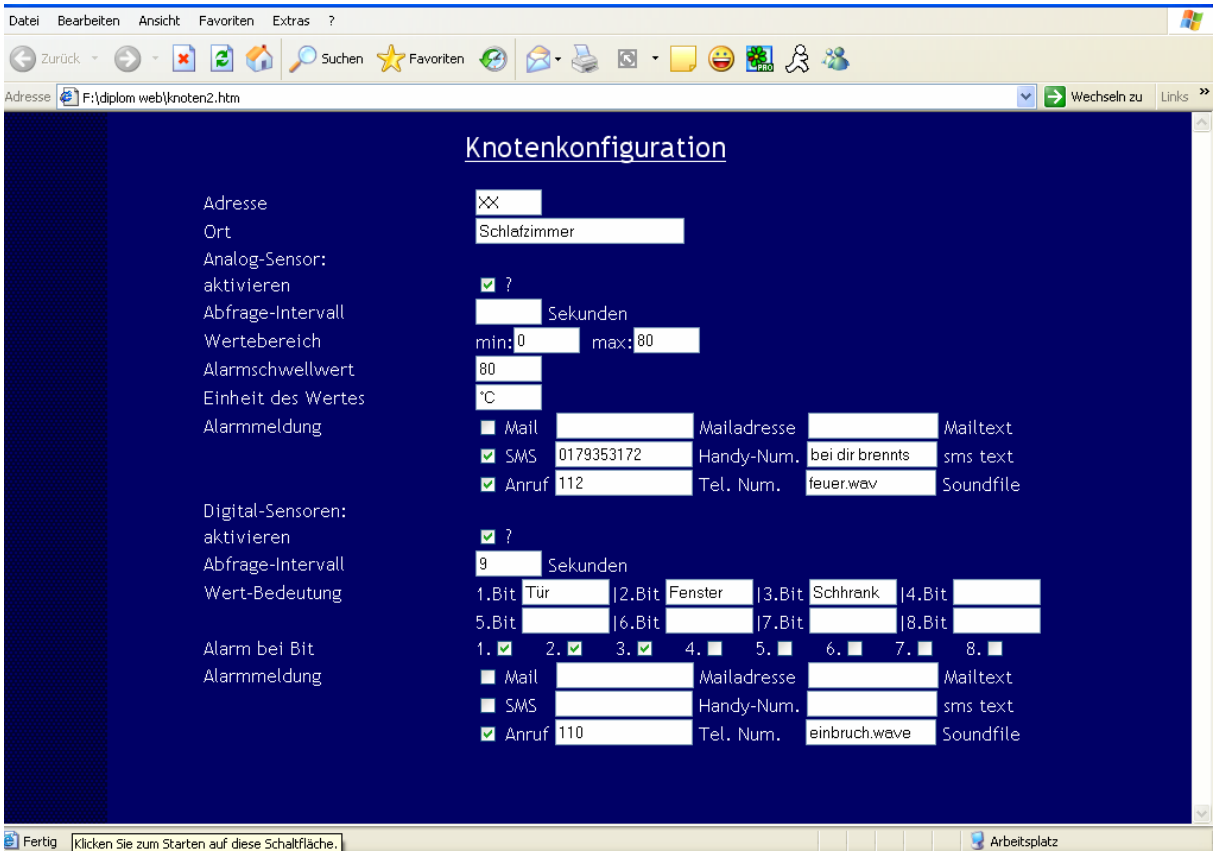
1. Eine Software, welche die Daten aus der RS 232 Schnittstelle einliest und diese in eine Datenbank überträgt.
2. Eine graphische Oberfläche zur Präsentation und Ansicht der Daten und der Konfiguration des Sensornetzwerkes.

Die graphische Oberfläche könnte folgendermaßen aussehen. Eine Webseite, welche die Sensorwerte darstellt und entsprechend deren Bedeutung interpretiert.



Bei diesem Beispiel sind nur die Analogwerte dargestellt, es soll auch nur eine Möglichkeit der Darstellung sein.

Zur Konfiguration der einzelnen Knoten könnte man auch jeweils eine eigene Seite erstellen. Hier ist das schon mal angedeutet, dadurch das die Knotennummern verlinkt sind. Aussehen könnte dies wie folgt:



Hier sieht man schon die Möglichkeiten, die sich durch die graphische Oberfläche bieten. Beispielsweise bei den Alarmwerten für die Digitalsensoren, wenn man jemanden bezahlt dafür das er nach dem Rechten schaut und Blumen gießt alle zwei Tage, kann man sich eine sms schicken lassen zum Zeitpunkt, wenn die Tür geöffnet wird. Man weiß Bescheid, das die Person auch seinen Auftrag wahrgenommen hat und nicht nur das Geld kassiert.

Durch die Benutzung eines Webservers als graphische Ausgabe der Werte ist das ganze System weltweit konfigurierbar und zu überwachen. Also im speziellen Fall einer Überwachung des Hauses bei längerer Abwesenheit, könnte man von jedem Rechner, mit Internetzugang , auf der Welt schauen, ob alles in Ordnung ist.

Durch die Anbindung des Servers an eine Telefonleitung kann das System im Notfall automatisch eine sms schicken oder im Falle eines Einbruchs sofort die Polizei informieren, durch abspielen vorgefertigter Nachrichten.

Wünschenswert wäre auch noch eine Seite, die den zeitlichen Verlauf der Sensorwerte darstellt. Dies sind aber alles Möglichkeiten, die durch die Benutzung der Datenbank gegeben sind und für den jeweiligen Anwendungsfall programmiert werden können, ohne das am Netz irgendetwas geändert werden muss.

Auch die graphische Darstellung der Netzstruktur, wäre eine weitere Bereicherung des Systems. Die Daten sind abrufbar, wie im Kapitel 5.2.7 beschrieben. Die Funktionalität ist beliebig erweiterbar und kann einen gewaltigen Umfang annehmen, je nach entsprechender Aufgabenstellung. Dies könnte also sowohl als Studien- oder vielleicht sogar als eine weitere Diplomarbeit bearbeitet werden.

7.4. Erweiterung der Software

Die bestehende Software sollte noch um einige Funktionalitäten erweitert werden, z.B. eine Autoadressattribierung. Diese macht für das Monitoring nicht immer Sinn, weil man gerne wissen will, wo der Sensor ist, mit dem man etwas überwacht. Ein Temperatursensor der in der Sonne liegt kann sehr heiß werden, was keine Gefahr der Umgebung widerspiegeln würde, allerdings ein Sensor hinter einem Schrank sollte niemals Temperaturen über 50°C anzeigen.

Für andere Monitoring Aufgaben kann es aber durchaus Sinn machen, nicht den genauen Standort zu kennen. Vorteil für diese Nutzung der Sensoren ist der Schutz gleiche Adressen zu nutzen, gerade wenn ein Netz auf selber Technik basierend in unmittelbarer Funkreichweite betrieben wird.

Erweiterungen der Software ist in vielen Bereichen denkbar, ob es nun um die Adressierung geht oder darum das ganze System auch für größere Netze zu nutzen.

Auch die Konvertierung eines TinyOS auf die CFMW1 Module wären wünschenswert, dazu sollten sich aber die Leser dieser Arbeit ihre eigenen Gedanken machen und vielleicht daraus ein eigenes Projekt entwickeln. Die entstandene Software darf von Jedem benutzt, erweitert und verändert werden.

8. Rückblick / Resümee

Die grundlegenden Ziele dieser Arbeit sind erreicht worden. Es wurde nun eine funktionierende Software erstellt, die für die gesetzte Aufgabe alle notwendigen Funktionalitäten liefert.

Erreicht wurden folgende Funktionalitäten / Aufgaben:

1. die Module können sich untereinander vernetzen
2. Daten werden per Multi-Hop durch das ganze Netz verbreitet
3. es können verschiedene Sensoren abgefragt werden, in unterschiedlichen Abfrageintervalle
4. es ist ein Verfahren implementiert, um ein Fluten des Netzes zu verhindern
5. alle Daten die erhoben werden können an einen PC ausgegeben werden
6. der Datentransfer unterliegt einem einheitlichen Frameformat
7. verschiedene Datenpakete wurden definiert
8. es kann die Struktur des Netzwerks ermittelt werden

Die Software ist somit einsatzbereit, allerdings derzeit nur auf dem Modul CC2420DBK. Eine Konvertierung auf das CFMW1 Board konnte nicht erzielt werden, da die Hardwareentwicklung parallel zur Softwareentwicklung verlief. Leider wurde in der zu Verfügung stehenden Zeit auch die Erstellung einer graphischen Oberfläche nicht erreicht..

Im Rückblick auf das vorgegebene Ziel und der daraus resultierenden Ausarbeitung, hätte man die Aufgaben weiter aufteilen sollen.
Allein im Bereich der Softwareerstellung, diese hätte man in verschiedene Teilaspekte differenzieren können:

- Protokollentwicklung, Definition Frameformat und deren Implementierung
- Entwicklung des Betriebssystem
- Graphisches User Interface (GUI)

Dies wäre aber sicher erst durch eine entsprechende Machbarkeitsstudie im Vorfeld genau abschätzbar gewesen.

Das Projekt der Entwicklung eines kostengünstigen Sensornetzwerkes für Monitoring Aufgaben sollte unbedingt weitergeführt werden. Es könnte daraus ein Netzsystem entstehen, was auch eine Anschaffung im privaten Bereich ermöglicht und natürlich auch den Einsatz.

Eine commerzielle Vermarktung wäre auch durchaus denkbar.
Dazu müssten aber noch Arbeiten, wie die Konvertierung auf das CFMW1 Modul, die Erstellung einer GUI und Veränderungen an der bestehenden Software erfüllt werden.

Zu den Änderungen an der Software zählen z.B.:

- mögliche andere Adressierungsarten
- ausgereiftere Formen der Kollisionsvermeidung
- Sicherung der Daten (Adresse, Einstellungen etc.) im eprom des Atmel Mega 128, für den Fall der Unterbrechung der Stromversorgung
- Datenverschlüsselung
- Änderung von Netzadressen

Viele dieser Probleme ließen sich durch bessere Softwarelibraries zum CC2420 lösen, dazu steht aber mehr im folgenden Punkt Kritik.

Kritik

Trotz der des Zugriffs auf die Softwarelibraries der Forma Chipcon, für den CC2420, muss das zur Verfügung stehende Softwarepaket kritisch betrachtet werden. So fehlen im verfügbaren ZigBee-Protokoll viele grundlegende Funktionalitäten, die eigentlich im ZigBee-Standard enthalten sind. Es wird durch Chipcon nicht der gesamte Protokollstack zur Verfügung gestellt.

Dies macht sich zum Beispiel bei der Adressierung bemerkbar. Es gibt keine Möglichkeit auf die MAC-Adresse eines Knotens zu zugreifen, dies schränkt die Möglichkeiten der Adressierung sehr ein.

Auch das Thema Netzsicherheit ist nicht verwirklicht, so ist es mit dem derzeitigen Stand der Software nicht möglich die Netze zu verschlüsseln. Obwohl der Einsatz von Netzwerkschlüsseln vorgesehen ist.

Das Auswerten von Acknowledgepaketen ist nicht möglich, dies wäre sinnreich um ein erneutes Senden von Daten sicherzustellen, falls es zur Kollision kam. Es ist zwar möglich die Versendung eines Aknowledgepaketes an zu fordern, jedoch nicht ersichtlich ob dieses Paket in der Hardware (CC2420) ausgewertet wird oder nicht.

Andere Funktionalitäten die im ZigBee-Protokoll festgeschrieben werden, wie *beacon* Frames etc. sind nicht durch die Software zu erreichen.

Die Software der Firma Chipcon ist extra für das Demonstrationsboard (CC2420DBK) geschrieben und ich bin der Meinung eine Demonstration mit so minimalistischen Möglichkeiten ist nicht ausreichend. Dies fällt als Erstes besonders auf, beim Empfang von Daten. Die ISR springt eine selbst zu schreibende Funktion an, die laut Hinweis von Chipcon sehr schnell arbeiten muss, weil die Daten nicht gepuffert werden. Ich erwarte von einem Demonstrationssystem zumindest eine saubere Pufferung der Daten .

Auch der Zugriff auf grundlegendere Funktionen ist nicht möglich, so z.B. der Zugriff auf Empfangs- und Sendestärke der Module.

Augenscheinlich werden von der Firma die entsprechenden Softwarebibliotheken nur sehr halbherzig gepflegt, selbst für Bereiche die nichts mit ZigBee zu tun haben. So musste ich die Headerdateien für den Zugriff auf die RS 232 Schnittstelle umschreiben, um ein aktives *polling* der Schnittstelle zu realisieren. Eine Konfigurationsdatei für die Einstellungen der Schnittstelle wäre sehr wünschenswert (polling, waiting, IRQ etc.).

Zwar stellen die Libraries der Firma Chipcon eine Anzahl Funktionalitäten zur Verfügung, jedoch sind diese nur sehr rudimentär und für die Entwicklung großer komplexer Netze absolut nicht ausreichend. Solange man nicht selbst ein Protokollstack programmiert oder zusätzlich kauft, ist man gezwungen auf eine Erweiterung der Software der Firma Chipcon zu warten.

Links und Literatur

(1) Sikora, A.: ZigBee: Grundlagen und Applikation; März 2004; S. 18 ff.

(2) Dreher, D.: Ad-Hoc Netze – Anwendungsgebiete von Ad-Hoc Netzen & Sensornetzwerke in der Automobilnetzwerke; Seminararbeit Systemengeneering; Juli 2004

(3) Homepage der ZigBee-Alliance

(4) http://de.wikipedia.org/wiki/Zigbee

(5) www.atmel.com

(6) www.chipcon.com

(7) http://www.sensorsmag.com/articles/0402/40/)

(8) http://www.cmt-gmbh.de/mpr5x0_mica2dot.htm)

(9) http://www.cmt-gmbh.de/mts510.htm)

(10) http://www-mtl.mit.edu/researchgroups/icsystems/uamps/research/images/uamp-1_large.jpg

(11) http://www.vs.inf.ethz.ch/res/proj/smart-its/btnode.html)

(12) http://rtl.e-technik.uni-rostock.de/~frei/website/index.php?id=to_dike_project&type=1

(13) http://www.sensorsmag.com/articles/0402/40/

(14) www.tinyos.net

(15) http://www.xbow.com/de/Funksensor-Netzwerk.htm

(16) http://www.iis.fraunhofer.de/medtech/sensor/monitoring/index_d.html

(17) www.fmncom.com

(18) http://www.heise.de/tp/r4/artikel/16/16312/1.html

(19) Chipcon: Quick Start Instructions SmartRF® CC2420DBK Demonstration Board Kit

(20) User Manual Rev. 1.3 SmartRF® CC2420DBK Demonstration Board Kit

(21) Tanenbaum, A.; Stehen, S., Maarten van: Verteilte Systeme Grundlagen und Paradigmen; 2003, Pearson Studium

(22) Tanenbaum, A.: Computernetzwerke 2003, Pearson Studium

(23) Zhao; Feng; Guibas; Leonidas: Wireless Sensor Networks An Information Processing Approach 2004, Morgan Kaufmann

(24) Hegering, H.-G., Abeck, S., Neumair, B.: Integriertes Management vernetzter Systeme Konzepte, Architekturen und deren betrieblicher Einsatz 1999, Dpunkt Verlag

(25) Bender, P.; Klein, M.; Mülle, J.; Schepperle, H.: Verteilung und Integration von Informationen im Verkehrsbereich; Seminar Sommersemester 2004, Universität Karlsruhe Oktober 2004

(26) Sikora, A.: Short-Range Wireless Networking mit IEEE802.15.4 und ZigBee: Möglichkeiten und Heruasforderungen; Berufsakademie Lörrach; Juli 2004

(27) Kosch, T.; Schwingenschlögl, C.; Bettstetter, C.; Situative IP-basierte Fahrerinformationssysteme: Szenarien, Routing und prototypische Realisierung; BMW Group Forschung & TU München, 2002

(28) Spiegl, W. Flügel, C.: Lokalisierung in Sensornetzwerken; Fraunhofer Institut Integrierte Schaltungen, Juli 2005

(29) Schößlin, J.: Seminar Sensornetzwerke TinyOS, 2004

(30) Weimerskirch, A.: Athentikation in Ad-hoc und Sensornetzwerken; Ruhr-Universität Bochum, 2004

(31) Pacelle, M.: Sensor-Ring: Völlig losgelöst; Millenial Net; 2005

(32) ZigBee/IEEE802.15.4 True One-Stop-Shop compliant solutions; Cipcon AS, Oslo, Juni 2005

(33) Wahl, T.: Smart Dust System-on-a-Chip Seminar SS2001, Universität Stuttgart, Juli 2001

Anhang

1. CD-Rom mit Quelle-Code und den angegeben digitalen Quellen

Versicherung über die Selbständigkeit der Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(4) bzw. §25(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den 15. August 2005