CARL VON OSSIETZKY universität OLDENBURG

DEPARTMENT FÜR INFORMATIK
SYSTEMSOFTWARE UND VERTEILTE SYSTEME

Entwurf und Implementierung eines Sensornetzwerks zur Erfassung eines Temperaturprofils in Wohn- und Büroräumen

Diplomarbeit

31. Mai 2010

Nils Schröder
Uhlhornsweg 5
26129 Oldenburg

Erstprüfer Prof. Dr.-Ing. Oliver Theel
Zweitprüfer Dipl.-Inform. Felix Oppermann

Erklärung zur Urheberschaft

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe.

Oldenburg, den 31. Mai 2010

Nils Schröder

Funkbasierte Sensornetzwerke ermöglichen die Erfassung von Umweltparametern mithilfe von Sensoren und das Steuern angeschlossener Aktoren. Hierbei ist ein wichtiges Designkriterium das verwendete Netzwerkprotokoll, welches energieeffizient und fehlertolerant sein muss. Anhand des Anwendungsfalls der Temperaturmessung innerhalb von Wohn- und Büroräumen wird in dieser Arbeit ein solches Sensornetzwerk entworfen, implementiert und getestet. Ebenfalls wird ein GUI-Programm zur Messdatenbetrachtung und Überwachung des Sensornetzwerks vorgestellt.

Wireless Sensor Networks permit to collect environmental parameters by using sensors and to control connected actuators. An important criterion of design is the use of a network protocol that has to be energy-efficient and fault-tolerant. Within this thesis, such a sensor network is designed, implemented and tested for the use case of temperature measurement within home and office buildings. Also, a GUI application to show the measuring data and to monitor the sensor network is presented.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Wohn- und Büroräume werden meist mithilfe einer Zentralheizung beheizt. Hierbei wird an zentraler Stelle ein Wärmeübertragungsmedium, in der Regel Wasser, im Wärmeerzeuger erhitzt und über Rohrleitungen zu den Heizflächen befördert. Der Weg zu den Heizkörpern wird auch Vorlauf genannt. In den Heizkörpern wird die Wärme an die Umgebung abgegeben und das erkaltete Medium fließt über den Rücklauf zurück zum Wärmeerzeuger.
Die Temperatur in den Räumen ergibt sich durch die für alle Räume global am Wärmeerzeuger geregelte Vorlauftemperatur und die lokal eingestellten Durchflussraten an den Heizkörpern.
Die Regelung der Vorlauftemperatur wird von einer durch die Außentemperatur und einen manuell am Wärmeerzeuger eingestellten Wert parametrisierten Heizkurve bestimmt. Hierbei gilt: Je niedriger die Außentemperatur, umso höher die Vorlauftemperatur und vice versa. Der Einstellwert des Wärmeerzeugers bestimmt die Steilheit der Heizkurve. In Wohngebäuden wird oft zusätzlich die Temperatur eines Raumes, meist des Wohnzimmers, zur Regelung der Vorlauftemperatur mit herangezogen. So ist es möglich, einen Raum genau auf eine Temperatur zu regeln und zudem bei der Heizkurve solare Einstrahlung durch Fenster und die Isolierung des Hauses mit einzubeziehen [Fira].
In den einzelnen Räumen wird die Temperatur anhand der Ventile an den Heizkörpern eingestellt. Je höher die Durchflussraten der Heizkörper im Raum, umso höher ist auch die Temperatur. Es kommen meistens Thermostatventile zum Einsatz. Bei diesen kann der Benutzer eine Raumtemperatur einstellen, welche je nach Hersteller innerhalb einer Bandbreite von ca. 2°C geregelt wird [The]. Die Regelung geschieht hier mit Hilfe eines Dehnstoffelements im Thermostatventil.
Eine oft gewünschte, im gesamten Gebäudekomplex einheitliche Temperatur lässt sich so jedoch nur sehr schwer realisieren. Verschiedene Gegebenheiten wie Heizleistung der einzelnen Heizkörper, Möbelstücke vor den Heizkörpern und die Wärmeisolierung erfordern verschiedene Thermostateinstellungen für verschiedene Räume und verschiedene Heizkörper. Offene Türen bewirken zudem eine Interaktion der Regelkreisläufe zwischen mehreren Räumen. Unter Umständen werden so Räume geheizt, welche bereits durch Nachbarräume ausreichend geheizt würden. Einstellparameter für die Thermostate sind so nur sehr schwer ohne eine längerfristige Temperaturverteilungserfassung zu ermitteln.
In Industriegebäuden kommt daher oftmals eine Warmluftheizung zum Einsatz. Dabei wird an zentraler Stelle mit einem Wärmetauscher warme Luft einer bestimmten Temperatur erzeugt. Die Warmluft wird dann über Lüftungsschächte in die einzelnen Räume befördert. Temperatursensoren bestimmen hierbei die Menge der zugeführten Luft.
Warmluftheizungen erfordern jedoch aufgrund der Wärmetauscher, der Sensorinfrastruktur und der aufwändigeren Lüftungsschächte einen deutlich höheren Installations- und Technikaufwand, wie anhand einer Werksbesichtigung des Bahlsenwerks in Varel nachvollzogen werden konnte. Aus diesem Grund kommt eine Warmluftheizung meist nur in Produktionshallen zum Einsatz. Der Verwaltungstrakt bei dem besichtigten Bahlsenwerk ist ebenfalls mit einer herkömmlichen Warmwasserheizung ausgestattet.
Warmluftheizungen erlangen in jüngster Zeit jedoch einen immer größeren Marktanteil. In großen Hochhäusern können die Fenster meist nicht geöffnet werden. Dies zieht die Installation eines Lüftungssystems nach sich, von dem dann auch die Klimatisierung übernommen wird. Auch Niedrigenergiehäuser besitzen häufig ein Lüftungssystem, da so die Klimatisierung und der Luftaustausch energieeffizienter realisiert werden können.
Energieeffizienz bei Heizsystemen in Gebäuden nimmt einen immer größeren Stellenwert ein. Um die Regelung bei Heizsystemen intelligenter zu gestalten, ist zum einen eine Erfassung der derzeitigen Umweltparameter sowohl innerhalb als auch außerhalb der Gebäude notwendig. Zum anderen müssen dezentrale Aktoren angesteuert werden können.

Um ein verteiltes Phänomen zu erfassen, bietet sich ein Sensornetzwerk an. Ein solches besteht aus kleinen batteriebetriebenen Geräten mit Sensorik und drahtloser Netzwerkfähigkeit, welche meist maximal die Größe einer Zigarettenschachtel besitzen. Eine einfache Installation ist aufgrund der portablen Energiequelle und der funkbasierten Kommunikation hierbei gegeben.
An den Sensorknoten angeschlossene Sensoren ermöglichen die Messung von Umweltparametern wie Temperatur, Luftfeuchte, Öffnungszustand von Fenstern und Türen oder Anwesenheit von Menschen. Weiterhin können Aktoren angeschlossen werden, welche in den hier skizzierten Beispielen die elektrischen Ventile der Heizkörper oder die Lüftungsklappen einer Warmluftheizung sind. Insbesondere bei den Warmluftheizungen kann, bedingt durch die automatisierte Temperaturregelung, nicht auf ein Sensornetzwerk verzichtet werden.
Damit ist man in der Lage, eine intelligentere Heizungsregelung zu realisieren. In einem Bürogebäude kann so die Heizleistung in einzelnen Räumen in der Nacht reduziert werden und dennoch zum Arbeitsbeginn bereits wieder die gewünschte Temperatur automatisch hergestellt sein.
In dieser Diplomarbeit wird ein solches Sensornetzwerk zur Erfassung der Temperaturverteilung entworfen, implementiert und getestet. Hierbei handelt es sich um eine reine Datenerfassung der Temperatur und weiterer Umweltgrößen, wie Luftfeuchte und Lichtwerte, an interessanten Punkten des Gebäudes. Die erfassten Daten können in einem Diagramm betrachtet werden, welches dann Auskunft über z. B. den Temperaturverlauf an einem Punkt des Gebäudes über einen bestimmten Zeitraum gibt. Diese Daten können als Grundlage für eine Regelung oder Regelungsplanung für die Heizleistungen der Heizkörper herangezogen werden. Dies ist jedoch nicht mehr Bestandteil dieser Arbeit. Direkt angewendet werden kann das entwickelte Sensornetz zur Aufdeckung von falschem Lüftungs- und Heizverhalten der Nutzer. Zum einen können über die Lichtwerte Aussagen über die Nutzung der Räume gemacht werden, was eine manuelle Heizleistungsreduzierung ergeben kann. Zum anderen führt Feuchtigkeit oft zu Schimmelbildung. Erkennt man in den Messwerten eine hohe Luftfeuchte bei sinkenden Temperaturen, so kommt es bei geschlossenen Fenstern zu Kondensation. Diese Situation kann nun erkannt und verhindert werden.

2 Anforderungsdefinition

In diesem Kapitel werden die Anforderungen an das System hergeleitet und detailliert beschrieben. Hierzu werden zunächst im Abschnitt 2.1 die Aufgabe und die hieraus resultierenden Anforderungen beschrieben. Darauffolgend werden im Abschnitt 2.2 einzelne Anforderungen definiert, welche aus der Aufgabenbeschreibung abgeleitet wurden und zur Evaluation des Systems herangezogen werden.

2.1 Aufgabenbeschreibung

Es soll ein System zur Erfassung der Temperaturverteilung innerhalb von Wohn- und Büroräumen entworfen und implementiert werden. Dieses besteht aus zwei Komponenten.
Die eine Kompnente ist ein wie in Kapitel 1 beschriebenes Sensornetzwerk zur Erfassung von Temperatur, Luftfeuchte und Lichtwerten an neuralgischen Stellen innerhalb des Gebäudes.
Die andere Komponente ist ein PC-Programm, welches in der Lage ist, die aufgezeichneten Sensorwerte persistent zu speichern und dem Benutzer eine einfache Betrachtung der Daten zu ermöglichen.
Bei dem Sensornetzwerk sind verschiedene Herausforderungen zu bewältigen, welche im Folgenden kurz beschrieben werden. Hierzu ist jeweils eine umfassende Literaturrecherche durchzuführen, der aktuelle Stand der Technik zu dokumentieren und eine geeignete Lösung für das jeweilige Problem auszuwählen oder zu entwickeln.
Die Betriebsdauer eines Sensorknotens ist bei fehlerfreiem Betrieb durch die Batterielaufzeit begrenzt [CES04]. So ist es enorm wichtig, möglichst schonend mit den Energiereserven umzugehen, um die Laufzeit zu maximieren. Hierzu soll der Knoten möglichst oft in einen Ruhezustand versetzt werden. Weiterhin ist für eine gleichzeitige Sensordatenerfassung und für die Kommunikation zwischen den Sensorknoten eine Zeitsynchronisation der Knoten untereinander erforderlich.
Ein weiterer wichtiger Punkt in einem Sensornetzwerk ist die verwendete Netzwerkstruktur. Hierzu müssen die Anforderungen an die Netzwerktopologie erarbeitet werden, und es muss insbesondere ein passendes Routingverfahren zum Einsatz kommen. Als Herausforderung ist die geforderte Möglichkeit zur Selbstorganisation der Sensorknoten zu nennen. Es soll möglich sein, dass sich ein Sensor durch Einschalten automatisch in ein bestehendes Sensornetz integriert.
Das PC-Programm enthält die grafische Benutzerschnittstelle des Systems und soll dem Anwender die Interaktion mit dem laufenden Sensornetzwerk sowie eine Betrachtung aufgezeichneter Daten in Form von Temperaturverläufen ermöglichen. Hier muss eine Datenbank genutzt werden, welche sowohl die Daten der Sensorknoten als auch deren Position im Raum speichert.
Zur Verdeutlichung der Möglichkeiten des PC-Programms ist ein Anwendungsfalldiagramm in Abbildung 2.1 gegeben. Der Benutzer kann diverse globale Einstellungen vornehmen. Damit das PC-Programm sich mit dem Sensornetzwerk verbinden kann, muss er die serielle Schnittstelle, an der ein Sensorknoten angeschlossen ist, angeben. Darüber hinaus müssen dem PC-Programm der Datenbankname und die Zugangsdaten bekannt gemacht werden. Der Benutzer hat auch die Möglichkeit, eine neue Datenbank anzulegen. Sind diese Vorraussetzungen erfüllt, so kann er die Messauflösung und die zu erfassenden Messgrößen angeben. Um die Sensoren auf einem Lageplan darzustellen, kann ein Gebäudegrundriss geladen werden. Hat der Benutzer bereits zuvor Sensoren darauf positioniert, so werden sie mitgeladen. Anschließend kann der Benutzer Sensoren auf dem Gebäudegrundriss hinzufügen, verschieben und entfernen. Die aktuelle Ansicht ist nach diesen Aktionen inklusive der Sensoren wieder speicherbar. Klickt der Benutzer einen der Sensoren an, werden ihm die aufgezeichneten Daten in einem Zeit-Wert-Diagramm präsentiert. Ein weiterer Anwendungsfall ist die Betrachtung von sowohl im PC-Programm als auch im Sensornetzwerk aufgetretenen Fehlern.

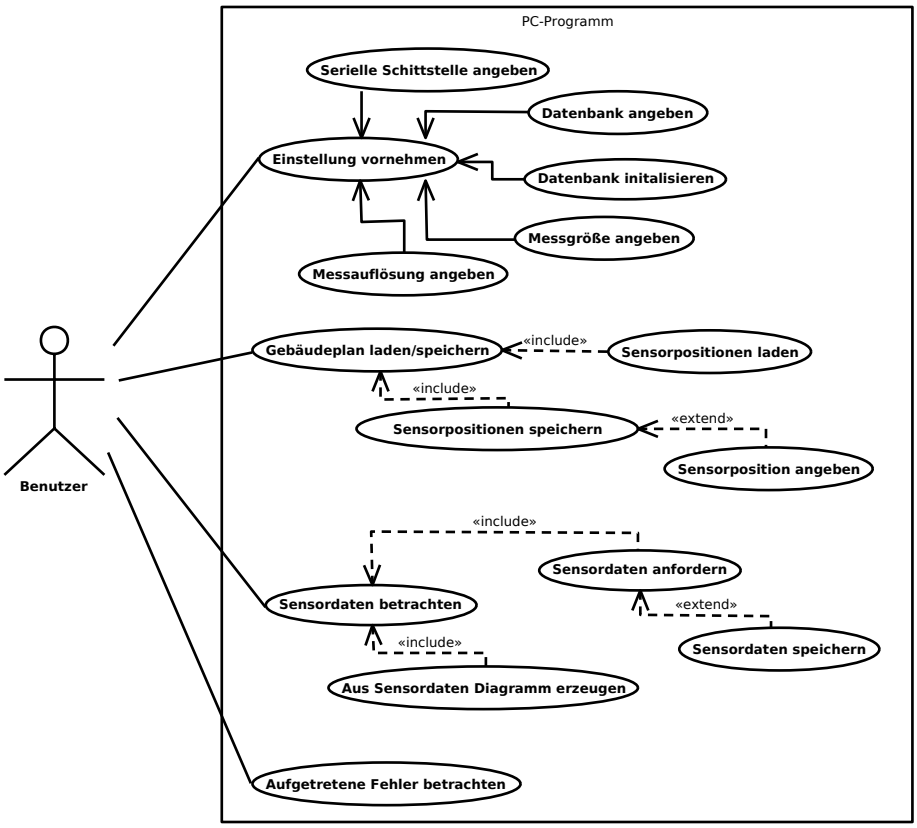


Abbildung 2.1: Das Anwendungsfalldiagramm für das PC-Programm

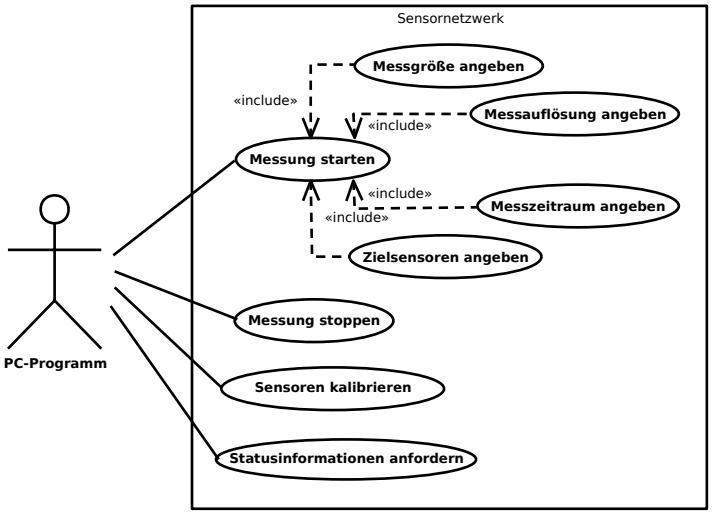


Abbildung 2.2: Das Anwendungsfalldiagramm für das Sensornetzwerk

Damit das PC-Programm die oben beschriebenen Funktionen anbieten kann, stellt es Anforderungen an das Sensornetzwerk, welche in dem in Abbildung 2.2 zu sehenden Anwendungsfalldiagramm dargestellt sind. Es kann eine neue Messung starten, wobei es auch die Möglichkeit haben muss, die Messgröße und die Auflösung sowie den Messzeitraum und die Zielsensoren, die die Messung durchführen sollen, anzugeben. Des Weiteren hat das PC-Programm die Möglichkeit, alle Messungen auf den Sensorknoten abzubrechen. Weitere Anwendungsfälle sind die Kalibrierung der Sensoren und das Abfragen von Statusinformationen. Es werden auch einige nichtfunktionale Anforderungen aufgestellt.
Mit ein Hauptgrund für die Entwicklung dieses Systems ist das Bestreben, erste Erfahrungen mit Sensornetzwerken für die Abteilung *Systemsoftware und verteilte Systeme* zu sammeln. Es ist daher dadrauf zu achten, bei Designentscheidungen auch ihre Bedüfnisse mit einzubeziehen.
Ferner ist eine ingenieurmäßige Vorgehensweise an den Tag zu legen. Das heißt, dass nach einem Entwicklungsmodell vorgegangen wird. Hier ist es das Wasserfallmodell. Es wird eine Anforderungsdefinition aufgestellt, ein Entwurf der einzelnen Komponenten findet statt, gefolgt von der Implementierung des Systems. Abschießend findet eine Testphase statt. Um das System in Zukunft nutzen und weiterentwickeln zu können, muss es zum einen leicht wartbar und zum anderen gut dokumentiert sein.

2.2 Anforderungen

Im Folgenden werden einzelne Anforderungen aus der vorangegangenen Aufgabenbeschreibung abgeleitet. Diese werden später für die im Abschnitt 6 stattfindende Evaluation herangezogen.
Hierbei wird unterschieden zwischen erforderlichen und wünschenswerten Anforderungen, anhand derer sich der Erfolg des implementierten Systems messen lassen kann.
Erforderliche Anforderungen sind für einen erfolgreichen Projektabschluss auf jeden Fall zu erfüllen. Die wünschenswerten Anforderungen sind hingegen als freiwillige Zusatzleistung anzusehen.

Erforderliche Anforderungen

E1 Datenerfassung: Die Sensorknoten müssen mindestens Umgebungstemperatur, Luftfeuchte und Lichtwerte erfassen können. Dies ist zwingend notwendig, um ein Wert-Zeit-Diagramm der genannten Größen zu erstellen. Es soll mindestens eine zeitliche Messauflösung von einem Wert jeder Messgröße pro Sekunde erreicht werden. Dieser Wert wurde willkührlich gewählt. Auf der einen Seite ist zu bedenken, dass Temperaturänderungen sich aufgrund der Trägheit der Sensoren nicht beliebig schnell erfassen lassen, jedoch die Messauflösung das Datenaufkommen und damit das zu verwendene Routingprotokoll stark beeinflusst. Auch ist davon auszugehen, dass sich die Temperatur in einem Raum nicht im Sekundentakt ändern kann.

E2 Laufzeit der Sensorknoten: Die Batterielaufzeit soll maximiert werden, da diese die Lebensdauer des Sensornetzwerks ohne Wartung begrenzt. Es ist wünschenswert, dass das Sensornetz bei Messungen im Minutentakt mindestens ein Jahr mit einem Satz Batterien operieren kann. Dies ist die Zeitspanne, die in der Literatur als typisch angesehen wird [CES04].

E3 Zeitsynchronisation: Die Uhren der einzelnen Sensorknoten müssen hinreichend genau synchronisiert werden, damit eine gleichzeitige Messwerterfassung und eine problemlose Kommunikation gewährleistet ist. Die Zeitsynchronisation wird bei der hier entwickelten Anwendung als hinreichend angesehen, wenn die Sensorknoten um nicht mehr als 250 Millisekunden voneinander abweichen. Dieser Wert entspicht einem Viertel der Messauflösung; so können immer zwei Messwerte verschiedener Sensorknoten eindeutig einander zugeordnet werden.

E4 Selbstorganisation der Sensorknoten: Die Sensorknoten müssen sich beim Einschalten selbstständig zu einem Netz organisieren. Neu hinzukommende Knoten sollen im laufenden Betrieb mit dem

Einschalten in das Sensornetz aufgenommen werden. Fällt ein Sensorknoten aus, so soll das verbleibende Sensornetz diesen Fehler eigenständig kompensieren können, wenn noch ein Pfad zwischen allen Sensorknoten vorhanden ist. Dies ist notwendig, um auftretende Fehler, bedingt durch defekte Hardware oder Energiemangel, kompensieren zu können.

E5 **Datenspeicherung:** Es müssen Messdaten, Zeitstempel der Messung und Position persistent gespeichert werden. Dies ist notwendig, um auch zu einem späteren Zeitpunkt die Daten betrachten zu können.

E6 **Aktoransteuerung:** Auch wenn Aktoren für den eigentlichen Inhalt dieser Arbeit nicht notwendig sind, so sollen sie schon im Entwurf des Sensornetzwerks bedacht werden, damit die Möglichkeit zum Hinzufügen einer Regelung zum System gegeben ist. Diese Anforderung leitet sich insbesondere aus dem Anspruch an die Erweiterbarkeit des Systems ab.

E7 **Datenvisualisierung:** Gespeicherte Daten müssen in Diagrammform vom Benutzer abgerufen werden können. Diese Anforderung ist wichtig, damit der Benutzer die erfassten Daten leicht auswerten und auf einen Blick sofort Tendenzen in den Daten erkennen kann. Zudem soll der Zugriff auf die momentanen Messwerte des laufenden Sensornetzes möglich sein.

Wünschenswerte Anforderungen

W1 **komplexe Visualisierung:** Es ist wünschenswert, dass eine umfangreiche Benutzeroberfläche für einen nicht mit dem System vertrauten Anwender entsteht. Sensorknoten sollen auf einem Lageplan positioniert werden können. Diese Anordnung sollte in mehreren Ebenen und dreidimensional erfolgen. Es ist weiterhin wünschenswert, dass aus den diskreten Messpunkten Temperaturen für beliebige Punkte errechnet werden können.

W2 **Trennung zwischen Benutzerschnittstelle und Sensornetzkontrolle:** Es ist wünschenswert, dass das PC-Programm in ein grafisches Client-Programm für die Benutzerinteraktion und ein Server-Programm zur Speicherung der Daten und Kontrolle des Sensornetzwerks unterteilt wird. GUI und Server sollen über das Netzwerk kommunizieren können. Der Benutzer muss damit nicht an demselben PC sitzen, an dem auch das Sensornetz angeschlossen ist.

W3 **komplexe Auswertung:** Es ist wünschenswert, dass Algorithmen gefunden werden, die auf Basis der aufgezeichneten Temperaturverläufe eine Empfehlung für die Heizleistung geben. Damit kann dann eine intelligente Heizungsregelung implementiert werden.

3 Techniken in Sensornetzwerken

Dieses Kapitel führt in die Welt der Sensornetze ein. Es werden Grundlagen und Begriffe geklärt und die Techniken, welche für den Entwurf und die Implementierung eines Sensornetzes zur Temperaturprofilerfassung in Wohn- und Büroräumen benötigt werden, vorgestellt. Dieses Kapitel ist auch als schneller Einstieg in den Themenbereich der Sensornetzwerke für Folgearbeiten zu diesem Thema innerhalb der Abteilung *Systemsoftware und verteilte Systeme* gedacht.
Zunächst werden in Abschnitt 3.1 Begriffe aus der Welt der Sensornetzwerke definiert.
Sensornetzwerke entwickeln sich zu einer wichtigen neuen Klasse von eingebetteten Systemen in unserer Umwelt [Bou09, Seite 21]. Sie werden für eine Vielzahl von verschiedensten Anwendungen eingesetzt, eine ausgewählte Übersicht in Abschnitt 3.2 gibt einen Einblick in die Anwendungsscenarien für Sensornetzwerke.
In Abschnitt 3.3 werden dann die von der Abteilung angeschafften Sensorknoten vorgestellt.

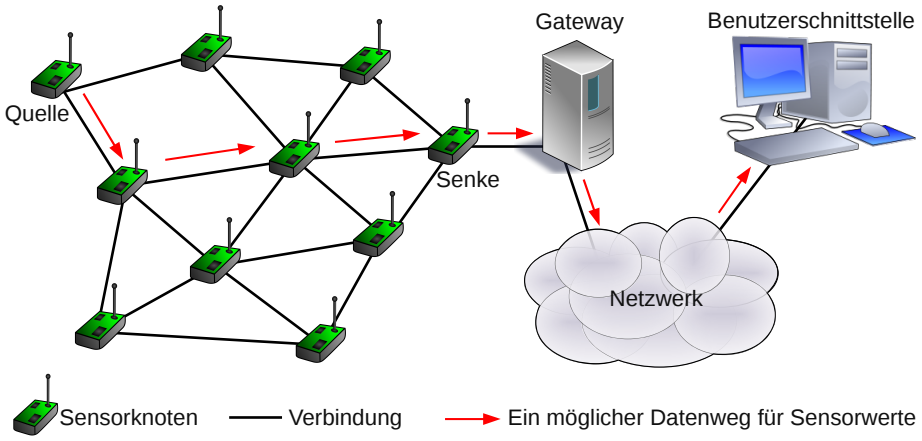


Abbildung 3.1: Der typische Aufbau eines einfachen Sensornetzwerks

Ein kabelloses Sensornetzwerk in seiner einfachsten Form kann laut [VDMC08, Seite 1] definiert werden als Netzwerk bestehend aus Sensorknoten, welche in der Lage sind, Umweltgrößen zu messen und die gesammelten Daten über eine kabellose Verbindung zu kommunizieren. Die Sensorknoten sind stationär oder beweglich, und sie können sich ihrer Position bewusst sein oder auch nicht. Die gesammelten Informationen werden, in einigen Fällen von anderen Sensorknoten weitergeleitet, an die Senke gesendet, wo sie lokal verarbeitet oder von einem angeschlossenen Gateway in ein anderes Netzwerk weitergeleitet werden. Vergleiche hierzu Abbildung 3.1. Das Gateway ist entweder ein Sensorknoten, ein kleines eingebettetes System, wie ein PDA, oder ein PC. Es besitzt meist eine unerschöpfliche Stromversorgung, größere Rechenleistung sowie Speicher und weitere Netzwerkhardware, um per Satellit, UMTS, GSM oder direkt per Ethernet mit anderen Netzwerken wie dem Internet verbunden zu sein. Von hier aus kann der Benutzer auf das Sensornetzwerk zugreifen.
Die Anwendungsenwicklung bei Sensornetzen unterscheidet sich in vielen Punkten von der Entwicklung für PCs. In Abschnitt 3.4 werden die Anforderungen bei der Entwicklung von Software für Sensornetzwerke erörtert.
In Abbildung 3.2 ist der typische Schichtaufbau der Software eines Sensornetzes zu sehen. Auf den Sensorknoten setzt ein Betriebssystem auf. In Abschnitt 3.5 wird die Aufgabe des Betriebssystems geschildert,

und es findet ein Vergleich von erhältlichen Betriebssystemen für Sensorknoten statt. Ein zentrales Element eines Betriebssystems für Sensorknoten ist das verwendete Netzwerkprotokoll, welches in Abschnitt 3.6 vorgestellt wird. Die Middleware ergänzt das Betriebssystems um Funktionen, beispielsweise zur Zeitsynchronisation (Abschnitt 3.7) und zur Positionsbestimmung (Abschnitt 3.8) [Sta08]. Auf der Middleware setzt die eigentliche Anwendung auf. Bei der Middleware und der Anwendung ist die Systemgrenze nicht in allen Fällen bei einem einzelnen Sensorknoten zu ziehen. Bei dem in dieser Arbeit vorgestellten System zur Temperaturprofilerfassung ist die Anwendung zwar auf Sensorknoten beschränkt, der Mechanismus zur Zeitsysnchronisation muss jedoch als systemweit angesehen werden. Alle Zeitsynchronisationsinstanzen der einzelnen Sensorknoten kommunizieren untereinander und beeinflussen sich gegenseitig.

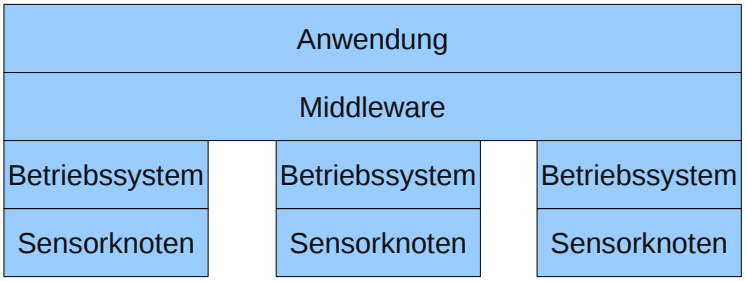


Abbildung 3.2: Die Softwareschichten in einem Sensornetzwerk (nach [RKJK03])

3.1 Begriffsdefinitionen

In diesem Abschnitt werden Definitionen zu Begriffen und Objekten aus dem Themenbereich der kabellosen Sensornetzwerke gegeben. Hier sind insbesondere wichtige Begriffe zu finden, die in mehreren Kapiteln dieser Arbeit Verwendung finden. Auch werden Begriffe aufgeführt, die in dieser Arbeit keine große Relevantz besitzen, jedoch für eine Übersicht über das Fachgebiet von Bedeutung sind.
Der überwiegende Teil der Fachliteratur ist in englischer Sprache gehalten, zum leichteren Verständnis werden die englischen Synonyme in Klammern angegeben.

Sensorknoten (Node oder Mote in Anlehnung an die Größe) Ein Sensorknoten besteht aus den vier Teilkomponenten Recheneinheit, Funkeinheit, Spannungsversorgungseinheit und Sensoreinheit [Bou09, Seite 22]. Je nach Anwendung kommen noch andere Komponenten, etwa zur Positionsbestimung oder Datenspeicherung, hinzu [ASSC02]. Sensorknoten besitzen typischerweise inklusive Stromquelle maximal die Größe einer Zigarettenschachtel. Des Weiteren ist die alle Komponenten verbindende Recheneinheit meist mithilfe eines Mikrocontrollers realisiert, welcher typischerweise mit einer Taktrate von 10 bis 20 MHz betrieben wird. Vergleiche hierzu die Hardwarespezifikation der verwendeten Sensorknoten im Abschnitt 3.3.

Kabelloses Sensornetzwerk (Wireless Sensor Network kurz WSN) Als kabelloses Sensornetzwerk wird ein Rechnernetz bestehend aus Sensorknoten, welche in der Lage sind, Umweltparameter zu erfassen und diese kabelungebunden zu kommunizieren, bezeichnet. Die Kommunikation findet nicht nur zwischen den direkten Nachbarn statt, sondern kann auch über mehrere Sensorknoten geroutet werden. Sensornetzwerke sind entweder autark oder mit einem anderen Netzwerk wie dem Internet verbunden. Die Sensorknoten können stationär oder beweglich sein. [VDMC08, Seite 1]

Sind die Sensorknoten in der Lage, angeschlossene Aktoren zu steuern, wird auch von kabellosen Sensor- und Aktornetzen gesprochen [VDMC08, Seite 4].

Heterogenes und homogenes Sensornetzwerk Homogenität oder Heterogenität beschreiben die im Sensornetzwerk verwendeten Sensorknoten. In homogenen Sensornetzwerken sind alle Sensorknoten von identischer Hardware, während in heterogenen Netzen mindestens zwei verschiedene Hardwareplattformen zum Einsatz kommen. Zur Zeit befnden sind überwiegend homogene Sensornetzwerke im Einsatz [Bou09, Seite 21]. Auch das in dieser Arbeit entwickelte Sensornetz ist homogen. Heterogene Sensornetze bestehen meist aus besonders energiesparenden kleinen Knoten zur Datenerfassung und vereinzelnten, durch Anschluss an ein Stromnetz mit energie versorgten, Sensorknoten mit mehr Rechenleisung und größeren Speichern zur Datenverarbeitung.

Man beachte, dass Homogenität und Heterogenität keine Aussage über die auf den Sensorknoten installierte Software machen.

Multi-Hop-Sensornetzwerk Typischerweise können in einem Sensornetzwerk nicht alle Sensorknoten, bedingt durch die begrenzte Funkreichweite, direkt miteinander kommunizieren. Zum Austausch von Daten zweier dieser Knoten müssen n dazwischenliegende Knoten die Daten weiterleiten, und es werden insgesamt $n + 1$ Transmissionen durchgeführt. Ist das Netzwerkprotokoll zu dieser Datenweiterleitung fähig, so spricht man von einem Multi-Hop-Sensornetzwerk [VDMC08, Seite 3].

Mikrocontroller Ein Mikrorechensystem ist ein Datenverarbeitungssystem bestehend aus Prozessorkern, Speicher, Ein- und Ausgabeschnittstellen sowie einem verbindenden Bussystem. Außerdem sind bereits ein oder mehrere Peripherie-Geräte, also A/D-Wandler, Sensoren oder Timermodule, angeschlossen. Ein Mikrocontroller ist ein Mikrorechner, der auf einem einzigen Chip realisiert wurde, um mit möglichst wenig Schaltungsaufwand Steuerungs- und Kommunikationsaufgaben zu lösen [BU07, Seite 1].

Analog-Digital-Wandler (A/D converter) Viele Sensoren sind analog und nutzen zur Ausgabe eine Spannung als Repräsentanten des Messwertes. Ein Analog-Digital-Wandler (A/D-Wandler) setzt diese Spannung in einen digitalen Wert um, welcher weiterverarbeitet und gespeichert werden kann. A/D-Wandler unterscheiden sich durch die Abtastfrequenz und die Messauflösung. Die Abtastfrequenz, angegeben in Hz, gibt Auskunft über die Anzahl der Wandeloperationen pro Sekunde. Die Auflösung, angegeben in Bit, bestimmt die Genauigkeitsgrenze für die Umsetzung des Spannungswertes in einen digitalen Wert. Es ist zu beachten, dass die Auflösung des A/D-Wandlers meist der begrenzende Faktor für die Schrittweite zwischen zwei Werten eines angeschlossenen Sensors ist.

Prozess oder Thread Ein Prozess ist formal definiert als die Abbildung $f : T \rightarrow S$. Hierbei ist T eine Menge diskreter Zeitpunkte und S die Menge aller Zustände des Systems. Ein Prozess kann auch als sequentielles Programm auf einem Prozessor in seiner Umgebung angesehen werden. Die Umgebung besteht aus dem Programm selbst, den Registern des Prozessors, den Daten im Hauptspeicher sowie Daten im allgemeinen Sinn, etwa belegten Betriebsmitteln [The05, Seite 32f].

Threads sind sogenannte Leichtgewichtsprozesse. Sie realisieren mehrere Kontrollflüsse innerhalb desselben Adressraums, welcher an einen Prozess gebunden ist. Sie gehören somit eindeutig zu genau einem Prozess und teilen sich mit ihm und untereinander auch die Umgebung [The05, Seite 136].

In der Welt der Sensornetzwerke wird nicht so streng zwischen einem Prozess und einem Thread unterschieden, die Begriffe werden synonym benutzt. Mikrocontroller besitzen keine Memory Management Unit (MMU) zur einfachen Implementierung getrennter Adressräume. Aufgrund der Ressourcenbeschränkungen (vergleiche Abschnitt 3.4) und des Einbenutzer-Betriebes implementieren die Betriebssysteme eine solche auch nicht.

In dem Betriebssystem TinyOS existieren keine Prozesse, es werden aber Threads mit einem optionalen Modul angeboten [Unb]. Entgegen der eigentlichen Threaddefinition können sie keinem anderen Thread zugeordnet werden, mit dem sie sich die Umgebung teilen.

In dem Betriebssystem Contiki existieren sowohl Prozesse als auch Threads. Prozesse realisieren Ereignisse und existieren innerhalb derselben Umgebung. Threads werden zwar eindeutig einem Prozess zugeordnet, sie besitzen jedoch eine eigene Umgebung.

Im weiteren Verlauf dieser Arbeit wird daher von Threads gesprochen, sofern die Quelle nicht explizit den Prozessbegriff nutzt.

Task Als Task wird in der Welt der Sensornetzwerke ein Thread bezeichnet, welcher keinen Umgebungswechsel hervorruft und auf die Variablen des erzeugenden Threads zugreifen kann. Er kann als sequentielles Programm angesehen werden, das in der Umgebung des taskerzeugenden Threads läuft. Weiterhin ist es wichtig zu wissen, dass Tasks eine Run-to-completion-Semantik besitzen: Sie werden bis zum Ende ununterbrechbar ausgeführt [RKJK03]. Im Gegensatz zu Funktionsaufrufen wird bei der Erzeugung des Tasks der Kontrollfluss des Erzeugers weitergeführt, und der Task wird zu einem späteren Zeitpunkt vom Scheduler dem Prozessor zugewiesen.

Quelle und Senke (Source and Sink) Als Quellen werden diejeniegen Sensorknoten im Sensornetzwerk bezeichnet, die mit ihren Sensoren Umweltparameter erfassen. Senken sind alle Sensorknoten im Netz, welche Daten von anderen Sensorknoten entgegennehmen und persistent speichern oder an ein anderes Netzwerk weiterleiten. Die Daten fließen somit immer von der Quelle zur Senke. Ein Sensornetzwerk kann mehrere Quellen und Senken beinhalten.

Ankerknoten (Anchor node) Ankerknoten sind Sensorknoten mit bekannter, nicht von dem Sensornetz selbst errechneter Position im Raum. Sie besitzen entweder ein GPS-Modul oder die Position wird beim Aufstellen festgehalten. Im Kontrast hierzu werden Sensorknoten mit unbekannter oder geschätzter Position unbekannte Knoten (unknown nodes oder agents) genannt. [VDMC08, 195f]

Soll die absolute Position des Sensornetzes oder einzelner Sensorknoten im Raum bekannt sein, so muss mindestens ein Ankerknoten im Netz vorhanden sein. Ansonsten kann man nur Aussgen über die relative Position der Sensorknoten zueinander machen.

Basisstation (Base station) Als Basisstation werden die Sensorknoten bezeichnet, welche das Sensornetzwerk mit einer bestehenden Kommunikationsinfrastruktur wie dem Internet verbinden, so dass ein Benutzer auf die gesammelten Daten zugreifen kann [AKK04]. Sie kann einen festen Standort besitzen oder mobil sein. Es können eine oder mehrere Basisstationen im Netz existieren. In fast allen Netzwerktopologien stellt die Basisstation gleichzeitig die Datensenke dar. Wird die Verbindung zu bestehenden Netzwerken von einem herkömmlichen PC mit Funknetzwerkkarte realisiert, so spricht man nicht von einer Basisstation sondern von einem Gateway. Basisstationen sind in diesem Fall alle direkt an den Gateway sendenden Sensorknoten.

Lebensdauer (Lifetime) Die Lebensdauer ist die Zeitspanne, während derer der Sensorknoten nach dem Einschalten seine Aufgabe erfüllt. Die Lebensdauer ist in der Regel durch die Energiereserve begrenzt. In seltenen Fällen ist der Datenspeicher der limitierende Faktor.

Arbeitszyklus (Duty cycle) Um beim Funkchip Energie zu sparen wird bei der Kommunikation ein Arbeitszyklus aus Schlafphase und Aktivphase eingeführt, der immer wieder periodisch wiederholt wird. In der Schlafphase ist der Funkchip ausgeschaltet, und in der Aktivphase ist er im Hochleistungsmodus. Solange keine Daten gesendet oder empfangen werden, entspricht die Länge der Aktivphase einem bis zwei Prozent der Länge des Arbeitszyklus.

Empfangssignalstärkenangabe (Received signal strength indication, kurz RSSI) Bei der Funkkommunikation gibt die Empfangssignalstärkenangabe Auskunft über die Stärke des empfangenen Funksignals. Sie wird häufig zur Entfernungsmessung zwischen Sensorknoten eingesetzt. In der Theorie nimmt die Empfangssignalstärke mit der Entfernung zum Sender im Quadrat ab. In der Praxis tritt bei einer Messung der Empfangssignalstärkenangabe ein Rauschen auf, das die

Positionsbestimmung in der Größenordnung von mehreren Metern ungenau werden lässt [Sto05, Seite 279f]. Die Empfangssignalstärke wird gemessen in Dezibel pro Milliwatt (dBm) [Bou09, Seite 312].

Aufstellung (Deployment) Das Ausbringen der Sensorknoten in die Umwelt wird als Aufstellen bezeichnet. Die Position der Sensorknoten im Raum hat Auswirkungen auf die Netzabdeckung, die Kommunikationskosten und das Ressourcenmanagment [CI05, Seite 19]. Es wird zwischen zwei Arten des Aufstellens unterschieden. Beim zufälligen Aufstellen (random deployment) kann die Lage der Sensorknoten ohne eine Positionsbstimmung nur grob geschätzt werden. Das ist z. B. der Fall, wenn sie von einem Flugzeug aus abgeworfen wurden. Beim deterministischen Aufstellen (deterministic deployment) ist zumindest dem Menschen die genaue Lage des Sensors bekannt. Die Sensorknoten werden einzeln genau an einer definierten Stelle aufgestellt [CI05, Seite 6f].

3.2 Ausgewählte Anwendungsbeispiele

Sensornetze werden in Zukunft einen immer größeren Stellenwert in unserer Gesellschaft einnehmen. Das Fachgebiet der Sensornetzwerkforschung ist eine sehr junge Disziplin, die Zahl der Fachaufsätze zu diesem Thema begann erst um das Jahr 2002 allmählich anzusteigen. Diese Entwicklung ist der kleiner und billiger werdenden Hardware insbesondere bei der Funktechnologie zu verdanken [ASSC02].
Mögliche Einsatzszenarien von Sensornetzwerken sind bei der Umweltüberwachung, im Gesundheitswesen, bei der Standortverfolgung von Tieren, in der Unterhaltungselektronik, bei stimmungsbasierten Anwendungen, im Transport und der Logistik, in Heim und Büro sowie in der Industrie zu finden [VDMC08, Seite 16].
Im Folgenden werden einige Anwendungen näher skizziert, um die Relevanz von Sensornetzwerken in unserer Gesellschaft herauszustellen.

Feuermeldesystem für Wälder Viele Regionen der Erde wie Spanien, Griechenland, Kalifornien oder Australien haben jedes Jahr mit schweren Waldbränden zu kämpfen. Es ist wichtig, die Brände so schnell wie möglich zu erkennen; je früher man den Brand entdeckt, umso kleiner und damit einfacher ist es, ihn zu löschen [VDMC08, Seite 17].

Ein noch nicht sehr weit entwickeltes Sensornetzwerk zur Erkennung von Waldbränden wurde von Conçalves et al. veröffentlicht. Die ausgebrachten Sensoren messen die Temperatur, die relative Luftfeuchte und Lichtwerte [SGS+06]. Die Daten werden an eine Basisstation weitergeleitet, wo sie weiterverarbeitet, gespeichert und zusätzlich per GSM an einen Server geschickt werden. Mit Hilfe einer eigens entwickelten Software namens *MonSense* können die Daten am Server oder per Laptop direkt an der Basisstation eingesehen werden. In ihrem letzten Feldeinsatz wurden 32 Sensorknoten auf einer Fläche von 20 Hektar ausgebracht, was bei einer gleichmäßigen Verteilung ca. einem Sensor pro 6250 m² entspricht. Mit den gegebenen Sensoren und der eingesetzten Software kann ein Waldbrand jedoch nur bei einem großen Feuer in unmittelbarer Nähe zum Sensor detektiert werden. Das Experiment hat nach Aussage der Autoren eher einen Forschungscharakter zur Ermittlung einer optimalen Sensorknotenverteilung und dient zur Untersuchung von Routingprotokollen.

Ein fortgeschritteneres und bereits im aktiven Ensatz befindliches System mit dem Namen *Fire Weather Net* (*FireWxNet*) wurde von den Autoren Hartung et al. vorgestellt [HHSH06]. Es hat nicht die Aufgabe, Feuer zu erkennen, sondern in einem Brandfall die Feuerwehr mit lokalen Wetterinformationen zu versorgen. Hiermit können der weitere Verlauf und die Ausbreitungsrichtung des Feuers bestimmt werden, was dann zur Einsatzplanung herangezogen wird. Auch ist eine bessere Brandgefahrabschätzung von noch nicht betroffenen Gebieten möglich. Ein wichtiges Designkriterium war daher, das Sensornetzwerk besonders schnell und einfach aufstellen zu können. Des Weiteren muss das Sensornetzwerk mit großen Höhenunterschieden zurechtkommen, um auf Brände Einfluss nehmende Inversionswetterlagen zu erkennen. Eine Inversionswetterlage bezeichnet eine entgegen der

Regel mit zunehmender Höhe steigende Temperatur. Das Sensornetzwerk besteht aus drei einzelnen Sensornetzwerken mit insgesamt 13 Sensorknoten. Hinzu kommen fünf Knoten mit Langreichweitenrichtfunk (maximal 50 km) sowie WLAN, wovon zwei mit Webcams ausgestattet sind. Die Basisstation besitzt zudem eine Satellitenschüssel, die eine Verbindung zum Internet herstellt. Die drei Sensornetze werden in Gebieten mit hoher Brandgefährdung ausgebracht. Sie messen Temperatur, relative Luftfeuchte, Windgeschwindigkeit sowie Windrichtung und senden die Daten alle 15 Minuten über die Richtfunkknoten zur Basisstation. Die Sensorknoten werden von zwei AA-Batterien mit Energie versorgt und besitzen eine Lebensdauer von vier Monaten. Die Richtfunkknoten sind mit Solarpanels ausgestattet.

Strukturelle Überwachung Mit Sensornetzwerken kann die strukturelle Integrität von Gebäuden, Schiffen, Flugzeugen, Brücken u. s. w. überwacht werden.

Ein Prototyp eines Sensornetzwerks zur Messung von Schwingungen in Gebäuden wurde von Whang et al. entwickelt [WXR+04]. Dieses soll, sobald es das Prototypenstadium verlassen hat, im *Four-Seasons-Gebäude* in Sherman Oaks (Kalifornien) aufgestellt werden. Es misst dann an 120 Stellen die auftretenden Vibrationen und mit einem Dehnungsstreifen an 96 Stellen die Verschiebung. Ein Erdbeben schädigte das Four Seasons 1994 so stark, dass es für den Abriss vorgesehen ist. Aus diesem Grund können an das leerstehende Gebäude erzwungene harmonische und seismische Schwingungen angelegt werden. Die vom Sensornetzwerk erhobenen Daten bilden die Grundlage für eine Erforschung der Auswirkung sowie Ausbreitung von Schwingungen in Gebäuden. Statiker können die Informationen zur Konstruktion zukünftiger Gebäude heranziehen. Schwingungen treten in Gebäuden nicht nur durch Wind, Erdbeben oder terroristische Anschläge auf, sondern auch kleinste Schwingungen, verursacht durch Aufzüge oder Menschen im Treppenhaus, sind von Belang.

Brücken müssen in regelmäßigen Abständen mit direkten Methoden wie Sichtinspektionen und Röntgenabtastungen und indirekten Methoden – hierzu zählen Veränderungen im Verhalten und der strukturellen Eigenschaften – auf Schäden untersucht werden. Sukun et al. haben ein bereits im produktiven Einsatz befindliches Sensornetzwerk zur indirekten Überwachung der Integrität an der Golden Gate Bridge angebracht [KPC+07]. Dieses ist zum einen zur kontinuierlichen Überwachung und zum anderen zur Katastrophenwarnung bei Erdbeben und Explosionen ausgelegt. Es misst mit 64 Sensorknoten auftretende Vibrationen mit einer Auflösung von 30µG. Hierbei werden pro Sekunde 1000 Werte erfasst und an eine Basisstation mit Internetanschluss gesendet. Besondere herausforderung bei dieser Installation ist die längliche Anordnung des Sensornetzwerks entlang der Brücke. Es musste ein Netzwerkprotokoll entwickelt werden, welches besonders gut bei die Multihop-Anforderung umsetzt. Weiter musste die salzige und feuchte Umgebung in Meeresnähe bei der Hardwareauswahl bedacht werden. Gegenüber dem vorher eingesetzten konventionellen System mit verkabelten Sensoren ist das Sensornetzwerk mit 600$ pro Knoten gegenüber mehreren tausend Dollar pro Messpunkt bei gleicher Datenqualität um einiges günstiger.

Vulkanüberwachung In einem Gemeinschaftsprojekt der Universitäten Harvard, North California, New Hampshire und des Instituto Geofisico ist ein Sensornetzwerk zur Überwachung der Aktivität von Vulkanen entstanden [WALW+06][WALJ+06]. Es kann sowohl als Frühwarnsystem für Vulkanausbrüche als auch zur Erhebung wissenschaftlicher Daten eingesetzt werden. In einem ersten Feldversuch wurde es 19 Tage am Vulkan Reventador in Ecuador ausgebracht und detektierte 229 interessante Ereignisse (Erdbeben, Erschütterungen und Eruptionen). Das Sensornetzwerk besteht aus 16 Sensorknoten, ausgestattet mit Seismometer und Mikrofon. Die Basisstation ist mit einem Langreichweitenfunkmodem verbunden, welches die gesammelten seismischen und akustischen Daten an einen Laptop im nahegelegenen Observationszentrum schickt. Ein GPS-Empfänger ist an einen Sensorknoten angeschlossen und dient der Zeitsynchronisation als Referenzzeitgeber. Die Sensorknoten werden von D-Batterien mit Energie versorgt. Aufgrund der Datenerfassungsrate von 100 Werten pro Sekunde beträgt die Lebensdauer jedoch nur rund eine Woche. Auch können so hohe Datenauf-

kommen nicht über die hier verwendete Funkschnittstelle versendet werden. Sie werden zunächst in einem Ringpuffer gespeichert. Ein in den Sensorknoten implementierter Ereignisdetektor informiert den Laptop über signifikante Abweichungen vom Durchschnitt. Melden dem Laptop mehrere Sensorknoten ein Ereignis, so fordert er von allen Sensorknoten die Daten des betreffenden Zeitraums an.

Patientenüberwachung Die kontinuierliche Überwachung der Vitalfunktionen von Patienten ist ein Schlüsselprozess in Krankenhäusern und findet heutzutage mit kabelgebundenen Sensoren statt, die an ein Gerät neben dem Bett angeschlossen sind. Nachteilig ist hierbei jedoch die Begrenzung der Sensoren durch die Eingänge am Gerät und die immer neu vorzunehmende Verkabelung des Patienten, wenn er andere Untersuchungsräume aufsuchen muss oder das Bett aus eigener Motivation verlassen möchte. Aus diesen Gründen haben Baldus et al. ein kabelloses Sensornetzwerk zur Patientenüberwachung entworfen [BKM04]. Das System besteht aus drei verschiedenen Geräten. Medizinische Sensorknoten werden direkt am Patienten angebracht und messen Größen wie z. B. den Blutdruck, die Sauerstoffkonzentration des Blutes und die Herzspannungskurve. Ebenfalls direkt am Patienten angebracht wird der Patientenidentifikationssensorknoten, welcher als Controller für die medizinischen Sensorknoten dient und den Patienten eindeutig identifiziert. Das dritte Gerät ist ein Setupstift, mit dem die zuvor beschriebenen Geräte nach der Anbringung am Patienten initalisiert werden. Das Sensornetz sendet seine Daten an einen neben dem Bett stehenden Monitor, wobei das System zuverlässig unter Echtzeitanforderung arbeitet.

Kostendruck im Gesundheitswesen zwingt Krankenhäuser zur Reduktion des Pflegepersonals. Insbesondere bei Nachtschichten, bei denen bereits heute nur noch eine Krankenpflegefachkraft für eine große Anzahl an Patienten zuständig ist, könnte das Personal durch eine Erweiterung des vorgestellten Systems durch eine Alarmfunktion entlastet werden. Fügt man den Patientenidentifikationssensorknoten eine Positionbestimmung hinzu und installiert im Krankenhaus eine Kommunikationsinfrastruktur, so können ansonsten auf keine anderen medizinischen Geräte angewiesene Patienten sich frei im Krankenhaus bewegen. Werden bendenkliche Vitalwerte gemessen, wird das Pflegepersonal über den Aufenthaltsort und die Art des Notfalls informiert [VDMC08, Seite 20].

Eine weitere Anwendung zur Patientenüberwachung kann in einem Altenheim zum Einsatz kommen. Senile oder demente Kunden können mit einm Sensornetzwerk in ihrem Verhalten überwacht werden. Sie erlangen durch diese neue Technologie ein großes Stück an Freiheit wieder, indem das Pflgepersonal weniger Einschränkungen bezüglich ihres Bewegungsfreiraumes machen muss und eine vermindete Aktivitätsüberwachung stattfinden kann.

3.3 Die verwendeten Sensorknoten

Von der Abteilung *Systemsoftware und verteilte Systeme* wurden insgesamt 130 der in Abbildung 3.3 zu sehenden Sensorknoten des Typs MTM-CM5000-MSP der koreanischen Firma Maxfor angeschafft. Diese Knoten sind kompatibel zu der bekannten Sensorplattform TelosB der Firma Crossbow, welche sich bereits als einer von mehreren großen Anbietern am Markt etabliert hat. Daher laufen auf dem MTM-CM5000-MSP-Sensorknoten bereits mehrere Betriebssysteme und freie Implementierungen von Treibern für die Sensorik bis hin zu Routingprotokollen.
Damit ist die Wahl der Sensorplattform für diese Arbeit von der Abteilung bereits abgenommen worden. Weil das Sensornetz in Gebäuden mit humanverträglichen klimatischen Bedingungen zum Einsatz kommt, entfällt eine in der Literatur beschriebene Auswahl der Hardware aufgrund der Umgebungsbedingungen ohnehin [ASSC02]. Das hier vorgestellte System wird so entworfen, dass es die gegebene Sensorplattform mit ihren Hardwareressourcen als Ausgangsbasis berücksichtigt.
In Abbildung 3.4 ist die Architektur dieser Sensorknoten zu sehen. Zentraler Bestandteil ist der Msp430-Mikrocontroller von Texas Instruments. Dieser wird mit 16 MHz getaktet und ist die zentrale Rechen-

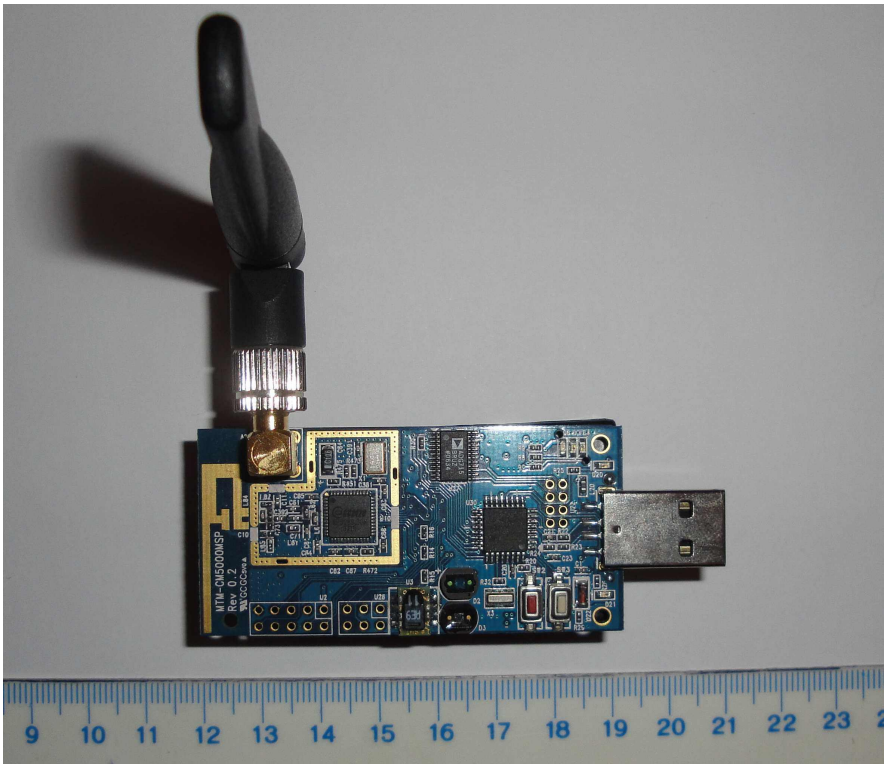


Abbildung 3.3: Der MTM-CM5000-MSP von Maxfor

einheit mit Arbeits- und Programmspeicher. Er besitzt zudem eine serielle Schnittstelle, die mithilfe eines externen USB-Chips die Kommunikation mit einem PC erlaubt. Des Weiteren hat der Msp430 einen internen Spannungssensor und einen A/D-Wandler mit acht Kanälen und zwölf Bit Auflösung. An den A/D-Wandler sind zwei Lichtsensoren angeschossen: Der eine misst die totale Lichteinstrahlung, der andere die photosynthetisch aktive Lichteinstrahlung. Über den I²C-Bus ist ein kombinierter Temperatur- und Luftfeuchtigkeitssensor angeschlossen. Eine weitere wichtige Komponente ist der per SPI-Bus angebundene Funkchip CC2420, welcher die Kommunikation zwischen den Sensorknoten realisiert. Außerdem ist als Hintergrundspeicher ein 1-MB-Flashmodul verbaut, und zur Anzeige stehen drei farbige LEDs zur Verfügung. Betrieben wird das Sensorboard mit einer Stromquelle auf Basis von zwei AA-Batterien [Firc]. In Tabelle 3.1 ist eine detallierte Übersicht über die technischen Daten der verwendeten Sensorknoten gegeben.

3.4 Ressourcenbeschränkungen und resultierende Anforderungen an Betriebssysteme und Anwendungen bei Sensornetzwerken

Sensorknoten besitzen im Gegensatz zu herkömmlichen PCs nur sehr beschränkte Fähigkeiten in Bezug auf Rechenleistung, Speicherausstattung und Bandbreite. Laut dem Moore'schen Gesetz verdoppelt sich die Anzahl der Transistoren auf einer konstanten und kosteneffektiven Chipfläche ca. alle 18 Monate. Dies führt bei PCs in der Regel zu einer Verdoppelung der Speicherkapazität und der Rechenleistung. Bei Sensorknoten hingegen erwarten die meisten Forscher, dass diese bei gleichbleibender Leistung kleiner und kostengünstiger werden [CES04]. Hiervon ausgehend wird man in den kommenden Jahren mit den derzeit angebotenen Ressourcen auskommen müssen.
Die beschränkten Ressourcen müssen aus diesem Grund bei Entwurf und Implementierung von Betriebssystemen und Anwendungen für Sensorkonten bedacht werden. Reddy et al. [RKJK03] und Stojmenovic

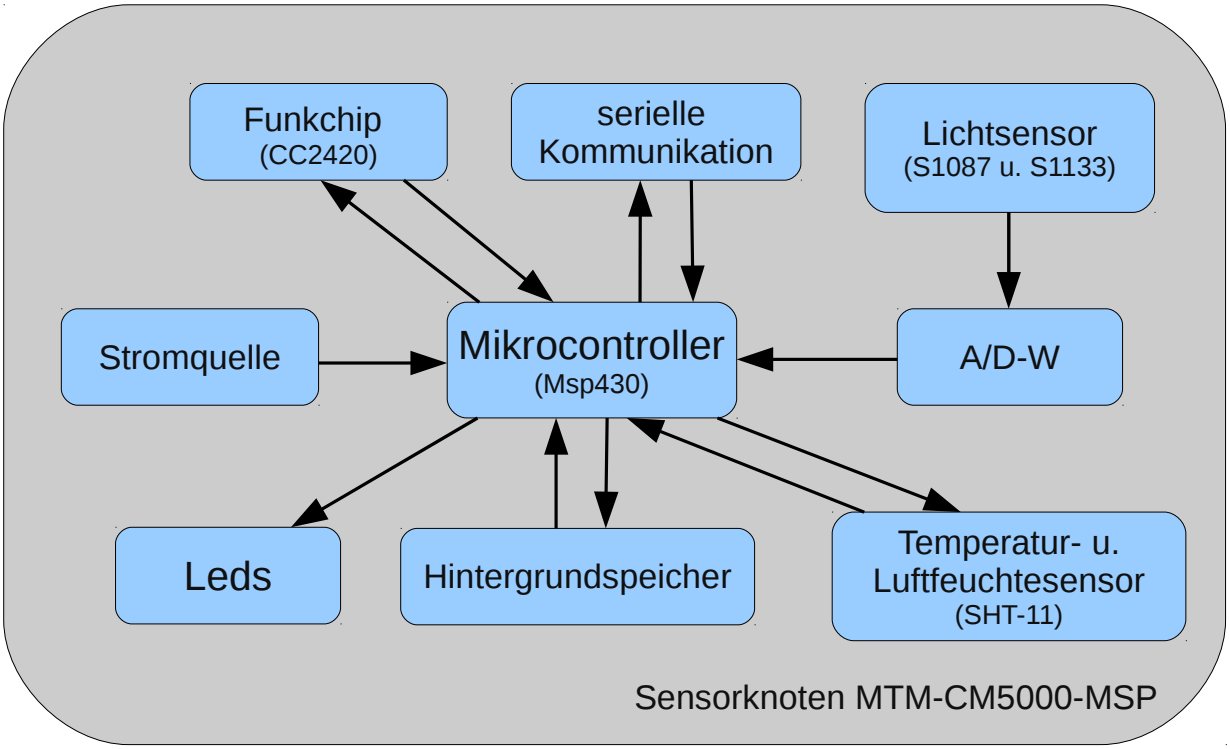


Abbildung 3.4: Die Architektur des MTM-CM5000-MSP

Tabelle 3.1: Die technischen Daten des MTM-CM5000-MSP [Firc]

Mikrocontroller	TI MSP430F1611 @$16MHz$
Programmspeicher	$48kB$
Arbeitsspeicher	$10kB$
Hintergrundspeicher	$1MB$
Stromaufnahme im Aktivmodus	$1,8mA$
Stromaufnahme im Schlafmodus	$5,1\mu$ A
Lichtsensor Hamamatsu S1087	$600nm$ peak
Temperatursensor Sensirion SHT11	-40°C - $123,8$°C, Auflösung: $0,01$°C, Genauigkeit: ±$0,4$°C
Luftfeuchtesensor Sensirion SHT11	0 - 100 %RH, Auflösung $0,03$ %RH, Genauigkeit: ±$3,0$ %RH
Analog-Digital-Wandler	8 Kanäle 12 Bit Auflösung
Funkchip	TI CC2420
Übertragungsrate	$250kb/s$
Reichweite	150m outdoor, 20 bis $30m$ in Gebäuden
Stromaufnahme	RX: $18,8mA$, TX: $17,4mA$, Schlafmodus: 1μA

[Sto05, Seite 177ff] beschreiben die im Folgenden genannten Einschränkungen und die hieraus resultierenden Anforderungen an Betriebssysteme. Die Grenzen zwischen Betriebssystem, API und Programm sind nicht so differenziert zu erkennen wie bei herkömmlichen PCs. Das Betriebssystem und die API stellen oft nur ein Framework für die Anwendung da, welche häufig stark mit diesen verzahnt ist. Daher gelten für Anwendungen dieselben Prinzipien wie für Betriebssysteme.

Rechenleistung Sensorknoten haben eine geringe Rechenleistung von wenigen MIPS. Rechenintensive Operationen sollten daher vermieden oder möglichst performant implementiert werden. Da nicht alle Betriebssysteme preemptives Scheduling unterstützen (vergleiche 3.5.1), sollte man lange laufende Tasks vermeiden oder in mehrere kleinere Aufgaben unterteilen. Dies gilt auch für die ereignisgetriebenen Betriebssysteme, welche ein Ereignis in der Regel bis zum Ende ausführen und in der Zwischenzeit nicht auf neue Ereignisse mit womöglich höherer Priorität reagieren können.

Speicher Neben der Energie ist der Speicher eine der größten Beschränkungen bei Sensorknoten. Die heute am Markt erhältlichen Sensorknoten besitzen typischerweise einen Programm- und Arbeitsspeicher zwischen 4 KB und 4 MB sowie Hintergrundspeicher in der Größenordnung von wenigen MB. Hier ist zu beachten, dass sowohl das Betriebssystem, die eigentliche Anwendung und unter Umständen eine Middleware in den Programmspeicher passen muss. Einige Betriebssysteme bieten zwar auch ein Nachladen von Modulen aus dem Hintergrundspeicher an, es ist aber dennoch eine optimale Speichrausnutzung Voraussetzung.

Energieverwaltung Sensorknoten werden in der Mehrzahl von Batterien mit Energie versorgt, die dann der begrenzende Faktor für die Lebensdauer sind. Es gibt mehrere Möglichkeiten, die Energiereserven zu schonen. Naheliegend ist es, Komponenten wie den Funkchip, den externen Flashspeicher oder den USB-Chip bei Nichtbenutzung auszuschalten. Eine andere Möglichkeit bieten die Schladmodi der Mikrocontroller. Ist derzeit keine Rechenarbeit zu erledigen, kann er vom Hochleistungsmodus in einen von mehreren Schlafmodi wechseln. Im Tiefschlaf sind lediglich noch die externen Interrupts aktiviert. Die letzte Möglichkeit, die hier genannt werden soll, ist der Programmierstil. Ein Zustandswechsel der oben genannten Komponenten ist meist relativ energieintensiv und sollte besser vermieden werden. Aufgaben können oftmals auch angesammelt und dann direkt nacheinander ausgeführt werden.

Stromsparmechanismen werden in zwei Kategorien unterteilt. Die impliziten Mechanismen funktionieren autark durch das Betriebssystem ohne eine Interaktion durch die Anwendung, wohingegen die expliziten Mechanismen mit einem Funktionsaufruf von der Anwendung initiiert werden. Je nach Betriebssystem sind die oben genannten Mechanismen zum Energiesparen implizit oder explizit.

Portabilität Im Gegensatz zu herkömmlichen PCs, die eine recht einheitliche Hardwarearchitektur mit nur wenigen verschiedenen Befehlssätzen besitzen, handelt es sich bei Sensorknoten meist um hoch angepasste Systeme. Zum einen ist das Gebiet der Sensorknoten noch recht jung und entwickelt sich fast täglich weiter. Zum anderen gibt es eine Fülle an verschiedenen Plattformen mit verschiedenen Mikrocontrollern, Sensoren, Speicherbausteinen, Funklösungen und weiteren Komponenten. Im Bereich der eingebetteten Systeme ist es auch nicht unüblich, für eine neue Anwendung eine neue Hardwareplattform zu entwickeln, um diese bestmöglich an die Aufgabe anzupassen. Die Portierung auf ein neues System wird enorm erleichtert, wenn der hardwareabhängige Programmcode in eigenen Dateien gekapselt wird und ein Mechanismus zum Austausch einzelner Komponenten vorgesehen ist.

Anpassbarkeit Anwendungen für Sensornetzwerke verteilen sich über verschiedenste Disziplinen. Dies reicht vom Messen von Umweltparametern, Steuern von Aktoren bis hin zur Zielverfolgung und anderem. Ein Betriebssystem muss für verschiedene Anwendungen auch verschiedene Anforderungen erfüllen. Diese könnten z. B. Echtzeitfähigkeiten, Neuprogrammierung im laufenden Betrieb oder ein nur unregelmäßiger Kontakt zur Basisstation sein.

Parallele Abläufe Zu einem Zeitpunkt können in einem Betriebssystem oder in der Anwendung mehrere Aufgaben auflaufen. Als Beispiel können die Erfassung neuer Sensorwerte und die Bearbeitung der letzten erfassten Sensorwerte genannt werden. Es kann auch zu jeder Zeit ein neues Datenpaket über Funk oder serielle Schnittstellen eintreffen. Aufgaben werden durch Threads, Tasks oder Ereignisse repräsentiert, welche oftmals auch auf denselben Datenstukturen agieren. Um bei derartigen Datenzugriffen keine Fehler entstehen zu lassen, muss das Betriebssystem Mechanismen zur Behandlung konkurrierender Operationen anbieten. Auch in der Welt der Sensornetzwerke sind Semaphore bekannt. Die meistgenutze Variante ist jedoch ein wechselseitiger Ausschluss, Mutex genannt, welcher auf Ausschalten der Interrupts beruht.

Ein weiterer wichtiger Punkt ist die beschränkte physikalische Fähigkeit von Mikrocontrollern bei parallelen Abläufen. Zum einen besitzt die Mehrzahl aller Mikrocontroller lediglich einen Kern. Zum anderen handelt es sich meist um RISC-Architekturen – der MSP430 besitzt 27 Grundinstruktionen und 24 emulierte, das heißt aus Grundinstruktionen zusammengesetzte Instruktionen. Einen Task zu posten und der Umgebungswechsel bei Threads ist somit auf Mikrocontrollern recht teuer. Aufgrund der Run-to-completion-Semantik von Tasks muss daher zwischen Ausführungsdauer mit hierbei ausgeschalteten Interrupts und der Anzahl der Taskwechsel abgewogen werden.

Netzwerk Die Kommunikation mit anderen Knoten oder mit einem PC ist eine wichtige Funktion von Sensorknoten. Anwendungen für Sensornetzwerke sehen so gut wie nie eine Benutzerinteraktion direkt an den Sensorknoten vor; nach dem Aufstellen operieren sie autark. Meist besitzen diese auch nur wenige LEDs zur Ausgabe und selten Knöpfe zur Eingabe. So stellt das Netzwerk die einzige Schnittstelle zum Menschen dar.

Obwohl die typische Paketgröße bei Sensornetzwerken nur ca. 32 Bytes beträgt, ist dies in Anbetracht der geringen Speicherressourcen von Mikrocontrollen nicht wenig. Netzwerkstapelimplementierungen mit Cross-Layer-Schnittstellen verhindern häufiges Kopieren von Speicherbereichen.

Zur Kommunikation mit anderen Sensorknoten kommt weitgehend Funk zum Einsatz, was energetisch die teuerste Option ist. Daher ist das Netzwerkprotokoll, speziell das eingesetzte Routingverfahren, stark an die Aufgabe der Anwendung angepasst. Außerdem sollte der Datenverkehr soweit wie möglich reduziert werden. Die Bandbreite des Funkkanals bei Sensornetzwerken ist stark begrenzt und liegt in der Regel zwischen 39 kb/s (CC1000) und 256 kb/s (CC2420, ZigBee). Dies muss auch bei dem Betriebssystem mitbedacht werden, wenn es z. B. um die Synchronisation zwischen mehreren Knoten geht.

Die Netzwerkimplementierung ist erfahrungsgemäß, auch in dem in dieser Arbeit vorgestelltem System, der Hauptanteil der Entwicklung. Eine detallierte Betrachtung der Netzwerkfunktionalität findet in Abschnitt 3.6 statt.

Sensorik Wie der Name Sensornetzwerk bereits impliziert, ist die Anbindung von Sensoren an die Sensorknoten essentiell. Moderne Sensoren werden zumeist über ein Bussystem wie SPI oder I²C angebunden. Diese Busse verhelfen zwar zu einem einheitlichen Zugriff auf die Sensoren bezüglich Timing u. s. w., jedoch verlangen die Sensoren oft spezifische Aufwärm- und Anfragesequenzen. Einen Treiber für Sensoren zu schreiben ist nicht trivial und verzögert die Entwicklung von Anwendungen. Darüber hinaus gibt es gegenüber der PC-Welt mit Maus und Tastatur als primären Eingabegeräten eine Fülle an verschiedenen Sensoren, um alle möglichen Umweltparameter zu erfassen. Zu guten Betriebssystemen gehört mindestens eine einfache Schnittstelle zum Einbinden neuer Sensorik; weitgehend bieten sie bereits Treiber für die gängigsten Sensoren an.

Dateisystem Es ist nicht immer möglich, die erfassten Daten sofort an die Senke zu schicken. Gründe hierfür sind Hardwarefeher, Umweltbedingungen oder Paketkollisionen auf dem Funkkanal. Sensorknoten müssen in der Lage sein, über einen längeren Zeitraum Daten zu sammeln und lokal zu speichern. Daher besitzen sie einen mehrere Megabyte großen Hintergrundspeicher zur Ergänzung

des sehr knappen Arbeitsspeichers. Ein Betriebssystem sollte eine Abstraktionsschicht zum einfachen, effizienten und verlässlichen Zugriff darauf für Anwendungen bereitstellen.

Ein zirkulärer Puffer erfüllt die Anforderungen simpler Programme; komplexere Anwendungen erfordern jedoch ein an die Ressourcenbeschränkungen in Sensornetzen angepasstes Dateisystem. Es gibt drei wesentliche Anwendungen an Dateisysteme für Sensorknoten. Diese sind das eher seltene Auslesen und Schreiben von Binärcodeabblidern sowie Konfigurationsparametern wie Abtastraten und Kalibrierungswerten. Der häufigste Anwendungsfall ist das sequentielle Schreiben und Lesen von Sensordaten. Zufallszugriffe sind hingegen in der Welt der Sensornetzwerke recht selten. In fast allen Fällen kommen auf Sensorknoten Flashspeicher zur persistenten Speicherung zum Einsatz. Es ist zu beachten, dass diese nur eine limitierte Anzahl von ca. 10.000 Schreiboperationen pro Seite vertragen. Zudem ist ein Schreibvorgang energetisch teuer, da eine Seite zunächst in den Arbeitsspeicher geladen, dort modifiziert und anschließend zurückgeschrieben werden muss.

Ein Beispiel für ein speziell an die oben genannten Anforderungen angepasstes Dateisystem ist das Extremely Low-Frequency (ELF) Flash File System.

Instandhaltung Sensornetzwerke bestehen gewöhnlich aus mehreren hundert Sensorknoten. Diese sind unter Umständen über eine große Fläche verteilt oder nicht mehr physikalisch erreichbar, z. B. nach einer Aufstellung per Flugzeug. Anwendung und Betriebssystem sollten eine Möglichkeit zur Fernwartung anbieten. Auch eine Möglichkeit zum Austausch des Betriebssystems und der Anwendung ist eine oft anzutreffende Anforderung.

3.5 Betriebssysteme für Sensorknoten

Bei vielen eingebetteten Systemen für ein einziges Anwendungsgebiet, wie DVD-Player oder Mikrowellen, kommt auch heute noch kein Betriebssystem zum Einsatz. Mehrzweckgeräte wie PDAs oder Handys besitzen meist eine speziell angepasste Variante der bekannten PC-Betriebssysteme wie z. B. Windows Mobile oder Embedded Linux. Sensorknoten befinden sich zwischen diesen beiden Extremen. Sie besitzen zwar meist eine spezifische Aufgabe, jedoch kommt auch hier in jüngster Vergangenheit oftmals ein Betriebssystem zum Einsatz [Sto05, Seite 174f].
Aufgabe eines Betriebssystems ist es, die systemnahen Funktionen von der Anwendung durch einheitliche Schnittstellen zu abstrahieren. Dies beinhaltet unter anderem eine Prozessor- und Speicherverwaltung, Multithreading, Multitasking, Scheduling und Zugriff auf die Peripherie, wie Funk, serielle Kommunikation sowie Ein- und Ausgabe [Tan92, Seite 1ff]. Hinzu kommt ein Mechanismus zur Koordination von parallelen Abläufen und, besonders wichtig bei Sensorknoten, eine Energieverwaltung.
In den folgenden Unterabschitten 3.5.1 bis 3.5.4 wird zunächst ein Vergleich zwischen erhältlichen Betriebssystemen für Sensorknoten durchgeführt, gefolgt von einer genaueren Betrachtung der Betriebssysteme TinyOS, MANTIS OS und Contiki. Die ersten beiden sind die am meisten eingesetzten Betriebssysteme in Sensornetzwerken, und Contiki wird neben TinyOS an der Universität Oldenburg und dem An-Institut OFFIS häufig benutzt.

3.5.1 Vergleich erhältlicher Betriebssysteme

Es existieren verschiedene Sensorknoten, speziell angepasst auf unterschiedliche Anwendungsfälle. Die in Abschnitt 3.4 beschriebenen Ressourcenbeschränkungen führen bei Betriebssystemen für Sensorknoten zu einer Abwägung zwischen Funktionalität und möglichst geringen Hardwareanforderungen, welche je nach Sensorknoten-Anwendungsfall-Kombination anders ausfallen muss. In diesem Abschnitt werden ausgewählte Betriebssysteme klassifiziert und miteinander verglichen.
Die Autoren Reddy et al. [RKJK03] schlagen das in Abbildung 3.5 zu sehende Klassifikationsschema vor. Sie sprechen hierbei von einem Klassifikationsschema anstatt einer Klassifizierung, da mehr als ein Charakteristikum zur Unterscheidung herangezogen wird. Unterschiede sind zu finden in der Architektur,

dem Ausführungsmodell, der Fähigkeit zur Neuprogrammierung, dem Scheduling, der Energieverwaltung, der Portabilität auf neue Hardwareplattformen und der Fähigkeit zur Simulation. Im Folgenden werden die einzelnen Punkte genauer erläutert und Betriebssysteme den jeweiligen Kategorien zugeordnet. Basis hierfür bildet nicht nur das Fachreferat der oben genannten Autoren, sondern auch [Sto05, Seiten 186-195].

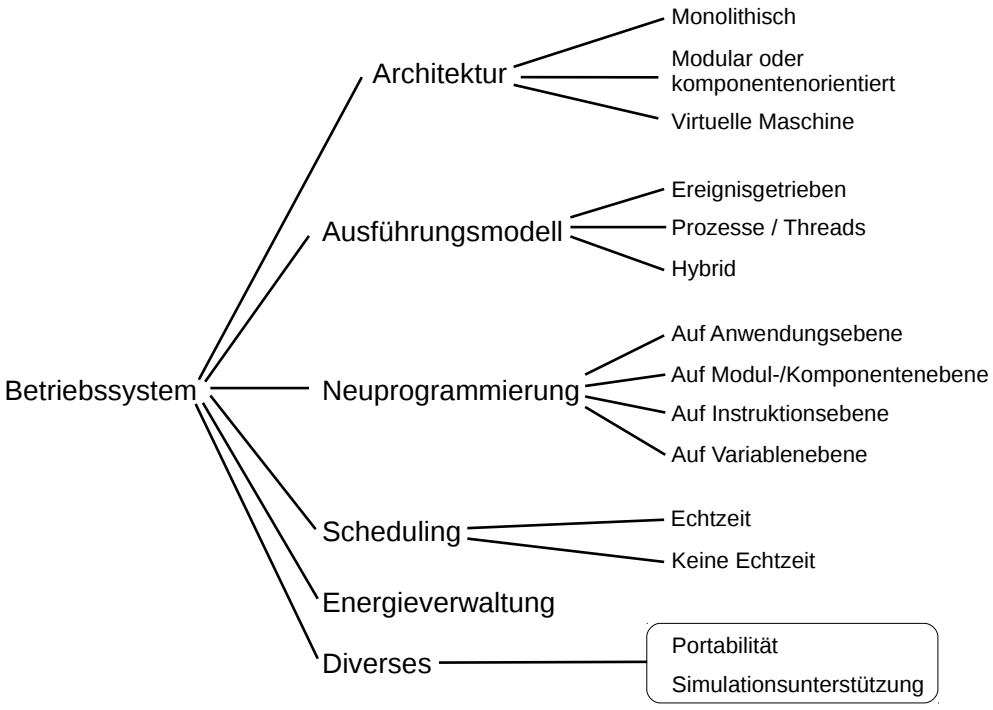


Abbildung 3.5: Das von Reddy et al. [RKJK03] vorgeschlagene Klassifikationsschema für Betriebssysteme

Architektur Die Architektur des Kernels beeinflusst maßgeblich die Art und Weise, wie das Betriebssystem Funktionen anbietet. Hierbei besteht ein Trade-off zwischen Performance und Flexibilität. Es wird zwischen drei Architekturen unterschieden:

1. *Monolithische Betriebssysteme* bestehen aus der Anwendung und den notwendigen Betriebssystemkomponenten. Sie formen ein Systemabbild für genau einen Sensorknoten. Das bringt Nachteile mit sich, wenn zu erwarten ist, dass sich Anforderungen an die Anwendung häufig ändern. Es muss in einem solchn Fall das gesamte Systemabbild neu auf den Sensorknoten geschrieben werden.
2. Bei *modularen Betriebssystemen* bestehen aus der Anwendung und Betriebssystemfunktionen aus einer Menge miteinander interagierender Module. Diese können zur Laufzeit geladen und entladen werden. Werden Änderungen bei der Anwendung erwartet, sollte eine modulare Architektur gewählt werden. Sie vereinfacht die Quellcodewartung und -modifikation. Nachteil modularer Betriebssysteme ist der Overhead zum Finden zusammenhängender Speicherbereiche beim Modulladevorgang. Hinzu kommt eine Adressumrechnung bei Modulen mit Verweisen in andere Speicherbereiche.
3. Eine *virtuelle Maschine* bietet der Anwendung eine Abstraktionsebene für dynamische und statische Komponenten an. Die Anwendung besteht aus Instruktionen für diese virtuelle Maschine, und eine Rekonfiguration ist besonders einfach. Das gesamte Sensornetzwerk wird als eine Entität angesehen, es gibt nur ein für alle Sensorknoten im Netzwerk einheitliches Sys-

temabbild. Virtuelle Maschinen können auf herkömmlichen Betriebssystemen basieren. Matè baut auf TinyOS auf und ContikiVM auf Contiki auf.

In Tabelle 3.2 werden Betriebssysteme der oben vorgetellten Architekturen genannt.

Ausführungsmodell Das Ausführungsmodell hat den größten Einfluss auf die Performanz der Betriebssysteme für Sensornetzwerke. Obwohl die Debatte über das beste Ausführungsmodell für Betriebssysteme noch anhält, besitzt die Mehrheit aller Betriebssysteme ein ereignisgetriebenes Modell. Die mit Abstand zweitgrößte Klasse sind die threadbasierten Betriebssysteme. Eine Zuordnung von Betriebssystemen zu diesen beiden Klassen findet in Tabelle 3.3 statt.

Bei *ereignisbasierten* Ausführungsmodellen wird die Berechnung innerhalb von Eventhandlern ausgeführt. Diese besitzen in den meisten Betriebssystemen eine Run-to-completion-Semantik. Da nur ein einziger Eventhandler zur Zeit laufen kann, gibt es nur einen Stack und somit keinen Overhead bei der Stackverwaltung. Es müssen nicht mehrere Speicherbereiche für verschiedene Stacks freigehalten werden. Hinzu kommt die Möglichkeit, schneller auf neu eintreffende Ereignisse mit höherer Priorität zu reagieren. Bei der Energieverwaltung ist dieses Modell von Vorteil. Wenn keine Aufgaben mehr anstehen, kann es sich schlafen legen, bis neue Ereignisse eintreffen. Aus diesen drei Gründen sind ereignisgetriebene Betriebssysteme zu wählen, wenn besonderer Wert auf die Effizienz gelegt wird. Nachteil ist jedoch eine geringere Fehlertoleranz aufgrund der nicht vorhandenen Speicherisolation von Anwendungen. Auch muss sich der Programmierer selber entscheiden, wann er die Kontrolle wieder an das Betriebssystem abgibt. Ein weiterer Punkt ist die Portierung von C-Quelltext. Dieser muss von der üblichen iterativen Programmierung aufwändig in ereignisgetriebenen Quelltext überführt werden.

Bei einem Ausführungsmodell mit *Threads* wird die Berechnung logisch nach Aufgaben auf mehrere Threads verteilt. Als Vorteile sind hier zu sehen, dass sich der Proggrammierer nicht um die Prozessorkontrolle und das Timing kümmern muss. Des Weiteren besteht eine Isolation zwischen den Speicherbereichen verschiedener Anwendungen, und eine Portierung von C-Quelltext ist leicht zu bewerkstelligen. Wird ein preemptiver Scheduler eingesetzt, so können mehrere Threads quasi gleichzeitig ausgeführt werden. Nachteil ist bei diesem Modell jedoch ein gerade auf Mikrocontrollern nicht unerheblicher Overhead durch den Umgebungswechsel bei einem Threadwechsel. Dieser ist auch für zeitnah zu erledigende Aufgaben wegen der häufigen Unterbrechung hinderlich. Es muss für den Stack eines jeden Threads ein Speicherbereich bereitgehalten werden. Dies ist nicht besonders günstig, weil bei Betriebssystemen für Sensornetzwerke aufgrund der nicht vorhandenen Memory Managment Unit bei Mikrocontrollern keine virtuelle Speicherverwaltung genutzt wird.

Weitere in der Literatur bekannte Ausführungsmodelle sind *zustandsorientiert*, basieren auf *Objekten* oder sind *datenzentriert*. Die zustandsorientierten Modelle verfolgen einen ähnlichen Ansatz wie endliche Automaten. Eine Anwendung besteht aus einer Menge von Zuständen und Antworten auf Ereignisse. Ereignisse führen basierend auf der Eingabe zu einem Zustandsübergang. Eine Neupro-

Tabelle 3.2: Die Architektur verschiedener Betriebssysteme [RKJK03]

Monolithisch	**Modular**	**VM**
TinyOS	SOS	VMSTAR
MagnetOS	Contiki	Matè
	MANTIS OS	MagnetOS
	CORMOS	
	Bertha	
	kOS	

Tabelle 3.3: Das Ausführungsmodell verschiedener Betriebssysteme [RKJK03]

Ereignisbasiert	Threadbasiert	Hybrid	Andere
TinyOS	MANTIS OS	Contiki (Ereignisse + Threads)	SenOS
SOS		kOS (Ereignisse + Objekte)	Nano-RK
CORMOS			
EYES			
PEEROS			

grammierung kann einfach durch einen Austausch der Zustandsübergangstabelle erreicht werden. Als Beispiel für ein zustandsorientiertes Betriebssystem kann SenOS genannt werden. Die objektbasierten und datenzentrierten Modell sind derzeit nicht von Bedeutung und sollen hier daher nicht besprochen werden.

Neuprogrammierung Die Fähigkeit zur Neuprogrammierung über das Netzwerk wird von den Sensornetzwerkforschern als wichtig angesehen, da oftmals die Sensorknoten nach dem Einsetzen in die Umwelt nicht mehr physikalisch erreichbar sind. Als Neuprogrammierung bezeichnet man die Fähigkeit zum Hinzufügen, Löschen und Ändern der installierten Software. Diese wird durch den Einsatz sogenannter Quelltextverteilungprotokolle erreicht. Eine Neuprogrammierung kann je nach Protokoll auf verschiedenen Ebenen möglich sein. Auf Anwendungsebene wird das gesamte Systemabbild ausgetauscht, auf Modulebene können einzelne Module oder Komponenten ausgetauscht werden. Es gibt auch Quelltextverteilungprotokolle, welche das Austauschen einzelner Instruktionen oder Variablen unterstützen. Als Beispiele für Quelltextverteilungsprotokolle sollen hier lediglich die für TinyOS unterstützen Protokolle XNP, MOAP und Deluge erwähnt werden.

Scheduling Anwendungsfälle für Sensornetzwerke können in die Kategorien periodisch, aperiodisch sowie hierbei in zeitkritisch und nicht-zeitkritisch unterteilt werden.

Periodische Anwendungen bestehen aus in definierten Zeitintervallen immer wiederkehrenden Aufgaben, als Beispiel kann das in dieser Arbeit entwickelte System zur Temperaturmessung genannt werden. Bei aperiodischen Anwendungen kann der Abarbeitungszeitpunkt nicht von der Anwendung vorhergesagt werden. Ein Beispiel hierfür ist ein Feuerüberwachungssystem. Alle Betriebssysteme beherrschen sowohl periodische als auch aperiodische Aufgaben.

Sicherheitskritische Anwendungen für Sensornetzwerke, etwa Feuerüberwachungssysteme, setzen eine Echtzeitfähigkeit vorraus. Diese wird im Betriebssystem mithilfe des Schedulers realisiert. Ein Scheduler ist genau dann echtzeitfähig, wenn er die Abarbeitung einer Aufgabe innerhalb einer vordefinierten Zeitspanne erledigt. Eine Übersicht über die Echtzeitfähigkeiten von Betriebssystemen für Sensorknoten wird in Tabelle 3.4 gegeben.

Energieverwaltung Die Energieverwaltung beinhaltet den Zugriff auf und die Kontrolle über alle für den Energieverbrauch wichtigen Komponenten des Sensorknotens. Das Betriebssystem bietet hierzu Energieverwaltungsschnittstellen an. Wichtige Komponenten sind insbesondere der Prozessor, der Funkchip und die Stromversorgung, wobei für letztere im Falle einer Batterie lediglich eine Überwachung angeboten wird. Jedes Betriebssystem hat seine eingene Vorgehensweise bei der Implementierung einer solchen Energieverwaltung. Die Mechanismen versuchen mit verschiedenen Ansätzen bestmöglich zu erkennen, wann eine Kompnente ausgeschaltet werden kann. Bei threadbasierten Betriebssystemen wird hierzu gerne der Scheduler benutzt.

Vergleiche zu diesem Themenbereich auch den Punkt Energieverwaltung im Abschnitt 3.4.

Tabelle 3.4: Echtzeitfähigkeit verschiedener Betriebssysteme [RKJK03]

Echtzeitfähig	Nicht echtzeitfähig
Nano-RK	TinyOS
CORMOS	SOS
PEEROS	Contiki
DCOS	EYES
t-kernel	SenOS
	OSSTAR
	MagnetOS
	kOS
	T2
	MANTIS OS

Portabilität Die Hardwareplattformen von Sensorknoten entwickeln sich noch recht rasch und in vielfältige Richtungen. Viele verschiedene sind bereits heute auf dem Markt erhältlich. Es ist bereits bei der Entwicklung von Betriebssystemen auf eine einfache Portierbarkeit zu achten. Möchte man sich heute für ein Betriebssystem für seinen Sensorknoten entscheiden, so ist eine vorhandene Unterstützung des Betriebssystems dafür von enormem Vorteil. Die in Wikipedia gepflegte Tabelle [Wik10] gibt Auskunft über die unterstützten Knoten von Betriebssystemen für Sensorknoten.

Simulationsunterstützung Eine Simulationsunterstützung ist genau dann gegeben, wenn derselbe Quelltext das gleiche Verhalten in der realen Welt wie in der Simulationsumgebung zeigt. Einige Betriebssysteme bieten eine Simulationsumgebung zum Testen der Anwendung vor dem Aufstellen an. Diese sind namentlich TinyOS, SOS, MANTIS OS und Contiki. Der Umfang der Simulationsumgebung ist nicht bei allen Betriebssystemen gleich.

3.5.2 TinyOS

TinyOS ist ein leichtgewichtiges und speziell an kabellose Sensorknoten mit wenig Leistung angepasstes Betriebssystem [LG09, Seite 5]. Es wurde anfangs von einer Gruppe Wissenschaftler der Universität Berkley um David Culler zu Forschungszwecken entwickelt. Mittlerweile hat es sich als eines der großen Betriebssysteme für Sensornetzwerke etabliert, ist quelloffen, steht unter der BSD-Lizenz und liegt seit dem 06.04.2010 in der stabilen Version 2.1.1 vor. In dieser Arbeit wird jedoch noch die Version 2.x aus dem CVS vom 16.07.2009 verwendet. Die Entwicklung von TinyOS wird heute von der TinyOS Alliance vorangetrieben, der Privatpersonen, Organisationen und Unternehmen aus Forschung und Industrie angehören. Das Entwicklungsmodell stützt sich auf die TinyOS Enhancement Proposals (TEPs), mit denen Neuerungen zunächst vorgschlagen und nach ihrer Aufnahme in TinyOS dokumentiert werden. Arbeitsgruppen können TEPs erstellen und einreichen. Eine Arbeitsgruppe besteht immer aus mindestens drei Menschen, die sich intern treffen und sowohl TEPs als auch Quellcode der Alliance vorschlagen. Spezielle Langzeitgruppen kümmern sich dabei um besonders wichtige Bereiche und Subsysteme, etwa Routing, Management, Plattformenunterstützung, Tests und Tools zum Programmieren [BCE+06].
TinyOS selbst und die Anwendungen für TinyOS werden in nesC geschrieben, das eine speziell für TinyOS entwickelte Spracherweiterung zu C ist. NesC-Quellcode wird zunächst mit dem nesC-Compiler in C-Quellcode überführt und dann mit einem C-Compiler für die jeweilige Sensorknotenplattform in das Binärformat übersetzt. Da TinyOS kein dynamsiches Linken unterstützt, hat der Compiler eine vollständige Sicht von dem Aufrufgraphen der Anwendung und kann den Quellcode optimal für den C-Compiler vorbereiten. Der C-Compiler erhält eine einzige C-Quelldatei und kann neben den üblichen Optimierungen –

z. B. Entfernen von unbenutzten Variablen und Code-Inlining – auch über die Aufrufgrenzen der einzelnen Komponenten hinweg Optimierungen vornehmen [LG09, Seite 12f].
TinyOS ist ereignisbasiert und besteht aus explizit durch Programmanweisungen miteinander verbundenen Komponenten, die jeweils eine bestimmte Funktion anbieten. Komponenten bieten Schnitstellen an und benutzen sie, um miteinander zu interagieren. Der Benutzer einer Schnittstelle stellt eine Anfrage an den Schnittstellenanbieter (call command), und der Anbieter antwortet mit einem Ereignis (signal event). Das Stellen von Anfragen und das Auslösen von Ereignissen verhalten sich hierbei wie Funktionsaufrufe in C. Die Programmierung in nesC wird anhand einer Hallo-Welt-Komponente, die nach dem Einschalten der Sensorknoten eine LED zum Leuchten bringt, im Folgenden erläutert.

```
1  module HalloWeltC
   {
3    uses interface Boot;
     uses interface Leds;
5  }

7  implementation
   {
9    event void Boot.booted()
     {
11     call Leds.led0On();
     }
13 }
```

Listing 3.1: Das Modul eines Hallo-Welt-Programms

Zunächst wird das im Listing 3.1 gegebene Modul erstellt. Ein Modul besteht aus der Moduldefinition, worin die benutzten und angebotenen Schnitstellen angegeben werden, und der Modulimplementierung, worin die Eventhandler und Kommandos der Schnittstellen implementiert werde. Im Fall des Hallo-Welt-Programms wird, sobald die Bootschnittstelle das Ereignis *booted* auslöst, das Kommando zum Einschalten einer LED bei der LED-Schnittstelle abgesetzt. Schnittstellen werden jeweils in einer eigenen Datei definiert, die den gleichen Namen wie die Schnittstelle trägt. Nutzt eine Komponente eine Schnittstelle, so muss sie alle dort definierten Ereignisse implementieren. Bietet eine Komponente eine Schnittstelle an, so müssen analog alle Kommandos implementiert werden. Im Listing 3.2 sind die Schnittstellendefinitionen der für dieses Programm nötigen Schnittstellen zu sehen. In dem hier vorgestellten Beispiel wird eine Schnittstelle zum Kernel benötigt, die das Ereignis *booted* auslöst, sobald das Betriebssystem hochgefahren ist. Hinzu kommt eine Schnittstelle zu der LED-Komponente.

```
1  interface Boot
   {
3    event void booted();
   }
5
   interface Leds
7  {
     command void led0On();
9    command void led0Off();
   }
```

Listing 3.2: Die benutzten Schnittstellen des Hallo-Welt-Programms

Mittels einer Konfiguration werden die Schnittstellen mehrerer Komponenten schlüssig miteinander verbunden. Es ist so eine übergeordnete Komponente entstanden, die mithilfe anderer Komponenten eine Funktion anbietet. In Listing 3.3 ist die Konfiguration der Hallo-Welt-Komponente zu sehen. Im oberen Bereich (Zeile 1) werden die von dieser Komponente benötigten und angebotenen Schnitstellen genannt.

```
configuration HalloWeltAppC {}

implementation
{
  components MainC, LedsC, HalloWeltC;
  MainC.Boot -> HalloWeltC.Boot;
  HalloWeltC.Leds -> LedsC.Leds;
}
```

Listing 3.3: Die Konfiguration eines Hallo-Welt-Programms

Da es sich bei dieser Konfiguration um die Hauptkonfiguration der Anwendung handelt, ist der Teil hier leer. Anschließend werden die benutzten Komponenten aufgeführt und die Schnittstellen miteinander verbunden [LG09, Seite 10ff].
TinyOS besteht aus einer Ansammlung von Komponenten und kann so immer passend für die jeweilige Anwendung zusammengestellt werden. So gibt es neben den Systemkomponenten, wie Scheduler, Timer oder Bootkomponente, für viele Aufgaben die passende Komponente. Dies umfasst zum einen Gerätetreiber für Hardware, wie einem Sensor oder Funkchip, zum anderen aber auch Software, wie ein Routingprotokoll oder Datenfilter.
Der Entwurf von Komponenten für TinyOS kann mittels Komponentendiagrammen verdeutlicht werden. In ihnen ist die Beziehung zwischen den einzelnen Komponenten und Modulen sowie der Schnittstellenverbindungen dazwischen auf einen Blick zu erkennen.

Der TinyOS-Kernel

In TinyOS werden alle Operationen von einem Task oder einer Unterbrechungsbehandlungsroutine durchgeführt. Tasks können zu jeder Zeit gepostet werden. Der TinyOS-Scheduler führt sie dann zu einem späteren Zeitpunkt nacheinander einzeln aus, das heißt, er ist nicht preemptiv und plant nach dem FIFO-Prinzip ein. Im Gegensatz dazu können Unterbrechungen zu jedem Zeitpunkt auftreten. Da sowohl Tasks als auch Unterbrechungsbehandlungsroutinen innerhalb der Module implementiert werden und hier Kommandos oder Ereignisse auslösen können, ist die Ausführung nicht nur auf das Modul beschränkt. Bei der Implementierung von Komponenten muss daher immer auf kritische Abschnitte geachtet werden. Einerseits kann man mit dem Schlüsselwort *async* Kommandos und Ereignisse für Unterbrechungsbehandlungsroutinen freigeben; ist das Schlüsselwort nicht gesetzt, meldet der Compiler einen Fehler [LG09, Seite 71]. Andererseits kann man einzelne Blöcke des Quellcodes mit dem Schlüsselwort *atom* als atomar definieren. Dann werden zur Ausführung die Interrupts ausgeschaltet.
Tasks in TinyOS besitzen keinen Rückgabewert und nehmen keine Parameter entgegen. Zudem kann ein Task nur genau dann gepostet werden, wenn kein gleicher Task bereits gepostet, aber noch nicht gestartet wurde. Somit kann ein Modul einen Task maximal einmal instantiieren, ein Task kann sich aber selber erneut posten [LS]. Jeder Task besitzt einen eigenen Stack, da aber nur ein Task zur selben Zeit laufen kann, kommt TinyOS mit einem Stack zu jedem Zeitpunkt aus.
Ein Arbeitsspeicherabbild eines laufenden TinyOS besteht unten aus einem Datenbereich, auf dem der Heap aufsetzt und nach oben wächst. Von oben wächst der Stack des aktuellen Tasks in Richtung des Heaps. TinyOS besitzt keinen Mechanismus, um ein Ineinanderlaufen von Heap und Stack zu verhindern. Des Weiteren ist auch kein Speicherschutz vorhanden, was aber mehreren Modulen den Zugriff auf dieselben Datenstrukturen erlaubt. Sie allozieren hierzu Speicher auf dem Heap mittels einer statischen Variablen und können über die Schnittstellen eine Referenz auf die Speicherstelle an andere Module übergeben [LG09, Seite 36ff].
In Abbildung 3.6 ist die Komponente MainC zu sehen, die jede Anwendung für TinyOS einbinden muss. Sie ist für den Bootvorgang verantwortlich.

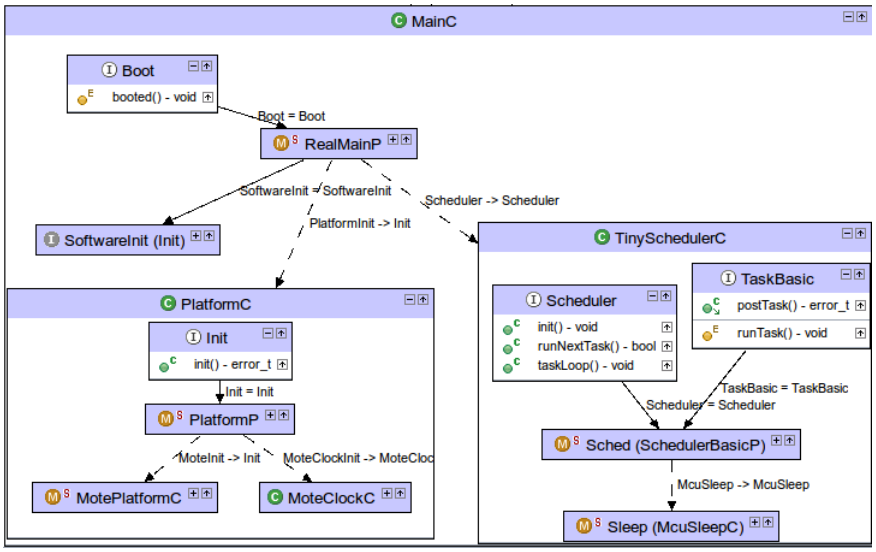


Abbildung 3.6: Die Komponente MainC von TinyOS

Der Bootvorgang wird von dem Modul RealMainP durchgeführt. Sie initalisiert den Scheduler und die Hardware, indem jeweils das Kommando *init()* aufgerufen wird. Die Komponente PlatformC implementiert die plattformabhängige Hardwareiinitalisierung, und die Komponente TinySchedulerC realisiert den Scheduler. Diese besitzt die Module Sched und Sleep. Sched implementiert den Scheduler und besitzt zwei Schnittstellen. Die Schnittstelle Scheduler bietet Kommandos zum Initalisieren, zum Ausführen des nächsten Tasks und für die Taskschleife an. Erzeugt wird ein neuer Task mit der Schnittstelle TaskBasic. Das Modul Sleep wird später bei der Energieverwaltung erläutert.

Sobald der Bootvorgang abgeschlossen ist, wird das Ereignis *booted* erzeugt, welches von Anwendungen mit der Schnittstlle Boot empfangen werden kann, und anschließend der Leerlauftask (TaskLoop) ausgeführt.

Der TinyOS-Netzwerkstapel

Zur Kommunikation der Sensorknoten über den Funkchip bietet TinyOS mehrere Komponenten, aus denen sich ein Netzwerkstapel bilden lässt. In TEP 126 [MHLC07] wird der Netzwerkstapel für den CC2420-Funkchip, welcher vom Maxfor-Sensorknoten verwendet wird, beschrieben. Er besteht aus mehreren Schichten, die jeweils eine spezielle Funktionalität anbieten, um eingehende und ausgehende Datenpakete zu modifizieren, zu filtern und zu übertragen. Jede Schicht kann Pakete aus der darunterliegenden Schicht entgegennehmen, bearbeiten und an die darüberliegende Schicht weiterreichen. Je nach Anwendung werden verschiedene Schichten zu einem Netzwerkstapel gestapelt. Der Netzwerkstapel des in dieser Arbeit entworfenen Sensornetzwerks ist in Abbildung 3.7 zu sehen.

Auf der Anwendungsschicht befinden sich die Anwendungen, für welche die Active-Message-Schicht die Kommunikation mit anderen Sensorknoten von dem Netzwerkstapel abstrahiert. Die Active-Message-Schicht bietet hierzu vier Schnittstellen. AMSend versendet neue Pakete an einen oder mehrere Sensorknoten, und die Schnittstelle Receive meldet neu eintreffende Pakete mittels eines Ereignisses. Hinzu kommt eine Kontrollschnittstelle zum Starten und Stoppen des Netzwerkstapels und eine Sendeschnittstelle zum Zugriff auf die Metainformationen des Pakets.

Unter der Active-Message-Schicht befindet sich die Unique-Send-Schicht. Diese fügt jedem Paket eine fortlaufende Nummer hinzu, um doppelte Pakete beim Empfänger in der Unique-Receive-Schicht ent-

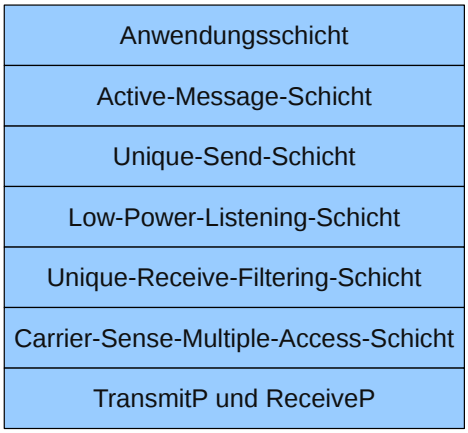


Abbildung 3.7: Der in dieser Arbeit verwendete engergieeffiziente Funk-Netzwerkstapel für TinyOS

decken zu können. Hierzu wird sich immer die letzte erhaltene Nummer zu jeder Sensorknotenadresse gemerkt, und Pakete gleicher Nummer vom selben Sender werden fallen gelassen.
Ist der CC2420-Funkchip eingeschaltet, so nimmt er einen Strom von 7 bis 19 mA auf. In TEP 105 [MHK] wird ein Low Power Listener (LPL) zum Energiesparen vorgestellt, welcher zwischen den im letzten Absatz beschriebenen Schichten sitzt und den Energieverbrauch des Funkchips reduziert. Hierzu wiederholt der Funkchip immer wieder einen Arbeitszyklus, der aus einer Schlafphase und einer Aktivphase besteht. Die Dauer des Arbeitszykluses und der prozentuale Anteil der Aktivphase kann mit einer Schnittstelle direkt aus der Anwendungsschicht heraus gesetzt werden. Solange der LPL keine Daten empfängt oder sendet, schaltet er den Funkchip während der Schlafphase aus, schaut in der Aktivphase nach, ob Daten eintreffen, und beginnt von neuem, wenn das nicht der Fall ist. Möchte ein Sensorknoten Daten senden, so sendet der LPL des Senders das Paket immer wieder erneut, bis der LPL des Empfängers den Empfang bestätigt oder eine Zeit von zwei Arbeitszyklen vergangen ist.
Der Funkchip greift auf den Kanal mittels CSMA-CA zu, um Kollisionen zu vermeiden. Ein Paket wird nur gesendet, wenn der Kanal derzeit nicht belegt ist. Ist der Kanal belegt, wartet der Funkchip eine kurze Zeit und überprüft erneut den Status des Kanals. In der Carrier-Sense-Multipe-Access-Schicht werden in dem Paket Wartezeiten für den CSMA-Algorithmus gesetzt, und es wird dem LPL das Ein- und Ausschalten des Funkchips ermöglicht.
Ganz unten befinden sich die Module TransmitP und ReceiveP. Sie implementieren die Unterbrechungsbehandlungen des Funkchips und binden diesen über den SPI-Bus an.
Ein Paket besteht laut TEP 111 [Lev] immer aus einem Header, Nutzdaten, einem Footer und Metadaten. Header, Footer und Metadaten sind vom Funkchip abhängig. Der Header enthält im Falle des CC2420-Funkchips unter anderem die Paketlänge, die Ziel- und Quelladresse und einem Active-Message-Type zur Unterscheidung von mehreren Funkverbindungen verschiedener Komponenten über einen Funkchip. Der Footer ist hingegen bei dem CC2420 leer, es wird lediglich eine zwei Byte große Prüfsumme angehängt. Die Metadaten werden nicht mitgesendet, sondern dienen den einzelnen Schichten zum Speichern und Austauschen von Informationen zu einem Paket. Hierunter fallen beim CC2420 die Sendeleistung, die Empfangssignalstärke, der Empfangszeitpunkt und zwei boolesche Variablen. Diese geben Auskunft über erhaltene Acknowledgements von versendenten Paketen und die Korrektheit der CRC-Summe bei empfangenen Paketen. Die Nutzdaten bestehen aus einem maximal TOS_DATA_LENGTH größen Struct, welcher per Voreinstellung auf 28 Bytes gestellt ist.
Eine Anwendung greift auf die einzelnen Bereiche des Paketes mittels der AMPacket-Schnittstelle zu. Hiermit kann sie ein neues Paket packen und wie oben beschrieben mittels AMSend über die Active-Message-Schicht verschicken oder aus erhaltenen Paketen die Nutzdaten und Metadaten extrahieren.

Analog zum Netzwerkstapel für die Funkkommunikation existiert ein Netzwerkstapel zur Kommunikation mit einem PC über die serielle Schnittstelle des Sensorknotens. Dieser wird in TEP 113 beschrieben [GL] und ist in Abbildung 3.8 zu sehen.

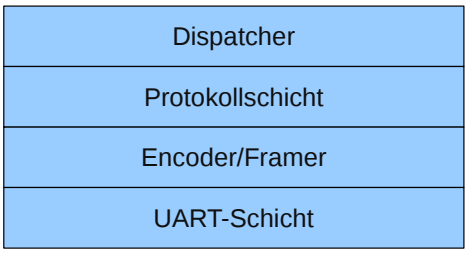


Abbildung 3.8: Der in dieser Arbeit verwendete serielle Netzwerkstapel für TinyOS (nach [GL])

In der UART-Schicht wird die Hardware zur seriellen Kommunikation des Mikrocontrollers angesprochen. Es werden die Übertragungsgeschwindigkeit und die Anzahl sowohl der Stoppbits als auch der zu sendenden und empfangenden Bytes gesetzt. Zudem realisiert diese Schicht das eigentliche Senden und Empfangen einzelner Bytes. Sie ist im Gegensatz zu den kommenden drei Schichten plattformabhängig.
Der Encoder/Framer erkennt bei dem eintreffenden Datenstrom die protokollspezifischen Trennzeichen und meldet sie der Protokollschicht.
Anhand der Meldungen aus dem Encoder/Framer nimmt die Protokollschicht den Datenstrom eines Paketes entgegen und berechnet und überprüft die Prüfsumme. Darüber hinaus kümmert sie sich um Protokollpakete, wie Acknowledgements.
Der Dispatcher erstellt aus dem Datenstrom ein Paket und reicht es an die Anwendungsschicht weiter. Er kann analog zu dem Netzwerkstapel für die Funkkommunikation mittels Send- und Receive-Schnitstellen Pakete von der Anwendung versenden oder ihr neue Pakete melden.
Das Paketformat der seriellen Kommunikation verhält sich bis auf das Fehlen der Metadaten analog zu dem der Funkkommunikation. Es besteht laut TEP 111 [Lev] aus einem Header und Nutzdaten. Der Header besteht aus der Zieladresse, der Nutzdatenlänge, einer Gruppen-ID und dem Active-Message-Type, um mehrere getrennte Verbindungen über einen Funkchip aufbauen zu können.

Die TinyOS-Energieverwaltung

Jeder Mikrocontroller besitzt mehrere Energiezustände, welche sich hinschtlich ihrer Rechenleistung, des Energieverbrauchs, der Zeit zum Wechseln in den Hochleistungsmodus und der verfügbaren Peripherie unterscheiden. Der im Maxfor verbaute Mikrocontroller MSP430 besitzt neben dem Hochleistungsmodus, in dem alle Komponenten eingeschaltet sind und Instruktionen verarbeitet werden, fünf Energiesparzustände. Diese reichen von LPM0, in dem lediglich die CPU und der Taktgeber ausgeschaltet sind, bis zu LPM4, in dem alle Komponenten bis auf die externen Unterbrechungen ausgeschaltet sind.
In TEP 112 [SLT+07] wird der Mechanismus zum Wechseln zwischen den Energiezuständen in TinyOS beschrieben. Ziel hierbei ist es, den Mikrocontroller immer in den möglichst besten Energiezustand zu versetzen. Sobald der Scheduler keinen Task mehr zum Einplanen vorfindet, also die Taskwarteschlange leer ist, ruft er das Kommando *McuSleep.sleep()* auf. Der Prozessor befindet sich nun in einem der oben vorgestellten Niedrigenergiemodi, bis eine Unterbrechung auftritt. Unterbrechungsquellen sind zum einen die Peripherie, wie Funkchip oder Sensoren, oder ein interner Timer des Mikrocontrollers.
Um zu entscheiden, in welchen Energiezustand die Sleepfunktion den Mikrocontroller überführt, nutzt TinyOS eine mikrocontrollerabhängige Funktion zur Berechnung des Energiezustandes für die Schlafphase. Ergebnis der Berechnungsfunktion ist der niedrigste Energiezustand, in dem alle derzeit benötigten Hardwaregeräte noch eingeschaltet sind. Läuft z. B. nur noch ein Timer im System, so wird in den tiefs-

ten Energiezustand gewechselt, in dem dieser noch voll funktionsfähig ist. Der Mikrocontroller wechselt mehrmals in der Sekunde den Energiezustand. Um nicht jedesmal einen neuen Energiezustand von der Berechnungsfunktion ermitteln zu lassen, wird sich der Energiezustand für die Schlafphase gemerkt. Er wird nur dann neu berechnet, wenn ein Dirty-Bit gesetzt ist. Die Hardwaretreiber der einzelnen Geräte sind dafür verantwortlich, das Dirty-Bit bei einer Änderung zu setzen.
Die hier beschriebene Energieverwaltung ist implizit, ein Anwendungsprogramm muss sich nicht um die Steuerung des Energiezustandes kümmern. Jedoch reagiert die Energieverwaltung lediglich auf die momentane Auslastungssituation. In einigen Fällen benötigt eine Anwendung viel Rechenleistung in kurzen Abständen oder geringe Latenzen, welche bereits vom Aufwachen aus dem niedrigsten Energiezustand überschritten werden. In solchen Fällen hat eine Komponente die Möglchkeit mittels der Schnittstelle McuPowerOverride den niedrigsten erlaubten Energiezustand anzugeben. Die Energieverwaltung bleibt zwar implizit, aber die Anwendung kann so die Energieverwaltung begrenzen oder ausschalten.
Neben dem Mikrocontroller ist der Funkchip der größte Energieverbraucher. Dieser wird mit dem LPL zeitweise automatisch ausgeschaltet. Eine Beschreibung des LPLs von TinyOS fand bereits im vorherigen Unterabschnitt statt.

3.5.3 MANTIS OS

Das MANTIS Operating System (MOS) ist ein weitverbreitetes Betriebssystem. Es ist angelehnt an UNIX und bietet mehr eingebaute Funktionen als TinyOS. MOS-Anwendungen werden in C geschrieben und innerhalb von Threads ausgeführt. Der MOS-Kernel beinhaltet einen Scheduler und Mechanismen zur Synchronisation [Sto05, Seite 188]. Hauptkonstruktionsmerkmale sind Flexibilität in Bezug auf Cross-Plattform-Unterstützung und Fernwartung in Form von Neuprogrammierung und Fernzugriff [BCD+05].
In Abbildung 3.9 ist der Aufbau von MOS zu sehen. Alle zum MOS gehörenden Komponenten sind in Blau gehalten, wohingegen die Anwendung aus den türkis gefärbten Threads besteht. Zentrales Element ist die MOS-System-API. Sie trennt die Anwendung von dem Betriebssystem und der Hardware ab und gewährleistet eine plattformübergreifende Unterstützung der Anwendung. Die API verbindet die Komponenten Kernel, Netzwerkschicht, Kommandoserver für die Fernwartung und die Geräteschicht.
In den folgenden Unterabschnitten werden Kernel inklusive Speicherverwaltung, die Geräteschicht sowie die Netzwerkschicht und die Energieverwaltung von MOS näher beschrieben und erläutert.

Der MOS-Kernel

MOS besitzt einen Multithreaded-Kernel mit einem Interface angelehnt an die POSIX-Semantik. Das Scheduling ist prioritätsbasiert mit Round Robin bei Threads gleicher Priorität. Für jeden Thread existiert ein Eintrag in der globalen Threadverwaltungstabelle. Die Größe dieser Tabelle wird zur Kompilierzeit festgelegt, wodurch es eine feste maximale Anzahl an Threads im System und einen fixen Speicher-Overhead gibt. In dem Tabelleneintrag werden alle wichtigen Informationen bis auf die Umgebung – diese wird auf dem Stack gespeichert – vorgehalten [BCD+05]. Der traditionelle POSIX-Ansastz ist Programmierern bereits bekannt und bietet daher einen einfachen Einstieg in die Anwendungsprogrammierung für MANTIS, jedoch bedingt er gegenüber ereignisbasierten Betriebssystemen längere Ausführungszeiten, einen größeren Quellcode und längere Unterbrechungsantwortzeiten [MLV09].
Der Arbeitsspeicher wird in zwei Bereiche aufgeteilt. Globale Variablen werden zur Kompilierzeit alloziert, der übrige Arbeitsspeicher wird als Heap verwaltet. Bei Erzeugung eines neuen Threads alloziert der Kernel Speicher vom Heap für den Threadstack. Da alle Threads in einem gemeinsamen Adressraum existieren, muss die maximale Stackgöße beim Erzeugen der Threads angegeben werden [Sto05]. Der Speicher wird wieder freigegeben, wenn der Thread beendet wird.
Aufgrund des gemeinsamen Adressraums können die Stacks der Threads auch ineinander wachsen und so Fehler produzieren. MOS besitzt keinen Schutzmechanismus dagegen, jedoch wird beim Erzeugen des

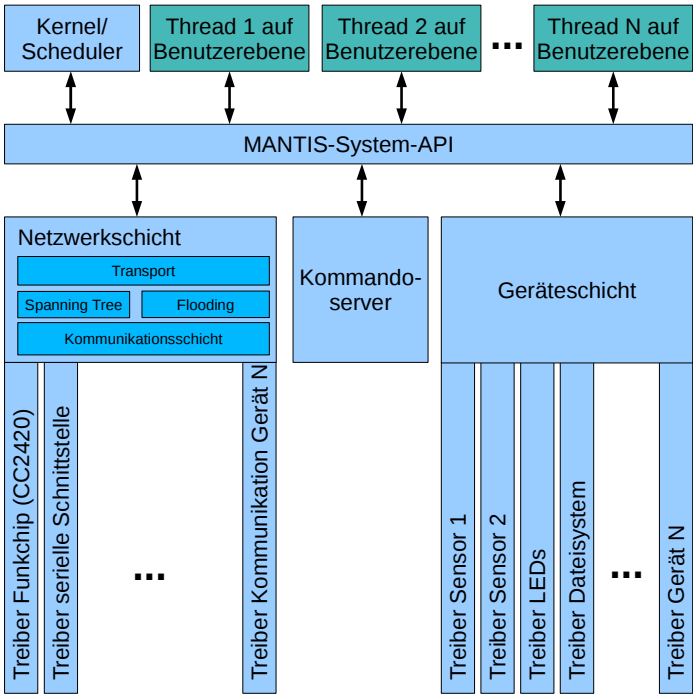


Abbildung 3.9: Die Architektur von MANTIS OS (nach [Sto05, Seite 190])

Threads der Stack mit 0xEF vollgeschrieben. So ist MOS in der Lage, bereits aufgetretene Fehler zu erkennen und zu reagieren [Sto05].
Der MOS-Kernel bietet des Weiteren binäre Semaphore (Mutex) und Semaphore mit Zählvariable an.

Die MOS-Geräte- und -Netzwerkschicht

MOS unterscheidet bei der Hardware zwischen Geräten mit asynchronem und synchronem Zugriff. Sie besitzen jeweils eine einheitliche Zugriffsschnittstelle.
Geräte mit synchronem Zugriff sind z. B. Sensoren, das Dateisystem, LEDs und der Zufallsnummerngenerator. Auf solche Geräte wird, angelehnt an UNIX, mit den Funktionen *read*, *write*, *mode* und *ioctl* zugegriffen. Mit *mode* können dabei Geräte an- und ausgeschaltet werden.
Geräte mit asynchronem Zugriff sind alle zur Kommunikation verwendeten Geräte, wie die Funk- und die serielle Schnittstelle. Solche Geräte erfordern die Fähigkeit, Daten im Hintergrund zu empfangen und zu senden. Es gibt bei diesen Geräten zusätzlich die Möglichkeit, einen synchronen Zugriff zu starten, wenn man z. B. auf ein neu eintreffendes Paket warten möchte. Hiebei können auch mehrere Netzwerkgeräte angegeben werden.
Die Gerätetreiber sind in MOS nicht in einem Thread, sondern in interruptgesteuerten Zustandsautomaten implementiert. Um Kopieroperationen zwischen den Schichten zu umgehen, besitzt die Netzwerkschicht eine zur Kompilierzeit anzugebende Anzahl an vorallozierten Puffern. Die Gerätetreiber fordern Puffer bei der Netzwerkschicht an, füllen diese und geben sie an den Anwendungsthread weiter. Ist die Anwendung mit der Verarbeitung der Daten fertig, so muss sie den Puffer wieder an die Netzwerkschicht zurückgeben [Sto05].

Die MOS-Energieverwaltung

MOS bietet sowohl explizite als auch implizite Funktionen zur Energieverwaltung.

Die explizite Energieverwaltung wird mit Hilfe der *mode*-Funktion realisiert. Es werden nicht nur die Stadien *an* und *aus* angeboten, sondern auch gerätespezifische Zwischenschritte wie *standby*.
Alle Geräte werden nach dem Einschalten des Sensorknotens ausgeschaltet gelassen, bis sie von der Anwendung explizit eingeschaltet werden. Sie verbleiben in diesem Zustand, bis die Anwendung die Geräte wieder ausschaltet. Greift die Anwendung synchron auf ein ausgeschaltetes Gerät zu, so wird es eingeschaltet, die Operation durchgeführt und wieder ausgeschaltet. Dennoch ergibt es einen Sinn, dass die Anwendung Geräte explizit einschalten kann. Einige Geräte verbrauchen beim Einschalten sehr viel Energie. Weiss die Anwendung bereits, dass sie ein Gerät häufig benutzt, so schaltet sie es explizit an und nur bei längeren Zugriffspausen aus.
Die CPU des eingebetteten Systems wird bei MOS mittels impliziter Energieverwaltung geregelt. Das Zustandsmodell des MOS-Kernels besteht aus den drei Zuständen *active*, *idle* und *power-save*. Im Zustand *active* ist die CPU im *High-power*-Modus und führt Threads mit maximaler Geschwindigkeit aus. Im *Idle*-Modus wird nur der Idlethread ausgeführt, die Interrupts und die Peripherie sind eingeschaltet. Im *Power-save*-Modus ist alles bis auf die externen Interrupts und einen Timer ausgeschaltet.
Zwischen den Zuständen wird wie folgt gewechselt: Ist noch mindestens ein Thread laufbereit, so verbleibt die CPU im *Power*-Modus. Wartet mindestens ein Thread an einem Interrupt, so verbleibt die CPU im *Idle*-Modus. Threads können nicht nur an einem Interrupt warten, sondern auch eine *Sleep*-Funktion aufrufen. Warten alle Threads an *sleep*, so wechselt der Kernel in den *Power-save*-Modus. Der oben erwähnte Timer wird hierbei dazu genutzt, die CPU wieder aufzuwecken [BCD+05][Sto05].

3.5.4 Contiki

Contiki ist ein von Adam Dunkels et al. entwickeltes leichtgewichtiges Betriebssystem, das sie 2004 mit ihrem Fachaufsatz [DGV04], aus dem alle in diesem Unterabschnitt zu findenden Informationen stammen, vorstellten.
Contiki ist in C implementiert und wurde bereits auf mehrere Plattformen portiert, unter anderem auch auf den MSP430 von Texas Instruments und auf die AVR-Reihe von Atmel. Gegenüber MANTIS OS und TinyOS besitzt Contiki das Alleinstellungsmerkmal, dynamisch Programme und Serviceroutinen nachladen zu können und dennoch den Ressourcenbeschränkungen in Sensornetzen gerecht zu werden. Darüber hinaus kann Contiki während der Laufzeit einzelne Programmteile über das Netzwerk austauschen. Dies führt zu einer sehr flexiblen Architektur.

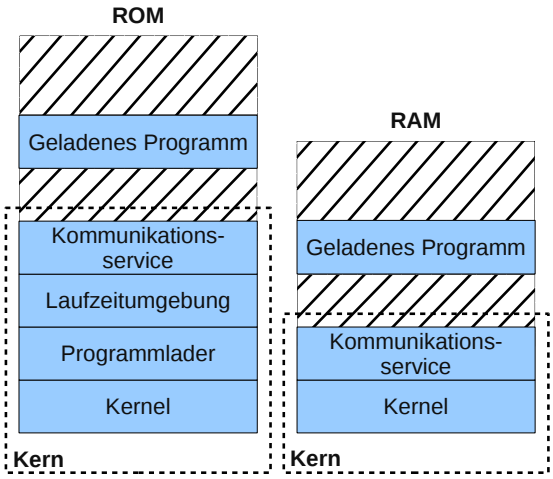


Abbildung 3.10: Ein Speicherabblild eines laufenden Contiki OS (nach [DGV04])

Wie in Abbildung 3.10 zu sehen ist, besteht ein laufendes Contiki aus dem Kernel, dem Programmlader,

der Laufzeitumgebung inklusive Bibliotheken und einer Menge von Prozessen. Prozesse können sowohl einzelne Eventhandler einer Anwendungen als auch Serviceroutinen sein. Die Interprozesskommunikation findet über den Kernel statt. Ebenfalls erkennt man anhand der Abbildung die Trennung zwischen Programm- und Arbeitsspeicher, es handelt sich somit um eine Harvard-Architektur.
Ein Contiki-System wird zur Kompilierzeit in zwei Teile partitioniert:
Zum einen in den an die Anwendung angepassten Kern. Dieser besteht typischerweise mindestens aus Kernel, Programmlader, Laufzeitumgebung und dem Kommunikationsservice inklusive Gerätetreiber für die Funkschnittstelle. Der Kern wird in ein einzelnes Binärabbild gebaut, welches zur Laufzeit nur noch komplett ausgetauscht werden kann.
Zum anderen besteht das Contiki-System aus einer Menge von Programmen und weiteren Services, welche vom Programmlader dynamisch zur Laufzeit hinzugefügt und entfernt werden können. Als Quelle hierfür sind sowohl der Hintergrundspeicher als auch das Netzwerk nutzbar.

Der Contiki-Kernel

Der Contiki-Kernel besteht aus einem leichtgewichtigen und prioritätsbasiertem Eventscheduler, der für neu auftretende Ereignisse einen Prozess erzeugt und in regelmäßigen Abständen die Pollinghandler der Prozesse aufruft. Der Programmablauf wird somit vollständig durch neue Ereignisse oder durch Pollingaufrufe gesteuert. Hierbei ist der Eventscheduler nicht preemptiv; Ereignisse besitzen eine Run-to-completion-Semantik. Sie können jedoch freiwillig den Prozessor abgeben.
Der Kernel unterstützt synchrone und asynchrone Ereignisse. Asynchrone Ereignisse werden vom Kernel in die Prozessschlange eingehängt und zu einem späteren Zeitpunkt ausgeführt. Synchrone Ereignisse hingegen verdrängen den Erzeugerprozess und werden sofort ausgeführt. Der Erzeugerprozess wird genau dann wieder dem Prozssor zugewiesen, wenn der Ereignisprozess bis zum Ende ausgeführt wurde. Aus Sicht des Erzeugerprozesses stellt das Erzeugen eines synchronen Ereignisses somit einen Funktionsaufruf dar.
Alle Prozesse teilen sich einen gemeinsamen Stack. Asynchrone Ereignisse reduzieren den Speicherbedarf des Stacks, da bis zum Beginn ihrer Behandlung alle vorherigen Prozesse bis zum Ende ausgeführt wurden und mit einem neuem leeren Stack begonnen werden kann.
Eine weitere Funktionalität des Contiki-Kernels ist der Pollingmechanismus. Dieser wird von nah an der Hardware operierenden Prozessen zur Abfrage von Statusinformationen der Hardware genutzt. Prozesse können bei der Pollingkomponente Interesse anmelden. Der Scheduler gibt der Pollingkomponente zwischen zwei Prozessen den Prozessor, so dass diese die eingetragenen Prozesse in der Reihenfolge der Pollingpriorität informieren kann.

Die Bibliotheken in Contiki

Der Kernel von Contiki bietet nur rudimentäre Funktionen zum Multiplexing der CPU und Funktionen zum Eventhandling. Der Rest des Systems ist mithilfe von Bibliotheken implementiert. So implmentiert z. B. eine Bibliothek preemptives Multithreading. Sie beinhaltet einen Scheduler und eine Stackverwaltung, welche jedem Thread einen eigenen Stack bereitstellt, auf dem auch die Register des Prozessors bei einem Threadwechsel gespeichert werden. Möchte ein Programm Multithreading nutzen, so muss es gegen diese Bibliothek gebaut werden.
In Contiki werden Programme auf drei verschiedene Arten mit Bibliotheken verlinkt.
Programme können statisch mit Bibliotheken innerhalb des Kerns verlinkt werden. Diese Art der Verlinkung kommt insbesondere bei sprachspezifischen und häufig genutzen Funktionen zum Einsatz.
Eine weitere Möglichkeit ist die statische Verlinkung mit Bibliotheken, die Bestandteil des ladbaren Programms sind. Anwendungsfall sind hier insbesondere Funktionen, die programmspezifisch sind und daher nur von einer Anwendung benutzt werden.

Letzte Möglichkeit ist die Implementiertung der Bibliotheksfunktionen mittels eines Services. Vorteil hierbei ist die Möglichkeit des dynamischen Austausches zur Laufzeit.

Die Contiki-Serviceroutinen

In Contiki realisiert ein Service häufig oder von verschiedenen anderen Prozessen genutzte Funktionen. Typische Beispiele sind der Kommunikationsstapel, Gerätetreiber für Sensoren und andere Hardware, aber auch übergeordnete Funktionen zur Datenberechnung wie Sensordatenfilter. Die Serviceroutinen können somit als eine Art gemeinsame Bibliothek angesehen werden.

Wie in Abbildung 3.11 zu sehen ist, werden die Serviceroutinen von der Service-Schicht verwaltet. Sie hält Informationen über die laufenden Sevices vor und ermöglicht anderen Prozessen, installierte Services zu finden. Ein Service besteht immer aus einem Serviceinterface und einem Prozess, der das Interface implementiert. Anwendungsprogramme, die einen Service nutzen wollen, werden zu diesem Zweck gegen ein Serviceinterfacestub gelinkt, das die Nutzung ermöglicht. Der Serviceinterfacestub kapselt die Nutzung von Services, so dass sie aus Sicht der Anwendungsprogramme einem Funktionsaufruf gleichen. Hierbei nutzt der Stub die Service-Schicht zur Lokalisation des Serviceinterfaces, vergleicht die Versionsnummer des Services mit sich selbst und führt bei positivem Ergebnis die Funktion aus.

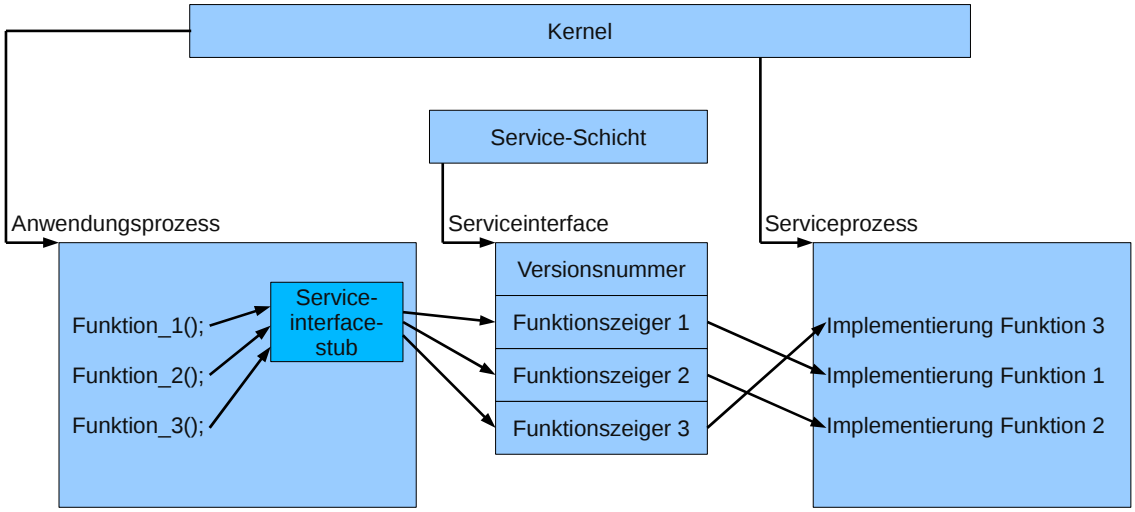


Abbildung 3.11: Der Funktionsaufruf von Serviceroutinen in Contiki OS (nach [DGV04])

Die Contiki-Energieverwaltung

Contiki besitzt keine Abstraktionsschicht für die Energieverwaltung. Es geht davon aus, dass die Energieverwaltung anwendungsabhängig ist und somit auch von dieser implementiert werden muss. Hierzu gibt der Scheduler Auskunft über die Anzahl der laufbereiten Prozesse, auf deren Grundlage die Anwendung das System herunterfahren und den Prozessor in einen Schlafmodus versetzen kann. Wacht der Prozessor durch eine externe Unterbrechung wieder auf, so springt der Pollingmechanismus an und benachrichtigt die Anwendung oder den Eventhandler der aufgetretenen Unterbrechung.

Analog muss sich der Kommunikationsstapel selbstständig um den Energiemodus des Funkchip kümmern.

3.6 Netzwerkprotokolle für Sensornetzwerke

Die Kommunikationsfunktionen werden bei Sensorknoten mit einem speziell an Sensornetzwerke angepassten Netzwerkprotokollstapel realisiert. In Unterabschnitt 3.5.2 wurde bereits der von TinyOS verwendete Protokollstapel vorgestellt; in diesem Abschnitt wird ein betriebssystemunabhängiger Blick auf die Kommunikation der Sensorknoten geworfen.
Von Akyildiz et al. wird der in Abbildung 3.12 zu sehende Netzwerkprotokollstapel beschrieben [ASSC02]. Auf der Anwendungsschicht befindet sich ein Protokoll, das den Anwendungen der Sensorknoten einen einheitlichen Zugriff auf das Netzwerk bietet. Anwendungen können Daten verschicken und erhalten und müssen sich nicht um die Implementierung des Netzwerks kümmern. Die Transportschicht ist nur nötig, wenn das Sensornetzwerk mit einem externen Netz wie dem Internet direkt kommunizieren können soll. Dies ist nur selten der Fall und wird daher hier auch nicht weiter besprochen. In der Vermittlungsschicht stellen die Routingprotokolle eine Verbindung zwischen den einzelnen Sensorknoten und der Datensenke her. Sie steuern den Paketfluss und organisieren so die Sensorknoten zu einem Sensornetzwerk. Die Verbindungsschicht ist für Punkt-zu-Punkt- und Punkt-zu-Multipunkt-Verbindungen verantwortlich. Sie erkennt Datenrahmenstrukturen und kümmert sich um die Datenströme der Bitübertragungsschicht. Hierbei ist ein Medium-Access-Control-Protokoll (MAC-Protokoll) für den Zugriff auf das Übertragungsmedium zuständig. In der Bitübertragungsschicht wird die Funkfrequenz gesetzt, die Trägerwelle generiert und moduliert, eingehende Funksignale detektiert und Daten verschlüsselt.

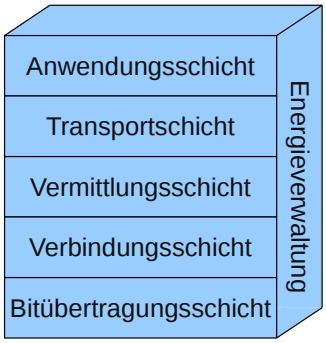


Abbildung 3.12: Ein allgemeiner Protokollstapel für Sensornetzwerke (nach [ASSC02])

Klassische MAC- und Routingprotokolle für funk- und kabelbasierte Netzwerkgeräte sind für kabellose Sensornetzwerke nicht geeignet. Kabellose Sensornetzwerke unterscheiden sich in vielen Punkten von herkömmlichen Netzwerken. Die Sensorknoten unterliegen den in Abschnitt 3.4 beschriebenen Ressourcenbeschränkungen, und insbesondere die Energieversorgung wird von den klassischen Protokollen zu sehr belastet [VDMC08, Seite 163]. Zudem werden die Sensorknoten in einigen Fällen zufällig aufgestellt, selten wird eine genaue Aufstellungsplanung mit vordefinierten Positionen vorgenommen. Sie müssen sich daher selbstständig zu einem Netzwerk organsieren können. Dabei sind die Sensorknoten nich immer gleich verteilt, einige haben viele Nachbarn und andere befinden sich in dünn bevölkerten Regionen. Darüber hinaus erzeugen in Sensornetzwerken Ereignisse viele Daten innerhalb eines kurzen Zeitraums, was von den Kommunikationsprotokollen bedacht werden muss. Im Gegensatz zum Internet mit seinem IP-Protokoll ist es in Sensornetzwerken oft wichtiger, die Daten zu bekommen als genau zu wissen, wo sie herstammen [VDMC08, Seite 173]. Zudem fließen die Daten in Sensornetzwerken in der Mehrheit von vielen Punkten zu genau einer Basisstation. Die Daten sind oft Beobachtungen derselben Umgebung zu bestimmten Größen und können auf den Weg zur Senke miteinander fusioniert werden.
Das von TinyOS verwendete MAC-Protokoll CSMA-CA mit dem LPL zum Energiesparen wurde bereits in Unterabschnitt 3.5.2 betrachtet. Eine Alternative zum LPL ist SMAC: es unterteilt den Zugriff auf den

Funkkanal in *Sleep*- und *Listen*-Phasen. Die Zeitpunkte werden mit einem Synchronisationspaket bekannt gegeben, und andere Sensorknoten können in der nächsten *Listen*-Phase Daten übermitteln [VDMC08, Seite 172f]. Diese und weitere Alternativen zu MAC-Protokollen liegen jedoch außerhalb des Fokus dieser Arbeit.
In diesem Abschnitt werden jetzt Routingprotokolle speziell für Sensornetzwerke betrachtet. Hierzu werden zunächst die Anforderungen genannt, und im Anschluss wird eine Taxonomie für Routingprotokolle vorgestellt.
Um den Ressourcenbeschränkungen insbesondere bei der Energieversorgung gerecht zu werden, muss das verwendete Routingprotokoll an die Anwendung und die Gegebenheiten des Sensornetzes angepasst sein. Im Folgenden werden Desingkriterien nach Al-Karaki und Kamal [AKK04] und Boukerche [Bou09, Seite 131f] genannt.

Sensorknotenaufstellung und Skalierbarkeit Die Sensorknoten können entweder zufällig oder deterministisch aufgestellt worden sein. Kennt man die Lage und Nachbarschaftsbeziehungen der Senorknoten, so können von dem Routingprotokoll feste Pfade zur Paketvermittlung genutzt werden. Andernfalls muss das Paket selber einen Weg von Quelle zur Senke und je nach Anwendung auch zwischen beliebigen Sensorknoten finden. In beiden Fällen muss jedoch die Aufstellungsverteilung bedacht werden. Sensornetzwerke können auf einem großen oder kleinen Gebiet aufgestellt sein. Je nach Lage und Anwendungsfall haben sie nur wenige Nachbarknoten bis hin zu mehreren hundert. Alle Routingprotokolle müssen mit der Größe skalieren und bei nicht gleichmäßigen Sensorknotenverteilungen Flaschenhälse in der Kommunikation berücksichtigen.

Fehlertoleranz und Quality of Service Die Sensorknoten können wegen Energiemangels, physikalischer Schäden oder Hardwaredefekten ausfallen. Hierbei sind die Ausfallraten deutlich höher als in herkömmlichen Netzwerken zwischen PCs. Das Routingprotokoll muss robust gegenüber Ausfällen sein, indem es diese schnell erkennt und Pakete über alternative Wege leitet. Um Ausfällen zu begegnen, muss jeder Sensorknoten erreichbare Nachbarknoten kennen und Regionen des Sensornetzwerks mit höherer Sensorknotendichte und mehr Energie in Richtung des Ziels finden. So kann vermieden werden, dass wenige Pfade immer wieder benutzt werden und sich so Daten stauen oder die Energieversorgung einzelner Sensorknoten übermäßig beansprucht wird. Es ist Aufgabe des Routingprotokolls, die Lebensdauer des Sensornetzwerks zu maximieren und stets eine verlässliche Kommunikation zu gewährleisten. Hierbei verlangen einige Anwendungen auch die Übertragung der Daten innerhalb einer bestimmten Zeit.

Dynamiken im Sensornetzwerk Die meisten Netzwerktopologien gehen davon aus, dass die Sensorknoten stationär sind. Einige Anwendungen erfordern jedoch auch mobile Sensorknoten oder Basisstationen. Das Routingprotokoll muss Datenpakete sicher von und zu mobilen Knoten leiten können. Darüber hinaus sind von Sensorknoten gemessene Ereignisse je nach Anwendung dynamisch oder statisch. Dynamische Eereignisse wie in Anwendungen zur Zielverfolgung erfordern eine periodische Übertragung und bedingen viele Daten, welche zur Basisstation geleitet werden müssen. Wird ein statisches Phänomen wie die Temperatur an einem Ort erfasst, so kann das Routingprotokoll reaktiv arbeiten und die Daten zu einem passenden Zeitpunkt an die Basisstation weiterleiten.

Aggregation von Daten Die Messdaten der einzelnen Sensorknoten stammen oft von demselben Phänomen ab, und so können Redundanzen auftreten. Die Routingprotokolle sollten ähnliche Daten in mehreren Paketen erkennen und zu einem aggregieren. Die Anzahl der Übertragungen wird damit reduziert, und sowohl Energieversorgung als auch Bandbreite können geschont werden. Die Aggregation wird hierbei von einer Aggregationsfunktion vorgenommen. Diese implementiert Methoden, die von der Filterung der Daten nach Duplikaten, Bildung von Minimum, Maximum sowie Durchschnittswert der Daten bis hin zu komplexen Berechnungen reichen. Es ist eine Fusion der Daten einer Umweltgröße von mehreren Sensorknoten mit verschiedenen Messauflösungen zu einem Datum

mit besserer Auflösung und geringerem Hintergrundrauschen denkbar. Die benötigte Rechenleistung für solche Verfahren muss vom Routingprotokoll bedacht werden. Die Aggregation der Daten ist nur sinnvoll, wenn die eingesparte Energie durch weniger Kommunikation den Mehraufwand bei der Berechnung wieder gutmacht.

Heterogenität des Sensornetzwerks Nicht in allen Sensornetzwerken besitzen alle Sensorknoten dieselben Fähigkeiten in Bezug auf Rechenleistung, Speichervermögen, Kommunikation und Energieversorgung. In heterogenen Sensornetzwerken können die einen Sensorknoten für die Datenfusion bestimmt sein, andere bilden einen sicheren Funkweg zur Basisstation, und wiederum andere Sensorknoten nur für die Datenerfassung zuständig. Das verwendete Routingprotokoll muss sich den Gegebenheiten der Sensorknoten anpassen und die Daten gegebenenfalls an andere Knoten mit speziellen Fähigkeiten weiterleiten.

Datenerfassungsmodell des Sensornetzwerks Je nach Anwendung ist die Sensordatenerfassung zeitgesteuert, ereignisgesteuert, anforderungsgesteuert oder ein Hybrid aus den dreien. Bei einem zeitgesteuertem Modell werden die Daten periodisch erhoben, wie beispielsweise bei der kontinuierlichen Erfassung der Temperatur. Bei ereignisgesteuerten Modellen wird immer dann eine Messung vorgenommen, wenn ein bestimmtes Ereignis in der Umwelt stattgefunden hat. Im Gegensatz dazu warten bei anforderungsgesteuerten Modellen die Sensorknoten auf eine Anfrage nach Sensorwerten. Darüber hinaus können Mischungen, etwa eine periodische Messung auf Anfrage für einen bestimmten Zeitraum, vorkommen. Die Routingprotokolle müssen das Datenerfassungsmodell berücksichtigen und gegebenenfalls das Datenaufkommen vorherbestimmen.

Energieverbrauch Der Energieverbrauch ist beim Routing der Pakete zwischen den Sensorknoten eine wichtige Größe und muss vom Routingprotokoll bedacht werden. Bei nahezu allen bisher vorgestellten Designkriterien findet eine Abwägung zwischen Energiesparen und Funktionalität statt. Zusätzlich können Routingprotokolle bei einigen Anwendungen zeitweise bestimmte Sensorknoten ausschalten, um die Lebensdauer des Sensornetzwerks zu erhöhen. Es muss aber auch bedacht werden, dass der Energiverbrauch des Funks mit der Entfernung zunimmt. Mehrere kurze Verbindungen sind somit günstiger als wenige weite Übertragungen. Das Routingprotokoll kann einzelne Wege nach ihrem Energieverbrauch bewerten und die Pakete entlang des besten Pfades zum Ziel befördern.

Aufgrund der großen Anzahl an verschiedenen, oft wiedersprüchlichen Anforderungen ist eine Vielzahl von Routingprotokollen mit unterschiedlichsten Ansätzen entstanden. Um einen Überblick über diese zu gewinnen, wird die von Boukerche vorgeschlagene Taxonomie vorgestellt und mit Ausführugen zu einer ähnlichen Taxonomie von Al-Karaki et al. ergänzt [Bou09, Seite 132ff][AKK04].

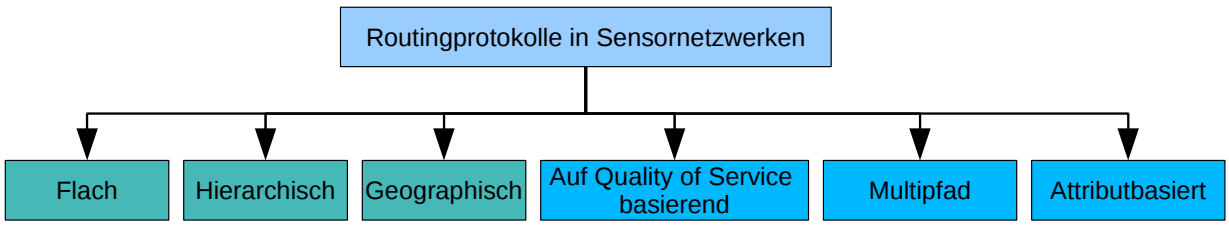
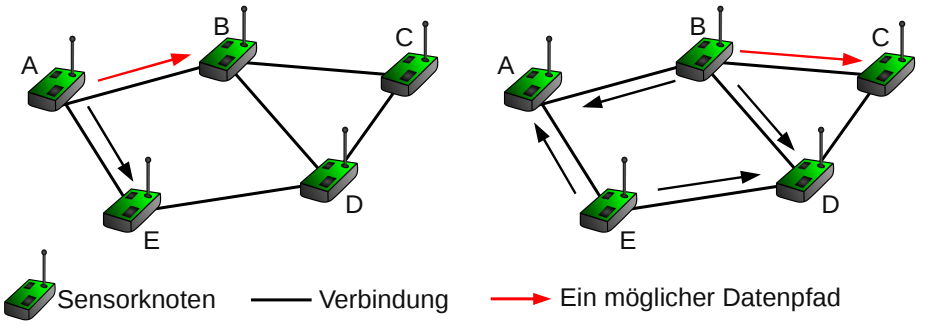


Abbildung 3.13: Ein Klassifikationsschema für Routingprotokolle (nach [Bou09, Seite 133])

Wie in Abbildung 3.13 zu sehen, können Routingprotokolle entsprechend ihrer Netzwerkstruktur in flache, hierarchische und geographische Protokolle unterteilt werden. In flachen Routingprotokollen sind alle Sensorknoten für ähnliche Aufgaben zuständig, und keiner nimmt im Netzwerk eine Sonderrolle ein.

In hierarchischen Routingprotokollen sind einzelne Sensorknoten besser ausgestattet oder bieten andere Funktionalitäten an. Es handelt sich also um ein Routingprotokoll für ein nicht heterogenes Sensornetzwerk, vergeliche hierzu die Ausführungen zu Heterogenität von Sensornetzwerken bei den Designkriterien für Routingprotokolle.
In den meisten Fällen zeichnen sich hierarchische Sensornetzwerke nicht durch unterschiedliche Sensorknotenhardware aus, sondern durch die Organisation des Sensornetzwerks in einer Hierarchie. Mittels eines Clusterverfahrens wird das Sensornetzwerk in kleinere Teilbereiche unterteilt, in denen mindestens ein Clusterhead eine Sonderrolle einnimmt. Die Clusterheads bilden miteinander ein eigenes Netzwerk, das Rückgrat des gesamten Sensornetzwerks. Über dieses können alle Sensorknoten miteinander kommunizieren. Die Stärke dieses Verfahrens ist insbesondere bei sehr großen Sensornetzwerken zu finden, denn das Sensornetzwerk wird stark vereinfacht. Die einzelnen Sensoren müssen nun nur noch mit einem typischerweise in der Nachbarschaft liegenden Clusterhead kommunizieren, und dieser kümmert sich um die Verbindung zur Senke oder zu anderen Sensorknoten.
Die dritte Klasse sind die geographischen Routingprotokolle; sie nutzen Standortinformationen, um die Daten zu routen. Die Sensorknoten kennen ihre eigene Position und die Position der Nachbarn. Trifft ein neues Paket ein, werden diese Informationen genutzt, um das Paket in Richtung des Zielgebietes weiterzuleiten.
Parallel zur Unterteilung nach Netzwerkstruktur können die Routingprotokolle anhand wichtiger Designentscheidungen unterschieden werden. Auf Quality of Service (QoS) basierende Routingprotokolle setzen den Schwerpunkt bei der verlässlichen Kommunikation. Wie bei den Designkriterien unter Fehlertoleranz und Quality of Service bereits beschrieben, wird versucht die Lebensdauer des Sensornetzwerks zu maximieren und hierbei die einzelnen Pakete sicher an ihr Ziel zu leiten. Multipfadprotokolle berechnen mehrere mögliche Pfade für die Datenpakete. Die Pakete werden dann entlang des besten Pfades zum Ziel gesendet. Fällt ein Pfad aus, so hat das Routingprotokoll alternative Wege zur Kompensation des Fehlers.
Die letzte Klasse sind die attributbasierten Routingprotokolle. Bei diesen werden die Pakete anhand der Daten in den Paketen weitergeleitet. Dieser Ansatz geht davon aus, dass alle Sensorknoten ähnliche Aufgaben erfüllen. Somit ist jeder von ihnen in der Lage zu entscheiden, ob ein Paket fallengelassen wird, ob es mit anderen Paketen aggregiert wird oder an welchen Sensorlknoten es weitergeleitet werden soll.
In Tabelle 3.5 ist eine Übersicht über Routingprotokolle speziell für Sensornetzwerke zu finden. Im Folgenden werden in den Unterabschnitten 3.6.1 und 3.6.2 die für diese Arbeit relevanten Routingprotokolle genauer beschrieben.

3.6.1 Flooding



(a) Sensorknoten A sendet das Paket an seine Nachbarn.

(b) Die Nachbarknoten B und E versenden das Paket ebenfalls an ihre Nachbarn.

Abbildung 3.14: Das Flooding-Protokoll für Sensornetzwerke

Tabelle 3.5: Eine Übersicht über verschiedene Routingprotokolle für Sensornetzwerke (nach [AKK04] und [Bou09, Seite 134 ff.])

	Klassifizierung	**Mobilität**	**Daten-aggregation**	**QoS**	**Skalierbarkeit**	**Multi-pfad**
SPIN	Flach	Möglich	Ja	Nein	Eingeschränkt	Ja
Flooding	Flach	Eingeschränkt	Nein	Nein	Schlecht	Ja
Directed Diffusion	Flach	Eingeschränkt	Ja	Nein	Eingeschränkt	Ja
Rumor Routing	Flach	Sehr eingeschränkt	Ja	Nein	Gut	Nein
GBR	Flach	Eingeschränkt	Ja	Nein	Eingeschränkt	Nein
MCFA	Flach	Nein	Nein	Nein	Gut	Nein
CADR	Flach	Nein	Ja	Nein	Eingeschränkt	Nein
Cougar	Flach	Nein	Ja	Nein	Eingeschränkt	Nein
Acquire	Flach	Eingeschränkt	Ja	Nein	Eingeschränkt	Nein
EAR	Flach	Eingeschränkt	Nein	Gut	Eingeschränkt	Nein
LEACH	Hierarchisch	Fixe BS	Ja	Nein	Gut	Nein
TEEN & APTEEN	Hierarchisch	Fixe BS	Ja	Nein	Gut	Nein
PEGASIS	Hierarchisch	Fixe BS	Nein	Nein	Gut	Nein
MECN & SMECN	Hierarchisch	Nein	Nein	Nein	Schlecht	Nein
SOP	Hierarchisch	Nein	Nein	Nein	Schlecht	Nein
HPAR	Hierarchisch	Nein	Nein	Nein	Gut	Nein
VGA	Hierarchisch	Nein	Ja	Nein	Gut	Ja
Sensor aggregate	Hierarchisch	Eingeschränkt	Ja	Nein	Gut	Nein
TTDD	Hierarchisch	Ja	Nein	Nein	Schlecht	Möglich
GAF	Geographisch	Eingeschränkt	Nein	Nein	Gut	Nein
GEAR	Geographisch	Eingeschränkt	Nein	Nein	Eingeschränkt	Nein
SPAN	Geographisch	Eingeschränkt	Nein	Nein	Eingeschränkt	Nein
MFR	Geographisch	Nein	Nein	Nein	Eingeschränkt	Nein
GOAFR	Geographisch	Nein	Nein	Nein	Gut	Nein
SAR	QoS	Nein	Ja	Ja	Eingeschränkt	Nein
SPEED	QoS	Nein	Nein	Ja	Eingeschränkt	Nein

Das Flooding-Protokoll ist der klassische Ansatz der einfachen Datenweiterleitung ohne komplexen Routingalgorithmus und Kenntnisse über die Netzwerkstruktur [VDMC08, Seite 174]. Beim Flooding wird von jedem Sensorknoten jedes erhaltene Paket an alle Nachbarn weitergeleitet. Dieser Prozess setzt sich fort, bis entweder eine bestimmte Anzahl an Hops überschritten wurde oder das Paket das Ziel erreicht hat. Ein Beispiel hierfür ist in Abbildung 3.14 gegeben, worin Sensorknoten A ein Paket an Sensorknoten C schicken möchte. Zunächst versendet A ein Paket an seine beiden Nachbarn B und E, vergleiche Abbildung 3.14(a). Da B und E jeweils nicht der Zielknoten sind, versenden sie das Paket weiter an alle ihre Nachbarn A,D und C, vergleiche Abbildung 3.14(b). C ist das Ziel und versendet im nächsten Schritt das Paket nicht weiter. Die anderen beiden Knoten A und D brechen die Weiterleitung aufgrund des Hop-Zählers ab, sofern die maximale Hop-Entfernung erreicht wurde.
Die Hop-Entfernung in diesem Sensornetz beträgt zwei, da jeder Sensorknoten jeden Sensorknoten mit minimal zwei Hops erreichen kann. Eine bessere Abbruchentfernung sind jedoch drei Hops: Hiermit kann ein beliebiger Sensorknoten ausfallen, und alle verbleibenden können sich noch erreichen. In diesem Fall würden in dem oben genannten Beispiel die Sensorknoten A und D jedoch noch ein weiteres Mal das Paket weiterleiten.
Die größte Schwäche von Flooding ist die schlechte Skalierbarkeit gegenüber der Sensornetzgröße. Es wird ein Vielfaches mehr an Paketen versendet als nötig. Überlappungen führen dazu, dass ein Sensorknoten mehrfach die Kopie desselben Paketes erhält [VDMC08, Seite 174]. Reflexionen führen dazu, dass sich gleiche Pfade mehrmals abdecken und die Nachrichtenzahl im Sensornetzwerk sprunghaft ansteigt. Diese Effekte verursachen nicht unerhebliche Probleme bei der Kommunikationsbandbreite und der Energieversorgung. Flooding ist blind für die Netzwerkstruktur und die aktuell vorhandenen Ressourcen, hat aber den Vorteil, dass es robust gegenüber Sensorknotenausfällen ist und die Pakete an alle Knoten im Netzwerk weiterleitet [CI05, Seite 12].

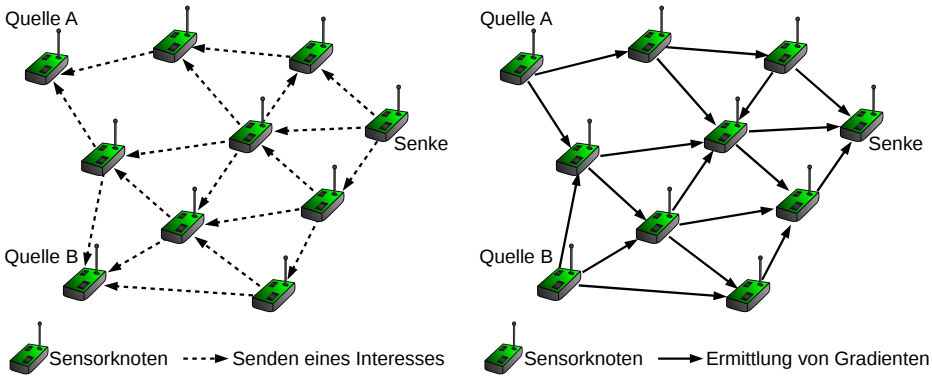
GOSSIP

Eine leichte Verbesserung von Flooding ist das GOSSIP-Protokoll. Erhält ein Sensorknoten ein Paket, so sendet es dies an genau einen zufällig ausgewählten Nachbarn weiter. Dieser Vorgang wird fortgesetzt, bis das Paket das Ziel erreicht hat. Hierdurch kann der sprunghafte Anstieg bei der Kommunikation vermieden werden; jeder Sensorknoten hat zu jedem Zeitpunkt nur ein einziges Paket [ASSC02]. Das Überlappungsproblem bleibt jedoch weiterhin bestehen [VDMC08, Seite 174]. GOSSIP hat zudem den großen Nachteil, dass die Übertragungszeit der Pakete sehr lang werden kann. Es eignet sich somit eher schlecht, um in regelmäßigen Abständen Daten von der Quelle zur Senke zu senden. Es wird vornehmlich eingesetzt, um eine Information im gesamten Netzwerk bekannt zu machen.

3.6.2 Directed Diffusion

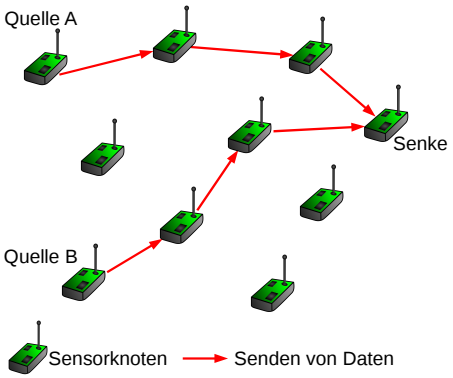
Directed Diffusion wurde von Intanagonwiwat et al. vorgeschlagen und gehört zu der Klasse der flachen, anforderungsbasierten Routingprotokolle [IGE+03].
Alle im Netz vorhandenen Messgrößen werden von jeweils einem Attribut-Wert-Paar beschrieben. Die Datensenke kann nun ein Interesse in Form einer Aufgabe an das Netzwerk melden. Ein Interesse besteht normalerweise aus einem Attribut, einem Zeitintervall, das beschreibt, in welchem Abstand die Datenquelle den Messwert des Attributs periodisch senden soll. Hinzu kommt der Gradient des Senders und ein weiteres Zeitintervall, welches beschreibt, wie lange eine Datenquelle ihre Messung vornehmen soll. Optional kann eine weitere beschreibende Komponente für die Zielknoten hinzugefügt werden. So kann entweder über Gradientenentfernung, räumliche Beschreibung der Sensorknoten oder Knotenadressbereiche die Zielsensorknotenmenge eingeschränkt werden. Ein solches Interesse wird im Netzwerk verbreitet, und Gradienten für den Weg von Datenquelle zu Datensenke werden hierbei aufgebaut. Im Folgenden wird beschrieben, wie die Interessen im Netz verbreitet, die Gradienten berechnet und die Daten zur Senke geschickt werden.

Jeder Sensorknoten pflegt eine Interessentabelle, in der jeder Eintrag für genau ein Interesse steht, also alle Attribute gleiche Werte annehmen. Die Interessentabelle speichert den Zeitstempel des letzten Eintreffens eines Interesses, für jeden Nachbarn einen Gradienten, von dem das Interesse eingetroffen ist, und Attribute des Interesses, um verschiedene Interessen voneinander unterscheiden zu können. Hinzu kommt ein Verfallsdatum, welches vom Zeitstempel und dem Zeitintervall des Interesses abgeleitet wird.
Trifft ein Interesse bei einem Knoten ein, so wird in der Interessentabelle das Vorhandensein geprüft. Ist es noch nicht vorhanden, so wird ein neuer Eintrag erzeugt, insbesondere wird auch ein Gradient berechnet und mit der eindeutigen Adresse des Senders gespeichert. Ist bereits ein Interesseneintrag vorhanden, aber noch kein Gradient zum Sender, wird dieser dem Eintrag hinzugefügt, und der Zeitstempel und das Verfallsdatum werden aktualisiert. Ist bereits ein Interesseneintrag und ein Gradient zum Sender vorhanden, werden nur Zeitstempel und Verfallsdatum aktualisiert.
Nachdem die Interessentabelle aktualisiert wurde, entscheidet der Knoten, ob er dieses Interesse weiter ins Netz propagiert. Um entscheiden zu können, ob ein Interesse überhaupt weiter ins Netz propagiert wird, kann man beim Erhalten des Interesses die Differenz des alten und des neuen Zeitstempels bilden und das Interesse verwerfen, wenn nicht eine Minimaldauer überschritten wird. Die naheliegende Lösung, alle Interessen zu verwerfen, bei denen bereits ein Eintrag in der Interessentabelle besteht, die man also bereits erhalten und weiterversendet hat, bietet sich nicht an. Hier könnte man kein Interesse erkennen, welches von gleicher Natur wie das ursprüngliche ist, also gleiche Sensorwerte zu einem späteren Zeitpunkt anfordert.
Hat man sich dafür entschieden, das Interesse weiter zu propagieren, so stellt sich die Frage, an welche Knoten. Hier sind mehrere Alternativen möglich. Die einfachste Form ist ein Broadcast an alle Nachbarknoten, was dem Floodingprotokoll entspricht. Dies wird häufig angewendet, da die Knoten oft keine Informationen über deren Nachbarn besitzen. Es wäre jedoch günstig, die Nachbarn zu kennen. Man kann alle Knoten exkludieren, von denen man bereits einen Gradienten zu dem aktuellen Interesse in der Interessentabelle gespeichert hat, von denen man also das Interesse bereits erhalten hat. Es ist hierbei jedoch darauf zu achten, dass so gleichartige Interessen sich nicht durch das Netz propagieren lassen. Eine Lösung für dieses Problem ist es, einen Zeitstempel mit in das Interesse aufzunehmen, um jedes Interesse eindeutig zu machen. Als weitere Herausforderung müssen die Nachbarn ständig aktuell gehalten werden. Insbesondere in dynamischen Sensornetzen, in denen zur Laufzeit neue Knoten hinzukommen und entfernt werden können, muss die Menge aller Nachbarn periodisch aktualisiert werden. Eine weitere Alternative zur Entscheidung der Interessenausbreitung stellt die Vorhaltung der bisher bedienten Interessen dar. Man kann sich merken, welche Knoten in der Vergangenheit bereits ein Interesse an gleichen Sensorwerten beantwortet haben, und diese gezielt wieder ansprechen. Auch hier muss periodisch nach bisher unbekannten Nachbarn geschaut werden. In allen Alternativen ist sichergestellt, dass ein Interesse, wie in Abbildung 3.15(a) zu sehen, am Ziel ankommt.
Wie bereits oben beschrieben wird bei jedem Eintreffen eines Interesses der Gradient zum Sender gebildet. Es entsteht ein Graph von Interessen, wie er in Abbildung 3.15(b) zu sehen ist. Zur Berechnung der Gradienten gibt es abermals mehrere Möglichkeiten. Wird mit einem Gradienten von 1 beim ersten Versenden gestartet und bei jedem weiteren Versenden 1 addiert, so gibt der Gradient Auskunft über die Hop-Entfernung zum Ziel. Alternativ kann mit 0 gestartet werden und jeder Sensorknoten addiert seine Entfernung zum Sender eines Interesses und nimmt diesen Wert als Gradienten für seine Interessentabelle und zum Weitersenden des Interesses. Als Wert für die Entfernung kann z.B. die RSSI genutzt werden, vergleiche hierzu Abschnitt 3.8. Alalog zu den Entfernungen kann die verfügbare Restenergie der Sensorknoten aufsumiert werden.
Möcht ein Sensorknoten nun ein Interesse bedienen, so sendet es die Daten zu dem Sensorknoten mit dem besten Gradienten des Interesses. So verfahren auch alle dieses Datenpaket erhaltenen Sensorknoten, bis das Paket am Ziel angekommen ist. Vergleiche hierzu Abbildung 3.15(c). Der beste Gradient ist hierbei je nach Berechnungsmethode der kleinste oder größte. So nehmen die Pakete autmatisch den besten Weg zurück zur Senke, egal ob kurze Entfernungen oder Wege mit der meisten Restenergie als optimal angesehen werden.



(a) Das Interesse wird von der Senke an beide Quellen gesendet.

(b) Beim Versenden der Interessen werden die Gradienten in Richtung der Senke gebildet.



(c) Die Datenpakete fließen entlang der besten Gradienten zur Senke.

Abbildung 3.15: Das Directed-Diffusion-Protokoll (nach [IGE+03])

Directed Diffusion skaliert deutlich besser mit der Netzwerkgröße als Flooding. Die Pakete von der Quelle zur Senke werden gezielt über den besten Weg gesendet. Dabei wird auf die Ressourcen und die zugrunde liegende Netzwerkstruktur eingegangen. Es werden nur so viele Pakete wie nötig verschickt. Beim verbreiten der Interessen kann durch geschicktes weiterversenden ebenfalls ein Vorteil gegenüber Flooding entstehen. Selbst wenn die Interessen per Flooding verbreitet werden, so ist dies nicht so dramatisch, da in der Regel nur selten neue Interessen an das Netzwerk gerichtet werden. Als Nachteil von Directed Diffusion ist die Ausrichtung auf das einsammeln von Daten zu nennen. Es geht davon aus, das die Daten vornehmlich von Quelle zu Senke fließen. Eine Kommunikation zwischen zwei beliebigen Sensorknoten ist so nicht vorgesehen. In einem solchen Fall könnte man sich jedoch eines speziellen Sendeinteresses bedienen, das extra einen Gradienten zu dem Zielknoten aufbaut. Benutzt man Flooding bei der Interessenverbreitung, so ist in diesem Fall Directed Diffusion sogar schlechter als Flooding.

3.7 Zeitsynchronisation in Sensornetzwerken

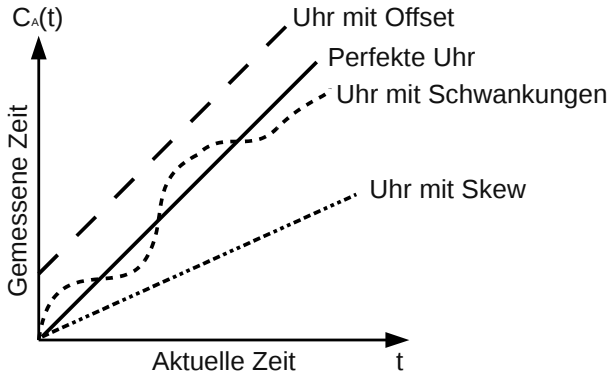


Abbildung 3.16: Die Beziehung zwischen aktueller und gemessener Zeit (nach [VDMC08, Seite 195])

Jeder Sensorknoten verwaltet seine eigene Uhr, welche aus Hardware- und Softwarekomponenten besteht. Ein verbauter Quarzoszillator schwingt mit einer bekannten Frequenz und löst nach einer festen Anzahl an Schwingungen eine Unterbrechung aus. Die Anzahl der Unterbrechungen wird in der Software mitgezählt. Eine solche Uhr auf dem Sensorknoten A wird C_A genannt und die von C_A gemessene Uhrzeit zum aktuellen Zeitpunkt t ist $C_A(t)$ [GvHS05].
Wie in Abbildung 3.16 zu sehen, ist die gemessene Zeit $C_A(t) \neq t$. Hierfür gibt es drei Gründe. Die Sensorknoten werden nicht alle zum selben Zeitpunkt eingeschaltet, wodurch es zu einer Offset genannten Zeitverschiebung zwischen den Uhren kommt. Der Offset ist definiert als die Differenz der Uhren zweier Sensorknoten, also Offset $O_(t) = C_A(t) - C_B(t)$. Ferner vollziehen die verbauten Quarze keine perfekte Schwingung. Sie unterliegen Schwankungen, die sich für ein großes Zeitintervall jedoch gegenseitig aufheben. Außerdem ist die Frequenz der Schwingung auf verschiedenen Sensorknoten, bedingt durch ungenaue Fertigung und Alterungsprozesse, nicht identisch. Die Frequenzabweichung zweier Sensorknoten wird als Skew bezeichnet und ist definiert als die Differenz der Takte zweier Sensorknoten über einen Zeitraum, also Skew $S_(t) = \frac{\delta C_A(t)}{\delta t} - \frac{\delta C_B(t)}{\delta t}$ [GvHS05][VDMC08, Seite 195]. Folglich gilt in der Regel für zwei verschiedene Sensorknoten $C_A(t) \neq C_B(t)$.
Dieser Abschnitt beschäftigt sich mit Protokollen zur Synchronisation der Sensorknotenuhren. Um die enorme Bedeutung der Zeitsynchronisation bei Sensornetzwerken herauszustellen, werden zunächst Anwendungen genannt und klassifiziert. Darauffolgend werden Designkriterien für Synchronisationsprotokolle vorgestellt. Anschließend wird in Unterabschnitt 3.7.1 die Funktionsweise der Zeitsynchronisation beschrieben, wobei insbesondere auf die möglichen Fehlerquellen eingegangen wird. Abschließend werden

in den Unterabschitten 3.7.2 bis 3.7.4 drei Zeitsynchronisationsprotokolle vorgestellt. Diese sind zum einen die bekanntesten Zeitsynchronisationsprotokolle für Sensornetzwerke, namentlich die Reference Broadcast Synchronization (RBS) sowie das Timing-sync Protocol for Sensor Networks (TPSN), und zum anderen das in dem in dieser Arbeit entworfenen Sensornetzwerk verwendete Flooding Time Synchronization Protocol (FTSP) [MKSL04].
Zeitkritische Anwendungen kann man mit der in Abbildung 3.17 gegebenen Klassifizierung in drei Klassen einteilen [Sto05, Seite 199ff]. Für die Anwendungen ist dabei nicht nur der reine Zugriff auf die Zeit wichtig, sondern insbesondere, dass die Uhren der einzelnen Sensorknoten miteinander synchronisiert sind.

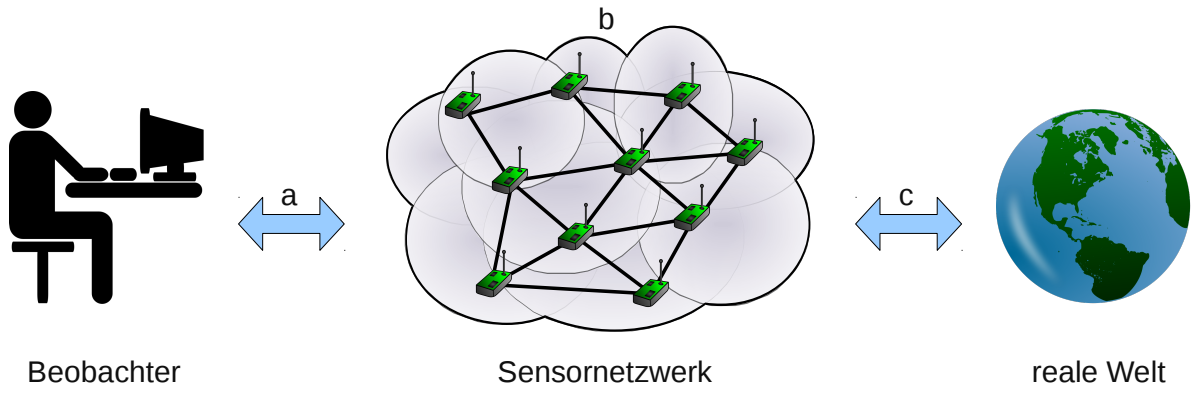


Abbildung 3.17: Die Klassifizierung der Anwendungen für Zeitsynchronisation in Sensornetzwerken (nach [Sto05, Seite 200])

Die erste Klasse (a) besteht aus Anwendungen, die einem externen Beobachter als Schnittstelle zur *Interaktion mit dem Sensornetzwerk* dienen. Dieser richtet Aufgaben an das Sensornetz, fragt Daten ab und nimmt Managmentfunktionen wahr. Er kann sowohl ein Benutzer als auch ein autonom agierendes Computersystem sein. Gestellte Aufgaben werden oftmals an einen Zeitpunkt oder ein Zeitfenster gebunden und sollen dann auch auf den einzelnen Sensorknoten gleichzeitig und zur angegebenen Zeit ausgeführt werden. Die Werte der erfassten Umweltgrößen müssen einen Zeitstempel erhalten, welcher für Messungen zum selben Zeitpunkt identisch ist. Nur so kann man die Daten später einem Zeitpunkt zuordnen, sie chronologisch sortieren und verschiedene Messgrößen desselben Ortes sowie dieselben Messgrößen verschiedener Orte miteinander in Zusammenhang bringen. Dies bildet für den Beobachter die Grundlage für eine aussagekräftige Auswertung der Daten.
Die zweite Klasse (b) besteht aus Anwendungen zur *Interaktion der Sensorknoten* untereinander. Oftmals werden Daten verschiedener Sersorknoten auf dem Weg von der Quelle zur Senke aggregiert. Voraussetzung hierfür sind exakt miteinander synchronisierte Uhren, damit nicht fälschlicherweise gleiche Messwerte von unterschiedlichen Zeitpunkten zusammengeführt werden. Des Weiteren wird eine synchronisierte Zeit von Netzwerkfunktionen benötigt. Es kann immer nur ein Sensorknoten gleichzeitig auf einem Funkkanal senden, ansonsten kommt es zu Kollisionen. Um dies zu unterbinden, wird beim Zeitmultiplexverfahren jedem Sensorknoten ein Timeslot zugeordnet, in dem er senden darf. Darüber hinaus wechselt der Funkchip, um Energie einzusparen, zeitweise in den Schlafmodus. Die Schlafzyklen der einzelnen Sensorknoten müssen aufeinander abgestimmt sein. Ferner nutzen auch andere Komponenten, wie die Sensorik oder der Hintergrundspeicher, Energiesparfunktionen. Zu all dem kommen noch einzelne Anwendungen für Spezialaufgaben hinzu. Als Beispiel seien hier Positionsbestimmungsverfahren genannt, bei denen Signallaufzeiten gemessen und Zeitstempel zwischen Sensorknoten ausgetauscht werden müssen.
Die dritte Klasse (c) besteht aus Anwendungen, die dem Sensornetzwerk zur *Interaktion mit der realen Welt* dienen. Wird dasselbe Ereignis von mehreren Sensorknoten erfasst, so müssen die einzelnen Daten fusioniert werden. Ein physikalisches Ereignis darf nicht durch eine parallele Erfassung von mehreren Sen-

sorknoten zu mehreren Ereignissen werden. Ein weiterer Aspekt bei der Datenfusion ist das Zusammenführen von unterschiedlichen Messgrößen zu einem Zeitpunkt oder derselben Messgröße zu verschiedenen Zeitpunkten. Als Beispiel für letzteres kann die Berechnung der Geschwindigkeit aus den von zwei Sensorknoten zu zwei Zeitpunkten erfassten Positionen eienes Objekts genannt werden. Weitere Anwendungen zur Interaktion mit der realen Welt sind bei der Ansteuerung von Aktoren zu finden. Komplexe und im Raum weit verteilte Steueroperationen werden von mehreren miteinander kooperierenden Sensorknoten durchgeführt. Hierzu ist ebenfalls eine einheitliche Sicht der Zeit die Grundvorraussetzung.
Zur Zeitsynchronisation im Internet hat sich das Network Time Protocol (NTP) durchgesetzt. Hierbei synchronisieren die Clients ihre Uhren mit der des Servers in einer Größenordnung von Millisekunden durch eine statistische Auswertung der Umlaufzeiten der Zeitstempelpakete. Bei dem in Sensornetzwerken typischen Funkkanal kommt es in der MAC-Schicht bei jedem Hop zu nichtdeterministischen Verzögerungen in der Größenordnung von mehreren 100 Millisekunden. Das NTP ist so nur bedingt für Sensornetzwerke geeignet [MKSL04].
Bei dem Entwurf von Zeitsynchronisationsprotokollen sind laut [Bou09, Seite 507f] verschiedene Designkriterien zu beachten. Dort wird eine Metrik zur Evaluation gegeben, wobei nicht alle Punkte gleichzeitig zu erfüllen sind. Je nach Anforderung der Anwendung besteht ein Trade-off zwischen den einzelnen Punkten.

Synchronisationsgenauigkeit Die von einem Synchronisationsprotokoll verlangte Genauigkeit richtet sich maßgeblich nach der Aufgabe, die mit den synchronisierten Uhren erfüllt werden soll. Es wird zwischen drei Ebenen unterschieden: Erstens eine reine zeitliche Sortierung der Daten, Events und Aufgaben; zweitens eine Synchronisation, bei der die einzelnen Sensorknoten einen Überblick über Skew und Offset der Nachbarknoten besitzen; die dritte und komplexeste Ebene ist die Synchronisation der Uhren gegenüber einer globalen Zeit innerhalb des Netzwerkes.

Effizienz Selbstverständlich unterliegen auch die Zeitsynchronisationsprotokolle den in Abschnitt 3.4 ausführlich besprochenen Ressourcenbeschränkungen. Sie müssen diese bei ihren Konzepten zur Synchronisation und bei der Implementierung mit berücksichtigen. Außerdem wird eine möglichst kurze Zeitspanne zur Synchronization der Uhren aus einem nicht synchronisierten Zustand heraus als effizient bezeichnet.

Netzwerkdynamik Ungeachtet dessen, ob die Sensorknoten zufällig oder deterministisch aufgestellt wurden, ist die Netzwerktopologie dynamisch. Gründe hierfür sind mobile Sensorknoten, eine dynamische Partitionierung des Netzwerkes zur Laufzeit und durch leere Batterien ausfallende Sensorknoten. Die Zeitsynchronisation muss daher auch in sich verändernden Netzwerktopologien zuverlässig funktionieren.

Lebensdauer Die Lebensdauer der Synchronisation bezeichnet den Zeitraum, in dem die Uhren nach erfolgter Synchronisation als synchron angesehen werden. Es wird unterschieden zwischen kurzzeitig und persistent. Bei einer kurzzeitigen Lebensdauer findet die Synchronisation nur dann statt, wenn sie benötigt wird. Bei der persistenten Synchronisation findet eine initiale Synchronisation statt, die dann bis zum Ende der Lebensdauer des Sensorknotens immer wieder korrigiert wird. Hier sind zu jeder Zeit synchronisierte Uhren vorhanden.

Infrastruktur Sensorknoten werden unter Umständen zufällig in unbekannten und gefährlichen Gebieten ausgebracht. Sie müssen sich selber zu einem Netzwerk organisieren, und es kann nicht davon ausgegangen werden, dass eine Infrastruktur vorhanden ist. Es ist nicht, wie bei dem NTP, gewährleistet, dass eine korrekte Uhrzeit nur wenige Hops entfernt ist. Die Aufgabe der Zeitsynchronisation muss auch mit mobilen Sensorknoten funktionieren und sollte gut mit der Netzgröße skalieren.

Kosten und Größe Sensorknoten sind sowohl klein als auch preiswert, und so muss auch die Zeitsynchronisation sich an diese Limitierungen halten. Ein teurer und großer GPS-Empfänger kann nicht

auf jedem Sensorknoten installiert werden. Zusätzliche Hardware sollte, wenn möglich, vermieden oder nur auf einer geringen Anzahl der Sensorknoten eingesetzt werden.

Verfügbarkeit und Bereich Alle Sensorkonten im Sensornetzwerk können sich entweder mit einer globalen Zeit synchronisieren, oder eine kleine Gruppen von benachbarten Knoten synchronisieren sich untereinander zu einer lokalen Zeit. Nicht alle Anwndungen benötigen eine globale Sicht der Zeit, und gerade in großen Sensornetzwerken kann durch lokale Zeitzonen viel Energie und Bandbreite bei der Synchronisation gespart werden. Hierbei ist jedoch wichtig, dass für jeden Sensorknoten eine für seine Aufgaben valide Zeit zur Verfügung steht.

3.7.1 Fehlerquellen bei der Zeitsynchronisation

Die überwiegende Mehrheit der Synchronisationsprotokolle beruht auf dem Austausch von Nachrichten zwischen den Sensorknoten. Eine Kommunikation zwischen den Sensorknoten wird nur dann nicht benötigt, wenn jeder Sensorknoten sich mit einer externen Infrastruktur wie z. B. dem GPS synchronisiert. Dies ist jedoch nur in wenigen Sensornetzwerken der Fall. Bei der Kommunikation der Sensorknoten untereinander kommt es zu nichtdeterministischen Fehlern, die in diesem Unterabschnitt näher betrachtet und erläutert werden. Unterschiedliche Synchronisationsprotokolle beziehen dabei unterschiedliche Fehlerquellen mit in ihre Synchronisation ein. Je mehr Fehlerquellen einbezogen werden, um so genauer aber auch aufwändiger und damit ressourcenintensiver ist das Synchronisationsverfahren.
Wie in Abbildung 3.18 zu sehen, kann die Kommunikation ihrem zeitlichen Verlauf nach in mehrere Abschnitte unterteilt werden. Dieses ist das feingranulärste Schema in aktuellen Fachaufsätzen und wurde auch von den in dieser Arbeit vorgestellten Zeitsynchronisationsprotokollen verwendet [MKSL04] [EGE02] [GKS03]. Im Folgenden werden die einzelnen Zeitintervalle der Kommunikation beschrieben:

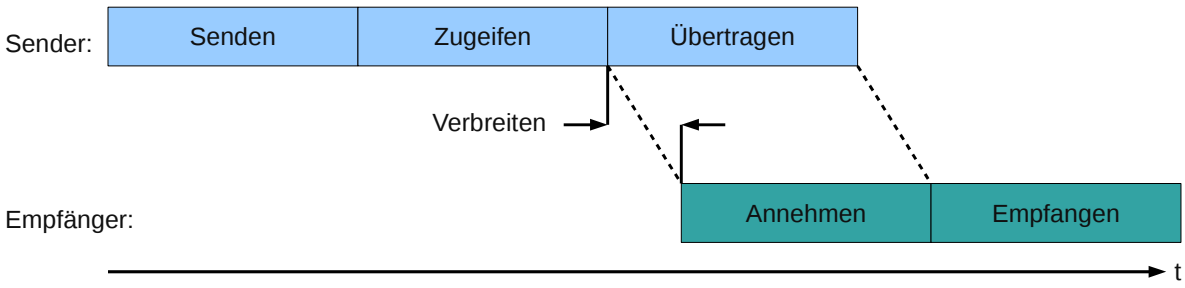


Abbildung 3.18: Die Fehlerquellen bei der Zeitsynchronisation (nach [MKSL04])

Zeit zum Senden Wenn sich ein Sensorknoten zum Versenden einer neuen Nachricht entscheidet, vergeht eine Zeitspanne von der ersten Intention des Sendens bis zum Eintreffen der Nachricht in der MAC-Schicht. Dies umfasst zum einen das Packen der Daten in ein Paket und zum anderen das Durchreichen durch alle Schichten des Netzwerkprotokolls. Je nach verwendetem Netzwerkprotokoll und Betriebssystem werden nichtdeterministische Zeiten für Kontextwechsel, Unterbrechungsbehandlungen, Systemaufrufe und zum Übertragen des Pakets an den Funkchip benötigt. Auch die aktuelle Prozessorauslastung durch andere Anwendungen spielt eine Rolle. Diese Zeit kann in einer Größenordnung von mehreren hundert Millisekunden liegen und ist häufig bereits größer als die zu erreichende Genauigkeit der Zeitsynchronisation.

Zeit zum Zugreifen Sobald das Paket in der MAC-Schicht auf der Senderseite eingetroffen ist, muss es eine Zeitspanne bis zum Versenden des ersten Bits warten. Einen Funkkanal kann immer nur ein Sender zur selben Zeit benutzen, und, bis der Kanal frei ist, können Millisekunden bis mehrere Sekunden

vergehen. Dies ist sowohl vom verwendeten MAC-Protkoll als auch von der derzeitigen Netzauslastung abhängig. Die Zeit zum Zugreifen auf den Kanal ist die am wenigsten deterministische Größe bei dem Austausch eines Datenpaketes zwischen zwei Sensorknoten.

Zeit zum Übertragen Dies ist die Zeit, die der Funkchip des Senders benötigt, um das Paket Bit für Bit auf der physikalischen Schicht über den Funkkanal zu versenden. Es wird eine Zeitspanne in der Größenordnung von mehreren zehn Millisekunden benötigt, welche jedoch mit der bekannten Paketgröße und der Übertragungsrate abgeschätzt werden kann. Übernimmt der Funkchip nicht die Aufgabe des MAC-Layers, so können jedoch nicht berechenbare Unterbrechungen im Mikrocontroller für weitere Verzögerung sorgen.

Zeit zum Verbreiten Es vergeht eine Zeitspanne zwischen dem Verlassen der einzelnen Bits auf der Senderseite bis zum Eintreffen auf der Empfängerseite. Hierbei vergeht bei der nahezu mit Lichtgeschwindigkeit stattfindenden Funkkommunikation weniger als eine Mikrosekunde bei Übertragungen über nicht mehr als 300 Meter. Verglichen mit den anderen Zeitspannen kann diese somit vernachlässigt werden.

Zeit zum Annehmen Analog zu der Zeit zum Übertragen benötigt der Empfänger eine gewisse Zeit um die Nachricht Bit für Bit entgegenzunehmen, bevor er sie an die MAC-Schicht weiterreichen kann. Die Zeit zum Annehmen ist genauso groß wie die Zeit zum Übertragen und überschneidet sich mit dieser.

Zeit zum Empfangen Nachdem alle Bits bei der MAC-Schicht des Empfängers eingetroffen sind, vergeht eine Zeitspanne, bis die Daten in der Anwendungsschicht angekommen sind. Es muss zunächst ein Paket gepackt werden, welches dann die einzelnen Schichten des Netzwerkstapels durchläuft. Hierbei können wieder die unter *Zeit zum Versenden* beschriebenen betriebssystemtypischen Verzögerungen auftreten. Diese Zeitspanne kann minimiert werden, wenn frühzeitig der Eintreffzeitpunkt eines neuen Pakets in einer tiefen Ebene des Betriebssystems festgehalten wird. Dies könnte z. B. innerhalb der Unterbrechungsbehandlung des Netzwerkgerätetreibers geschehen.

Darüber hinaus kommt es noch zu einem Fehler, hervorgerufen durch das Bytealignment, der je nach verwendeter Hardware und Implementierung der Zeit zum Annehmen oder Empfangen zugeordnet werden kann. Einige Funkchips sind nicht in der Lage, das Bytealignment des ankommenden Datenstroms zu erfassen. In diesem Fall muss der Netzwerkstapel anhand des bekannten Synchronisationsbytes die einkommende Nachricht an die korrekte Position verschieben. Hierdurch kommt es zu einer Verzögerung von bis zu 400 µs, welche jedoch mit dem Bitoffset und der Übertragungsrate bestimmt werden kann [MKSL04].

3.7.2 Reference Broadcast Synchronization (RBS)

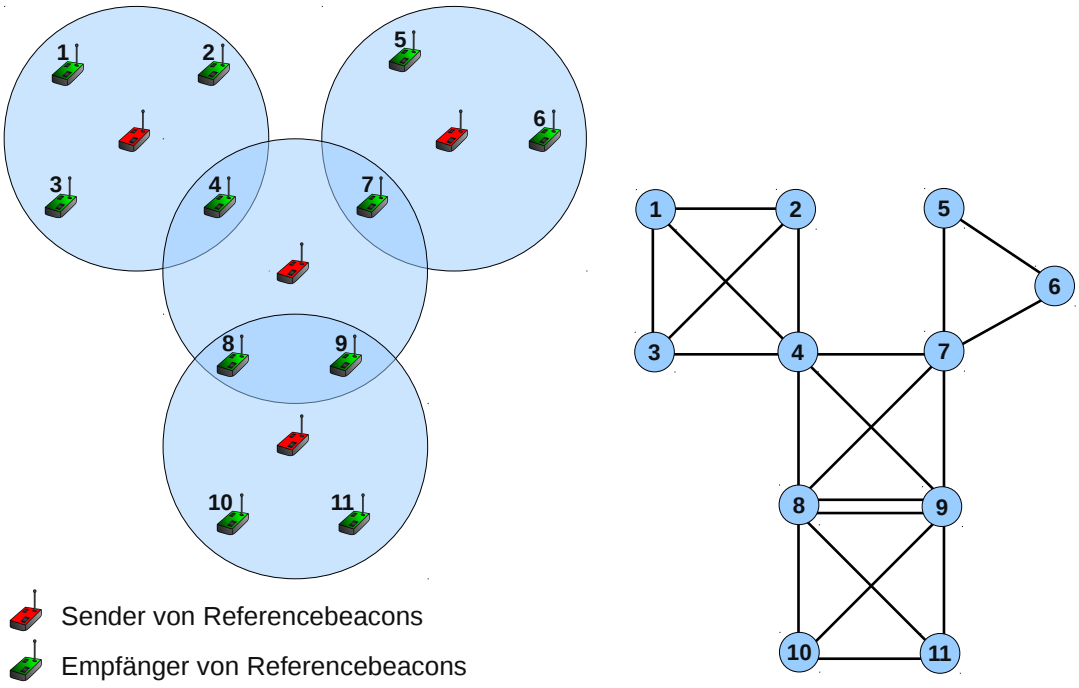
Die Reference Broadcast Synchronisation (RBS) wurde von Elson et al. im Jahr 2002 vorgeschlagen und zählt damit zu den ersten Synchronisationsprotokollen speziell für Sensornetzwerke [EGE02]. Bei der RBS senden die Sensorknoten sogenannte Referenzbeacons mit Hilfe eines Broadcasts an ihre Nachbarknoten. Das Referenzbeacon enthält keinen Zeitsempel, der Inhalt des Pakets spielt für RBS keinerlei Rolle. Stattdessen halten die Empfänger den Empfangszeitpukt fest und nutzen diesen als Referenz zum Vergleich ihrer Uhren. Wichtig ist hierbei, dass der Broadcast vom Funkchip implementiert wird. Nur so ist gewährleistet, dass alle empfangenden Sensorknoten denselben Zeitpunkt festhalten. Durch den Verzicht auf Zeitinformationen innerhalb der Synchronisationspakete wird der kritische Pfad auf die Fehlerquellen beim Empfänger reduziert. Lediglich die Zeit zum Annehmen und Empfangen spielt eine Rolle, und die besonders große und nichtdeterministische Zeit zum Senden ist komplett eleminiert. Vergleiche hierzu 3.7.1. Darüber hinaus muss der Sender lediglich einen Broadcastping starten, was vergleichsweise ressourcenschonend ist. RBS realisiert laut Aussage der Autoren ein energieeffizientes Synchronisationsprotokoll mit einer Genauigkeit von ca. fünf Mikrosekunden.

Funktionsweise von RBS

Gegeben sei eine Menge benachbarter Sensorknoten. Einer davon sendet ein Referenzbeacon, welches von den Knoten A und B empfangen wird. Sie halten den Empfangszeitpunkt $C_A(t)$ und $C_B(t)$ im Kernel innerhalb der Unterbrechungsbehandlungsroutine des Funkchipgerätetreibers fest, um die Zeit zum Empfangen möglichst klein zu halten. Danach senden sie diese Information an den jeweils anderen Sensorknoten. Somit besitzen sie beide sowohl den eigenen Empfangszeitpunkt als auch den des anderen und können den Offset als Differenz der Werte bestimmen. Die Sensorknoten sind jetzt relativ zueinander synchronisiert. Dies ist für viele Anwendungen in Sensornetzen bereits ausreichend. Um einen Fehler durch seltene Verzögerungen bei der Zeit zum Empfangen zu vermeiden, z. B. hervorgerufen durch eine kurzzeitig hohe Prozessorauslastung, wird eine Sequenz von m Referenzbeacons versendet. Der Offset O wird als Mittelwert der einzelnen Offsets bestimmt. Allgemein gilt damit für die Offsets von n empfangenden Sensorknoten:

$$\forall A \in n, B \in n: \ \mathrm{O}[A,B] = \frac{1}{m}\sum_{k=1}^{m}(C_{B,k}(t_k) - C_{A,k}(t_k))$$

Bisher wurde nur die Kompensation des Offsets betrachtet. Um das bisherige Modell um eine Kompensation des Skews zu erweitern, wird die oben gegebene Gleichung angepasst. Anstatt einen Mittelwert über alle Offsets zu bilden, wird eine lineare Regression der kleinsten Abweichungsquadrate durchgeführt. So erhält man eine Gerade, die Auskunft über die Frequenz der eigenen Uhr relativ zu der des anderen Knotens gibt.



(a) Ein Multi-Hop-Netzwerk mit vier Zonen relativ zueinander synchronisierter Sensorknoten.

(b) Der Graph der Zeitpfade des in Abbildung 3.19(a) dargestellten Multi-Hop-Netzwerks. Jede Kante spiegelt eine Zeitreferenz wider.

Abbildung 3.19: Die Reference Broadcast Synchronization in Multi-Hop-Netzwerken (nach [EGE02])

Auf diese Weise können jedoch nur Zeitrelationen für benachbarte Sensorknoten berechnet werden. Um dennoch eine Synchronisation im gesamten Netzwerk zu erreichen, werden Pfade zur Konvertierung gesucht. In Abbildung 3.19(a) ist ein Multi-Hop-Netzwerk mit vier Referenzbeaconsendern dargestellt. Die

Sensorknoten 4, 7, 8 und 9 empfangen die Referenzbeacons von jeweils zwei Sendern. Sie können daher zwischen den anderen Sensorknoten vermitteln. Will beispielsweise Knoten 1 die Zeit eines erfassten Ereignisses in die Zeit des Knotens 7 konvertieren, so wird sie erst in die Zeit des Knotens 4 konvertiert und dann in die Zeit von Knoten sieben. Um in einem Netzwerk solche Zeitpfade zur Konvertierung zu organisieren, wird ein wie in Abbildung 3.19(b) zu sehender Graph konstruiert. In diesem sollte möglichst der kürzeste Weg zwischen den Punkten gefunden werden, da sich bei jeder Konvertierung die Ungenauigkeiten addieren.
Diese Vorgehensweise skaliert nicht gut mit der Größe des Netzwerkes. Die Autoren schlagen vor, dass die Zeitkonvertierung Bestandteil des Routingprotokolls sein könnte. In diesem Fall werden die erfassten Daten auf dem Weg von der Quelle zur Senke bei jedem Hop in die Zeit des Empfängers konvertiert. Alle bei der Senke eintreffenden Daten liegen dann in der Zeit der Senke vor.
Ein weiterer Punkt ist die Synchronisierung relativ zu einer Zeit wie der koordinierten Weltzeit (UTC). RBS bietet von Haus aus eine Möglichkeit, die Knoten relativ zu einer anderen Zeit zu synchronisieren. Es liegt daher nahe, einen Sensorknoten mit einem GPS-Empfänger auszustatten und allen Sensorknoten im Netzwerk bekannt zu machen, dass man sich bei diesem einen Sensorknoten mit der UTC-Zeit synchronisieren kann.

3.7.3 Timing-sync Protocol for Sensor Networks (TPSN)

Das Timing-sync Protocol for Sensor Networks (TPSN) wurde von Ganeriwal et al. im Jahr 2003 vorgeschlagen [GKS03]. Im Gegensatz zu RBS beruht es nicht auf einer Empfänger-Empfänger-Synchronisierung, sondern verfolgt den klassischen Ansatz der Sender-Empfänger-Synchronisierung. Die Autoren sind der Ansicht, dass ein Handshake zwischen zwei Sensorknoten mit einem direkten Austausch von Zeitinformationen zwischen zwei Sensorknoten der bessere Weg ist. Dies begründen sie mit der Möglichkeit moderner Funkchips zum Zeitstempeln der Pakete direkt in der MAC-Schicht. So nimmt auch bei TPSN die Zeit zum Senden keinen Einfluss auf die Genauigkeit. Des Weiteren wird bei TPSN nicht die Zeit eines Sensorknotens relativ zu einem anderen berechnet, sondern eine globale Zeit ist auf jedem Sensorknoten direkt und zu jeder Zeit verfügbar.
TPSN erzeugt eine sich selbst organisierende Ebenenhierarchie von Deamons ähnlich zu NTP, die in ihrer eigenen Ebene als Client und nach unten hin als Server agieren. Jeder Sensorknoten befindet sich in genau einer Ebene und kann mit mindestens einem Sensorknoten der höherliegenden Ebene direkt kommunizieren. Auf der obersten Ebene befindet sich nur ein einzieger Sensorknoten, der Rootnode genannt wird. Dieser stellt die Masteruhr dar und kann z. B. mit einem GPS-Empfänger ausgestattet werden, wenn sich das Netzwerk gegenüber der UTC-Zeit synchronisieren soll.

Funktionsweise von TPSN

Die Zeitsynchronisation mit TPSN wird in zwei Phasen realisiert. In der Level-Discovery-Phase wird die Hierarchie aufgebaut, und in der Synchronisationsphase werden die einzelnen Uhren miteinander synchronisiert.
Die Level-Discovery-Phase beginnt, sobald das Netzwerk augfgestellt wird. Der Rootnode ist vordefiniert und weist sich selber die Ebene 0 zu. Danach sendet er ein Level-Discovery-Paket mit seiner eindeutigen Adresse und seiner Ebene an die Broadcastadresse. Alle Sensorknoten, die dieses Paket erhalten, weisen sich selber die erhaltene Ebene + 1 zu und versenden selbst ein neues Level-Discovery-Paket mit ihrer Adresse und Ebenennummer. Hat ein Sensorkonten eine Ebene erhalten, so ignoriert er fortan alle Level-Discovery-Pakete. Dieser Vorgang wiederholt sich, bis sich alle Sensorknoten einer Ebene zugewiesen haben. Es ist eine Hierarchie entstanden wie in Abbildung 3.20.
Es kann passieren, dass ein Sensorknoten keiner Ebene zugewiesen wird. Dies kann der Fall sein, wenn es zu Kollisionen auf der MAC-Schicht kam oder der Sensorknoten erst eingeschaltet wurde, als die Level-Discovery-Phase bereits abgeschlossen war. Aus diesem Grund wartet der Sensorknoten nach dem Ein-

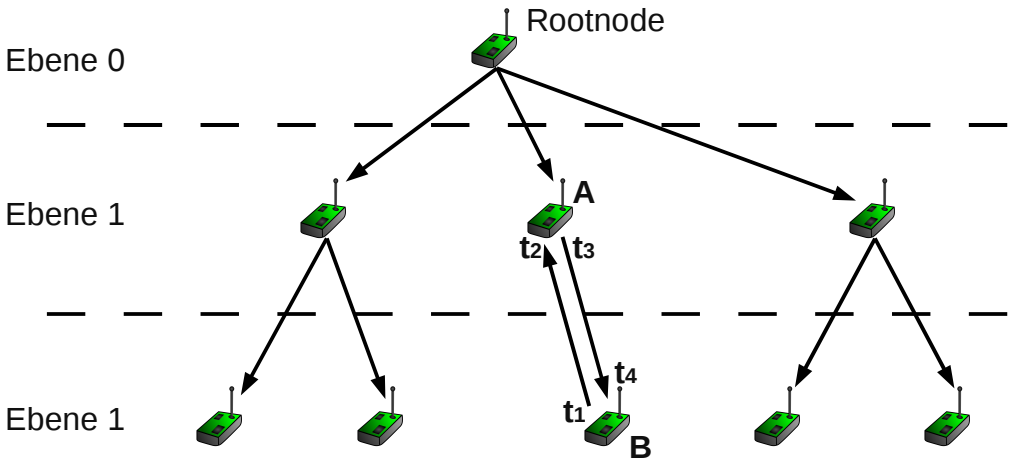


Abbildung 3.20: Der hierarchische Aufbau des TPSN (nach [VDMC08, Seite 229])

schalten immer eine gewisse Zeit. Hat er dann keine Ebene zugewiesen bekommen, so verschickt er ein Level-Request-Paket. Die Nachbarknoten antworten mit einem Level-Discovery-Paket, und der Sensorknoten weist sich die kleinste erhaltene Ebene + 1 zu. Analog handelt der Sensorknoten, wenn er zu einem späteren Zeitpunkt keinen Kontakt mehr zu der höheren Ebene besitzt. Geht der Kontakt zum Rootnode verloren, so wählen die Sensorknoten der Ebene 1 einen neuen Rootnode, was erneut zu einer Level-Discovery-Phase führt und eine neue Hierarchie formt.
Nach einer gewissen Zeit initiiert der Rootnode die Synchronisationsphase, indem er ein Zeitsynchronisationspaket schickt. Dieses enthält seine Ebenennummer 0. Alle Knoten, die dieses Paket erhalten und der Ebene 1 angehören, warten eine zufällige Zeit ab, um Kollisionen zu vermeiden. Danach starten sie eine Zwei-Wege-Synchronisation mit dem Rootnode, die im folgenden Absatz beschrieben wird. Die Sensorknoten der Ebene 2 empfangen ebenfalls zumindest die Nachricht des Sensorknotens der Ebene 1, da sie ebenfalls innerhalb seiner Funkreichweite sind. Sie ignorieren die Nachricht als solche, nutzen sie jedoch als ihr Startsignal. Sie warten eine zufällige Zeit plus die Zeit, welche die Sensorknoten der Ebene 1 zum Synchronisieren benötigen, um dann mit einem Sensorknoten der Ebene 1 ebenfalls eine Zwei-Wege-Synchronisation durchzuführen. Dieser Vorgang wiederholt sich, bis sich alle Ebenen synchronisiert haben.
Die Zwei-Wege-Synchronisation wird von Sensorknoten A gestartet, der seine Uhr an die Uhr des Sensorknotens B angleichen möchte. Wie in Abbildung 3.20 zu sehen, sendet A hierzu ein Synchronisationspulspaket an B, welcher die Ebenennummer von B und den Zeitpunkt des Versendens $\mathrm{C}_A(t_1)$ enthält. B erhält das Paket zum Zeitpunkt $\mathrm{C}_B(t_2)$, wobei gilt: $\mathrm{C}_B(t_2) = \mathrm{C}_A(t_1) + \Delta + \mathrm{d}$. Dabei bezeichnet Δ den Offset der Knoten und d die Zeit zum Verbreiten, Annehmen und Empfangen. Es wird hierbei angenommen, dass die Zeit zum Übertragen deckungsgleich mit der Zeit zum Annehmen ist und somit nicht berücksichtigt werden muss. Zum Zeitpunkt $\mathrm{C}_B(t_3)$ sendet B ein Acknowledgementpaket an A, welches die Ebenennummer von B und die Zeiten $\mathrm{C}_A(t_1)$, $\mathrm{C}_B(t_2)$ und $\mathrm{C}_B(t_3)$ enthält. A merkt sich den Zeitpunkt des Eintreffens $\mathrm{C}_A(t_4)$ und bestätigt B mit einem Acknowledgementpaket den Erhalt. Nun kann A den Offset und die Zeit zum Übertragen berechnen und seine Uhr synchronisieren. Es gilt:

$$\Delta = \frac{(C_B(t_2) - C_A(t_1)) - (C_A(t_4) - C_B(t_3))}{2} \; und \; d = \frac{(C_B(t_2) - C_A(t_1)) + (C_A(t_4) - C_B(t_3))}{2}$$

Hierbei wird davon ausgegangen, dass während der kurzen Zeitspanne des Vorgangs der Offset der Uhren von A und B konstant bleibt. Darüber hinaus wird der Skew bei der Synchronisation nicht berücksichtigt. Stattdessen wird die Synchronisationsphase periodisch wiederholt. Um die Periodendauer festzulegen,

muss man bei der zu verwendenden Sensorplattform den maximalen Drift der Sensorknotenuhren empirisch ermitteln. Bei einer gegebenen maximalen Synchronisationsungenauigkeit kann man dann berechnen, in welchen Zeitintervallen die Synchronisationsphase wiederholt werden muss. Die Autoren haben für eine Genauigkeit von 10 ms ein Intervall von 34 min für Berkleysensorknoten ermittelt.

3.7.4 Flooding Time Synchronization Protocol (FTSP)

Das Flooding Time Synchronization Protocol (FTSP) wurde von Maróti t al. im Jahr 2004 vorgestellt [MKSL04]. Es stellt analog zu TPSN eine globale Uhrzeit zur Verfügung und verfolgt den Ansatz einer initalen Synchronisation, welche die gesamte Lebensdauer des Sensornetzwerks aufrechterhalten wird. Wie der Name FTSP bereits impliziert, erreicht dieses Protokoll eine Zeitsynchronisation, indem es in regelmäßigen Abständen das gesamte Netzwerk mit Synchronisationsnachrichten flutet. Hierdurch ist das FTSP besonders robust gegen Kommunikationsfehler und dynamische Netzwerktopologien. Die Synchronisationsnachrichten werden wie bei dem TPSN in der MAC-Schicht mit einem Zeitstempel versehen, so dass die Zeit zum Senden außer acht gelassen werden kann. Die Autoren behaupten ebenfalls von ihrem Protokoll, dass es besonders ressourcenschonend sei, insbesondere bei der Bandbreite der Kommunikation. Ein Vergleich der Synchronisationsprotokolle RBS und TPSN mit dem FTSP findet am Ende dieses Unterabschnittes statt.
Das FTSP wird für das in dieser Arbeit entworfene Sensornetzwerk zur Erfassung eines Temperaturprofils in Wohn- und Büroräumen zur Zeitsynchronisation benutzt. Begründet wird diese Auswahl in Unterabschnitt 4.1. Dieses Synchronisationsverfahren wird daher besonders ausführlich besprochen, und die Ausführungen beziehen sich auf die TinyOS beiliegende Implementierung. Die Quelltexte sind in voller Länge auf der beiliegenden CD enthalten, siehe A.

Funktionsweise von FTSP

Das FTSP synchronisiert die Uhren aller Sensorknoten im Sensornetzwerk zu der Uhr eines Root genannten Sensorknotens. Hierbei ist die lokale Zeit des Roots die globale Zeit des Sensornetzwerks.
Zunächst wird die Synchronisation der Sensorknoten mit direkter Funkverbindung zu dem Root betrachtet. Hierzu sendet der Root in periodischen Abständen Broadcastnachrichten mit einem Zeitstempel des Versendezeitpunktes.
TinyOS bietet ein Modul zum Versenden spezieller Synchronisationspakete an. Wie bereits beschrieben wurde, findet das Hinzufügen des Zeitstempels beim Versenden und das Erfassen der Ankunftszeit beim Empfangen auf der MAC-Schicht innerhalb des Kernels statt. Es ist somit auch die Aufgabe des Betriebssystems, diese Operationen durchzuführen. Da alle Synchronisationsprotokolle ohne externe Infrastruktur denselben im Unterabschitt 3.7.1 beschriebenen Fehlerquellen bei der Funkübertragung unterliegen, bietet es sich für das Betriebssystem an, eine Schnittstelle zum Versenden von Synchronisationspaketen bereitzustellen. Die Implementierungen der einzelnen Synchronisationsprotokolle bleiben somit plattformunabhängig, und die Forschungen zur Bereinigung der Zeitfehler bei der Übertragung können zentral an einer Stelle umgesetzt werden.
In TinyOS existiert die Schnittstelle TimeSyncAMSend, mit der TimeSyncPackets versendet werden können [MS08]. Wird von einer Anwendung ein Event zum Versenden eines solchen Pakets ausgelöst, so merkt sich der Eventhandler die korrespondierende lokale Zeit und stößt das Versenden an. Sobald der Funkchip mit der Übertragung beginnt, löst er einen Interrupt aus, und in der Unterbrechungsbehandlungsroutine wird die Zeitdifferenz zwischen Event und Start der Übertragung berechnet. Diese Differnz wird dem Paket beigefügt. Trifft das Paket beim Empfänger ein, so merkt er sich in der Unterbrechungsbehandlungsroutine den Eintreffzeitpunkt. Anschließend wird die mitgeschickte Zeitdifferenz vom Eintreffzeitpunkt abgezogen und das Ergebnis in den Metadaten des Pakets gespeichert. Diese Zeit repräsentiert den Zeitpunkt des Sendeevents beim Sender in der lokalen Zeit des Empfängers.

Für diese Implementierung muss der Funkchip das Verändern des Endes der Nachricht nach dem Start der Übertragung unterstützen. Tut er dies nicht, wird alternativ dem Synchronisationspaket nicht die Zeitdifferenz zwischen Sendeevent und Start der Übertragung beigefügt, sondern lediglich die Sendeeventzeit. Sobald die Übertragung startet, merkt sich der Sender den Zeitpunkt, sendet diesen in einer zweiten Nachricht hinterher, und die oben beschriebene Differenz wird erst beim Empfänger gebildet.
Für beide Varianten wurde somit die Zeit zum Senden, Zugreifen und Empfangen komplett eliminiert, und die Zeit zum Annehmen wird auf die Dauer einer Unterbrechungsbehandlung begrenzt. Die Zeit zum Verbreiten wird nicht berücksichtigt, ist aber, wie bereits beschrieben, bei der Funkkommunikation bei Reichweiten unter 300 Metern zu vernachlässigen.
Für die Anwendungen, und hier im Speziellen das FTSP, verhalten sich diese Sychronisationspakete wie normale Pakete auch. Trifft jetzt bei den Sensorknoten ein vom Root gesendetes Paket ein, so wird es an die FTSP-Instanz weitergereicht. Diese entnimmt dem Paket den vom Root hinzugefügten Zeitstempel der globalen Zeit zum Sendeevent aus dem Datenbereich des Pakets und erfragt bei dem Betriebssystem die oben beschriebene lokale Zeit des Sendeevents. Eine solche Kombination von Zeitstempeln wird auch Synchronisationspunkt genannt. Aus ihm lässt sich durch Differenzbildung der Offset der Uhren bestimmen.
Alle Sensorknoten besitzen eine Tabelle, in der sie die letzten n (Defaultwert: n = 8) Synchronisationspunkte, bestehend aus der eigenen lokalen Zeit des Sendeevents beim Root und dem berechneten Offset, vorhalten. Mit Hilfe dieser Tabelle wird eine lineare Regression durchgeführt, um den Skew zu berechnen, welche bei jedem neu eintreffenden Synchronisationspaket neu durchgeführt wird.
FTSP bietet anderen Modulen eine Schnittstelle zum Zugriff auf die globale Zeit und zum Umrechnen zwischen lokaler und globaler Zeit an, dessen Implementierung in Listing 3.4 dargestellt wird. Um die lokale Zeit in die globale Zeit umzurechnen, wird der Durchschnitt aller Offsets und der durch die unterschiedlich schnell laufenden Uhren hervorgerufenen Fehler addiert, welcher sich aus der Zeitdifferenz der aktuellen lokalen Zeit und der mittleren Zeit aller Synchronisationspakete multipliziert mit dem Skew ergibt.

```
async command error_t GlobalTime.getGlobalTime(uint32_t *time)
{
   *time = call GlobalTime.getLocalTime();
   return call GlobalTime.local2Global(time);
}

async command error_t GlobalTime.local2Global(uint32_t *time)
{
   *time += offsetAverage + (int32_t)(skew * (int32_t)(*time -
      localAverage));
   return is_synced();
}

async command error_t GlobalTime.global2Local(uint32_t *time)
{
   uint32_t approxLocalTime = *time - offsetAverage;
   *time = approxLocalTime - (int32_t)(skew * (int32_t)(
      approxLocalTime - localAverage));
   return is_synced();
}
```

Listing 3.4: Die Schnittstellenimplementierung der Zeitschnittstelle vom FTSP

Im Folgenden wird jetzt beschrieben, wie FTSP in Multi-Hop-Netzwerken funktioniert.
In die Synchronisationsnachrichten wird neben der globalen Zeit ebenfalls die Sensorknotenadresse des Roots und eine Sequenznummer mit aufgenommen. In periodischen Abständen feuert auf jedem Sensorknoten ein Timer, der neue Synchronisationsnachrichten verschickt, wenn er selber der Root ist oder genügend Daten in seiner Tabelle besitzt, um davon auszugehen, dass er synchronisiert ist. Sensorknoten, die nicht der Root sind, setzen als Sequenznummer die letzte erhaltene Sequenznummer. Der Root erhöht

die Sequenznummer mit jedem Verschicken einer neuen Nachricht. Vergleiche hierzu Zeilen 9 bis 18 in dem Pseudocodelisting 3.5.

Erhält ein Sensorknoten eine neue Synchronisationsnachricht, so nimmt er sie nur bei einer kleineren oder gleichgroßen Rootadresse als von bisher erhaltenen Paketen entgegen, denn der Root ist per Definition immer der Sensorknoten mit der niedrigsten Sensorknotenadresse. In großen Sensornetzwerken sind häufig sehr viele Synchronisationsnachrichten unterwegs, und beim Flooding der Pakete kann es passieren, dass man ein und dasselbe Synchronisationspaket mehr als einmal erhält. Daher werden alle Pakete mit einer kleineren Sequenznummer als der aller bisher erhaltenen Synchronisationspakete ebenfalls abgelehnt. Anschließend wird anhand des angenommenen Synchronisationspakets die eigene Synchronität überprüft. Ist der Sensorknoten nicht mehr synchron zum Root, so wird die Tabelle mit den Synchronisationspunkten geleert. Abschließend wird der neue Synchronisationspunkt in die Tabelle aufgenommen und Offset sowie Skew neu berechnet. Vergleiche hierzu das Event *Radio.receive()* im Listing 3.5.

Betrachtet man das Protokoll nach der Aufstellung des Sensornetzwerks, so beginnt zunächst nur der Root, seine Nachbarn zu synchronisieren. Diese wiederum fangen selber damit an, Synchronisationspakete zu versenden, sobald sie genügend Daten besitzen. Dieser Prozess setzt sich fort, und es entsteht indirekt eine Baumstruktur ähnlich der beim TPSN – siehe dazu auch Abbildung 3.20. Durch das Zusammenspiel des Timers und des Events *Radio.receive()* wird danach weiterhin das gesamte Sensornetz periodisch mit Synchronisationsnachrichten geflutet und die Synchronität aufrecht erhalten.

Es bleibt noch die Frage zu klären, wie der Root gewählt wird. Erhält ein Sensorknoten ein Synchronisationspaket von einem Root niedriger Adresse als der eigenen, so setzt er die Variable *Heartbeat* auf 0. Im Eventhandler des periodischen Timers wird bei jedem Auslösen des Timers die Variable *Heartbeat* erhöht. Übersteigt sie den Wert ROOT_TIMEOUT, so wird angenommen, dass keine Verbindung zum Root besteht. Er ist entweder gestorben, es besteht keine Kommunikationsverbindung mehr zu ihm oder das Sensornetzwerk wurde gerade erst aufgestellt. In diesen Fällen übernimmt der Sensorknoten die Funtkion des Roots, bis er ein Synchronisationspaket eines anderen Knotens mit niedrigerer Adresse erhält.

```
event Timer.fired()
{
  ++heartBeats;

  if( myRootID != myID
    && heartBeats >= ROOT_TIMEOUT )
    myRootID = myID;

  if( numEntries >= NUMENTRIES_LIMIT
     || myRootID == myID ){

    msg.rootID = myRootID;
    msg.seqNum = highestSeqNum;
    Radio.send(msg);

    if( myRootID == myID )
      ++highestSeqNum;
   }
}

event Radio.receive(TimeSyncMsg *msg)
{
  if( msg->rootID < myRootID )
    myRootID = msg->rootID;
  else if( msg->rootID > myRootID
       || msg->seqNum < highestSeqNum )
    return;
```

```
28
    highestSeqNum = msg->seqNum;
30  if( myRootID < myID )
      heartBeats = 0;
32
    if( numEntries >= NUMENTRIES_LIMIT
34    && getError(msg) > TIME_ERROR_LIMIT )
      clearRegressionTable();
36
      addEntryAndEstimateDrift(msg);
38  }
```

Listing 3.5: Die wesentlichen Funktionen von FTSP in Multi-Hop-Netzwerken (angelehnt an [MKSL04]))

Vergleich von FTSP mit RBS und TPSN

In diesem Unterabschitt wird das FTSP mit den in den vorherigen Unterabschnitten 3.7.2 und 3.7.3 vorgestellten Synchronisationsprotokollen RBS und TPSN verglichen [MKSL04]. Dieser Vergleich bietet sich an, weil RBS und TPSN schon häufig in der Praxis verwendet wurden – es liegen somit auch Messdaten vor –, weil beide ebenfalls spezielle Synchronisationsprotokolle für Sensornetzwerke sind und weil beim Entwurf vom FTSP Ideen dieser beiden Protokolle aufgegriffen wurden.
RBS nutzt nur Zeitstempel auf Seiten des Empfängers und eliminiert dadurch die Fehlerquellen auf Seiten des Senders vollständig. FTSP fügt die Zeitstempel in der MAC-Schicht hinzu und ist damit nur unwesentlich ungenauer. TPSN ist in der Lage, auch die Zeit zum Verbreiten zu entfernen. In diesem Protokoll wird jedoch nicht die Zeit zum Senden kompensiert. Durch die Zwei-Wege-Synchronisation muss einer der Knoten von dem Empfangsmodus in den Sendemodus wechseln, was bei einigen Funkchips eine lage Zeit in Anspruch nimmt und somit Fehler verursacht. Ein weiterer Nachteil der Zwei-Wege-Synchronisation ist der erhöhte Kommunikationsaufwand. Bei FTSP und RBS können mit Broadcastpakten alle Nachbarknoten auf einmal synchronisiert werden, bei TPSN synchronisieren sich hingegen immer nur zwei Sensorknoten miteinander [MKSL04].
Die Synchronisationsgenauigkeit bei RBS liegt nach Aussage der Autoren bei fünf Mikrosekunden auf Mica-Sensorknoten [EGE02]. Die Entwickler von TPSN haben ebenfalls RBS und zusätzlich ihr TPSN auf Mica-Sensorknoten bei der Synchronisation zweier benachbarter Knoten getestet und konnten eine Genauigkeit von 16,9 µsec für TPSN und 29,1 µsec für RBS messen [GKS03]. Die Urheber von FTSP stellten eine Synchronisationsgenauigkeit von 1,48 µsec bei zwei benachbarten Sensorknoten und 3 µsec bei zwei sechs Hops voneinander entfernten Knoten fest [MKSL04].
FTSP und TPSN besitzen beide einen Baum zum Verbreiten der Synchronisationsnachrichten. Bei TPSN wird dieser jedoch explizit gebildet und muss bei einer sich verändernden Netzwerktopologie ständig neu gebildet werden. Beim FTSP bildet sich der Baum indirekt durch die Synchronisationspakete. Hierdurch ist FTSP auch bei mobilen Sensorknoten im Sensornetzwerk ausgezeichnet geeignet [MKSL04].
Bei einer gegebenen Synchronisationsperiode von t Sekunden verschickt jeder Sensorknoten alle t Sekunden ein Paket bei FTSP, zwei Pakete bei TPSN und 1,5 Pakete bei RBS. Bei letzterem wurde mit einem Paket pro Zeitaustausch und 0,5 Paketen für das Referenzpacket gerechnet.

3.8 Positionsbestimmung von Sensorknoten

Es ist häufig sehr wichtig, genau zu wissen, wo sich die Sensorknoten befinden, wie die im Folgenden genannten Gründe verdeutlichen [Bou09, Seite 109f]. Man muss den erfassten Sensordaten einen Ort zuordnen können, um eine qualifizierte Aussage über das Ausbringungsgebiet des Sensornetzes aus ihnen ableiten zu können. Kennt man die Position der einzelnen Knoten, so kann man beim Transport der Daten zur

Senke eine Korrelation zwischen den Daten benachbarter Sensorknoten ermitteln und das Datenaufkommen reduzieren. Hinzu kommt die Möglichkeit zur Evaluation der Abdeckung des Gebietes mit Sensoren sowie die Sensorendichte darin. Des Weiteren ist bei einigen Anwendungen eine eindeutige Adressierung der Sensorknoten anhand ihrer Position sinnvoll. Sensordaten können damit gezielt für einen Standort oder ein Gebiet angefordert werden, und die Klasse der geographischen Routingprotokolle nutzt eine solche Benennung als Grundlage zum Routen der Datenpakete.
Das Problem der Positionsbestimmung wird von den Sensornetzwerkforschern derzeit noch nicht als gelöst angesehen. Es wurde zwar schon eine Vielzahl von Verfahren vorgeschlagen, diese sind aber noch nicht an realistische Einsatzbedingungen angepasst. Die einen Positionsbestimmungsverfahren machen vereinfachte Annahmen, wie etwa eine Sichtlinie zwischen den Sensorknoten, eine hohe Dichte an Ankerknoten oder Wissen über die Aufstellungsverteilung. Die anderen benötigen spezielle Hardware, welche zuviel Energie verbraucht, zu groß oder zu teuer ist. In großen Sensornetzwerken halten die Annahmen unter realistischen Bedingungen nicht, und es kann nicht jeder Sensorknoten mit zusätzlicher Hardware ausgestattet werden. Andererseits sind Positionsbestimmungsverfahren generell zu ungenau, wenn nicht vereinfachende Annahmen getroffen werden. Die Herausforderung an die Sensornetzwerkforscher ist es, Positionsbestimmungsverfahren zu entwickeln, die hinreichend genau zu möglichst geringen Kosten in großen Sensornetzwerken unter realistischen Bedingungen arbeiten [SSS07].
In diesem Abschnitt werden zunächst die Anforderungen an ein Positionsbestimmungsverfahren betrachtet und ein Klassifikationsschema vorgestellt. Darauffolgend werden Verfahren zur Distanz- und Winkelbestimmung beschrieben. Anschließend werden in den Unterabschnitten 3.8.1 bis 3.8.3 ausgewählte Positionsbestimmungsverfahren vorgestellt. Dabei handelt es sich um ein allgemein bekanntes Positionsbestimmungsverfahren (GPS), um ein einfaches Positionsbestimmungsverfahren (Bounding Box) und um ein komplexes, aber dafür genaueres Positionsbestimmungsverfahren (SpinLoc).
Eine wichtige Anforderung an ein Positionsbestimmungsverfahren ist der effiziente Umgang mit den Ressourcen (vergleiche Abschnitt 3.4). Die Positionsbestimmung ist zwar wichtig, aber in den meisten Fällen nicht die eigentliche Aufgabe des Sensornetzwerks. Daher sollte das verwendete Verfahren möglichst wenig Speicher und Rechenleistung konsumieren, um unter anderem einen geringen Energieverbrauch zu gewährleisten. Darüber hinaus sollte der Funkkanal nicht mehr als nötig belastet werden. Eine weitere Anforderung ist die Skalierbarkeit. Das Verfahren sollte sowohl in kleinen als auch in großen sowie in dicht als auch in dünn besiedelten Netzen gleich gut mit den Ressourcen umgehen. Dabei sollte die Genauigkeit der Positionsbestimmung nicht von diesen Parametern abhängig sein. Hinzu kommt die Forderung nach Robustheit. Das Verfahren sollte tolerant bei ungenauen Distanzen und Winkelangaben zu Ankerknoten sowie bei Kommunikationsfehlern sein, und die Beschaffenheit des Aufstellungsgebietes sollte keine Rolle spielen. Als letzte Anforderung ist die Selbstorganisation zu nennen. Die Positionsbestimmung sollte aus dem Netz selber erfolgen, es sollte keine zusätzliche Infrastruktur vorausgesetzt werden [Bou09, Seite 310].
Kein bisher bekanntes Positionsbestimmungsverfahren wird allen Anforderungen gerecht, es muss passend zu der Anwendung eine Abwägung zwischen den oben genannten Anforderungen getroffen werden. Aus diesem Grund ist eine Vielzahl von Verfahren entwickelt worden, die nach den folgenden Punkten klassifiziert werden können.

Verteiltes oder zentrales Verfahren Bei den zentralen Positionsbestimmungsverfahren wird die Positionsbestimmung von einem oder wenigen speziellen Sensorknoten vorgenommen. Diese senden die ermittelten Positionen an die jeweiligen Sensorknoten. Bei den verteilten Positionsbestimmungsverfahren berechnet jeder Sensorknoten selbstständig seine Position, was nicht bedeuten muss, dass es nicht auch hier spezielle Ankerknoten gibt, welche Referenzsignale aussenden [Bou09, 324]. Zentrale Verfahren können rechenintensive Operationen auf wenige darauf ausgelegte Sensorknoten verlagern. Als Nachteile sind jedoch ein Overhead bei der Kommunikation und die schlechte Skalierung in Bezug auf die Netzgröße zu nennen [SSS07].

Passives oder aktives Verfahren Aktive Positionsbestimmungsverfahren nutzen vom Sensornetz ge-

nerierte Signale zur Positionsbestimmung. Die passiven Verfahren benötigen entweder eine zusätzliche Infrastruktur oder verwenden globale Ereignisse, wie den Schattenwurf vorbeiziehender Wolken beziehungsweise akustische Signale der Umwelt [KA08].

Verfahren mit oder ohne Infrastruktur Ein Verfahren kann entweder eine zusätzliche Infrastruktur benötigen oder auch nicht. Eine Infrastruktur sind z. B. die Satelliten des GPS aber auch selbst aufgebaute Komponenten wie ein Projektor, der ein bekanntes Muster auf das Sensorfeld projiziert. Des Weiteren wird auch von benötigter Infrastruktur gesprochen, wenn die vorherige Sensornetzinfrastruktur zur Positionbestimmung angepasst werden muss. Dies ist z. B. bei einer Aufstellung der Ankerknoten an fest definierten bekannten Positionen oder bei zusätzlich benötigter Hardware der Fall [Bou09, 324].

Verfahren mit oder ohne Sichtlinie zwischen den Sensorknoten Eine Sichtlinie zwischen den Sensorknoten ist genau dann gegeben, wenn sich keine Objekte auf der direkten Linie zwischen zwei beliebigen Sensorknoten in einem Sensornetzwerk befinden [SSS07]. Die Unterscheidung der Positionsbestimmungsverfahren zwischen denjenigen mit und ohne Sichtlinie ist eng mit den vorherigen beiden Klassifizierungsmerkmalen verbunden. Bei den aktiven Verfahren wird die Ausbreitung des generierten Signals mehr oder minder stark von Objekten zwischen den Sensorknoten beeinflusst. Gleiches gilt bei den Verfahren mit einer zusätzlichen Infrastruktur. Darüber hinaus könnte es auch nötig sein, bei der Ausbringung die Sensorknoten so aufzustellen, dass eine Sichtlinie gegeben ist. Wie oben erwähnt, wird ein solcher Eingriff in das Design auch als Positionsbestimmungsverfahren mit Infrastruktur bezeichnet.

Ankerknoten im Sensornetz vorhanden oder nicht Von manchen Positionsbestimmungsverfahren werden Ankerknoten im Netzwerk vorausgesetzt, welche dann auch bei den zentralen und aktiven Verfahren die Generierung von Referenzsignalen übernehmen. Die Verfahren mit Ankerknoten unterscheiden sich in deren benötigter Anzahl, denn die Genauigkeit der Positionsbestimmung wird hiervon nicht selten beeinflusst [Bou09, 324].

Verfahren für Innenräume oder fürs Freie Nicht alle Positionsbestimmungsverfahren sind sowohl für den Einsatz innerhalb als auch außerhalb von Gebäuden geeignet, weil sich die Umgebungen stark unterscheiden [Bou09, Seite 324]. In Gebäuden ist das Sensornetz in vielen Fällen dichter, und es sind andere Umwelteinflüsse als in der freien Natur vorhanden. Außerhalb von Gebäuden ist eher eine Sichtlinie zwischen den Sensorknoten zu erwarten.

Relative oder absolute Positionsbestimmung Relative Positionsbestimmungsverfahren ermitteln die Position der Sensorknoten relativ zu einem festen Punkt im Sensornetz, meist der Basisstation. Absolute Verfahren bestimmen die globale Position der einzelnen Knoten, die häufigste Form sind Längen- und Breitengrad sowie die Höhe über dem Meeresspiegel [Bou09, 324].

Entferungsbasierte und nicht entfernungsbasierte Verfahren Bei entfernungsbasierten Positionsbestimmungsverfahren wird die Sensorknotenposition mithilfe der Entfernungen zwischen den Sensorknoten ermittelt. Dies geschieht entweder mit spezieller Hardware zur Entfernungsmessung, die jeder Knoten besitzen muss, oder durch eine genaue Systemkalibrierung und Profilierung des Aufstellungsgebietes. Ein Beispiel für letzteres ist die Positionsbestimmung anhand der Empfangssignalstärke. Nicht entfernungsbasierte Verfahren ermitteln die Position mit ihren Sensoren, einer zusätzlichen Infrastruktur oder anhand ihrer Nachbarknoten [ZH09].

Mobile oder statische Sensorknoten Nicht alle Positionsbestimmungsverfahren sind in der Lage, die Position von mobilen Sensorknoten zu ermitteln. Die Mehrzahl der Verfahren wurde für statische Sensorknoten entworfen. Sie sind zwar in der Lage, sich verändernde Positionen zu erfassen, indem sie immer wieder neu gestartet werden, aber dies zieht einen nicht unerheblichen Rechen- und

Kommunikationsaufwand nach sich. Diese Methodik ist je nach Verfahren wegen der Positionsberechnungsperiodendauer nur bei langsamen Bewegungen akzeptabel [XOL+07].

Zeitsynchronisation erforderlich oder nicht Die einen Positionsbestimmungsverfahren setzen eine Zeitsynchronisation zwischen den Sensorknoten voraus, die anderen nicht. Bei den Verfahren, welche eine Zeitsynchronisation voraussetzen, nimmt die Genauigkeit der Zeitsynchronisation einen großen Einfluss auf die Genauigkeit der Positionsbestimmung [VDMC08, Seite 194].

Ermittlung zwei- oder dreidimensionaler Position Es wird unterschieden zwischen Verfahren zur Positionsbestimmung in der Fläche und im Raum. Nutzt man ein Positionsbestimmungsverfahren, welches davon ausgeht, dass sich alle Sensorknoten in einer Ebene befinden, so ist die ermittelte Position fehlerhaft, wenn sich die einzelnen Knoten in verschiedenen Höhen befinden. Nur ein Positionsbestimmungsverfahren für den Raum ist in der Lage, die Aufstellungshöhe der Sensorknoten zu erfassen.

Die Klasse der entfernungsbasierten Positionsbestimmungsverfahren benötigt eine Möglichkeit zur Bestimmung von Entfernungen, aus denen dann die Position der Sensorknoten berechnet werden kann. Daher werden im Folgenden einige Verfahren zur Entfernungsbestimmung vorgestellt.
Wie bereits in Abschnitt 3.1 beschrieben wurde, macht die Empfangssignalstärke des Funksignals eine Aussage über die Entfernung zum Sender. Hierzu wird in regelmäßigen Abständen von den Sensorknoten ein Datenpakt beliebigen Inhalts an die benachbarten Sensorknoten gesendet oder die reguläre Kommunikation genutzt, wenn man sich darauf verlassen kann, dass diese regelmäßig stattfindet. In der Theorie nimmt die Empfangssignalstärke im Quadrat zur Entfernung ab. Eine empirische Untersuchung zu diesem Zusammenhang haben Z. Zhong und T. He durchgeführt [ZH09]. Sie haben festgestellt, dass kein netzwerkweiter monotoner Zusammenhang besteht. Man kann aber die Empfangssignalstärke als sehr guten Indikator für relative Entferungen – im Sinne von nah oder fern – für benachbarte Sensorknoten benutzen. Problematisch bei der Entfernungsbestimmung anhand der Empfangssignalstärke sind der nicht perfekte quadratische Zusammenhang in der Praxis, unberechenbare Verluste verursacht durch Hindernisse, Reflexionen an Objekten, Hintergrundrauschen, unzureichende Genauigkeit der Hardware bei der Erfassung der Empfangssignalstärke und Aufstellungshöhe sowie die Orientierung der Sensorknotenantenne. In eigenen Experimenten konnten die Erkenntnisse der Autoren nachvollzogen werden: Eine Wand macht schnell einen Unterschied von zwei bis drei Metern. Dessen ungeachtet wird die Empfangssignalstärke gerne für Positionsbestimmungsverfahren herangezogen. Funkchips bieten das Auslesen der Empfangssignalstärke von Haus aus an; es muss so keine zusätzliche Hardware verbaut werden, und die Kosten sowie der Energieverbrauch werden kleingehalten. Außerdem wird erwartet, dass in Zukunft die Genauigkeit der Funkchips bei der Messung der Empfangssignalstärke zunimmt [Bou09, Seite 312f]. Auch jetzt schon kann eine Verbesserung der Aussagekraft erzielt werden, indem man nach dem Aufstellen für einen Knoten die Empfangssignalstärke in festgelegten Entfernungen misst, in das Berechnungsmodell zudem die Umgebung mit einbezieht und beim Aufstellen auf gleiche Orientierugen der Sensorknotenantennen achtet.
Deutlich genauer bei der Entfernungsmessung ist das Time-of-Arrival-(ToA-)Verfahren, das in zwei Varianten existiert. In der ersten Variante wird von den Sensorknoten ein Datenpaket mit einem Zeitstempel des Versendezeitpunktes verschickt. Der empfangende Sensorknoten kann aus der Zeitdifferenz zwischen Versenden und Eintreffen mit der bekannten Ausbreitungsgeschwindigkeit des Signals die zurückgelegte Strecke berechnen. Diese Variante benötigt im Gegensatz zur zweiten Variante eine Zeitsynchronisation zwischen den Knoten. In der zweiten Variante versendet ein Sensorknoten A ein Paket zur Anforderung einer Antwort an einen anderen Knoten B und merkt sich den Versendezeitpunkt. B antwortet mit einem Paket, so dass A anhand des Empfangszeitpunktes die Dauer des gesamten Vorgangs und damit wieder die Entfernung berechnen kann. Hierbei ist zu beachten, dass die Zeit zwischen Eintreffen und Versenden bei B abgezogen werden muss. Diese Zeit kann empirisch ermittelt werden. Beide Varianten von ToA haben jedoch den Nachteil, dass sich Funksignale nahezu mit Lichtgeschwindigkeit ausbreiten und die Signallaufzeiten deshalb extrem kurz sind. Dies kann nur mit dedizierter Hardware gemessen werden. Eine

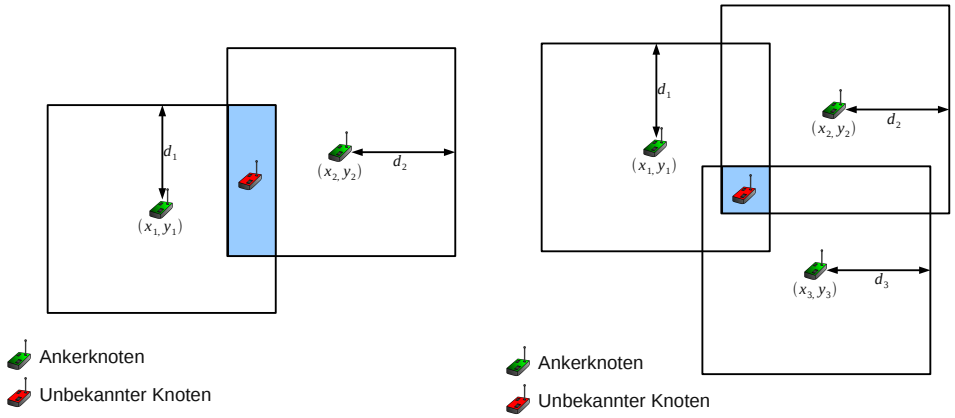
Lösung für diesen Ansatz ist, auf langsamere Signale wie Schall auszuweichen. Hierbei wird allerdings eine Sichtlinie vorausgesetzt [VDMC08, Seite 197ff].
Ein weiteres Verfahren ist Time Difference of Arrival (TDoA). Hierbei sind alle Sensorknoten mit Lautsprecher und Mikrofon ausgestattet. Sensorknoten A sendet zunächst ein Paket per Funk, welches ein folgendes akustisches Signal ankündigt, wartet eine festgelegte Zeitspanne t und versendet dann das akustische Signal. Erhält Sensorknoten B das Funkpaket, so merkt er sich den Zeitpunkt und nutzt die Zeitspanne t zum Einschalten seines Mikrofons. B kann nach dem Empfang des akustischen Signals dessen Signallaufzeit aus der Zeitdifferenz der Eintreffzeitpunkte abzüglich t berechnen. Bei dieser Berechnung wird vereinfachend von einem instantanen Eintreffen des Funkpaketes nach dem Versenden ausgegangen, es definiert somit den Zeitpunkt des Losschickens minus t. Das akustische Signal breitet sich hingegen deutlich langsamer aus, so dass die hier beschriebene Zeitdifferenz von den Sensorknoten gemessen und zur Berechnung der Entfernung genutzt werden kann. Dieses Verfahren ist sehr genau bei Umgebungen mit einer Sichtlinie zwischen den Sensorknoten, es funktioniert allerdings am besten mit kalibrierten Lautsprechern sowie Mikrofonen und in Umgebungen ohne Echo [Sto05, Seite 282].
Eine weitere genutzte Größe, welche als Grundlage zur Positionbestimmung herangezogen wird, ist der Winkel von eintreffenden Signalen (Angle of Arrival [AoA]) zu einem Referenzknoten, einem Funkmast oder der mit einem elektronischen Kompass ermittelten geographischen Orientierung. Der Winkel kann mithilfe eines Feldes von Empfängern bestimmt werden. Es sind mehrere Funkempfänger oder Mikrofone bei einem Sensorknoten so angeordnet, dass die Richtung des eintreffenden Signals aus der Zeitdifferenz des Eintreffens zwischen den Empfängern bestimmt werden kann. Hierbei wird eine Genauigkeit in der Größenordnung von wenigen Grad erzielt. Nachteilig sind jedoch die großen Hardwarekosten und die Unterbringung der Hardware auf der typischen kleinen Sensorknotenfläche [Sto05, 283]. Einen anderen Ansatz bei der Winkelbestimmung verfolgen Chang et al. [CTL+08]. Sie befestigen einen Ankerknoten an einem Motor, der den Ankerknoten auf einer Kreisbahn bewegt. Hierdurch kann mithilfe des Dopplereffektes der Winkel unbekannter Knoten und einem Referenzknoten bestimmt werden. Das von den Autoren vorgeschlagene Verfahren zur Positionsbestimmung wird in Unerabschnitt 3.8.3 beschrieben.

3.8.1 Global Positioning System (GPS)

Das Global Positioning System (GPS) besteht aus 24 Satelliten. Sie umrunden die Erde zweimal am Tag in einer solchen Umlaufbahnenkonstellation, dass zu jeder Zeit an jedem Ort mindestens vier Satelliten empfangen werden können. Ein GPS-Empfänger berechnet die Entfernung zu mindestens drei Satelliten per Time of Arrival. Ein vierter Satellit wird zur Zeitsynchronisation des Empfängers mit den Satelliten benötigt. Anhand des Zeitpunktes wird ferner im GPS-Empfänger dank der bekannten Satellitenbahnen die Position der Satelliten berechnet. Man besitzt nun drei Punkte und deren Entfernung und kann per Trilateration die Position des GPS-Empfängers bestimmen [Bou09, Seite 334]. Als Lösung erhält man im dreidimensionalen Fall zwei valide Lösungen, denn drei Kugeln um verschiedene Punke schneiden sich in maximal zwei Punkten. Eine Lösung befindet sich allerdings weit im Weltraum oberhalb der Satelliten. Die Genauigkeit der Positionsbestimmung beträgt je nach verwendetem GPS-Empfänger und den derzeitigen Satellitenpositionen zwei bis 15 Meter [Bou09, Seite 334]. Es werden der Längen- und der Breitengrad sowie die Höhe berechnet, des Weiteren stehen die Uhrzeit und, bei mehreren Messungen, die Geschwindigkeit und Richtung zur Verfügung.
GPS hat sich weltweit als dominantes Positionsbestimmungsverfahren im zivilen und militärischen Bereich etabliert. Es ist allerdings zur Positionsbestimmung in Sensornetzwerken nur bedingt geeignet. GPS-Module sind teuer und konsumieren inakzeptabel viel Strom. Ein großes Problem ist eine gute Verbindung zu den Satelliten. In Gebäuden, in urbanen Umgebungen, unter Wasser und bei schlechtem Wetter ist der Empfänger nicht in der Lage, Signale zu empfangen. Aus diesen Gründen kommt GPS oftmals nur auf einer kleinen Anzahl der Sensorknoten zum Einsatz, um beispielsweise einem relativen Positionsbestimmungsverfahren eine Abbildung des lokalen Koordinatensystems auf das globale zu ermöglichen. In dem in Abschnitt 3.2 vorgestellten System zur Vulkanüberwachung wird ein einziger Sensorknoten mit einem

GPS-Empfänger ausgestattet. GPS wird hier lediglich als Referenzzeitgeber für die Zeitsynchronisation genutzt [WALJ+06].
Bei GPS handelt es sich um ein absolutes, entfernungsbasiertes, verteiltes und passives Positionsbestimmungsverfahren ohne Ankerknoten für statische oder mobile Sensorknoten. Es wird eine Infrastruktur in Form der Satelliten benötigt. Diese sind zwar weltweit kostenlos vorhanden, aber für ein Sensornetzwerk auf fremden Planeten muss dies bedacht werden. Es wird keine Sichtlinie zwischen den Sensorknoten benötigt, aber Kontakt zu den Satelliten. Somit ist es am besten für das Freie und nur sehr bedingt für Innenräume geeignet. GPS setzt keine Zeitsynchronisation voraus und ermittelt dreidimensionale Koordinaten.

3.8.2 Bounding-Box-Verfahren



(a) Bounding Box mit zwei Ankerknoten (b) Bounding Box mit drei Ankerknoten

Abbildung 3.21: Die Positionsbestimmung mit Bounding Boxes

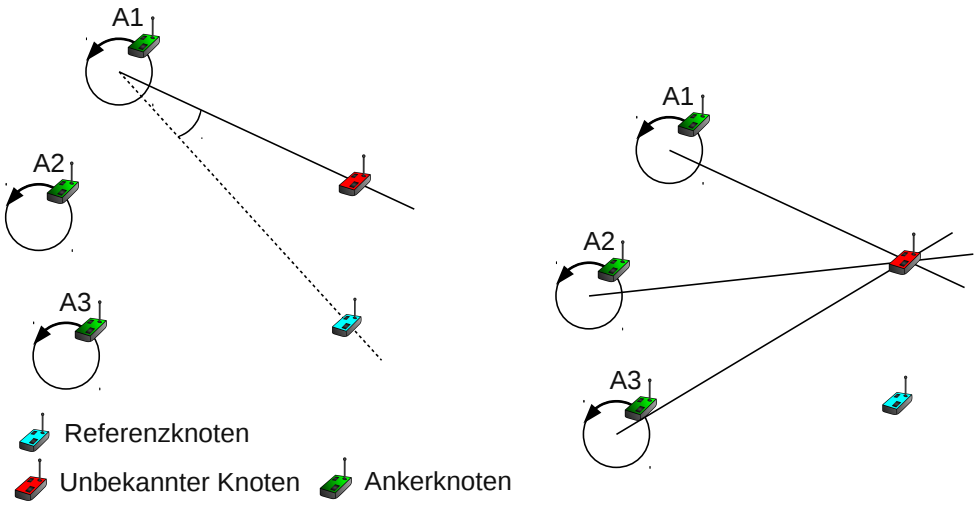
Die Bounding-Box-Methode ist ein einfach zu berechnendes Positionsbestimmungsverfahren. Es grenzt die Fläche, in der sich die unbekannten Sensorknoten befinden können, ein. Wie in Abbildung 3.21 dargestellt, wird um jeden Ankerknoten i mit der Position (x_i, y_i) innerhalb der Reichweite des unbekannten Sensorknotens ein Quadrat mit der Seitenlänge $2 \cdot \mathrm{d}_i$ gelegt. Dabei bezeichnet d_i den Abstand zwischen dem Ankerknoten i und dem unbekannten Sensorknoten. Nun kann die blau gefärbte Schittfläche der einzelnen Quadrate wie folgt berechnet werden:

$$[\max(x_i - d_i), \max(y_i - d_i)] \times [\min(x_i + d_i), \min(y_i + d_i)], \ \mathrm{m}it\ i = 1..n$$

Als Position des unbekannten Kontens wird die Mitte der berechneten Schnittfläche genommen. In Abbildung 3.21(b) ist eine solche Situation mit drei Ankerknoten abgebildet. Die Ungenauigkeit ist groß, wenn die Ankerknoten und der Sensorknoten auf einer Linie liegen, welche die Quadrate nahe der Mitte ihrer Seiten schneidet. Vergleiche hierzu Abbildung 3.21(a). In einem solchen Fall könnte es auch keine Schnittfläche geben, wenn die berechneten Entfernungen zu kurz bestimmt wurden. Am besten ist das Ergebnis bei unbekannten Knoten in den Ecken der Quadrate [Sto05, 289].
Dieses Positionsbestimmungsverfahren eignet sich am besten bei Sensorknoten mit wenig Rechenleistung. Die Berechnung ist einfach und ohne den Einsatz von Gleitkommaoperationen durchzuführen. Die Genauigkeit ist jedoch nicht so gut wie bei mathematisch aufwändigeren Verfahren.
Bounding Box ist ein verteilts, passives Verfahren ohne Infrastruktur für eine zweidimensionale Positionsbestimmung. Es wird keine Sichtlinie zwischen den Knoten vorausgesetzt, solange die Entfernungsmessung eine solche nicht verlangt. Es müssen Ankerknoten im Netz vorhanden sein, von denen die unbekannten Knoten die Position anfordern können. Ist diese absolut, so berechen auch die unbekannten Knoten eine

absolute Position. Des Weiteren handelt es sich um ein entfernungsbasiertes Positionsbestimmungsverfahren. Ist man nicht in der Lage, Entfernungen zu ermitteln, kann aber auch die maximale Funkreichweite als Distanz angenommen werden. Das Verfahren kann innerhalb und außerhalb von Gebäuden zum Einsatz kommen und benötigt keine Zeitsynchronisation. Aufgrund der einfachen Berechnung kann es auch für mobile Sensorknoten eingesetzt werden.

3.8.3 Spinning Indoor Localization (SpinLoc)



(a) Der Winkel zwischen Referenzknoten und Ankerknoten wird bestimmt.

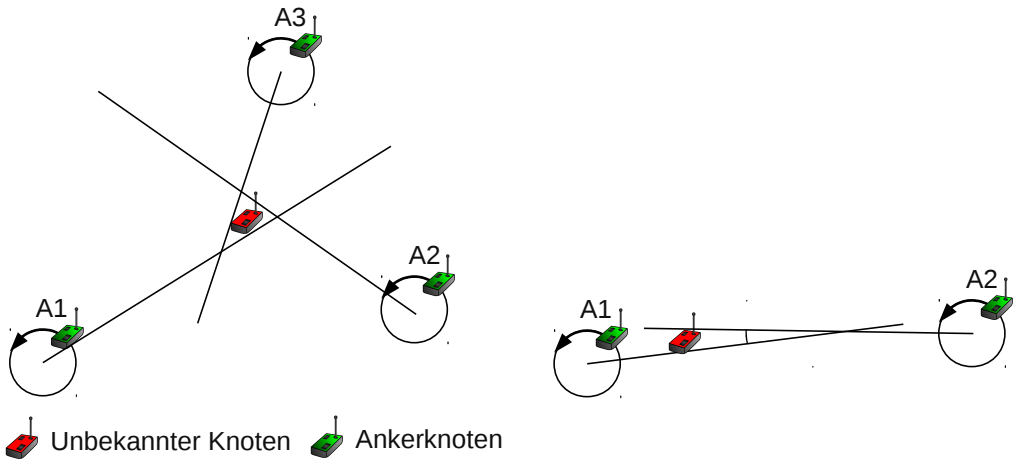
(b) Die Position der unbekannten Knoten wird durch den Schnitt der Geraden bestimmt.

Abbildung 3.22: Die Positionsbestimmung mit SpinLoc (nach [CTL+08])

SpinLoc wurde von Chang et al. vorgeschlagen und berechnet die Position der unbekannten Knoten mithilfe von Winkeln zwischen Sensorknoten [CTL+08]. Der Name SpinLoc steht für Spinning Indoor Localization und beruht auf der Winkelbestimmung mittels zentrisch um einen Punkt rotierender Ankerknoten. Infolgedessen kommt es beim Senden zu einem Dopplereffekt, der je nach Richtung zum Ankerknoten unterschiedlich ausgeprägt ist. Sensorknoten senden im Bereich von 400 MHz bis 2,4 GHz. Diese Frequenzen können mit den typischen Taktraten von 10 MHz nicht erfasst werden. Als Lösung hierfür wird das Signal der Frequenz f_1 von einem Assistenzknoten mit einem Signal sehr ähnlicher aber fixer Frequenz f_2 überlagert. Es entsteht ein dopplereffektmoduliertes Interferenzmuster im Ausbringungsgebiet. Ein Referenzknoten und die unbekannten Knoten können jetzt die Interferenzfrequenz $|f_1 - f_2|$ mit einem am A/D-Wandler angeschlossenen Empfangssignalstärkesensor messen, mit einem Zeitstempel versehen und an die Basisstation senden. Diese berechnet mit den bekannten Positionen der Ankerknoten und des Referenzknotens für jeden Ankerknoten aus den Messwerten zu einem Zeitpunkt den Winkel zwischen Referenzknoten und unbekanntem Knoten. Vergleiche hierzu Abbildung 3.22(a). Dieser Vorgang wird anschließend für die anderen Ankerknoten im Sensornetzwerk wiederholt.

Besitzt die Basisstation die Winkel aller Ankerknoten, so berechnet sie, wie in Abbildung 3.22(b) zu erkennen ist, die Position der unbekannten Knoten als Schnittpunkt der Geraden von den Ankerknoten durch den unbekannten Knoten.

In der Praxis kann der Winkel nicht hochgenau bestimmt werden, was dazu führt, dass sich bei mehr als zwei Ankerknoten die Geraden nicht in einem Punkt schneiden. Eine solche Situation wird in Abbildung 3.23(a) verdeutlicht. Die Autoren schlagen hierfür vor, den gewichteten Schwerpunkt der Schnittpunkte als Sensorknotenposition anzunehmen. Die Gewichte der Punkte ergeben sich aus dem spitzen Winkel, der



(a) Die Geraden schneiden sich in der Praxis selten in einem Punkt.

(b) Die Geraden schneiden sich in einem kleinen spitzen Winkel.

Abbildung 3.23: Die möglichen Fehler bei SpinLoc (nach [CTL+08])

zwischen den beiden Geraden am Schnittpunkt auftritt. Je kleiner dieser ist, um so größer ist die Auswirkung von ungenauen Geradengleichungen auf den Positionsfehler. Vergleiche hierzu Abbildung 3.23(b). SpinLoc ist ein zentrales, aktives Positionsbestimmungsverfahren mit Infrastruktur. Die Autoren sind sich noch nicht sicher, inwieweit eine Sichtlinie benötigt wird. Untersuchungen hierzu sind geplant. Die Positionsbestimmung erzielte im Testaufbau bei 600 Hz Interferenzfrequenz und drei Ankerknoten mit 133 Umdrehungen pro Minute für 90% der Werte eine Genauigkeit von 70 cm [CTL+08]. Dies kann mit einer größeren Anzahl an Ankerknoten verbessert werden, was jedoch aufgrund der nacheinander auszuführenden Winkelmessungen zu größeren Latenzzeiten führt. Abhilfe schafft ein noch zu entwickelndes Scheduling, welches in großen Netzen mehrere Messungen von nicht in Nachbarschaft stehenden Ankerknoten gleichzeitig durchführt. Die Basisstation benötigt zur Berechnung der Position aus den Messdaten 0,47 Sekunden pro unbekanntem Knoten, somit ist das Verfahren nur für statische oder wenige, sich langsam bewegende Sensorknoten geeignet. SpinLoc wurde explizit für Innenräume entworfen, ist nicht entfernungsbasiert, setzt eine Zeitsynchronisation voraus und berechnet die Position relativ zur Basisstation. Derzeit ist nur eine zweidimensionale Positionsbestimmung möglich, es kann jedoch mit hinzugefügten, vertikal statt horizontal rotierenden Ankerknoten auch die Höhe bestimmt werden.

4 Entwurf und Implementierung des Sensornetzwerks

In diesem Kapitel werden der Entwurf und die Implementierung des Sensornetzwerks zur Erfassung eines Temperaturprofils in Wohn- und Büroräumen vorgestellt. Das Sensornetzwerk dient dem Zweck, innerhalb der einzelnen Räume des Gebäudes Temperatur, Luftfeuchte, Lichtwerte und die Spannung der Sensorknotenbatterie zu messen.
Dazu werden in Abschnitt 4.1 zunächst die zu verwendenden Technologien ausgewählt, wobei die in Kapitel 3 gewonnenen Erkenntnisse berücksichtigt und erste Entwurfsentscheidungen getroffen werden. Die Auswahl der Technologien stellt entscheidende Weichen und bedingt die Struktur des Sensornetzwerks. Es ist bereits hier aufgrund der beschränkten Ressourcen eine Abwägung zwischen Funktionalität und Performanz zu treffen, und dieser Abschnitt ist somit bereits dem Entwurf des Sensornetzwerks zuzuordnen.
Nachdem die Struktur des Sensornetzes feststeht, wird in Abschnitt 4.2 der Entwurf fortgesetzt. Dazu wird zunächst ein kleiner Überblick über den Aufbau des Sensornetzwerks gegeben, und anschließend werden die einzelnen Komponenten vorgestellt. Dann werden die Funktionsweise der einzelnen Komponenten und wichtige Datenstrukturen erläutert.
Abschließend werden die Implementierungen der Basisstation sowie der Sensorknoten in den Abschnitten 4.3 und 4.4 vorgestellt.

4.1 Technologieauswahl

Anhand der Anforderungsdefinition aus Kapitel 2 und insbesondere der erforderlichen Anforderungen E1 bis E6 werden in diesem Abschnitt die für das Sensornetzwerk benötigten Technologien festgelegt.
Zunächst einmal ist ein Betriebssystem für die einzelnen Sensorknoten zu wählen. Ohne ein Betriebssystem müsste die Anwendung für den Sensorknoten in C von Grund auf neu geschrieben werden, d. h. inklusive Netzwerkstapel und Gerätetreiber für die Sensoren und den Funkchip. Dies ist innerhalb der Projektlaufzeit nicht zu realisieren. In Abschnitt 3.5 wurden ausführlich Betriebssysteme für Sensorknoten besprochen und miteinander verglichen. Die Wahl für das in diesem Sensornetz verwendete Betriebssystem für Sensorknoten fällt auf TinyOS. Hierfür sprechen mehrere Gründe: Da der MTM-CM5000-MSP-Sensorknoten von Maxfor von der Abteilung *Systemsoftware und verteilte Systeme* gestellt wird und als gegeben angesehen werden muss, ist es von enormem Vorteil, dass TinyOS diese Plattform bereits unterstützt. Es sind sogar bereits Treiber für die verbauten Sensoren vorhanden. Die Entwicklung kann damit zügig starten und die Anforderung E1 für die Datenerfassung ist ohne die Entwicklung eines Gerätetreibers zu realisieren. Des Weiteren besitzt TinyOS eine Energieverwaltung für den Mikrocontroller und den Funkchip. Ein möglichst effizienter Umgang mit den Energieressourcen ist oberstes Gebot in Sensornetzwerken und wird auch in Anforderung E2 verlangt. Ein weiterer Grund für TinyOS ist das ereignisbasierte Ausführungsmodell. Dieses ist besonders ressourcenschonend – vergleiche Unterabschnitt 3.5.1 – und passt zu dem Programmfluss in der hier entworfenen Anwendung. Abschließend fiel die Entscheidung auch deswegen auf TinyOS, weil bereits Erfahrungen damit an der Universität Oldenburg gemacht wurden und somit Wissen über TinyOS lokal vorhanden ist.
TinyOS wird in der Version 2.x aus dem TinyOS CVS vom 16.07.2009 verwendet, da diese zu Beginn der Implementierung die neueste Version ohne bekannte Fehler war (Bezugsquelle: http://sourceforge.net/projects/tinyos/). Als Entwicklungsumgebung kommt Eclipse mit dem von der ETH Zürich herausgegebenen YETI-2-Plugin für TinyOS zum Einsatz (Bezugsquelle: http://tos-ide.ethz.ch/wiki/). Der Quellcode wird mit dem nesC-Compiler der Version 1.3.1 in C übersetzt und mit dem msp430-gcc in der Version 3.2.3 für den Sensorknoten gebaut. Uisp in der Version 20050519tinyos wird genutzt, um das Binärabbild auf den Sensorknoten zu schreiben. Die letzten drei

Werkzeuge wurden aus dem Ubuntu-Repository verwendet. Obwohl man die Komponenten auch mit der UML-2-Spezifikation beschreiben kann [Bac09], kommt in dieser Arbeit das YETI-2-Plugin zur Erstellung der Komponentendiagramme zum Einsatz. Dieses ist in der Lage, automatisch aus dem Quellcode solche Diagramme zu erzugen.
Nachdem das verwendete Betriebssystem geklärt wurde, steht der Netzwerkstapel fest (vergleiche Unterabschnitt 3.5.2). Mit diesem ist eine energieeffiziente Kommunikation zwischen benachbarten Sensorknoten möglich. Es fehlt ein geeignetes Routingprotokoll, um die Kommunikation zwischen mehr als ein Hop entfernten Sensorknoten zu gewährleisten. Im Folgenden werden die Eigenschaften des Sensornetzwerks näher untersucht, um darauf aufbauend ein Routingprotokoll auszuwählen.
Hierzu wird zunächst das typische Datenaufkommen in dem Sensornetzwerk betrachtet. Die Sensorwerte müssen regelmäßig zu einer oder mehreren Datensenken gesendet werden, wo sie das PC-Programm entgegennimmt. Die Paketgröße eines Datensatzes, bestehend aus Sensorknotenadresse, Zeitstempel, Temperatur, Luftfeuchte, Licht- und Spannungswert beträgt 14 Byte für die Daten plus 11 Byte für den Header und 2 Byte für die Prüfsumme [Lev]. Dies ergibt eine Paketgröße von 27 Byte. Die Abteilung *Systemsoftware und verteilte Systeme* hat insgesamt 130 Sensorknoten angeschafft, und für diese Arbeit stehen 10 davon zur Verfügung. Sollen alle Sensorknoten die Daten mit derselben Messauflösung erfassen und an eine Senke im Sensornetzwerk schicken, so ist dieser Sensorknoten der Falschenhals bei der Kommunikation. Der Funkchip des Sensorknotens kann mit maximal 250 kb/s, also 31,25 kB/s, Daten übertragen (vergleiche Abschnitt 3.3). Daraus ergibt sich die maximale theoretische Anzahl an Paketen, die jeder Sensorknoten pro Sekunde in dem Sensornetzwerk mit nur einer Datensenke versenden dürfte:

$$\frac{31,25\frac{KB}{s}}{27B \cdot 130} \approx 9\ \frac{\mathrm{P}akete}{s}$$

Im praktischen Fall kommen noch unvorhersehbare Verluste durch Kollisionen hinzu, verursacht von den Sensorknoten, die um das Medium konkurrieren. Dazu kann nicht immer ausgeschlossen werden, dass im selben Frequenzband operierende WLANs die Kommunikation behindern. Von Anforderung E1 wird eine Messauflösung von einem Datenpaket pro Sekunde gefordert, was fast um den Faktor zehn kleiner ist als der maximale theoretische Wert. Das Sensornetzwerk kommt also mit einer Datensenke aus; diese wird im Folgenden Basisstation genannt.
Zudem kann auf eine Aggregation der Daten verzichtet werden; es wäre aber nützlich, wenn das verwendete Routingprotokoll eine solche für spätere Erweiterungen bereits vorsähe.
Die Sensorknoten werden fest an einer Stelle aufgestellt. Das Routingprotokoll muss daher keine mobilen Sensorknoten berücksichtigen.
Darüber hinaus sind alle Sensorknoten von gleicher Bauweise, es handelt sich also um ein homogenes Sensornetzwerk. Aus diesem Grund bietet es sich nicht an, bestimmte Sensorknoten für bestimmte Aufgaben vorzusehen. In diesem Sensornetzwerk sind alle Sensorknoten, bis auf die Basisstation, mit derselben Software bespielt. Das Routingprotokoll muss daher keine Pakete zu Sensorknoten mit speziellen Fähigkeiten weiterleiten können. Außerdem bietet es sich hierdurch nicht an, eine hierarchische Netzwerkstruktur zu wählen. Kein Sensorknoten besitzt eine besondere Funkreichweite oder Lebensdauer, um sich gut zur Organisation eines Subnetzes zu eignen. Dazu ist die Netzwerkgröße mit 130 Sensorknoten noch überschaubar. Ein flaches Routingprotokoll ist damit ausreichend.
Aus der Anforderungsdefinition geht hervor, dass Pakete von der Basisstation zu den Sensorknoten geschickt werden müssen, zum einen, um Messungen zu starten und die Sensorkalibrierung vorzunehmen, zum anderen, um nach Anforderung E6 Aktoren anzusteuern. Des Weiteren müssen Messwerte und Statusnachrichten von den Sensorknoten an die Basisstation geschickt werden können. Eine Kommunikation zwischen beliebigen Sensorknoten ist aus der Anforderungsdefinition nicht abzuleiten. Geht man zusätzlich davon aus, dass Messungen nur selten neu gestartet werden müssen und man die Sensorkalibrierung nur einmalig nach dem Aufstellen vornimmt, so übersteigt das Paketaufkommen zur Basisstation hin das Paketaufkommen von ihr weg. Das Datenaufkommen der Aktoren spielt bei dieser Betrachtung eine untergeordnete Rolle, die Sensordatenerfassung liegt im Fokus dieser Arbeit.

Anhand der Betrachtung von Routingprotokollen in Abschnitt 3.6 wurde Directed Diffusion als Routingprotokoll für das Sensornetzwerk gewählt. Es passt gut auf die oben beschriebenen Eigenschaften des hier vorliegenden Sensornetzwerks. Directed Diffusion ist flach, unterstützt eine Datenaggregation, ist multipfadfähig und organisiert die Sensorkonten selbstständig zu einem Multi-Hop-Netzwerk, indem es Interessen verbreitet. Durch die letzten beiden Fähigkeiten wird es somit auch der Anforderung E4 bezüglich Selbstorganisation und Robustheit gegenüber Ausfällen gerecht. Von der Basisstation werden Interessen an die Sensorknoten gesendet, und diese können danach Daten an die Basisstation schicken. Ein reaktives Routingprotokoll ist ausreichend, da statische Umweltparameter erfasst werden (vergleiche Abschnitt 3.6). Directed Diffusion wird in einer auf das Sensornetzwerk zugeschnittenen Form für TinyOS innerhalb dieser Arbeit in Abschnitt 4.2 entworfen und seine Implementierung in Abschnitt 4.4 vorgestellt.
Im Folgenden geht es um die Auswahl des Zeitsynchronisationsprotokolls, vergleiche hierzu auch Abschnitt 3.7. Das Routingprotokoll Directed Diffusion benötigt keine Zeitinformationen und stellt daher keine Anforderungen an das Zeitsynchronisationsprotokoll. Anforderung E3 verlangt dagegen eine Genauigkeit von 250 ms, um die Sensorwerte eindeutig einem Zeitpunkt zuordnen zu können. Alle vorgestellten Synchronisationsprotokolle bieten eine Genauigkeit weit unter diesem Wert an. RBS scheidet jedoch aus, da es speziell für relative Zeitsynchronisation ausgelegt ist. In diesem Sensornetzwerk ist eine einheitliche globale Zeit erforderlich, um Messungen gleichzeitig zu starten und Sensorwerte dezentral mit einheitlichen Zeitstempeln zu versehen. Das Zeitstempeln der Datenpakete auf die Basisstation zu beschränken ist nicht ausreichend, da der Paketfluss nicht-deterministischen Zeitspannen unterliegt.
In dieser Arbeit wird das FTSP verwendet. Gegenüber dem TPSN synchronisiert es genauer und organisiert seinen Baum implizit. Ausfallende Sensorknoten beinflussen nicht die Zeitsynchronisation des FTSP. Besteht keine Verbindung mehr zu einem Teilbereich des Sensornetzes, so hält das FTSP sogar beide Teilnetze untereinander synchron. Es synchronisiert dann die beiden Teilnetze miteinander, sobald sie wieder verbunden sind. Anforderung E4 nach Selbstorganisation der Sensorknoten und Robustheit gegenüber Ausfällen wird somit vom verwendeten Zeitsynchronisationsprotokoll erfüllt. Das FTSP benötigt keine Infrastruktur oder zusätzliche Geräte. Es macht somit keine Einschränkungen hinsichtlich des Aufstellungsortes des Sensornetzwerks, die Kosten der Sensorknoten werden nicht erhöht, und es wird keine Energie für externe Geräte verschwendet. Auch liegt bereits eine fertige Implementierung des FTSP für TinyOS vor, die auch den LPL des Netzwerkstapels nutzt und damit als energieeffizient angesehen werden kann. Zudem verschickt es nur halb so viele Pakete wie TPSN (vergleiche Unterabschnitt 3.7.4). Das FTSP ist damit auch effizient in Bezug auf die Bandbreite. Es geht also bescheiden mit den vorhandenen Ressourcen um und steht Andorderung E2 nach einer langen Lebensdauer des Sensornetzes nicht im Weg.
Die TinyOS beiliegende Implementierung von FTSP wurde in Unterabschnitt 3.7.4 bereits vorgestellt. Die Verwendung innerhalb dieses Sensornetzwerks wird in Abschnitt 4.3 beschrieben.
Abschießend wird geklärt, wie die Sensordaten einer Position zugeordnet werden können. Ein solche Zuordnung ist laut Abschnitt 3.8 für eine Datenauswertung unabdingbar. Das Routingprotokoll benötigt demgegenüber keine Kenntnisse über die Sensorknotenpositionen. Zudem werden die Sensorknoten deterministisch von einem Menschen aufgestellt und sind nicht mobil. Insgesamt ist somit aus den Anforderungen nicht abzuleiten, dass die Sensorknoten selbst über eine Positionsbestimmung verfügen müssen. Die Daten werden eindeutig einem Sensorknoten zugeordnet und erst in dem PC-Programm mit Positionsangaben zusammengebracht. Der Benutzer gibt hierzu nach dem Aufstellen der Sensorknoten die Positionen in eine Karte im PC-Programm an. Damit entfällt auch eine Umrechnung der globalen Koordinaten in das Koordinatensystem der Karte.
Für die manuelle Positionsbestimmung spricht auch das Fehlen eines geeigneten und realisierbaren Positionsbestimmungsverfahrens. In jedem Raum des Gebäudes können sich mehrere Sensoren befinden. Um dabei einzelne Sensorknoten zu unterscheiden, sollte die Positionsbestimmung auf einen Meter genau sein und im dreidimensionalen Raum funktionieren, um Sensorknoten auf verschiedenen Höhen und Etagen zu berücksichtigen. Einfache Verfahren ohne Infrastruktur, wie z. B. das Bounding-Box-Verfahren aus Unterabschnitt 3.8.2, erreichen eine solche Auflösung nicht. Verfahren mit Infrastruktur bzw. zusätzlich benötigter Hardware eignen sich nicht, weil weitere Kosten entstehen, mehr Energie verbraucht wird

und sie zu aufwändig sind. Somit fallen auch GPS und SpinLoc aus. Daraus folgt zudem, dass die große Klasse der entfernungsbasierten Verfahren die Entfernungen in diesem Sensornetzwerk mithilfe der Empfangssignalstärke berechnen müssten. Es ist jedoch keine Sichtlinie zwischen den Sensorknoten gegeben, Reflexionen und Hindernisse verfälschen den Wert der Empfangssignalstärke signifikant, so dass keine hinreichend genaue Positionsbestimmung für diese Anwendung erzielt wird [ZH09].

4.2 Feinentwurf

Nachdem in Abschnitt 4.1 die verwendeten Technologien ausgewählt wurden, steht der Aufbau des Sensornetzwerks fest. Es ist homogen, hat eine flache Netzwerktopologie, wird deterministisch aufgestellt und die Sensorpositionen werden vom Benutzer im PC-Programm angegeben. Es kommt TinyOS als Betriebssystem, Directed Diffusion als Routingprotokoll und FTSP als Zeitsynchronisationsprotokoll zum Einsatz.

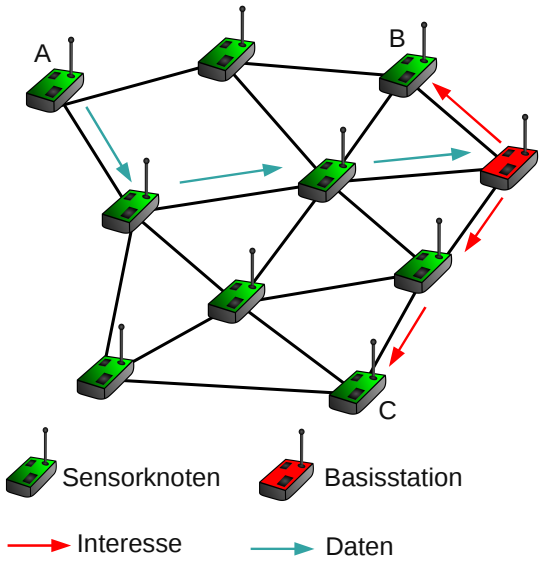


Abbildung 4.1: Ein Überblick über das entworfene Sensornetzwerk

Das Sensornetzwerk besteht, wie in Abbildung 4.1 zu sehen, aus einer Basisstation und unter Berücksichtigung der Betrachtungen in Unterabschnitt 4.1 aus bis zu 130 Sensorknoten. Die Basisstation ist die Schnittstelle zwischen dem PC-Programm und dem Sensornetzwerk, sie ist direkt an einen PC über die serielle Schnittstelle angeschlossen. Das PC-Programm kann so Anfragen in Form eines Interesses an die einzelnen Sensorknoten schicken, siehe Sensorknoten B und C. Die Sensorknoten sind für die Messungen der Umweltgrößen verantwortlich und senden diese an die Basisstation, welche die Daten an das PC-Programm weiterreicht, siehe Sensorknoten A.
In Abbildung 4.2 sind die einzelnen Softwareschichten der Sensorknoten zu sehen. Direkt auf den verwendeten Sensorknoten vom Typ MTM-CM5000-MSP setzt TinyOS mit seinem Netzwrkstapel auf. Directed Diffusion bietet mit seinen Interessen eine direkte Möglichkeit, die Anwendung auf dem Sensorknoten zu steuern. Das verwendete Interessensystem wird vom Routingprotokoll implementiert, die Interessen werden jedoch von der Anwendung bedient. Somit ist das Routingprotokoll eng mit der Anwendung verzahnt und befindet sich deshalb im Netzwerkstapel auf der Anwendungsschicht zwischen der Anwendung und der Active-Message-Schicht, siehe Unterabschnitt 3.5.2. Die Anwendung bedient die Interessen, sie implementiert also die im Sensornetzwerk definierten Interessen. Hierzu kann sie die Sensordatenerfassung als Bezugsquelle für Sensorwerte nutzen. Als Middleware zur Zeitsynchronisation ist das FTSP über alle Sensorknoten verteilt und stellt eine globale Zeit zur Verfügung.

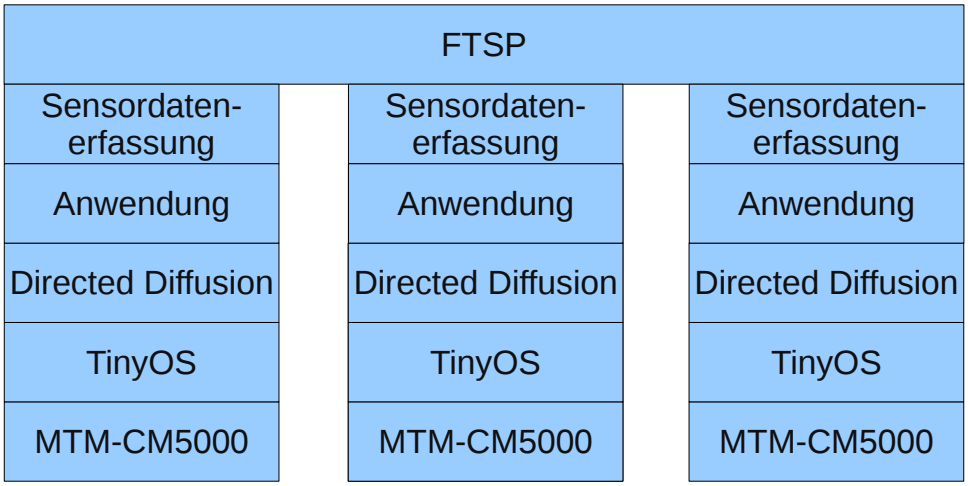


Abbildung 4.2: Die Softwareschichten des entworfenen Sensornetzwerks

Im Folgenden wird der Entwurf von Directed Diffusion, der Anwendung und der Sensordatenerfassung für die Sensorknoten vorgestellt. Die Basisstation benötigt keinen eigenen Entwurf, da sie lediglich Pakete zwischen serieller Schnittstelle und Funkschnittstelle vermittelt. Alle wichtigen Informationen zur Basisstation sind im Abschnitt 4.3 zu finden.

4.2.1 Entwurf von Directed Diffusion

Durch eine Anpassung von Directed Diffusion auf den gegebenen Anwendungsfall der Temperaturprofilerfassung kann das Routingprotokoll effizienter in Bezug auf Bandbreite und Rechenleistung gemacht werden. Damit wird implizit auch der Energieverbrauch gemindert. Darüber hinaus ist das Format der Interessentabelle und der Interessen ebenfalls anwendungsabhängig.
Zunächst werden die Interessen im Sensornetzwerk näher betrachtet. Es können insgesamt zwei verschiedenartige Interessen ausgemacht werden. Zum einen ist es die Hauptaufgabe des Sensornetzwerks, die Sensordaten in periodischen Abständen zu erfassen und an die Basisstation zu schicken. Zum anderen gibt es Aufgaben, die nur eine einmalige Aktion bedingen, wie die Aktorenanstuerung und das Setzen von Kalibrierungswerten für die Sensoren. Die beiden Typen werden im Folgenden periodisches und einmaliges Interesse genannt. In Listing 4.1 sind die definierten Paketformate für diese zwei Interessenarten aufgeführt. Beide Typen bestehen aus der Sensorknotenadresse, dem Gradienten des Senders, der Art des Interesses, einer eindeutigen Interessen-ID und einem Zieladressbereich zur Bestimmung des Empfängers. Darüber hinaus besitzt ein periodisches Interesse noch eine Startzeit, eine Zeit für das Messende und die Messauflösung. Ein einmaliges Interesse besitzt hingegen noch eine Variable, um Aktoren Werte mitgeben zu können. Es werden die nx-Datentypen von TinyOS verwendet, welche eine plattformunabhängige Kommunikation zwischen verschiedenen Sensorknoten sicherstellen. Die Bitbreite der Variablen orientiert sich an den Gegebenheiten. Adressen sind in TinyOS 16 Bit breit, FTSP bietet eine 32 Bit breite Zeitinformation an. Die Art des Interesses ist 8 Bit breit, ein Sensorknoten kann damit zwischen 256 verschiedenen Interessen pro Interessentyp unterscheiden.

```
typedef nx_struct PeriodicInterestMsg
{
  nx_uint16_t nodeID;
  nx_uint8_t interest;
  nx_uint16_t interestID;
  nx_uint32_t startTime;
```

```
  nx_uint32_t endTime;
  nx_uint16_t interval;
  nx_uint16_t targetadress;
  nx_uint16_t targetadressEnd;
  nx_uint8_t gradient;
}PeriodicInterestMsg_t;

typedef nx_struct OneShotInterestMsg
{
  nx_uint16_t nodeID;
  nx_uint8_t interest;
  nx_uint16_t interestID;
  nx_int16_t value;
  nx_uint16_t targetadress;
  nx_uint16_t targetadressEnd;
  nx_uint8_t gradient;
}OneShotInterestMsg_t;
```

Listing 4.1: Die Nutzdaten der Interessenpakete im Sensornetzwerk

Directed Diffusion funktioniert vom Prinzip her in dem hier entworfenen Sensornetzwerk genau so, wie es in Unterabschitt 3.6.2 anhand des Fachaufsatzes von Intanagonwiwat et al. vorgestellt wurde. Das PC-Programm versendet über die Basisstation Interessen per Flooding an das Sensornetzwerk, wobei Gradienten aufgebaut werden und die Sensorknoten ihre Antwort auf das Interesse anhand der Gradienten zurück zur Basisstation leiten.
In dem hier entworfenen Sensornetzwerk gehen alle Interessen von der Basisstation aus – beliebige Sensorknoten schicken sich keine Nachrichten. Es ist daher nicht nötig, die Gradienten an ein Interesse zu binden. Es wird zusätzlich zu der Interessentabelle eine Tabelle geführt, in der alle Gradienten aufgenommen werden. Das hat mehrere Vorteile: Einerseits müssen Paare aus Gradienten und Sensorknotenadresse nicht mehrfach vorgehalten werden. Gibt es zwei Interessen mit gleichen Gradienten-Sensorknotenadressen-Paar, so würde die von Intanagonwiwat et al. vorgestellte Variante das Paar doppelt speichern. Andererseits muss bei Erhalt eines neuen Interesses und damit eines neuen Gradienten das Interesse nicht in der Interessentabelle gesucht werden. Dies gilt auch für das Routing der Daten von der Quelle zur Basisstation. Durch eine Trennung zwischen Interessen- und Gradiententabelle müssen nur die Interessen, die von diesem Sensorknoten bedient werden sollen, in die Interessentabelle aufgenommen werden. Es kann somit Speicher, Rechenleistung und dadurch auch Energie gespart werden. Darüber hinaus ist es bei dem hier vorliegenden Sensornetzwerk nicht von Bedeutung, zu welchem Interesse Daten gehören. Sie sollen alle an die Basisstation gesendet werden. Indem man eine Gradiententabelle für alle Daten zum Routen nutzt, ist sichergestellt, dass die Daten immer auf dem besten Weg versendet werden, aber nicht nur das: Hinzu kommt eine größere Sicherheit gegenüber Ausfällen im Sensornetzwerk. Periodische Interessen können mitunter einen sehr langen Zeitraum im Sensornetzwerk bestehen. Fällt während der Bedienung des Interesses der bei seiner Verbreitung gefundene Pfad zur Senke aus, so kann ein Pfad eines anderen, womöglich jüngeren Interesses genutzt werden. Der Aufbau eines Eintrages in der Gradiententabelle besteht, wie in Listing 4.2 dargestellt, aus der Adresse des benachbarten Sensorknotens, dem zugehörigen Gradienten und dem letzten Zeitpunkt, an dem die Verbindung überprüft wurde.
Nachdem die Interessen und die Gradiententabelle definiert wurden, fehlt als letzte Datenstruktur für Directed Diffusion die in Listing 4.3 angegebene Definition der Interessentabelle. Ein Interessentabelleneintrag besteht immer aus der Art des Interesses, der Messauflösung und dem Zeitspanne, in welchem es periodisch bedient werden soll. Die beiden übrigen Spalten *waitedServeIntervals* und *isServed* werden von dem Mechanismus zur Interessenbedienung benötigt und im weiteren Verlauf erläutert.
Im Folgenden wird der Programmfluss von Directed Diffusion anhand von Flussdiagrammen erläutert.
Directed Diffusion wurde für das Sensornetzwerk in zwei Teilen entworfen. Zum einen gibt es den Mechanismus, der Interessen im Netzwerk verbreitet und sowohl die Interessentabelle als auch die Gradienten-

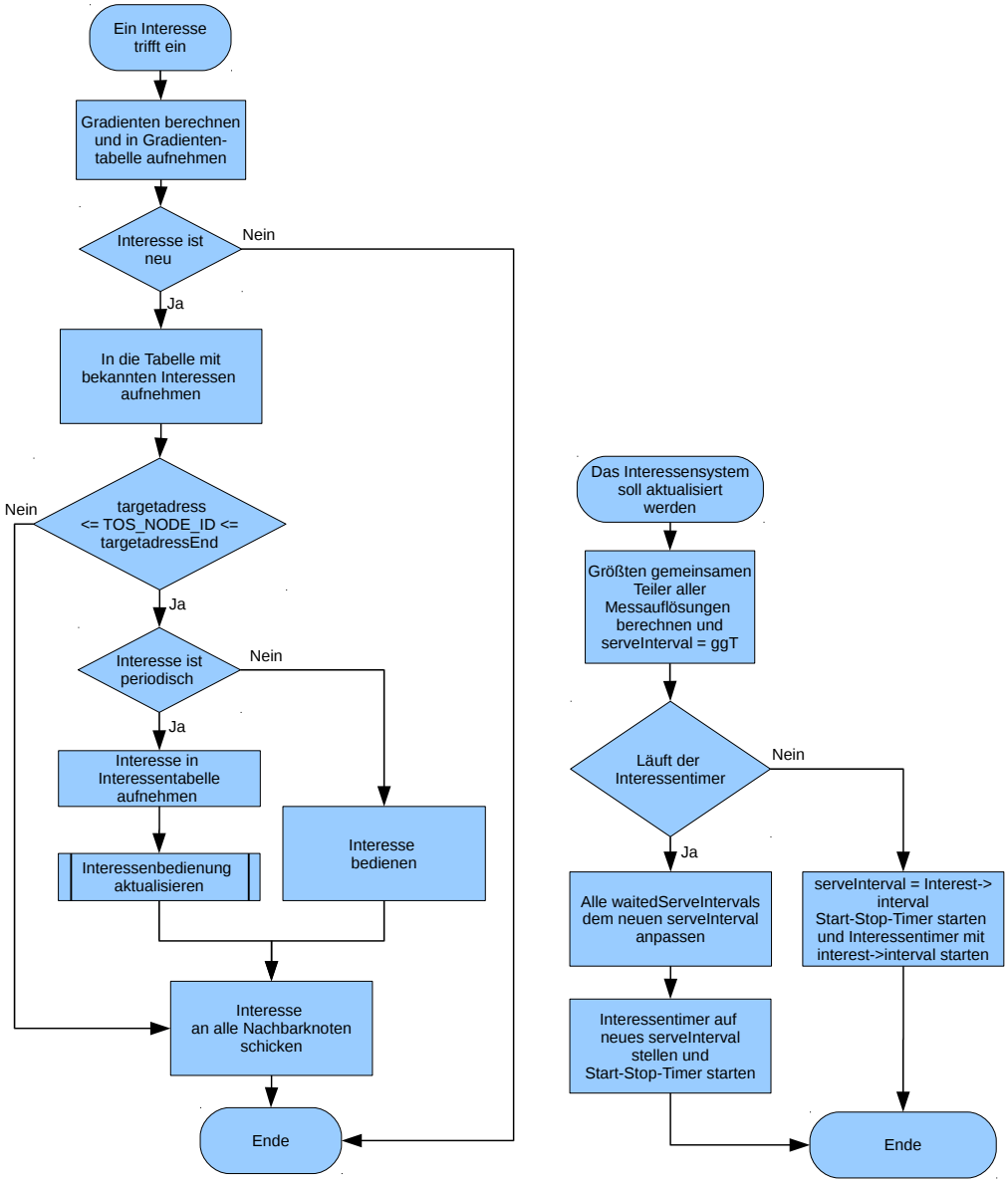
```
typedef struct gradienttableEntry
{
  uint16_t nodeID;
  uint16_t gradient;
  uint32_t lastChecked;
  bool   isFree;
}gradienttableEntry_t;
```

Listing 4.2: Der Aufbau eines Gradiententabelleneintrags

```
typedef struct interesttableEntry
{
  uint8_t interest;
  uint32_t interval;
  uint32_t waitedServeIntervals;
  uint32_t startTime;
  uint32_t endTime;
  bool isServed;
  bool isFree;
}interesttableEntry_t;
```

Listing 4.3: Der Aufbau eines Interessentabelleneintrags

tabelle pflegt. Zum anderen gibt es den Mechanismus, der anhand der Interessentabelle die periodischen Interessen bedient.
Inital wird ein neues Interesse von der Basisstation an alle Nachbarknoten gesendet. Erhält ein Sensorknoten ein neues Interesse, so kümmert sich der Verbreitungsmechanismus, wie in Abbildung 4.3(a) zu sehen, um Verbreitung und Datenhaltung. Zunächst wird der Gradient zum Sender gebildet und in die Gradiententabelle an der richtigen Stelle aufgenommen. Die Gradiententabelle ist sortiert, damit beim deutlich häufiger vorkommenden Routen von Datenpaketen zur Basisstation nicht lange der beste Gradient gesucht werden muss. Anschließend wird anhand der eindeutigen Interessen-ID geprüft, ob der Sensorknoten das Interesse bereits erhalten hat. Hierzu wird eine Liste mit den n letzten erhaltenen Interessen-IDs geführt. Ist die Interessen-ID nicht in dieser Liste vorhanden, so wird das Interesse als neu angesehen. Ist ein Interesse neu, so wird es in die Tabelle mit den bekannten Interessen aufgenommen. Die Überprüfung der Interessen-ID hat den Vorteil, dass gleiche Interessen nicht noch mal bearbeitet werden. Darüber hinaus kann beim Flooding der Interessen zur Verbreitung im Sensornetz die Anzahl der Pakete reduziert werden. Es treten zwar immer noch Reflexionen auf, aber Mehrfachreflexionen und Zyklen werden verhindert. Mit den Ressourcen wird schonend umgegangen, indem doppelt eintreffende Interessen verworfen werden.
Danach wird überprüft, ob das Interesse für den empfangenden Sensorknoten bestimmt ist. Ist dies nicht der Fall, so wird es nur noch an alle Nachbarknoten weiterversendet. Ansonsten wird überprüft, ob es sich um ein periodisches oder einmaliges Interesse handelt. Einmalige Interessen werden nicht in die Interessentabelle aufgenommen, sondern direkt bedient, indem in der Anwendung ein Ereignis ausgelöst wird. Hierbei werden Art des Interesses und der mitgegebene Wert als Parameter übergeben. Periodische Interessen werden in die Interessentabelle aufgenommen und der Mechanismus zur Interessenbedienung auf den neuesten Stand gebracht. Beim Aufnehmen der Interessen in die Interessentabelle muss nachgeschaut werden, ob ein gleichartiges Interesse bereits vorhanden ist. Ein Interesse ist genau dann gleichartig, wenn sowohl die Art des Interesses (*interest*) und die Messauflösung (*interval*) übereinstimmen. In einem solchen Fall wird bei dem vorhandenen Interessentabelleneintrag das Zeitfenster der Messung auf die Werte des neuen Interesses gesetzt. Dies ist eine Designentscheidung, die getroffen wurde, um Interessen auch nachträglich anpassen zu können. Abschließend werden sowohl periodische als auch einmalige Interessen an alle benachbarten Sensorknoten weiterversendet.

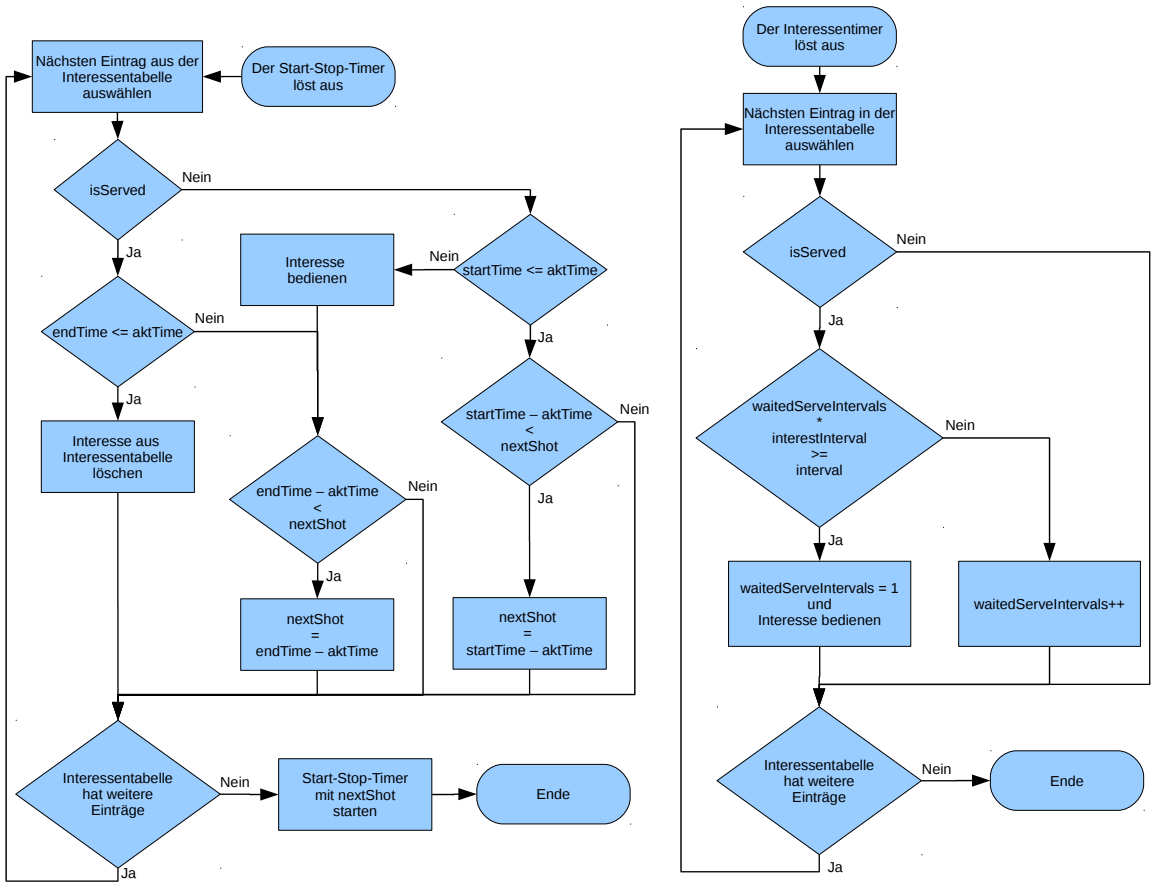


(a) Ein neues Interesse trifft bei dem Sensorknoten ein und wird verarbeitet.

(b) Bei periodischen Interessen muss das Interessensystem auf den neusten Stand gebracht werden.

Abbildung 4.3: Die Flussdiagramme zum Aufnehmen eines neuen Interesses in die Interessentabelle

Der Mechanismus zur Interessenbedienung besteht aus zwei Timern. Der *Start-Stop-Timer* sorgt dafür, dass die Bedienung eines periodischen Interesses gestartet oder beendet wird, indem *isServed* bei einem Interesse in der Interessentabelle entsprechend gesetzt wird. Der *Interessentimer* veranlasst die Bedienung der Interessen mit *isServed* = *true* in der vorgesehenen Messauflösung. Der Status der Interessen ändert sich relativ zu der Bedienung der Interessen in der Regel nur selten. Durch die Unterteilung in aktive und inaktive Interessen muss bei der Bedienung nicht überprüft werden, ob ein Interesse gestartet oder beendet werden soll. Dies spart viel Rechenzeit, was in diesem Fall nicht nur den knappen Ressourcen zugute kommt, sondern auch die maximal mögliche Messauflösung erhöht.



(a) Mit dem Start-Stop-Timer wird geprüft, ob Interessen ab sofort bedient werden sollen oder ein Interesse abgearbeitet wurde.

(b) Der Interessentimer löst aus, und alle ansthenden Interessen werden bedient.

Abbildung 4.4: Die Flussdiagramme des Systems zur Interessenbedienung

Es wird zunächst der Eventhandler des *Start-Stop-Timers* näher betrachtet, vergleiche hierzu das Flussdiagramm in Abbildung 4.4(a). Dieser ändert nicht nur den Status der Interessen, sondern berechnet auch den Zeitpunkt *nextShot*, zu dem die nächste Änderung eines Interessenstatus zu erwarten ist. Hierdurch feuert der *Start-Stop-Timer* immer genau in den Abständen, in denen auch Arbeit ansteht. Auch hier wurde wieder auf einen schonenden Ressourcenumgang geachtet.

Löst der *Start-Stop-Timer* aus, so wird der erste belegte Eintrag in der Interessentabelle selektiert. Wird das Interesse bereits bedient, so wird überprüft, ob das Ende der Messung erreicht ist. Bei Messende wird der Eintrag aus der Interessentabelle gelöscht. Wenn das Ende noch nicht erreicht ist, wird die Zeit bis zum Messende berechnet und mit *nextShot* verglichen. Der Wert von *nextShot* wird auf die berechnete Zeit gesetzt, wenn diese kleiner als *nextShot* ist. Wird das Interesse hingegen noch nicht bedient, so wird geprüft, ob der Startzeitpunkt gekommen ist. In diesem Fall wird *isServed* auf *true* gesetzt und die Zeit bis

zum Messende mit *nextShot* verglichen. Soll das Interesse noch nicht bedient werden, so wird die Zeit bis zum Starten der Messung mit *nextShot* verglichen. Der Wert von *nextShot* wird in beiden Fällen auf den berechneten Wert gesetzt, wenn dieser kleiner als *nextShot* ist. Dieser Vorgang wird für alle Interessentabelleneinträge wiederholt, und anschließend kann der *Start-Stop-Timer* mit dem Wert von *nextShot* neu gestartet werden.
Nun wird der Eventhandler für die Interessenbedienung der periodischen Interessen betrachtet, siehe das Flussdiagramm in Abbildung 4.4(b). Es musste ein Weg gefunden werden, bei dem alle Interessen mit einem Timer bedient werden, da sich die Anzahl der Timer nur zur Kompilierzeit festlegen lässt. Würde man einen bestimmten Pool an Timern vorsehen, so würde entweder viel Speicher unnütz verwendet werden oder die Anzahl der maximal im System vorhandenen Interessen wäre stark beschränkt. Es wurde die Variable *serveInterval* eingeführt, die den Wert des größten gemeinsamen Teilers aller Messauflösungen der Interessen im System annimmt. Der *Interessentimer* löst immer in Perioden der Dauer *serveInterval* aus und überprüft, welche Interessen bedient werden müssen. Die Interessen merken sich innerhalb der Interessentabelle in der Spalte *waitedServeIntervals* die Anzahl der verstrichenen *serveIntervals*.
Löst der *Interessentimer* aus, so wird der erste belegte Eintrag selektiert, der einen aktiven Status besitzt. Ist die Anzahl der gewarteten Intervalle multipiziert mit dem *serveInterval* größer oder gleich der Messauflösung des Interesses, so wird es bedient, indem in der Anwendung ein Ereignis ausgelöst wird. Anschließend kann die Anzahl der gewarteten Intervalle auf 1 gesetzt werden. Andernfalls wird lediglich *waitedServeIntervals* um 1 erhöht. Dieser Vorgang wiederholt sich, bis alle Einträge der Interessentabelle abgearbeitet wurden.
Es wird nun auch deutlich, warum der Mechanismus für neu eintreffende Interessen die Interessenbedienung aktualisieren muss, sobald ein neues Interesse in die Tabelle aufgenommen wird. Dieser Vorgang ist in Flussdiagramm 4.3(b) dargestellt. Zunächst wird der größte gemeinsame Teiler aller Messauflösungen berechnet. Wenn der *Interessentimer* nicht läuft, ist derzeit noch kein Interesse im System vorhanden. Das *serveInterevall* wird dann auf den Wert der Messauflösung des neuen Interesses gesetzt und sowohl *Interessentimer* als auch *Start-Stop-Timer* werden gestartet. Der Aktualisierungsvorgang ist beendet. Läuft der *Interessentimer* bereits, so müssen die Werte der *waitedServeIntervals* angepasst werden. Hierzu werden die gewarteten Intervalle mit dem alten *serveInterevall* multipliziert und anschließend durch das neue *serveInterevall* dividiert. Der *Interessentimer* wird auf das neue *serveInterevall* gesetzt und der *Start-Stop-Timer* wird einmalig ausgelöst, damit das neue Interesse in die Berechnung des *nextShot*-Intervalls eingehen kann.
Bisher wurde lediglich die Bearbeitung eingehender Interessen betrachtet. Es bleibt zu klären, wie mit eintreffenden Datenpaketen von Nachbarknoten und allen vom Sensorknoten zu verschickenden Paketen umgegangen wird. Hierzu wird im Folgenden die Interaktion von Directed Diffusion mit dem Netzwerk erläutert.
TinyOS bietet mit dem Active Message Type die Möglichkeit, mehrere Verbindungen über einen Funkchip aufzubauen, siehe Unterabschnitt 3.5.2. Pakete eines AM-Typs besitzen jeweils einen eigenen Eventhandler und können so auseinandergehalten werden. In dem hier entworfenen Sensornetzwerk wird zwischen drei Paketarten mit eigenem AM-Typ unterschieden. Es gibt Interessenpakete, Datenpakete und Informationspakete. Letztere enthalten die Sensorknotenadresse und Statusinformationen in Form einer acht Bit breiten Zahl. Sie werden insbesondere für Fehlermeldungen genutzt. Informations- und Interessenpakete sollen gegenüber den Datenpaketen priorisiert werden.
Die Active-Message-Schicht bietet ein reines Versenden von Paketen an. Ist zum aktuellen Zeitpunkt keine Verbindung zum Zielknoten herzustellen, so wird das Paket verworfen und ein Fehler gemeldet. Darüber hinaus kann es vorkommen, dass gleichzeitig sowohl die Anwendung als auch Directed Diffusion ein Paket verschicken möchte. Die Anwendung könnte auch zwei Datenpakete so schnell nacheinander senden wollen, dass Directed Diffusion noch mit der Verarbeitung des ersten Paketes beschäftigt ist, wenn das zweite aufläuft.
Um diesen Problemen zu begegen, kann man zwei Warteschlangen für den sequentiellen Zugriff auf den Funk einführen. Steht ein neues Paket zum Versenden an, so kann man es unter gegenseitigem Ausschluss in die jeweilige Warteschlange einfügen. Es wird dort vorgehalten, bis es erfolgreich versendet wurde. Es

gibt zwei Warteschlangen, um eine Priorisierung von Interessen- und Informationspaketen vor den Datenpaketen zu realisieren. Die *fastQueue* nimmt Pakete auf, die schnellstmöglich versendet werden sollen. Die *slowQueue* nimmt Datenpakete auf und sendet sie erst dann, wenn eine bestimmte Anzahl an Paketen zusammengekommen ist. Dies geschieht aus zweierlei Gründen. Zum einen kann man so vor dem Senden der Datenpakete eine Aggregation vornehmen. Zum anderen erreicht man damit einen energieeffizienteren Zugriff auf das Medium. Wie bereits in Unterabschnitt 3.5.2 erläutert wurde, sendet der LPL ein Paket immer wieder neu, bis der Empfänger aufwacht und das Paket entgegennimmt. Ist ein Paket beim Empfänger eingegangen, so erwartet er weitere und bleibt für eine gewisse Zeit lauschbereit. Sendet man also viele Pakete in einem Schwung, so muss lediglich das erste Paket mehrfach versendet werden.
Abschließend wird betrachtet, wie die Pakete geroutet werden. Beim Eintreffen eines neuen Interessenpaketes wird es, wie oben beschrieben, an den Mechanismus zur Interessenverarbeitung weitergeleitet. Trifft hingegen ein neues Informationspaket ein, so wird es in die *fastQueue* eingereiht. Möchte die Anwendung oder Directed Diffusion ein Interesse oder eine Information versenden, so wird ein neues Paket gepackt und ebenfalls in die *fastQueue* eingereiht. Veranlasst die Anwendung hingegen das Verschicken eines neuen Datensatzes, so wird ein Datenpaket gepackt und, genau wie eintreffende Datenpakete, in die *slowQueue* eingereiht.
In Abbildung 4.5 ist das Flussdiagramm des Sendevorgangs zu sehen. Dieser wird immer genau dann gestartet, wenn entweder ein neues Paket in die *fastQueue* oder das n-te Paket in die *slowQueue* eingehängt wurde. Zunächst wird der beste Gradient aus der Gradiententabelle selektiert. Dies ist aufgrund der Sortierug der erste Eintrag. Ist kein Eintrag vorhanden, so wird der Vorgang abgebrochen. Ansonsten wird zuerst die *fastQueue* auf wartende Pakete überprüft. Wenn hier Pakete vorhanden sind, wird das nächste selektiert. Handelt es sich dabei um ein Interessenpaket, wird es an alle Nachbarknoten geschickt. Alle anderen Pakete werden an die Sensorknotenadresse des Gradienten versendet. Sind keine Pakete in der *fastQueue* vorhanden, wird die *slowQueue* auf Pakete überprüft. Wenn dort auch keine Pakete vorhanden sind, wird der Vorgang beendet. Ansonsten wird das nächste Paket der *slowQueue* selektiert und an den Sensorknoten des Gradienten verschickt. In allen Fällen wird anschließend gewartet, bis der Sendevorgang abgeschlossen ist. Wenn kein Fehler auftritt, wird das aktuelle Paket aus seiner Warteschlange entfernt. In jedem Fall beginnt der Vorgang anschließend von vorne.
Die *fastQueue* ist neutral gegenüber dem AM-Typ des Pakets. Jedes Paket aus der *fastQueue* wird auf dem AM-Kanal gesendet, der in dem Paket gesetzt wurde. Die *slowQueue* hingegen sendet alle Daten auf dem Datenkanal. Bei einer möglichen Erweiterung kann so die *fastQueue* auch für beliebige andere Pakete genutzt werden, die *slowQueue* bleibt Datenpaketen vorbehalten, damit hier eine eventuelle Aggregation der Messwerte stattfinden kann.

4.2.2 Entwurf der Anwendung

Im Folgenden wird die Anwendung entworfen. Hierzu wird zunächst geklärt, welche Interessen in dem System innerhalb dieser Arbeit implementiert werden sollen.
Anforderung E1 verlangt die Datenerfassung, siehe Abschnitt 2.2. Die Sensorknoten besitzen Sensoren für die Temperatur, die Luftfeuchte, den Lichtwert und die Batteriespannung. Es soll möglich sein, die Werte in jeder beliebigen Kombination erfassen zu können. Daher gibt es insgesamt $2^4 = 16$ verschiedene periodische Interessen. Die unteren vier Bit von der Art des Interesses stehen hierbei jeweils für eine Messgröße.
Anforderung E6 verlangt die Ansteuerung von Aktoren. Die Sensorknoten besitzen drei LEDs, die hier als Attrappe für einen Aktor herhalten sollen. Das erste einmalige Interesse ist die Ansteuerung der LEDs, wobei der mitgegebene und acht Bit breite Wert den Zustand jeder einzelnen LED kodiert. Die unteren drei Bit geben Auskunft über den zu setzenden Zustand der LED, und die folgenden drei Bit geben an, ob der zu setzende Zustand für die jeweilige LED wirklich gesetzt werden soll. So kann auch der Zustand einer LED verändert werden, ohne die anderen LEDs zu beeinflussen.
Des Weiteren müssen die Sensoren kalibriert werden können. Hierzu werden vier einmalige Interessen

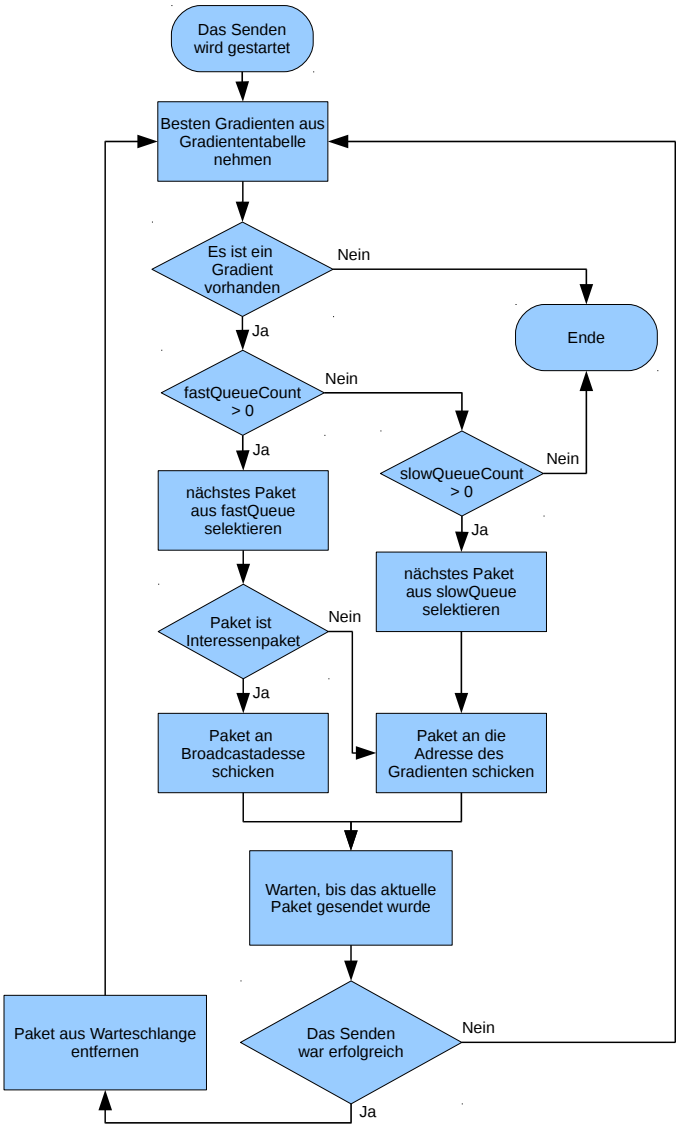


Abbildung 4.5: Das Flussdiagramm zum Senden von Paketen

implementiert, die je einem Sensor einen Offset zum aktuellen Messwert diktieren. Hinzu kommen noch zwei einmalige Interessen, die für Debuggingzwecke gedacht sind. Zum einen kann die Sensorknotenadresse des derzeit besten Gradienten mit einer Informationsnachricht versendet werden, zum anderen kann die Interessentabelle gelöscht werden.

Es sind damit 16 periodische und 7 einmalige Interessen vordefiniert.

Directed Diffusion kapselt bereits die Interessenauslösung. Die Anwendung besteht daher lediglich aus zwei Eventhandlern, welche periodische und einmalige Interessen abarbeiten.

Der Eventhandler der periodischen Interessen schaut nach, ob das Interesse zu den definierten periodischen Interessen gehört. Ist dies nicht der Fall, wird eine Informationsnachricht mit einem Fehler verschickt. Ansonsten werden die zu messenden Werte von der Sensordatenerfassung samt Zeitstempel der Messung angefordert und an die Basisstation versendet. Die Nutzdaten des Pakets bestehen, wie in Listing 4.4 zu sehen, aus der Sensorknotenadresse, dem Zeitstempel der Messung und je einem Wert für jeden Messwert. Da die Temperatur auch negative Werte annehmen kann, ist dieser Wert als einziger mit einem Vorzeichen behaftet.

```
typedef nx_struct dataMsg
{
  nx_uint16_t nodeID;
  nx_uint32_t timestamp;
  nx_uint16_t voltage;
  nx_int16_t temperature;
  nx_uint16_t humidity;
  nx_uint16_t light;
}dataMsg_t;
```

Listing 4.4: Die Nutzdaten der Datenpakete

Es wurde entschieden, immer alle vier Werte zu übertragen, egal wie viele Umweltgrößen tatsächlich erfasst wurden. Diese Entscheidung beruht auf der Annahme, dass bei einer durchgeführten Messung in den meisten Fällen alle Daten erfasst werden sollen, die mit der gegebenen Hardware erfassbar sind. Würde man nur je einen Wert senden, so müsste das Paket einen weiteren Wert mit übertragen. Dieser gäbe an, um welche Umweltgröße es sich handelt. Eine solche Vorgehensweise stellt eine Verschwendung der Bandbreite da. Zudem ist es bei den minimalen Paketgrößenunterschieden günstiger, ein einziges Paket zu versenden, anstatt vier nur etwas kleinere. Es müsste sonst für jedes Paket einzeln bei jedem Hop per CSMA der Zugriff auf das Funkmedium erfolgen. Außerdem besitzt jedes Paket einen Header, wodurch abermals Bandbreite verloren geht. In den wenigen Fällen, in denen nicht alle Werte erfasst werden sollen, setzt die Sensordatenerfassung einen speziellen Wert, um die Nichterfassung zu signalisieren, der außerhalb des Wertebereichs des jeweiligen Sensors liegt.

Der Eventhandler für die einmaligen Interessen schaut analog nach, ob das vorliegende Interesse definiert ist. Auch hier wird eine Informationsnachricht mit einer Fehlernummer bei einem negativen Ergebnis des Prüfvorgangs versendet. Ist das Interesse jedoch vorhanden, so wird ein Task gepostet, der das Interesse implementiert. Einmalige Interessen können mitunter sehr unterschiedlich sein. Sie werden innerhalb eines Tasks implementiert, da nicht abzusehen ist, wie lange die Ausführung dauert.

Sowohl die periodischen als auch die einmaligen Interessen können sehr leicht erweitert werden. Bei periodischen Interessen kann man innerhalb des Eventhandlers einfach einen neuen Fall bei der Fallunterscheidung der Interessenart hinzufügen. Bei einmaligen Interessen implementiert man einen neuen Task und lässt diesen analog innerhalb der Fallunterscheidung im Eventhandler der einmaligen Interessen starten.

4.2.3 Entwurf der Sensordatenerfassung

Abschließend erfolgt der Entwurf der Sensordatenerfassung. Sie muss in der Lage sein, von der Anwendung den Befehl zum Erfassen der Werte entgegenzunehmen, dann die entsprechen Sensorwerte abzufragen und diese anschließend mit dem Zeitstempel der Messung wieder an die Anwendung zu melden. Hierbei ist zu beachten, dass unter Umständen bereits neue Werte angefragt werden, während die alten noch nicht fertig erfasst wurden. Dies ist genau dann der Fall, wenn mehrere Messungen mit sehr geringer Messauflösung gleichzeitig in der Interessentabelle vorhanden sind. In einem solchen Fall soll die zweite Messung dennoch stattfinden. Sie findet dann zwar nicht zu dem richtigen Zeitpunkt statt, dieser Fehler kann jedoch später anhand des Zeitstempels erkannt werden. Die Sensordatenerfassung soll jede in Auftrag gegebene Messung durchführen. Um dies zu erreichen, wird eine Warteschlage für die Messungen eingeführt.

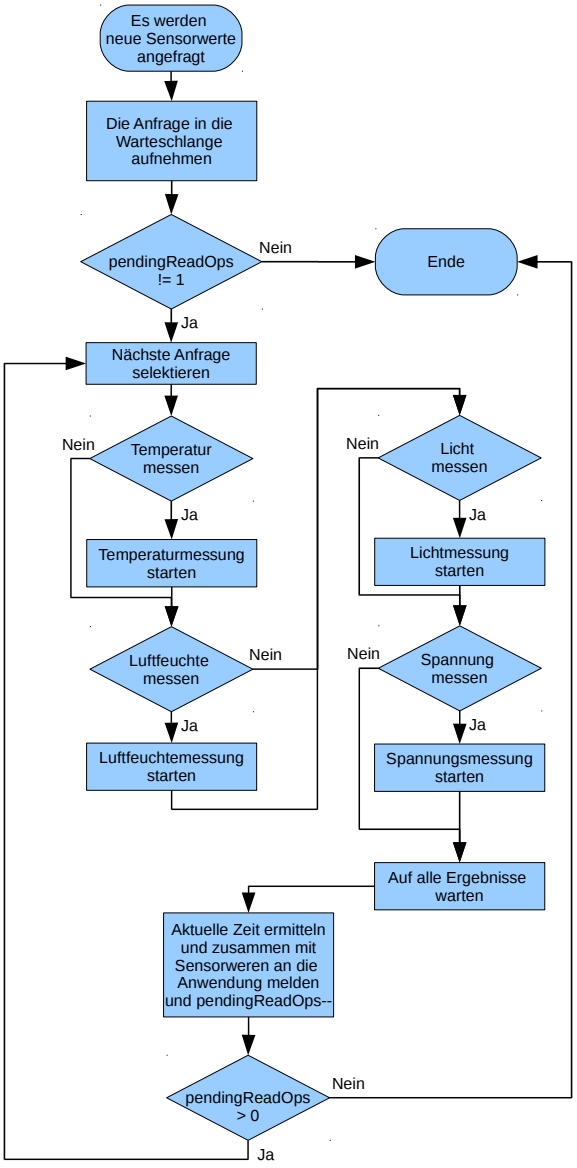


Abbildung 4.6: Das Flussdiagramm zum Anfordern neuer Sensordaten

In Abbildung 4.6 ist das Flussdiagramm für die Sensordatenerfassung dargestellt. Eine neue Anfrage wird

in die Warteschlage aufgenommen. Anschließend wird überprüft, ob bereits Messungen laufen. Ist dies der Fall, ist diese Anfrage hier zunächst beendet. Die aktuelle Messung wird die neue Anfrage starten, sobald sie abgeschlossen und die neue Messung an der Reihe ist. Andernfalls wird die nächste Messung aus der Warteschlange selektiert. Darauffolgend wird die Messung der relevanten Größen gestartet. Anschließend wird auf das Ende aller Messungen gewartet, die aktuelle Zeit beim Synchronisationsprotokoll erfragt und an die Anwendung gemeldet. Sind alle anstehenden Messungen aus der Warteschlange abgearbeitet, endet die Sensordatenerfassung. Ansonsten wird der Messvorgang mit der nächsten Messung wiederholt.
Bei der Sensordatenerfassung wurde insbesondere darauf geachtet, dass alle Messungen zunächst gestartet werden und dann erst auf alle Ergebnisse gewartet wird. Von der Implementierung her ist es deutlich einfacher, die Messungen sequentiell durchzuführen. In einem solchen Fall würde sich die Dauer des Messvorgangs jedoch verlängern. Die Sensoren sind als externe Komponenten anzusehen, die alle parallel arbeiten können. Diese Möglichkeit sollte unbedingt genutzt werden, damit der Zeitstempel möglichst nahe an der tatsächlichen Erfassungszeit liegt.

4.3 Implementierung der Basisstation

Die Basisstation hat die Aufgabe, an der seriellen Schnittstelle eintreffende Pakete an das Sensornetzwerk zu senden und vom Sensornetzwerk empfangene Pakete an die serielle Schnittstelle weiterzuleiten. Dabei werden weder die Nutzdaten noch die Informationen im Header der Pakete verändert. Es bleibt im Header insbesondere der AM-Typ und die Zieladresse erhalten. Darüber hinaus ist die Basisstation die Masteruhr. Die Sensorknoten synchronisieren sich immer gegenüber der Uhr des Sensorknotens mit der niedrigsten Adresse. Daher erhält die Basisstation immer die Sensorknotenadresse 0. Zudem ist die Basisstation dafür zuständig, dem PC-Programm in regelmäßigen Abständen die aktuelle Zeit des Sensornetzwerks mitzuteilen. Das PC-Programm nutzt diese Daten, um die Zeitstempel der Messwerte in die aktuelle mitteleuropäische Zeit umzurechnen.
Für die Implementierung der Basisstation wird die Komponente BaseStation, welche TinyOS als Beispielimplementierung beiliegt, als Grudlage genutzt. Diese implementiert bereits die Vermittlung der Pakete zwischen serieller und Funkschnittstelle. Sie wurde so erweitert, dass sie ebenfalls den LPL zum energieeffizienten Zugriff auf das Funkmedium nutzt. Die Basisstation ist zwar direkt an den PC angeschlossen und verfügt somit über eine unerschöpfliche Energieversorgung, jedoch muss zur Kommunikation mit den Sensorknoten selbstverständlich dasselbe Zugriffsprotokoll genutzt werden. Darüber hinaus wurde die Basisstation um das FTSP und einen Timer erweitert, dessen Eventhandler die aktuelle Zeit an den PC sendet. Die Originalquelltexte der Komponente BaseStation befinden sich zum Vergleich auf der beiliegenden CD, siehe im Anhang A.
In Abbildung 4.7 ist das Komponentendiagramm der Basisstation zu sehen. Die Komponente der Basisstation besteht aus den Komponenten MainC, TimeSyncC, TimerMilliC, LedsC, SerialActiveMessageC, ActiveMessageC und CC2420ActiveMessageC sowie aus dem Modul mit der Implementierung der Basisstation. MainC bootet den Sensorknoten und bindet die Kernkomponenten von TinyOS ein, und TimeSyncC stellt eine globale Uhrzeit zur Verfügung. Die Komponenten SerialActiveMessageC, ActiveMessageC und CC2420ActiveMessageC bilden die beiden Netzwerkstapel.
Über die Schnittstelle Boot, die von der Komponente MainC angeboten wird, erhält das Modul der Basisstation ein Boot-Event, sobald der Sensorknoten gebootet wurde. Im Booteventhandler wird Timer1 periodisch mit einem Intervall von zwei Sekunden gestatet. Dies bedeutet, dass er fortan alle zwei Sekunden ein Timerevent auslöst. Darüber hinaus werden die Schnittstellen zur Kommunikation über die SplitControl-Schnittstelle gestartet, die Paketvermittlung ist danach aktiv. Zur Paketvermittlung werden die beiden Komponenten Serial und Radio mit ihren Schnittstellen zur Active-Message-Schicht genutzt. Die Komponente CC2420ActiveMessageC stellt dem Basisstationsmodul die Schnittstelle zum Setzen der Zyklendauer des LPLs zur Verfügung.
Die Komponente TimeSyncC implementiert das FTSP, das ebenfalls die Bootschnittstelle nutzt, um nach

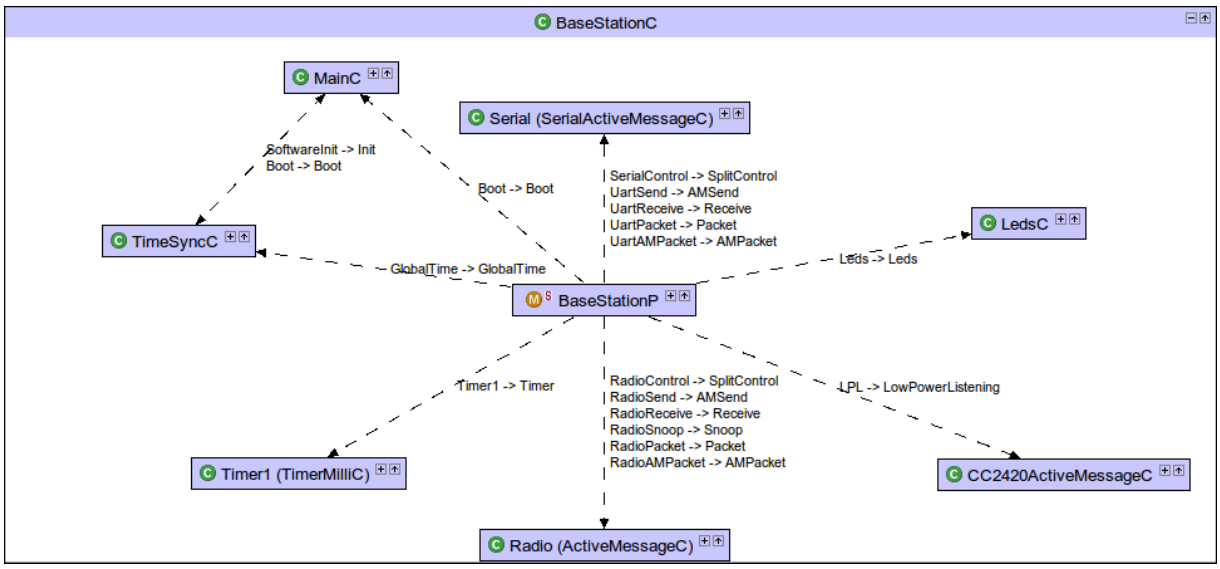


Abbildung 4.7: Das Komponentendiagramm der Basisstation

dem Programmstart einen eigenen Timer zu starten. Dieser sendet fortan die Synchronisationspakete an das Sensornetzwerk, wie es in Unterabschnitt 3.7.4 beschrieben wurde.
Löst Timer1 aus, so wird im Eventhandler die von TimeSyncC implementierte Schinttstelle GlobalTime genutzt, um die aktuelle Zeit zu erfragen und auf Validität zu überprüfen. Sieht das FTSP den Sensorknoten als synchron an, was nach einer initalen Setupphase immer der Fall ist, weil der Sensorknoten die niedrigste Adresse besitzt, so sendet es die Zeitinformationen an den PC. Hierbei wird lediglich ein Zeitstempel übertragen, vergleiche Listing 4.5.
Die Komponente LedsC implementiert die Ansteuerung der LEDs. Beim Weiterleiten eines Pakets wird der Zustand der ersten LED geändert; muss ein Paket aufgrund eines Fehlers fallen gelassen werden, ändert sich der Zustand der zweiten LED.

```
1 typedef nx_struct timeMsg
  {
3   nx_uint32_t timestamp;
  }timeMsg_t;
```

Listing 4.5: Die Nutzdaten der Zeitpakete an den PC

4.4 Implementierung der Sensorknoten

In diesem Unterabschnitt wird die Implementierung der Sensorknoten vorgestellt.
Das Komponentendiagramm der Anwendung zur Erfassung eines Temperaturprofils für Wohn- und Büroräume ist in Abbildung 4.8 zu sehen. Die Komponente SensorMoteAppC besteht aus den Komponenten MainC, TimerMilliC, LedsC, TimeSyncC, DirectedDiffusionC, SensorikC und dem Modul, das die Anwendung implementiert. MainC bindet analog zur Implementierung der Basisstation die Kernkomponenten von TinyOS ein, und die Anwendung nutzt die Schnittstelle Boot. TimeSyncC stellt wieder die Schnittstelle GlobalTime zur Verfügung und implementiert das FTSP. Die Komponente SensorikC implementiert die Sensordatenerfassung und bietet die Schnittstelle Sensorik an, während die Komponente DirectedDiffusionC das Routingprotokoll implementiert, welches die Schnittstellen DirectedDiffusion und

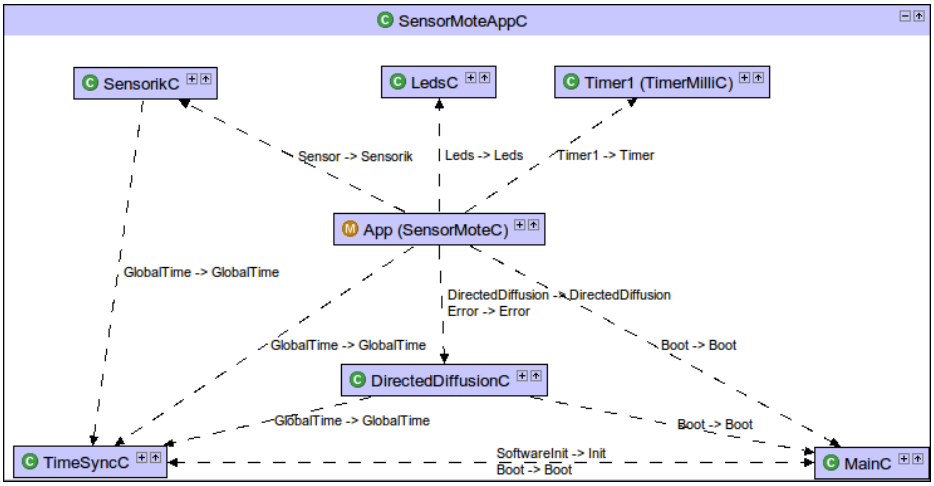


Abbildung 4.8: Das Komponentendiagramm der Sensorknoten

Error anbietet. Die Konfiguration der Komponente SensorMoteAppC versorgt SensorikC und DirectedDiffusionC mit der Schnittselle GlobalTime sowie TimeSyncC und DirectedDiffusionC mit der Schittstelle Boot.
Das Modul SensorMoteC implementiert die Anwendung, indem es für alle Ereignisse der Schnittstellen Boot, Timer, Sensorik und DirectedDiffusion je einen Eventhandler implementiert. Sobald TinyOS gebootet hat, wird über die Bootschnittstelle das Ereignis *Booted* ausgelöst. Die Anwendung startet draufhin einen periodischen Timer. Löst der Timer aus, wird nachgeschaut, ob sich der Sensorknoten mit dem Sensornetzwerk synchronisiert hat. Besitzt der Sensorknoten eine valide globale Uhrzeit, so wird DirectedDiffusionC mit dem Kommando *NodeInSync()* freigegeben, und der Sensorknoten ist bereit. Der Timer wird daraufhin gestoppt. Andernfalls beginnt der Vorgang von neuem, und die LEDs werden als optisches Zeichen zum Blinken gebracht. Die Ereignisse der Schnittstellen Sensorik und DirectedDiffusion werden zu einem späteren Zeitpunkt beleuchtet.

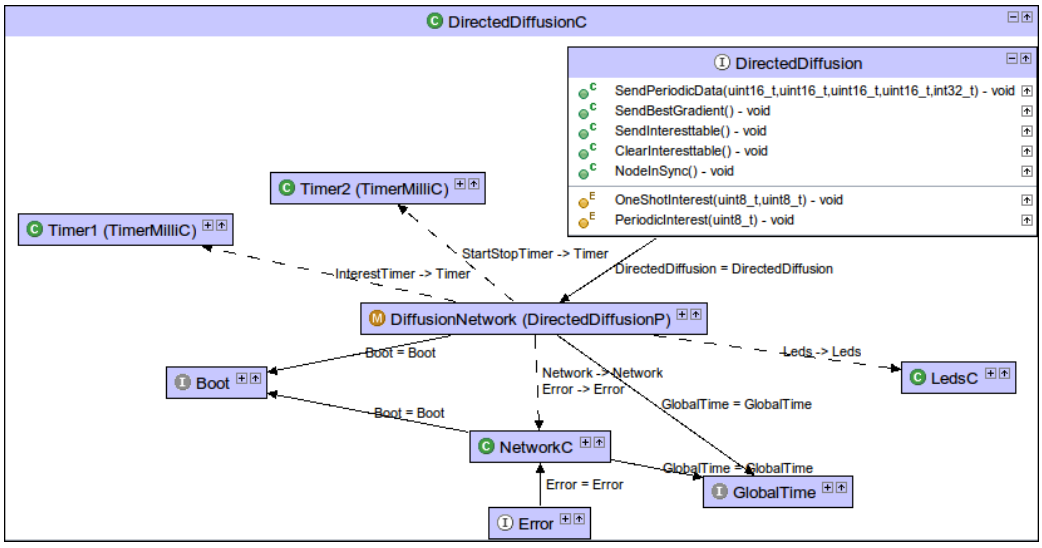


Abbildung 4.9: Das Komponentendiagramm von Directed Diffusion

Die Komponente DirectedDiffusionC besteht, wie im Komponentendiagramm in Abbildung 4.9 zu se-

hen, neben dem Modul DirectedDiffusionP aus den Komponenten NetworkC, LedsC und zweimal der Komponente TimerMilliC. Die Implementierung von Directed Diffusion wurde in zwei logische Blöcke unterteilt. Das Modul DirectedDiffusionP implementiert die Mechanismen zur Interessenverbreitung und -verarbeitung, und die Komponente NetworkC kümmert sich um Paketverwaltung und -versand. Dazu bietet sie die beiden Schnittstellen Network und Error an. Die beiden Timer sind zum einen der *Interessentimer* und zum anderen der *Start-Stop-Timer*. Vergleiche hierzu auch Unterabschnitt 4.2.1. Darüber hinaus verbindet die Konfiguration der Komponente DirectedDiffusionC die Schnittstellen Boot und GlobalTime mit NetworkC.
Das Modul DirectedDiffusionP implementiert die von der Anwendung zur Interessenbedienung benötigten Kommandos der Schnittstelle DirectedDiffusion. Die Kommandos *sendPeriodicData()*, *sendBestGradient()* und *sendInterestTable()* rufen mit den übergebenen Parametern ein gleichnamiges Kommando der Schnittstelle Network auf, da die Kommandos in der Paketverwaltung bearbeitet werden. Der Befehl *clearInterestTable()* hält die beiden Timer an und löscht den Inhalt der Interessentabelle. Der Aufruf *nodeInSync()* wird ebenfalls an NetworkC delegiert.
Die beiden Ereignisse *OneShotInterest* und *PeriodicInterest* werden ausgelöst, wenn die Anwendung ein Interesse beantworten soll.

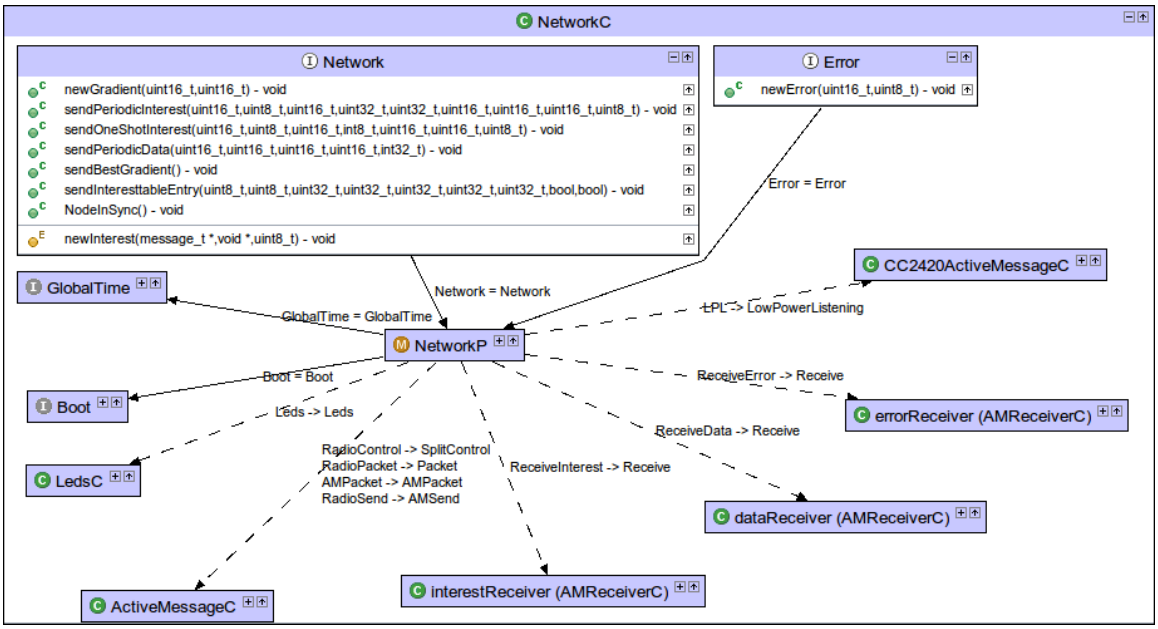


Abbildung 4.10: Das Komponentendiagramm von NetworkC

In Abbildung 4.10 ist das Komponentendiagramm von der Komponente NetworkC abgebildet. Diese besteht aus den Komponenten LedsC, ActiveMessageC, CC2420ActiveMessageC und dreimal AMReceiverC. ActiveMessageC und CC2420ActiveMessageC implementieren den energieeffizenten Netzwerkstapel für den Funk und bieten die Schnittstellen zum Starten des Funkchips, zum Senden von Paketen, zum Setzen des Schlafintervalls des LPL und zum Bearbeiten der Header der Pakete an. Die drei Instanzen der Komponente AMReceiverC bieten die Schnittstelle zum Empfangen von Paketen aus der Active-Massage-Schicht an. Jeder dieser Empfänger gehört zu einem der in Unterabschnitt 4.2.1 eingeführten Kanäle. So werden drei verschiedene Ereignisse ausgelöst, je nachdem, ob ein Interessenpaket, ein Informationspaket oder ein Datenpaket eingetroffen ist.
Sobald der Sensorknoten gebootet ist, wird auch im Modul NetworkP das Ereignis *Booted()* ausgelöst. Daraufhin wird der Netzwerkstapel gestartet und die Datenstrukturen initialisiert. Interessenpakete werden jedoch erst angenommen, wenn das Modul SensorMoteC die Synchronität meldet. Daten- und Informati-

onspakete treffen noch nicht ein, da dieser Sensorknoten noch keine Interessen gesendet und daher kein anderer ihn in seiner Gradiententabelle hat. Würde er dennoch solche Pakete erhalten, so würde er sie in eine der Warteschlangen einreihen und versenden, sobald er selber ein Interesse und damit einen Gradienten empfängt.
Das Modul NetworkP bietet die Schnittstellen Network und Error an und implementiert daher dessen Kommandos.
Die Kommandos *sendPeriodicData()*, *sendBestGradient()* und *sendInterestTable()* sorgen wie bereits erwähnt für die Interessenbedienung der Schnittstelle Network. Das Kommando *sendPeriodicData()* packt aus den übergebenen Daten ein Datenpaket und reiht es in die *slowQueue* ein, während *sendBestGradient()* die Sensorknotenadresse des besten Gradienten nimmt und sie in einem Informationspaket verpackt, welches in die *fastQueue* eingereiht wird. Bisher nicht implementiert ist *sendInterestTable()*. Die Kommandos *sendOneShotInterest()*, *sendPeriodicInterest()* und *newGradient()* werden vom Mechanismus der Interessenverbreitung im Modul DirectedDiffusionP genutzt, um neue Interessen zu versenden und einen neuen Gradienten in die Gradiententabelle aufzunehmen. Beim Versenden neuer Interessen wird aus den übergebenen Werten das jeweilige Interessenpaket gepackt und in die *fastQueue* eingehängt. Wird ein neuer Gradient übergeben, so wird zunächst die Gradiententabelle aktualisiert, also veraltete Gradienten entfernt, und anschließend der neue Gradient einsortiert.
Darüber hinaus definiert die Schnittstelle Network das Event *newInterest()*. Dieses Ereignis wird von NetworkP beim Eintreffen eines neuen Interessenpakets ausgelöst. DirectedDiffusionP implementiert einen Eventhandler, der das Interesse entgegennimmt und verarbeitet.
Die Schnittstelle Error bietet allen Komponenten die Möglichkeit an, Informationspakete zu verschicken. NetworkP packt analog zu den oben vorgestellten Kommandos ein Informationspaket aus den übergebenen Parametern und reiht es in die *fastQueue* ein. Die Informationspakete werden derzeit hauptsächlich für Fehlernachrichten verwendet; es sind die in Listing 4.6 vordefinierten Werte mit den Bedeutungen zu entnehmen.

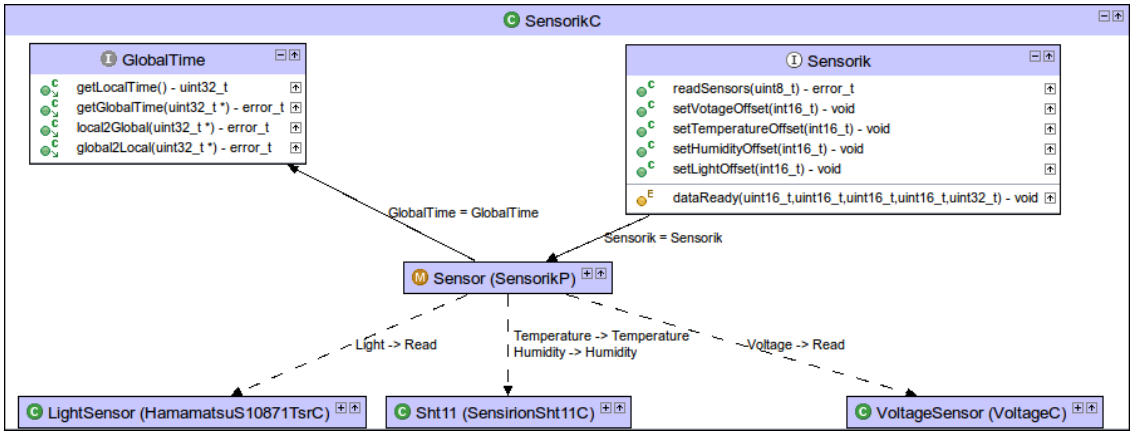


Abbildung 4.11: Das Komponentendiagramm der Sensordatenerfassung

Das Komponentendiagramm für die Sensordatenerfassung findet sich in Abbildung 4.11. Die Komponente SensorikC besteht neben dem Modul SensorikP aus den Komponenten VoltageC, SensirionSht11C und HamamatsuS10871TsrC. Dies sind die Treiber für den Spannungssensor, den Temperatur- sowie Luftfeuchtesensor und den Lichtsensor. Sie bieten jeweils eine Schnittstelle zum Durchführen einer Messung an.
Das Modul SensorikP implementiert die Kommandos der Schnittstelle Sensorik. Diese sind zum einen die vier Kommandos zum Setzen der Kalibrierungswerte und zum anderen das Kommando *readSensor()* zum Initiieren einer neuen Messung. Wird *readSensor()* aufgerufen, so wird wie in Unterabschnitt 4.2 erläutert

```
enum{
  //Die Interessentabelle ist voll.
  INTERESTTABLE_IS_FULL = 1,
  //Es trat ein Fehler beim Senden auf.
  SENDDONE_ERROR = 2,
  //Beim Senden wurden zwei Datenpakete vertauscht (sollte nicht
     vorkommen knnen).
  SEND_ERROR = 3,
  //Es ist kein Gradient in der Gradiententabelle vorhanden.
  NO_GRADIENT_ERROR = 4,
  //Ein eintreffendes Paket hat nicht die Laenge, die es haben sollte.
  PACKET_VERIFICATION_ERROR = 5,
  //Die Zeitsynchroniation ist derzeit nicht valide.
  NOT_IN_SYNC_ERROR = 6,
  //Die Gradiententabelle ist voll.
  GRADIENT_TABLE_FULL_ERROR = 7,
  //Das empfangene Interesse ist weder ein periodisches noch ein
     einmaliges.
  UNKNOWN_INTEREST_MSG_ERROR = 8,
  //Das One-Shot-Interessensystem ist derzeit beschaeftigt, ein
     Interesse wurde fallengelassen.
  ONE_SHOT_INTEREST_DROP_ERROR = 9,
  //Die Art des Interesses ist der Anwendung nicht bekannt.
  INTEREST_NOT_IMPLEMENTED = 10,
};
```

Listing 4.6: Die im System vordefinierten Fehler

eine Messung der übergebenen Messgrößen gestartet. Ist eine Messung beendet, so erzeugt SensorikP das Ereignis *dataReady()* und übergibt so der Anwendung das Ergebnis.

Fragt man bei einem der Sensorentreiber einen Messwert an, so erhält man als Ergebnis zunächst nur einen Auslesewert, welcher in die jeweilige Messgröße innerhalb des Moduls SensorikP umgerechnet wird.

Die Umrechnung der Temperatur T in C° und der relativen Luftfeucht RH_{linear} ohne Temperaturkompensation und RH_{true} mit Temperaturkompensation in % kann bei 3 Volt Versorgungsspannung und dem Auslesewert SO laut dem Datenblatt des SHT11 [Fird] berechnet werden mit:

$$T = -39,6 + 0,1 \cdot SO_T$$

$$RH_{linear} = -4 + 0,0405 \cdot SO_{RH} - 2,8E^{-6} + SO^2$$

$$RH_{ture} = (T - 25) \cdot (0,01 + 0,00008 \cdot SO_{RH}) + RH_{linear}$$

Der Wert für die Spannung wird über den A/D-Wandler des Mikrocontrollers erfasst. Dieser besitzt eine Auflösung von 12 Bit. Um die Spannung zu erhalten, wird der A/D-Wandlerwert durch die Messauflösung dividiert und mit der Referenzspanung des Wandlers multipliziert:

$$V = \frac{SO}{4096} \cdot 3$$

Der Lichtwert wird mit einer Fotodiode bestimmt, die einen Widerstand von 100 kOhm besitzt. Man kann mit dem Ohmschen Gesetz ($V = I \cdot R$) den gemessenen Spannungsabfall über dem Sensor in den Stromfluss durch den Sensor umrechnen und anhand des Datenblattes [Firb] damit den Wert der totalen solaren Einstrahlung berechnen:

$$I = \frac{\frac{SO}{4096}}{100000}$$

$$TSR = 1E^5 \cdot I \cdot 1000$$

Die Werte für die Spannung und die Temperatur werden in SensorikP dabei jedoch noch mit 100 multipliziert, um mit den ganzzahligen Datenpaketen für diese Werte zwei Nachkommastellen zu übertragen.
Abschließend soll das Zusammenspiel der einzelnen Komponenten anhand eines Ablaufdiagrammes verdeutlicht werden. Dieses ist angelehnt an die Sequenzdiagramme bei der objektorientierten Programmierung. Eine spezielle Form der Sequenzdiagramme für TinyOS existiert nicht.

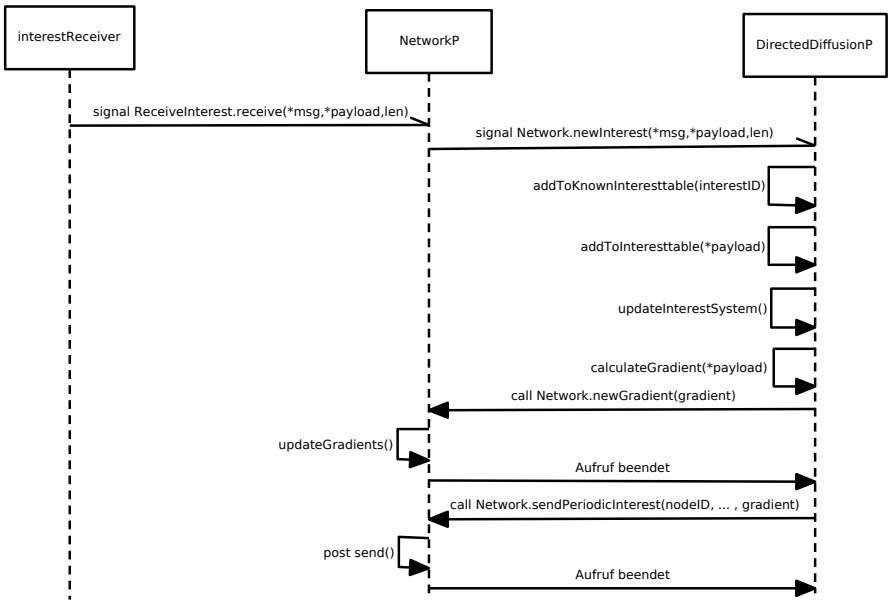


Abbildung 4.12: Das Ablaufdiagramm für ein eintreffendes periodisches Interesse

Im Ablaufdiagramm in Abbildung 4.12 wird angenommen, dass den Sensorknoten ein neues periodisches Interesse erreicht, welches an ihn gerichtet ist. Wenn ein neues Interessenpaket eintrifft, wird in NetworkP der Eventhandler zum Empfangen von periodischen Interessen aufgerufen. Dieser erzeugt über die Schnittstelle das Ereignis *newInterest* bei dem Modul DirectedDiffusionP und übergibt an dessen Eventhandler eine Referenz auf das Paket sowie die Nutzdaten und Nutzdatenlänge. In DirectedDiffusionP wird die Interessen-ID in die Liste der bekannten Interessen aufgenommen und das Interesse in die Interessentabelle eingepflegt. Darauffolgend wird das Interessenverarbeitungssystem auf den neusten Stand gebracht und der Gradient berechnet. Der Gradient ist in der derzeitgen Implementierung die Hop-Entfernung zur Datensenke. Bei der Berechnung des Gradienten wird also zu dem Gradienten einfach 1 addiert. Danach wird der neue Gradient an das Modul NetworkP übergeben. Hier wird zunächst die Gradiententabelle aufgeräumt und dann der Gradient aufgenommen. Nachdem der Kommandoaufruf zurückgekehrt ist, ruft DirectedDiffusionP das Kommando *sendPeriodicInterest* der Schnittstelle Network auf und übergibt dabei die Daten des neuen Interesses. Dies sind die Daten des alten Interesses mit dem neu berechneten Gradienten. NetworkP packt ein neues Interessenpaket, reiht es in die *fastQueue* ein und startet den Sendetask. Der Kommandoaufruf kehrt erneut zurück, und das neue Interesse wurde verarbeitet. Jetzt startet der Sendetask und versendet über die Active-Message-Schicht das Paket an alle benachbarten Sensorknoten.
Es vergeht eine bestimmte Zeit, bis das Interesse bedient werden soll. Dann startet der Programmfluss, wie er im Ablaufdiagramm in Abbildung 4.13 zu sehen ist. Der *Start-Stop-Timer* feuert und löst ein Ereignis in DirectedDiffusionP aus. Der *isServed*-Eintrag des Interesses wird jetzt in der Interessentabelle auf

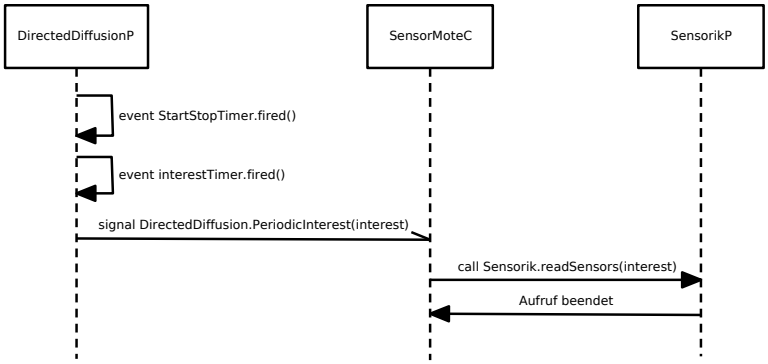


Abbildung 4.13: Das Ablaufdiagramm für die Bedienung eines periodischen Interesses (1 von 2)

wahr gesetzt und der *Interessentimer* gestartet. Er feuert, sobald das Interesse bedient werden soll. Daraufhin wird das Ereignis *PeriodicInterest* der Schnittstelle DirectedDiffusion erzeugt und damit im Modul SensorMoteC der passende Eventhandler gestartet. Dieser bekommt die Art des Interesses mit übergeben. SensorMoteC ruft das Kommando *readSensors* der Schnittstelle Sensorik auf und gibt so die Anfrage nach dem Interesse weiter. In SensorikC läuft der Eventhandler an, der bei den Treibern der Sensoren ein Ereignis zur Anfrage der Sensorwerte erzeugt. Der Programmfluss kehrt zu SensorMoteC zurück, und die Interessenbedienung ist vorerst beendet.

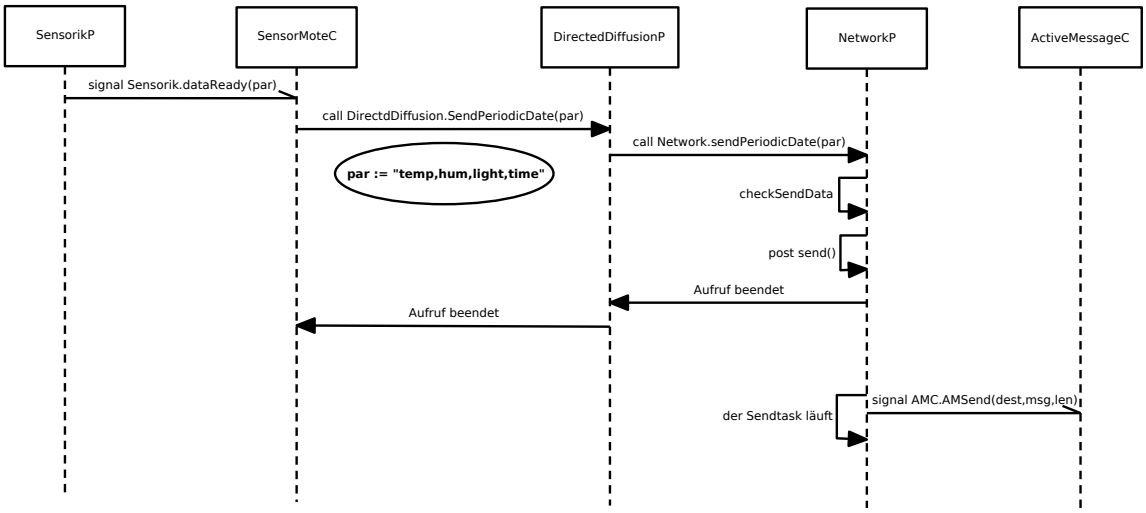


Abbildung 4.14: Das Ablaufdiagramm für die Bedienung eines periodischen Interesses (2 von 2)

In Abbildung 4.14 ist das Ablaufdiagramm für den zweiten Teil der Interessenbedienung zu sehen. Sobald die Sensoren die Werte liefern, erfragt das Modul SensorikP über die Schnittstelle GlobalTime bei der Zeitsynchronisation die aktuelle globale Zeit, prüft, ob sie valide ist, und erzeugt dann das Ereignis *dataReady*. SensorMoteC nimmt die Werte und den Zeitstempel im Eventhandler entgegen und ruft damit das Kommando *SendPeriodicData()* bei DirectedDiffusionC auf. Dieses Modul leitet den Aufruf abermals mit einem gleichnamigen Kommando an das Modul NetworkP weiter. Hier wird aus den Daten ein Datenpaket gepackt und als Zieladresse der beste Gradient genutzt. Dann wird das Paket in die *slowQueue* eingereiht. Mittels *checkSendData()* wird danach überprüft, ob bereits genügend Datensätze zum Senden vorliegen. Ist dem so, wird der Sendetask gepostet. Anschließend kehrt der Kommandoaufruf bis in das Modul Sen-

sorMoteC zurück, und die Bedienung des Interesses ist beendet. Jetzt kann der Sendetask den Prozessor bekommen und über ActiveMessageC das Paket an die Zieladresse senden.

4.4.1 Einstellungsparameter

Um den Ressourcenbeschränkungen gerecht zu werden, sollten die verwendeten Datenstrukturen immer so klein wie möglich gewählt werden. Ist eine Tabelle zu groß, so geht Speicher und beim Arbeiten in der Tabelle auch Rechenleistung verloren. Aus diesem Grund wurden in der Datei *defines.h* Konstanten definiert, welche die Größe der wichtigen Datenstrukturen an einer zentralen Stelle definieren. So kann der Sensorknoten immer auf den aktuellen Anwendungsfall angepasst werden. Diese Konstanten werden im Folgenden mit ihrem Standardwert gennannt und erläutert.

FASTQUEUELEN = 20 Gibt die Anzahl der Pakete an, die in der *fastQueue* aufgenommen werden können.

SLOWQUEUELEN = 50 Gibt die Anzahl der Pakete an, die in der *slowQueue* aufgenommen werden können.

GRADIENTTABLEENTRIES = 10 Gibt die Anzahl der Einträge in der Gradiententabelle an.

GRADIENTAGE = 360000 Wenn die Gradienten älter als GRADIENTAGE Millisekunden sind, werden sie aus der Gradiententabelle gelöscht, sofern noch andere, jüngere Gradienten vorhanden sind.

GRADIENTUPDATEPACKETGECOUNT = 30 Die Überprüfung der Gradienten ist sehr rechenintensiv. Um Ressourcen zu schonen, wird die Überprüfung des Gradientenalters nur alle GRADIENTUPDATEPACKETCOUNT Pakete durchgeführt.

INTERESTTABLEENTRIES = 20 Gibt die Anzahl der Einträge in der Interessentabelle an.

INTERESTHISTORYCOUNT = 10 Gibt die Anzahl der Einträge in der Tabelle mit den bekannten Interessen an. Es ist zu beachten, dass nicht mehr als INTERESTHISTORYCOUNT verschiedene Interessen innerhalb einer Zeitspanne von ca. einer Sekunde versendet werden sollten.

PACKETSTOHOLD = 25 Gibt die Anzahl der zu sammelnden Datenpakete in der *slowQueue* an, bevor sie versedet werden sollen. Es ist zu beachten, dass PACKETSTOHOLD immer echt kleiner als SLOWQUEUELEN sein muss. Ansonsten würden nie Datenpakete versendet werden. Es wird empfohlen, die Hälfte zu wählen, damit in einem Fehlerfall noch Datenpakete zwischengespeichert werden können.

MAXSENSORREADINGSPENDING = INTERESTTABLEENTRIES Gibt die Größe der Warteschlange für Sensoranfragen in der Sensordatenerfassung an. Jedes Interesse in der Interessentabelle könnte gleichzeitig eine Anfrage stellen. Daher ist es logisch, genau so viele Anfragen auch annehmen zu können.

5 Entwurf und Implementierung des PC-Programms

In diesem Kapitel wird der Entwurf und die Implementierung des PC-Programms zur Speicherung und Betrachtung der Daten aus dem Sensornetzwerk vorgestellt. Mit diesem kann der Benutzer wie in der Anforderungsdefinition in Kapitel 2 festgelegt das Sensornetzwerk betreiben.

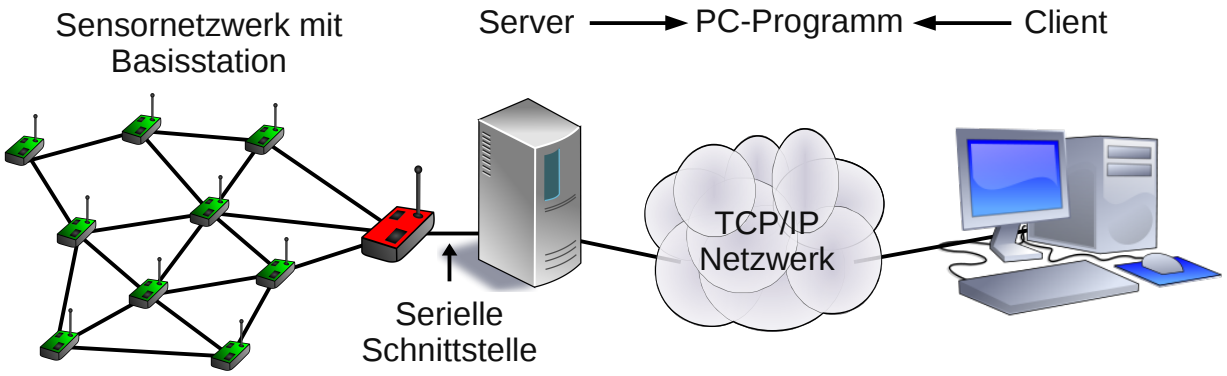


Abbildung 5.1: Eine Übersicht über das System zur Erfassung der Temperaturverteilung innerhalb von Wohn- und Büroräumen

Die Anforderung W2 wünscht eine Trennung des PC-Programms in einen Server und in ein Client-Programm. Zusammen mit dem in Kapitel 4 vorgestellten Sensornetzwerks ergibt sich somit das in Abbildung 5.1 dargestellte System zur Erfassung der Temperaturverteilung innerhalb von Wohn- und Büroräumen. Der Server steuert das Sensornetzwerk mithilfe der Interessenachrichten über die mittels serieller Schnittstelle angeschlossene Basisstation und speichert alle eintreffenden Messdaten. Via TCP/IP-Protokollstapel kommuniziert er über ein Netzwerk mit dem Client. Dieser dient dem Benuter zum Setzen von Konfigurationsparametern für das Sensornetzwerk und ermöglicht die Datenbetrachtung.

Im Folgenden wird in Abschnitt 5.1 die Auswahl der Technologien, mit denen das PC-Programm umgesetzt wurde, vorgestellt. Anschließend werden in den Abschnitten 5.2 und 5.3 der Entwurf und die Implementierung des Servers und des Clients beschrieben.

Ohne das PC-Programm könnte man das Sensornetzwerk zwar nicht betreiben, aber im Mittelpunkt dieser Arbeit steht das Sensornetzwerk. Um rasch voranzuschreiten, wurde hier entgegen der Entwicklung des Sensornetzwerks nicht das Wasserfallmodell eingesetzt. Der Entwurf und die Implementierung des PC-Programms fand ein einem Schritt statt und wurde mittels Rapid Prototyping durchgeführt.

5.1 Technologieauswahl

Die Entwicklung des PC-Programms wird unter GNU/Linux durchgeführt. Daher ist dieses System auch die Zielplattform, es soll aber dennoch bei der Entwicklung auf Plattformunabhängigkeit geachtet werden. Es finden aber keine Tests für andere Plattformen statt, weshalb diese nicht offiziell unterstützt werden.

Das PC-Programm wurde in der Programmiersprache C++ implementiert. GUI-Programme implementiert man heute typischerweise mit einer objektorientierten Sprache. Die Wahl fiel auf C++, weil das Sensornetzwerk mittels nesC implementiert wurde und so mögliche künftige Entwickler des in dieser Arbeit vorgestellten Systems typischerweise C und C++ beherrschen. Zudem sind diese Sprachen bereits weitver-

breitet, und es existieren Compiler für diverse Betriebssysteme und Plattformen. Hier wird gcc in Version 4.4.1 eingesetzt, der über die Internetadresse http://gcc.gnu.org bezogen werden kann.
Der Server kommuniziert mit der Basisstation mittels der TinyOS beiliegenden Implementierung des seriellen Protokollstapels. Dieser greift unter Linux mit termios, welches ein Bestandteil der GNU-C-Bibliothek ist, auf die serielle Schnittstelle zu. Unter Windows wird die Windows-System-API genutzt.
Zur Umsetzung der graphischen Benutzerschnittstelle kommt die C++-Klassenbibliothek Qt zum Einsatz. Diese wurde von der Firma Trolltech entwickelt und steht seit der Übernahme von Trolltech durch Nokia unter der LGPL. Für Qt spricht, dass es plattformunabhängig ist und bereits vom Autor dieser Arbeit eingesetzt wurde. Darüber hinaus bietet Qt Module für den Zugriff auf das Netzwerk und auf Datenbanken an, die auch für diese Zwecke vom PC-Programm genutzt werden. Qt wird in Version 4.5 eingesetzt und kann im Internet über http://qt.nokia.org bezogen werden.
Qt verwendet den Meta-Object-Compiler (moc) genannten Präprozessor, um C++ um das Signal-Slot-Konzept zu erweitern. Dieses ermöglicht eine einfache Kommunikation zwischen verschiedenen Objekten. Hierzu können diese Signale definieren, die sie erzeugen, sobald ein bestimmtes Ereignis in einem Objekt eintritt. Des Weiteren definieren Objekte Slots, die von anderen Objekten über Ereignisse in Kenntnis gesetzt werden wollen, um daraufhin Operationen durchzuführen. Mittels *connect()* können Signale mit mehreren Slots mehrerer Objekte verbunden werden. Einzige Bedingung hierfür ist eine identische Funktionssignatur, das Signal-Slot-Konzept ist damit datentypsicher. Erzeugt jetzt ein Objekt ein Signal, so werden alle verbundenen Slots benachrichtigt [Fire].
Die Anzeige der Messdaten in einem Zeit-Werte-Diagramm wird mit Hilfe von Qt Widgets for Technical Applications (Qwt) realisiert. Dies ist eine Bibliothek für Qt, welche Widgets für technische Anwendungen bereitstellt. Die Bibliothek ist zu beziehen über http://qwt.sourceforge.net.
Als Entwicklungsumgebung wird der Qt beigelegte Qt-Creator verwendet.

5.2 Entwurf und Implementierung des Servers

In diesem Abschnitt werden der Entwurf und die Implementierung des Servers vorgestellt. Wie bereits in der Einleitung dieses Kapitels beschrieben, hat der Server die Aufgabe, das Sensornetzwerk zu steuern und dem Benutzer so die Interaktion mit diesem zu ermöglichen.
In Abbildung 5.2 ist das Aktivitätsdiagramm zur Steuerung des Sensornetzwerks durch das Versenden von Interessen dargestellt. Dieses Aktivitätsdiagramm entspricht der Realisierung des Anwendungsfalldiagramms für das Sensornetzwerk – siehe Anforderungsdefinition in Abschnitt 2.1 – und wurde um die Möglichkeiten des in Kapitel 4 vorgestellten Sensornetzwerks erweitert.
Zentrales Element des Servers ist eine Liste von Messungen, die durchgeführt werden sollen. Sie entspicht der Menge aller im System vorhandenen periodischen Interessen, also der Menge aller Interessentabelleneinträge aller Sensorknoten. In periodischen Abständen von fünf Minuten feuert ein Timer, der die Liste durchgeht und für jeden Eintrag ein Paket für perodische Interessen packt. Dieses wird im Anschluss an die Basisstation gesendet und damit im Sensornetzwerk verbreitet. Ein Messlisteneintrag besteht aus der Art des Interesses, der Messauflösung, dem Zeitraum, in dem die Messung durchgeführt werden soll, und dem Adressbereich der Sensorknoten, welche die Messung durchführen sollen. Des Weiteren wird beim Packen des Interesse-Pakets eine fortlaufende Nummer als Interessen-ID und 0 als Gradient in den periodischen Interessen gesetzt.
Die Liste der Messungen wird regelmäßig an das Sensornetzwerk gesendet, damit auch später hinzukommende Sensorknoten die Interessen erhalten. Darüber hinaus werden so die Gradienten der Sensorknoten aktualisiert.
Das Client-Programm steuert das Sensornetzwerk über die auf der linken Seite des Aktivitätsdiagramms eintreffenden Objekte, die den Server jeweils zu einer Aktion veranlassen, welche im Folgenden beschrieben werden.
Das Client-Programm kann eine Messung starten, indem es ein eine Messung beschreibendes Objekt sen-

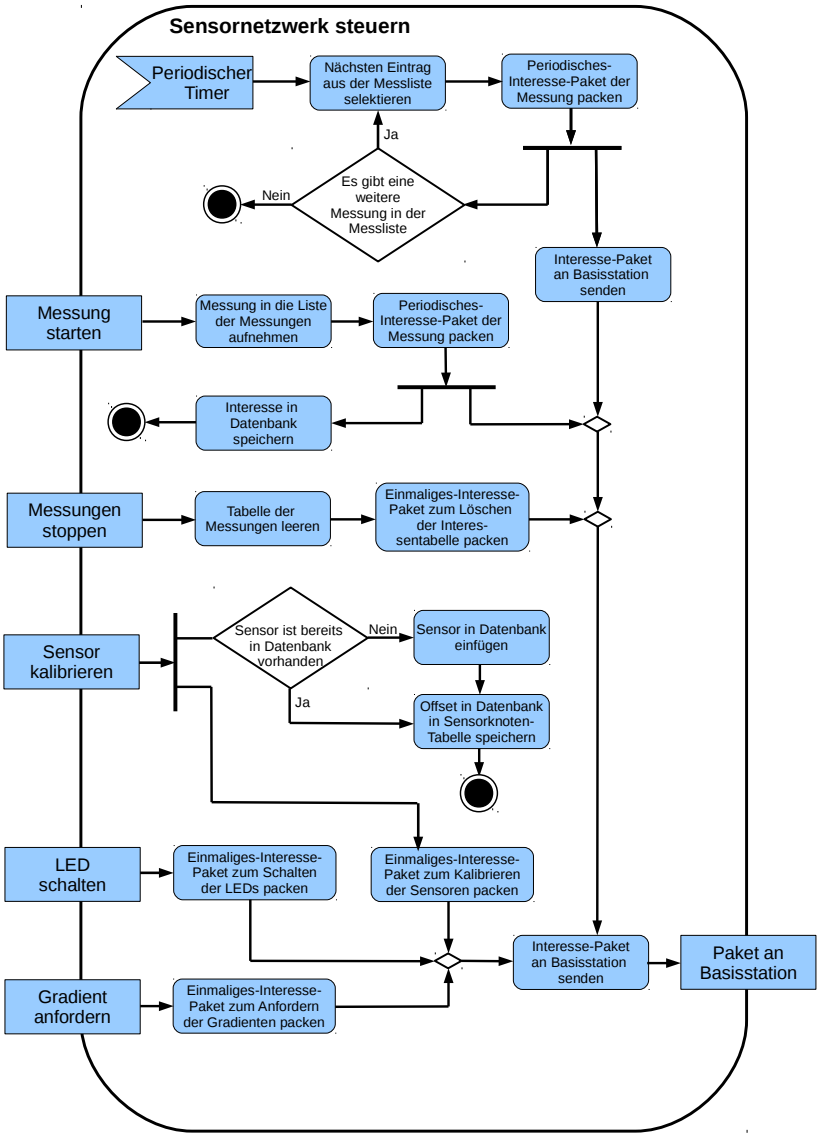


Abbildung 5.2: Das Aktivitätsdiagramm zum Steuern des Sensornetzwerks

det. Die Messung wird zunächst in die Liste der Messungen aufgenommen. Hierzu wird geschaut, ob bereits eine identische Messung existiert. Eine Messung ist analog zur Definition bei den Sensorknoten identisch, wenn Art des Interesses, Messauflösung und Adressbereich übereinstimmen. Ist dies der Fall, so wird lediglich der Messzeitraum neu gesetzt. Des Weiteren muss das Client-Programm nicht zwangsläufig einen Messzeitraum mit angeben. Ist kein Zeitraum angegeben, so wird der Messstart auf 0 und das Messende auf die Konstante MAXINT gesetzt, damit die Messung permanent stattfindet. Anschließend wird, wie bereits beim Timer beschrieben, ein Paket für periodische Interessen gepackt und an die Basisstation gesendet. Parallel hierzu wird das Interessen-Paket an die Datenbank gegeben und dort gespeichert.
Derzeit unterstützt das System lediglich das Stoppen aller Messungen. Wird dieses von dem Client-Programm in Auftrag gegeben, so wird die Liste mit den Messungen geleert und ein einmaliges Interessenpaket zum Löschen der Interessentabelle der Sensorknoten sowohl gepackt als auch an die Basisstation gesendet.
Möchte das Client-Programm einen Sensorknoten kalibrieren, so wird ein Paket für einmalige Interessen zur Kalibrierung eines Sensors gepackt und an die Basisstation gesendet. Parallel hierzu wird nachgeschaut, ob der Sensor bereits in der Datenbank vorhanden ist. Ist dies nicht der Fall, so wird der Sensor der Datenbank hinzugefügt. In beiden Fällen wird anschließend der Offset des Sensors bei dem Sensorknoten in der Datenbank gespeichert.
Um das Ansteuern von Aktoren zu zeigen, kann das PC-Programm die LEDs schalten. Hierzu wird lediglich ein Paket mit einem einmaligen Interesse zum Schalten der LEDs gepackt und an die Basisstation gesendet.
Analog erfolgt die Anforderung der Sensorknotenadresse des besten Gradienten der Sensorknoten.
Bei allen hier vorgestellten Abläufen wird beim Packen der Interessen-Pakete im Header der Pakete die Quelladresse 0 gesetzt, weil diese die Adresse der Basisstation ist, und als AM-Typ der Interessenkanal angegeben.
Nachdem das Sensornetzwerk jetzt gesteuert werden kann, bleibt zu klären, wie die Daten aus dem Sensornetzwerk entgegengenommen und dem Benutzer zugänglich gemacht werden. Hierbei verlangt Anforderung E5 auch die persistente Speicherung der Messdaten, vergleiche Abschnitt 2.2.
In Abbildung 5.3 wird das Aktivitätsdiagramm zum Entgegennehmen, Bereitstellen und Speichern der Daten aus dem Sensornetzwerk vorgestellt. Dieses entspricht den serverseitigen Anwendungsfällen des Anwendungsfalldiagramms für das PC-Programm, siehe Anforderungsdefinition in Abschnitt 2.1.
Trifft ein neues Paket von der Basisstation ein, so wird anhand der Länge der Nutzdaten des Paketes der Pakettyp bestimmt.
Handelt es sich um ein Synchronisationspaket der Basisstation, so wird die Zeit des Servers mit der Zeit des Sensornetzwerks synchronisiert. Hierzu wird ein an RBS (siehe Unterabschnitt 3.7.2) angelehntes Verfahren benutzt. Analog zu RBS wird der Offset zwischen der mitteleuropäischen Zeit des PCs mit der Zeit des Sensornetzes gebildet. Der Offset zwischen den beiden wird dann als Mittelwert aller Offsets berechnet.
Trifft nun ein Datenpaket ein, so wird der Zeitstempel aus der Zeit des Sensornetzwerks durch Addition des berechneten Offsets in die mitteleuropäische Zeit umgerechnet. Anschließend wird das Datenpaket zum einen an die Datenbank zur Speicherung weitergereicht und zum anderen an das Client-Programm gesendet. Dies kann die Messdaten als die aktuellsten verfügbaren Daten dem Benutzer anzeigen.
Ist das neu eingetroffene Paket ein Informationspaket, so wird nachgeschaut, ob die beinhaltete Zahl einer definierten Fehlernummer entspicht – zu den Fehlernummern siehe Abschnitt 4.4. Ist dies der Fall, wird eine ausformulierte Fehlermeldung zunächst in eine Logdatei geschrieben und anschließend an das Client-Programm gesendet. Handelt es sich nicht um einen Fehler des Sensornetzwerks, so wird geprüft, ob es sich um eine Gradientenadresse handelt. Dies ist derzeit der Fall, wenn die beinhaltete Zahl größer oder gleich 20 ist, da der Bereich unter 20 für Fehler reserviert bleiben soll. Dann wird 20 subtrahiert, da bei der Speicherung der Gradientenadresse im Sensornetzwerk zur Differenzierung zwischen Fehlern und Gradienten 20 addiert wurde, und der Gradient an den Server gesendet.
Möchte der Benutzer am Client-Programm Messdaten einer in der Vergangenheit liegenden Messung betrachten, so kann das Client-Programm diese Daten beim Server anfragen. Hierzu gibt der Benutzer über

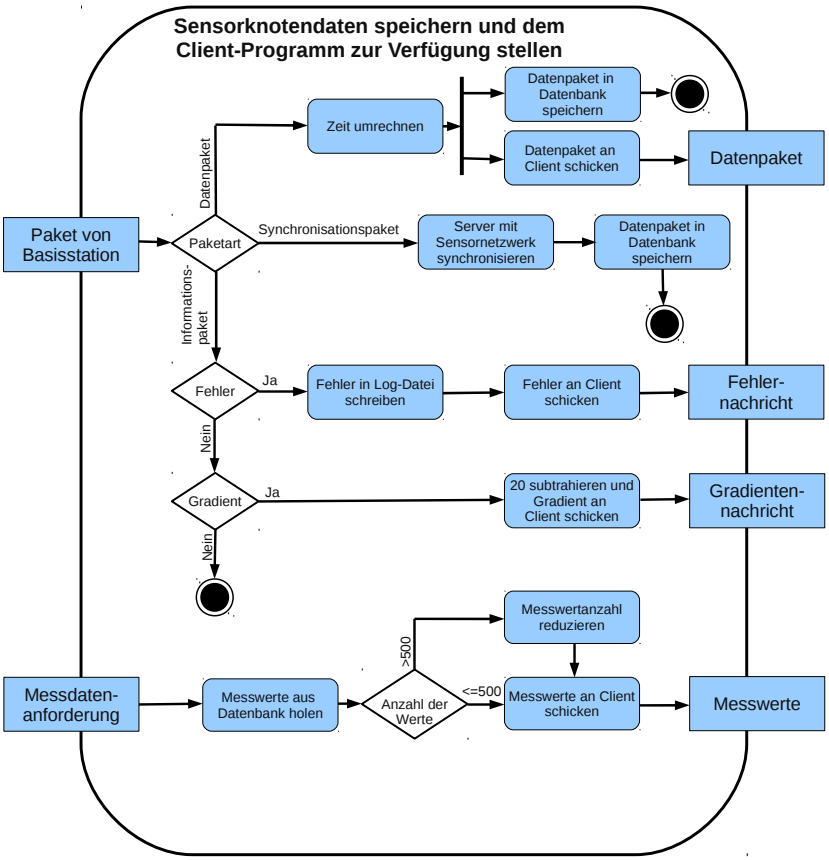


Abbildung 5.3: Das Aktivitätsdiagramm zum Entgegennehmen der Daten aus dem Sensornetzwerk und zum Senden von Messdaten an das Client-Programm

den Client einen Zeitraum, den Sensorknoten und die Messgröße an. Der Server holt daraufhin die Daten aus der Datenbank. Wenn die Anzahl der Werte größer als 500 ist, so wird die Datenmenge reduziert, um Bandbreite bei der Kommunikation zu sparen. Die Datenreduktion geschieht per gleitendem Mittelwertfilter. Ist $g(x)$ die Folge der aus der Datenbank geholten Messwerte, c deren Anzahl und $f(x)$ die zu berechnende Folge der 500 Werte, so ergibt sich mit ganzzahlig aufgerundetem n:

$$\forall x \in [0, ..., 499] : f(x) = \frac{1}{n} \sum_{k=n \cdot x}^{n \cdot x + n - 1} g(k), n = \lceil \frac{c}{500} \rceil$$

Der Wert 500 ist gewählt worden, da dies die maximale Anzahl der gleichzeitig darstellbaren Werte des Client-Programms ist. Möchte der Benutzer die Daten genauer betrachten, so kann er den Zeitraum der Messung reduzieren. Abschießend werden die Messwerte an das Client-Programm geschickt.

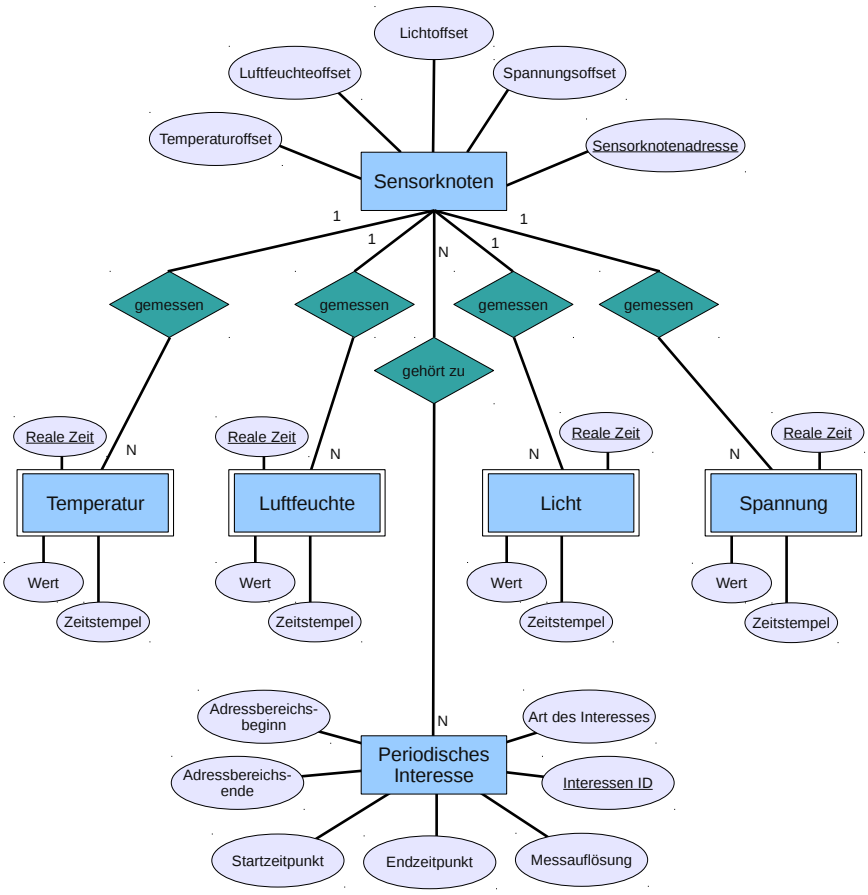


Abbildung 5.4: Das Entity-Relationship-Modell der Datenbank zur Speicherung der Messdaten

Nachdem die Funktionsweise des Servers geklärt wurde, fehlt noch die Definition des Datenbankschemas. Dieses wird mit dem Entity-Relationship-Modell in Abbildung 5.4 illustriert. Zunächst gibt es eine Tabelle für die Sensorknoten, in der die Sensorknotenadresse und Offsets für die Sensoren gespeichert werden. Der Primärschlüssel ist die Sensorknotenadresse. Darüber hinaus gibt es je eine Tabelle für die Messgrößen, welche die reale Zeit, also hierzulande die mitteleuropäische Zeit, den Zeitstempel der Messung in der Zeit des Sensornetzwerks und den Messwert als Attribute besitzen. Hinzu kommt noch die Adresse des messenden Sensorknotens, welche zusammen mit der realen Zeit den Primärschlüssel ergibt. Jeder Sensorknoten besitzt beliebig viele Messwerte der Messgrößen, jeder Messwert gehört aber eindeutig zu einem

Sensorknoten. Außerdem werden in der Datenbank die periodischen Interessen mit ihren Werteausprägungen gespeichert. Der Primärschlüssel ist hierbei die eindeutige Interessen-ID, wobei zu beachten ist, dass lediglich die erste genutzte ID gespeichert wird. Wenn der Timer im Server das Interesse erneut versendet, wird jeweils eine neue Interessen-ID generiert, welche aber nicht in der Datenbank gespeichert wird. Die Tabelle mit den Interessen findet in dem hier vorgestellten System noch keine Verwendung. Sie wurde lediglich der Vollständigkeit halber mit aufgenommen.

In Abbildung 5.5 ist das Klassendiagramm mit den wichtigen Klassen des Servers zu sehen. Funktionen und Attribute, die für das Verständnis unwesentlich sind, wurden weggelassen.

Es wird zunächst die Schnittstelle zum Sensornetzwerk betrachtet. Ein Interesse ist entweder ein Objekt der Klasse *periodicInterest* oder *oneShotInterest*. Beide wurden von der Klasse *tnterest* abgeleitet, um per Polymorphie an Funktionen derselben Signatur übergeben werden zu können, und besitzen als Attribute die interessenspezifischen Größen, auf welche per Setter und Getter zugegriffen wird. Hinzu kommen zwei Funktionen, um aus den Werten der Attribute das Paket inklusive Header zu packen und umgekehrt.

Die Klasse *serialInterface* bietet einen Slot zum Senden der Interessenachrichten an. Hierzu reiht sie das übergebene Interesse in die Sendeliste des seriellen Threads ein. Der eigentliche Sendevorgang wird von der Klasse *serialThread* ausgeführt, welche den seriellen Thread implementiert. Dieser schaut immer wieder nach, ob neue Datenpakete bereitstehen oder ob Interessen gesendet werden sollen. In diesem Fall findet er das neue Interesse vor und sendet es an die Basisstation. Der eigentliche Zugriff auf die serielle Schnittstelle wird von der Klasse *serialSource* realisiert, der den seriellen Netzwerkstapel von TinyOS implementiert.

Wenn ein neues Paket vom Sensornetzwerk eintrifft, so reicht *serialThread* es an die Klasse *serialInterface* weiter, wo die Art des Datenpakets bestimmt wird. Je nach Art wird dann analog zu den Interessenobjekten ein Datenobjekt erstellt und per Signal an die anderen Komponenten des Servers weitergeleitet.

Das Senden und Empfangen wird mit einem Thread implementiert, damit langandauernde Operationen im Server nicht das Empfangen von Paketen behindern können. So ist sichergestellt, dass der Nachrichtenpuffer des Betriebssystems nicht überlaufen kann und somit keine Pakete verloren gehen.

Den Zugriff auf die Datenbank realisiert die Klasse *databaseInterface*, welche Slots zum Speichern der Daten-, Interessenpakete und Gradienten bereitstellt. Hinzu kommt ein Slot und ein Signal zum Auslesen der gespeicherten Sensorwerte. Der Zugriff auf die Datenbank ist ebenfalls in einem Thread realisiert. Insbesondere Anfragen aus dem Client-Programm nach aufgezeichneten Daten können lange dauern. In dieser Zeit werden andere auflaufende Anfragen in zwei Listen vorgehalten.

Die Schnittstelle zum Netzwerk und damit zum Client-Programm realisiert die Klasse *networkInterface*. Diese instanziiert den von Qt mitgelieferten TCP-Server. Die Netzwerkschnittstelle bietet Signale und Slots für die bereits beschrieben Anfragen des Client-Programms an. Hierbei werden zwischen Server und Client-Programm Daten mithilfe von Objekten ausgetauscht, z. B. Interessenobjekte oder Ergebnisobjekte einer Datenanfrage. Um die Objekte über die serielle Netzwerkverbindung zu übertragen, werden sie serialisiert. Zur Serialisierung wurden die Stream-Operatoren der einzelnen Objekte überschrieben. Damit können Objekte in einen Datenstrom geschrieben werden und umgekehrt.

Zentrales Element des Servers ist die Klasse *control*, welche je eine Instanz der beschriebenen Klassen besitzt. Zu dem beherbergt sie die Liste aller Messungen und den Timer zum periodischen Senden der periodischen Interessen. Bootet der Server, so startet *control* die beiden Threads, setzt die serielle Schnittstelle zum Sensornetzwerk und veranlasst die Datenbankschnittstelle dazu, sich mit der Datenbank zu verbinden. Darüber hinaus verbindet *control* die Signale mit den Slots. So wird z. B. das Signal *newSensorReading* mit dem Slot *saveSensorReadings* der Datenbankschnittstelle und dem Slot *sendSensorInfo* der Netzwerkschnittstelle verbunden.

Jedes Qt-Programm muss die QCoreApplication einbinden, welche die Eventschleife von Qt beinhaltet. Diese realisiert das Signal-Slot-Konzept, indem es bei neuen Signalen die entsprechenden Slots aufruft. Die Klasse *main*, welche nach dem Start des Servers der Einstiegspunkt ist, besitzt ein Objekt dieser Klasse und die *control*-Instanz.

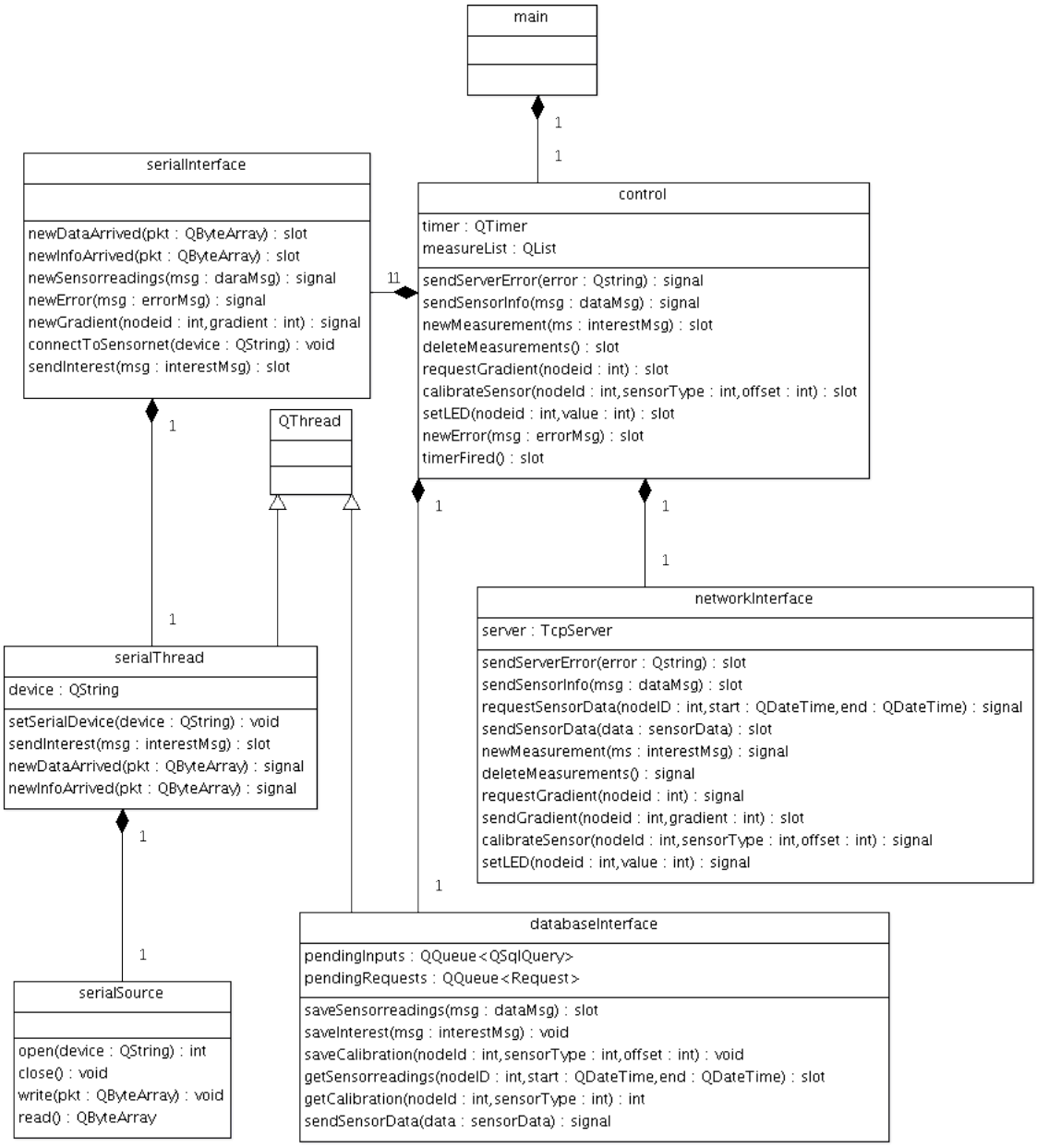


Abbildung 5.5: Das Klassendiagramm des Servers

Abschließend bleibt noch zu erwähnen, dass lediglich die QtCore-Bibliothek für den Server verwendet wird. Dadurch benötigt er keine graphische Benutzeroberfläche.

5.3 Entwurf und Implementierung des Clients

In diesem Abschnitt werden der Entwurf und die Implementierung des Client-Programms beschrieben. Dieses ermöglicht dem Benutzer die Messdatenbertachtung und Steuerung des Sensornetzwerks. Um die folgenden Ausführungen besser verständlich zu machen, werden zunächst die Benutzeroberfläche und die dahintersteckenden Funktionen grob beschrieben. Eine ausführliche Anleitung zur Benutzung des Systems wird mit dem auf der beiliegenden CD zu findenden Benutzerhandbuch geboten.

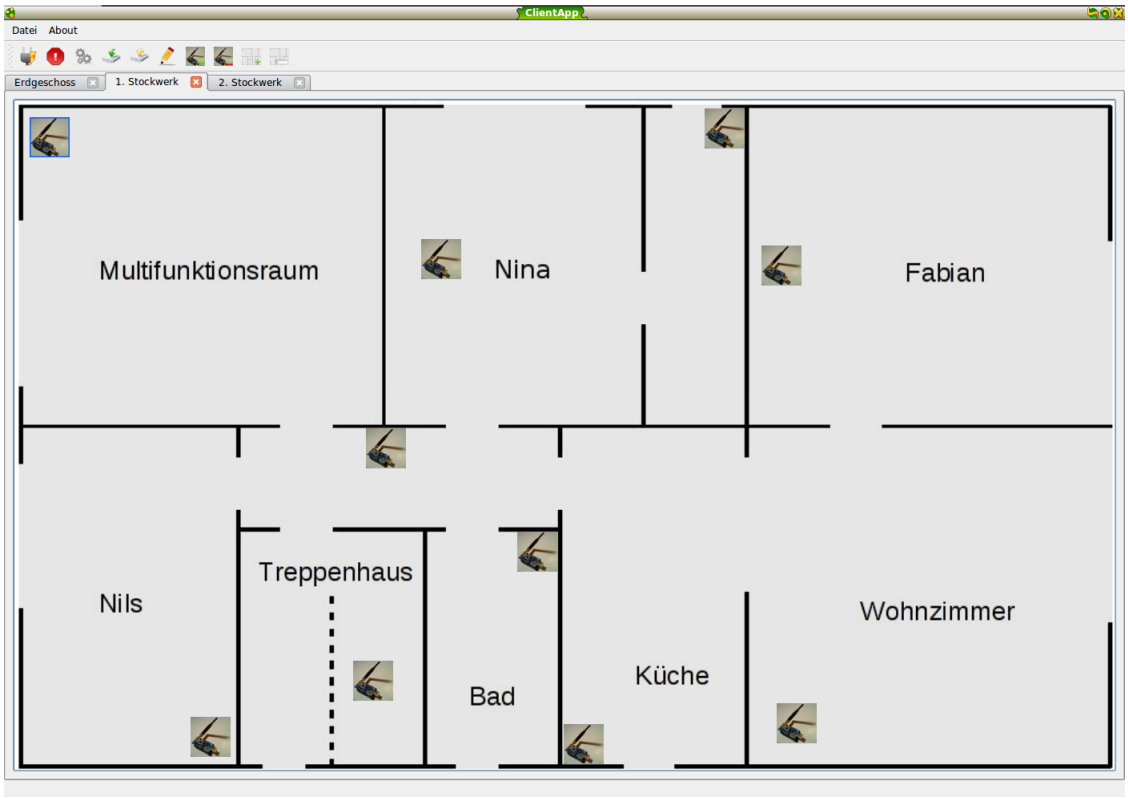


Abbildung 5.6: Ein Bildschirmfoto des Client-Programms

In Abbildung 5.6 ist ein Bildschirmfoto des Client-Programms zu sehen. Anforderung W1 wünscht eine komplexe Visualisierung, vergleiche Abschnitt 2.2. Diese wurde umgesetzt, indem jede Ebene des Gebäudes eine Bilddatei als Lageplan nutzt, in der die Sensoren per Drag-and-Drop vom Benutzer platziert werden können. Jedes Stockwerk eines Gebäudes wird in einem Tab dargestellt.
In der Drucktastenleiste befinden sich folgende Knöpfe, die von links nach rechts beschrieben werden: Das erste Icon öffnet ein Fenster, in dem die Adresse des Servers angegeben wird und die Verbindung zum Server hergestellt werden kann. Das Icon daneben öffnet ein Fenster, in dem alle aufgetretenen Fehler sowohl des Sensornetzwerks als auch des Servers aufgelistet werden. Das dritte Icon öffnet das Fenster für die globalen Einstellungen. Hier werden die Datenbankadresse, die serielle Schnittstelle zum Sensornetzwerk und die Messeinstellungen angegeben. Die folgenden zwei Icons speichern und laden den aktuellen Gebäudeplan. Das sechste Icon wechselt zwischen dem Bearbeitungs- und dem Ansichtsmodus. Im Bearbeitungsmodus werden mit den letzten vier Knöpfen Sensoren hinzugefügt und entfernt und Ebenen hinzugefügt und umbenannt. Zudem sind die Sensorknotenicons verschiebbar. Im Ansichtsmodus kön-

nen die Sensorknotenicons angeklickt werden, woraufhin sich ein Fenster zur Betrachtung der Daten des Sensorknotens und zur Ansteuerung angeschlossener Aktoren öffnet.
Im Folgenden wird jetzt die Funktionsweise des Client-Programms erklärt. In Abbildung 5.7 ist das Aktivitätsdiagramm zur Verwaltung des Lageplans des Gebäudes dargestellt. Alle Abläufe werden hierbei durch die Aktion des Benutzers gestartet und entspringen daher einem Signal. Zudem muss für die ersten vier Abläufe der Bearbeitungsmodus aktiviert sein. In der Implementierung sind die jeweiligen Knöpfe im Ansichtsmodus deaktiviert, so dass die hier zu sehende Überprüfung nie stattfinden muss und der Benutzer bei einem Klick auf einen aktivierten Knopf in jedem Fall die Aktion durchführen kann.
Möchte der Benutzer ein neues Stockwerk hinzufügen, so öffnet sich ein Fenster zur Auswahl der Bilddatei des Lageplans. Nach erfolgter Auswahl wird ein Image-Label-Objekt mit dem Bild erzeugt und einem neu instanziierten Floor-Objekt hinzugefügt, welches in einem neuen Tab-Objekt als Widget gesetzt wird. Besitzt das Bild eine Auflösung kleiner oder gleich 1024 x 768, so wird es immer auf Fenstergröße skaliert. Ist es größer, so befindet sich das Floor-Objekt innerhalb einer Scrollbox in dem Tab. Abschließend wird das neue Tab-Objekt in die Liste der Gebäudeebenen aufgenommen und angezeigt.
Möchte der Benutzer ein Stockwerk entfernen, so werden alle Sensorknotenobjekte, die in diesem Stockwerk positioniert wurden, gelöscht. Anschließend wird der Tab aus der Liste der Gebäudeebenen entfernt und gelöscht.
Möchte der Benutzer einen neuen Sensorknoten hinzufügen, so öffnet sich ein Fenster mit einem Sensorknotenicon und der Aufforderung zur Eingabe der Sensorknotenadresse. Hat der Benutzer die Sensorknotenadresse angegeben, so kann er das Icon per Drag-and-Drop auf den Lageplan ziehen. Jedes Floor-Objekt eines Stockwerkes besitzt eine Liste mit Sensorknotenobjekten. Es wird nun ein neues Sensorknotenobjekt erzeugt, die Position auf dem Lageplan darin gesetzt und das Objekt in die Liste der Sensorknoten des Floor-Objektes aufgenommen.
Soll ein Sensorknoten aus dem Gebäudeplan entfernt werden, so wird das Sensorknotenobjekt aus der Liste der Sensorknoten des Floor-Objektes gelöscht.
Soll der Plan des Gebäudes gespeichert werden, so wählt der Benutzer zunächst einen Dateinamen. Danach wird die Liste aller Ebenen durchgegangen. Für jede Ebene wird der Name der Ebene und der Pfad zur Bilddatei in einer Binärdatei gespeichert. Darauffolgend werden die Anzahl der Sensorknoten der Ebene und die Sensorknoten selbst mit den Positionsangaben gespeichert.
Zum Laden eines Gebäudeplans gibt der Benutzer wiederum zunächst die Datei an, in der die zu ladende Gebäudeansicht gespeichert ist. Danach wird für jede in der Datei vorhandene Ebene, analog zum Hinzufügen einer neuen Ebene, wieder ein Tab erzeugt. Diesem werden dann die neu instanziierten Sensorknotenobjekte hinzugefügt, die mit den in der Datei gespeicherten Informationen erstellt worden sind. Abschließend wird der Tab in die Liste der Ebenen aufgenommen und angezeigt.
Um die Ebenen und Sensorknoten auf einfache Art und Weise laden und speichern zu können, wurden die Stream-Operatoren überschrieben. So kann ein Floor- oder Sensorknotenobjekt mit dem Stream-Operator in einen Dateistrom geschrieben oder aus ihm gelesen werden.
In Abbildung 5.8 ist das Aktivitätsdiagramm zur Datenbetrachtung und Interaktion mit dem Sensornetzwerk abgebildet. Um sensorknotenspezifische Funktionen durchzuführen, klickt der Benutzer auf das Icon eines Sensorknotens, woraufhin sich ein Fenster öffnet.
Möchte der Benutzer die Messdaten betrachten, so kann er hier ein Zeitfenster angeben, woraufhin das Client-Programm die Daten beim Server anfragt. Treffen angeforderte Daten ein, so wird der Sensorknoten anhand der Sensorknotenadresse in allen Listen der Sensorknoten in allen Tabs gesucht. Danach werden die Messdaten in den Datenpuffer des QWT-Anzeigewidgets zum Erzeugen von Datenkurven geschrieben. Des Weiteren kann der Benutzer in dem Fenster des Sensorknotens dessen Sensoren kalibrieren. Hierzu gibt er einen Offset an, welcher anschließend von dem Client-Programm an den Server gesendet wird. Analog kann der Benutzer hier auch den Zustand der LEDs setzen oder die Sensorknotenadresse des besten Gradienten anfordern. Auch hierzu erzeugt das Client-Programm eine entsprechende Anfrage beim Server.
Im Einstellungsfenster des Client-Programms kann der Benutzer die Messparameter setzen oder die Messung stoppen. Der Client unterstützt derzeit lediglich das Starten globaler Messungen für alle Sensorknoten

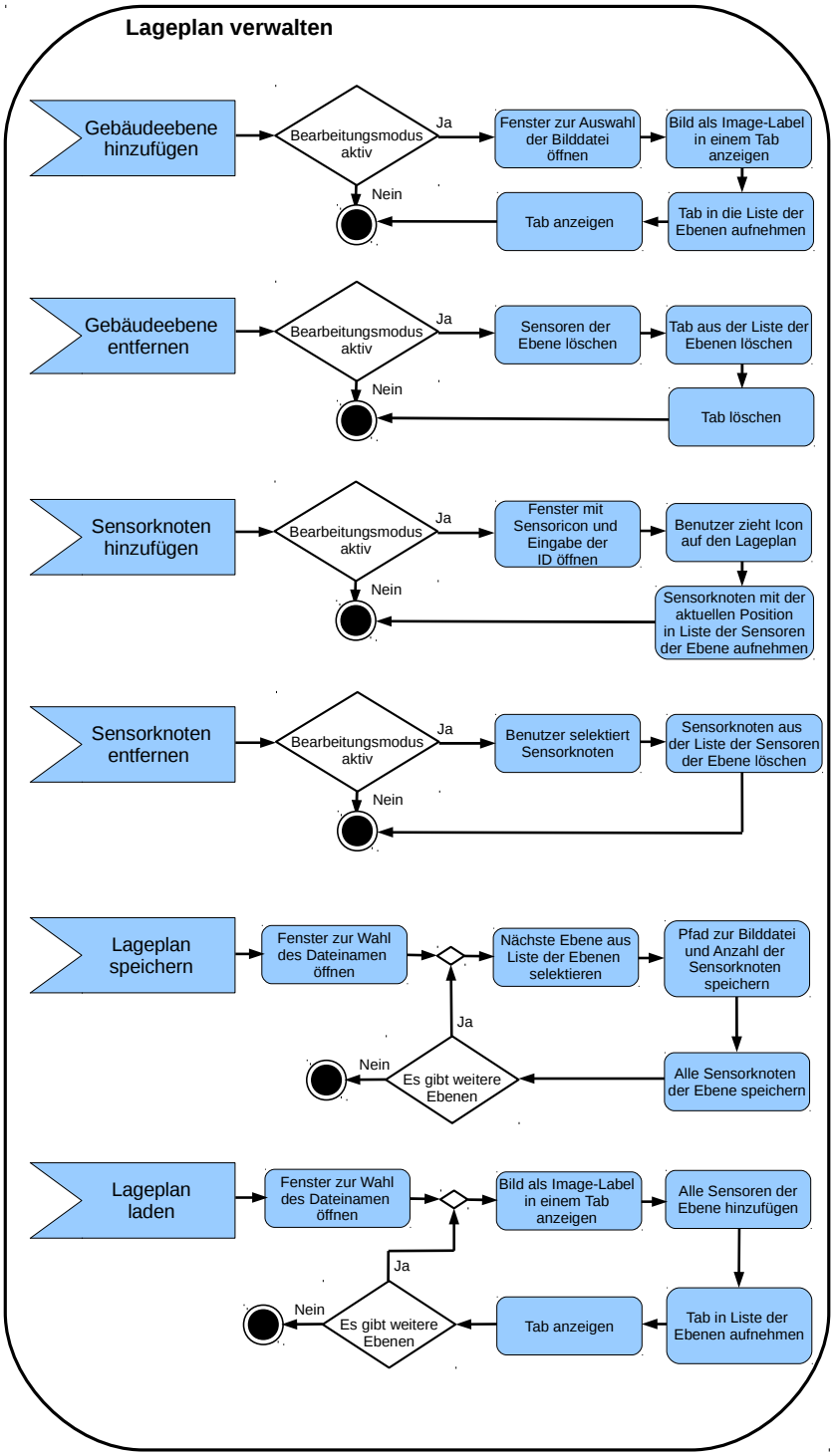


Abbildung 5.7: Das Aktivitätsdiagramm zum Verwalten des Lageplans im Client-Programm

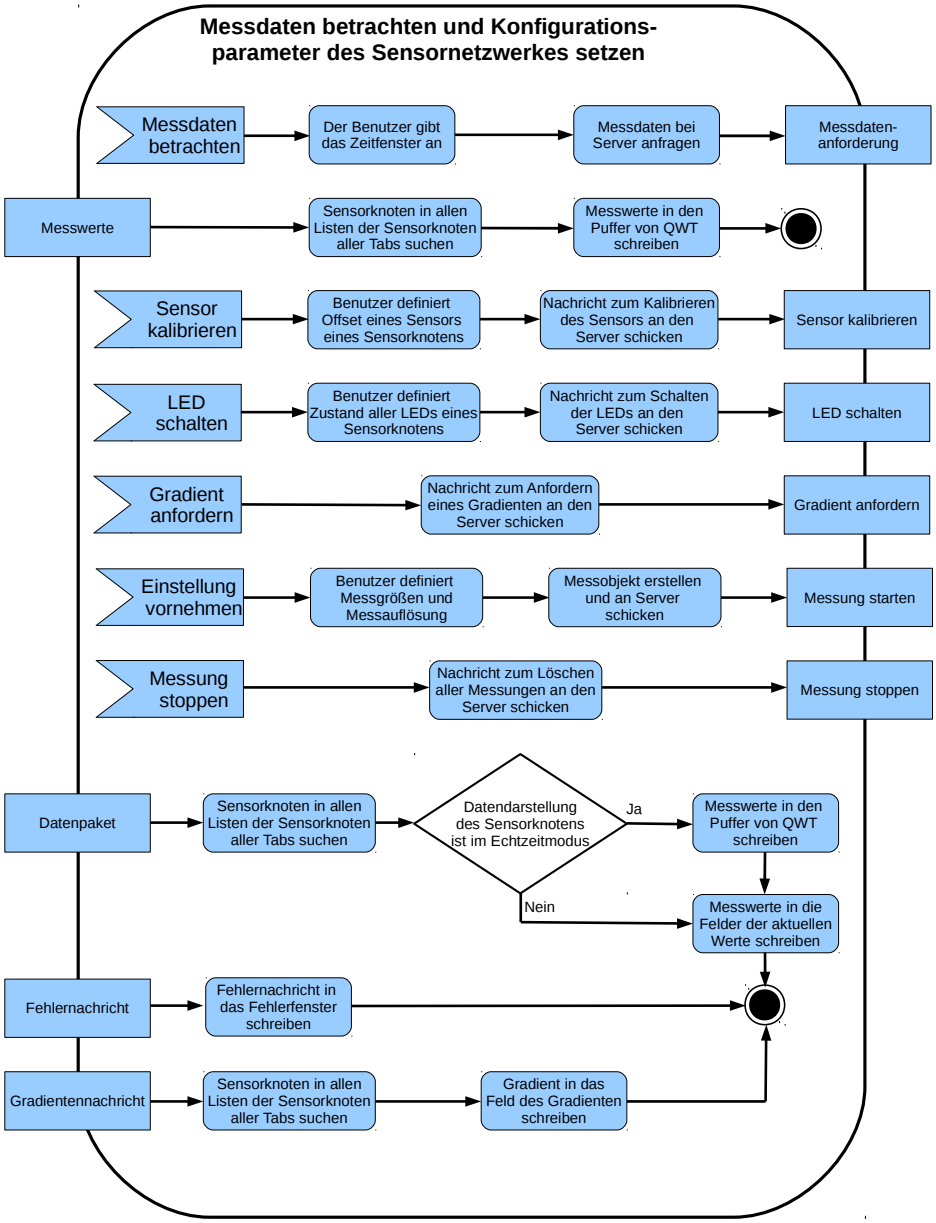


Abbildung 5.8: Das Aktivitätsdiagramm zur Datenbetrachtung und Interaktion mit dem Sensornetzwerk

des Netzwerkes. Hierzu gibt der Benutzer Messauflösung und Messgrößen an, und das Client-Programm schickt eine Messnachricht an den Server. Analog wird eine Nachricht zum Löschen der Interessentabelle versendet, wenn der Benutzer die Messungen stoppen möchte.
Erhält der Server ein neues Datenpaket aus dem Sensornetzwerk, so sendet er es an das Client-Programm weiter. Das Clientprogramm sucht wieder den passenden Sensor und setzt die Messwerte in einer Übersicht, in der die letzten gemessenen Werte jeder Größe angezeigt werden. Wünscht der Benutzer bei der Datenbetrachtung Werte aus dem Sensornetzwerk statt der Datenbank (Echtzeitmodus), so werden die eintreffenden Daten auch in den Puffer des Anzeigewidgets gesetzt. Analog werden neu eintreffende Gradienten ebenfalls in die Übersicht des Sensorknotens aufgenommen.
Trifft eine neue Fehlernachricht ein, so wird der zugehörige String in das Fenster mit den Fehlern geschrieben.

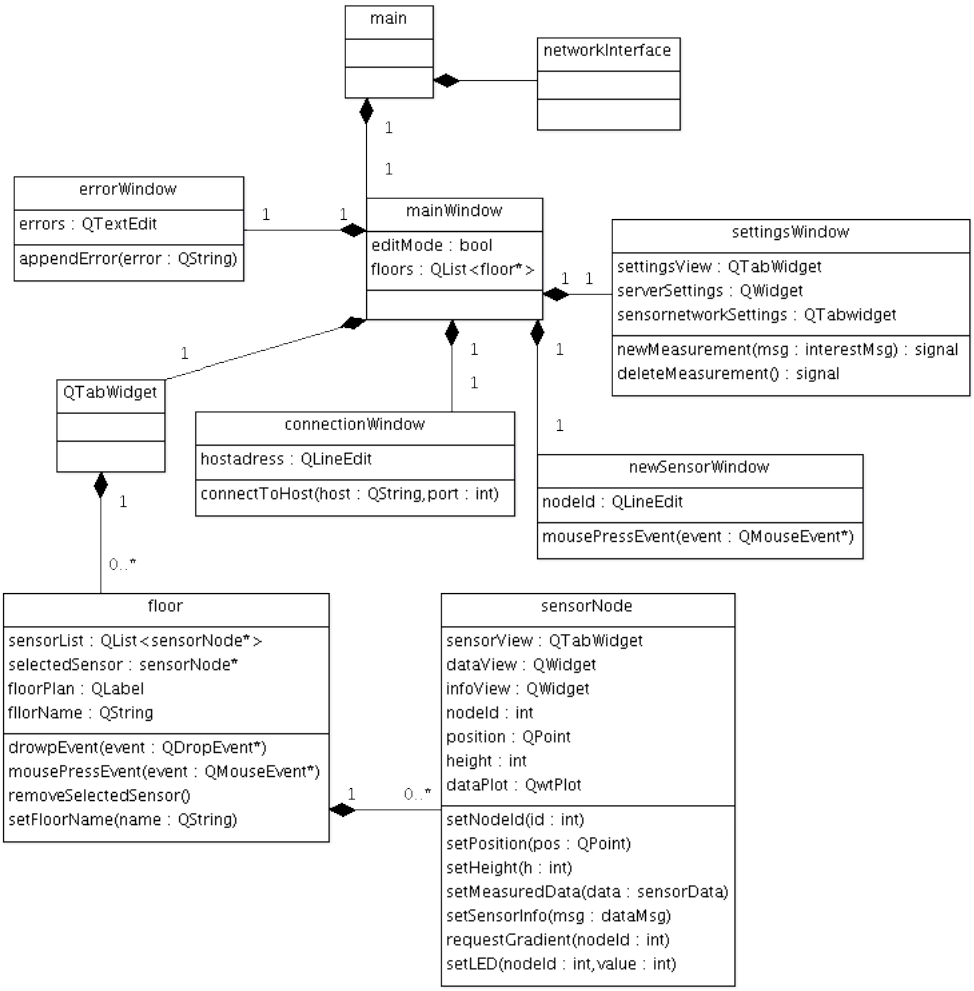


Abbildung 5.9: Das Klassendiagramm des Client-Programms

In Abbildung 5.9 ist das Klassendiagramm des Client-Programms zu sehen. Die Klasse *main* besitzt je eine Instanz der Netzwerkschnittstelle und des Hauptfensters *mainWindow*. Die Netzwerkschnittstelle ist das exakte Gegenstück zu der Netzwerkschnittstelle des Servers. Die Klasse *mainWindow* implementiert die

Bedienelemente und besitzt für jeden Knopf einen Eventhandler, in dem die entsprechenden Operationen, etwa das Öffnen der Nebenfenster, durchgeführt werden. Dafür besitzt sie je eine Instanz für das Fehlerfenster, das Einstellungsfenster, das Serververbindungsfenster und das Fenster zum Hinzufügen neuer Sensorknoten in eine Ebene.
Außerdem besitzt das Hauptfenster die Liste mit den Ebenenobjekten. Jedes Ebenenobjekt besitzt wiederum eine Liste mit Sensorknotenobjekten. Zieht der Benutzer einen neuen Sensorknoten aus dem *newSensorWindow* auf den Lageplan, so wird in dem entsprechenden Floorobjekt das Ereignis *dropEvent* ausgelöst, und der Sensor kann in die Sensorknotenliste aufgenommen werden.
Jede *sensorNode*-Instanz besitzt ein eigenes Tab-Widget als Fenster, in dem zwei Tabs für die Darstellung der Messdaten, die Übersicht über die aktuellen Sensorknotenwerte und die Ansteuerung der Aktoren stattfindet.

5.4 Der Debugserver

Neben den in den letzten beiden Abschnitten vorgestellten Programmen für den PC ist auch ein Debugserver entstanden. Dieser soll hier nur der Vollständigkeit halber erwähnt werden. Aufgabe des Debugservers ist es, einem Entwickler des Systems eine einfache Möglichkeit zum Testen des Sensornetzwerks an die Hand zu geben. Er kann damit direkt periodische und einmalige Interessen erstellen und die empfangenen Daten betrachten.
Der Debugserver besteht aus der seriellen Komponente, die auch im Server genutzt wird. Benutzt wird diese mit einer kleinen GUI, in der die periodischen und einmaligen Interessen jeweils in einem Tab feldweise – jedes Datum der Nutzdaten kann direkt eingegeben werden – definiert werden können. In einem QTextEdit-Widget werden die ankommenden Datenpakete in Textform ausgegeben. In einem weiteren QTextEdit-Widget werden die Informationsnachrichten mit der Sensorknotenadresse des Senders und der enthaltenen Nummer ausgegeben.
Der Debugserver wurde auch in dieser Arbeit bei der Implementierung des Sensornetzwerks genutzt, um das Verhalten der einzelnen Komponenten nachzuvollziehen und zu testen.

6 Evaluation

In diesem Kapitel wird das entworfene und implementierte System zur Erfassung eines Temperaturprofils in Wohn- und Büroräumen evaluiert. Hierzu werden zunächst die durchgeführten Tests vorgestellt und anschließend die in der Anforderungsdefinition in Abschnitt 2.2 aufgestellten Anforderungen auf ihre Umsetzung überprüft.
Den Anfang machen die Tests des Sensornetzwerks:

Test der Sensordatenerfassung Der Test der Sensordatenerfassung sollte zeigen, ob diese Komponente in der Lage ist, die Sensorwerte richtig zu erfassen, und insbesondere die im Abschnitt 4.4 vorgestellten Formeln zur Umrechnung des erfassten Wertes in die jeweilige Umweltgröße überprüfen.

Hierzu wurde eine kleine Testanwendung geschrieben, die alle fünf Sekunden die Messwerte von Temperatur, Luftfeuchte, Licht und Spannung mit der Sensordatenerfassungskomponente misst und an die unmodifizierte Basisstation aus den Anwendungsbeispielen von TinyOS sendet. Auf der Seite des PCs wurde der Debugserver zur Anzeige der Daten benutzt.

Der Sensorknoten wurde ins Badezimmer neben einen USB-Stick PCE-HT71 der Firma PCE Groupe Europe gelegt, welcher Temperatur und Luftfeuchte misst. Anschließend wurde die Dusche angestellt, um die Luftfeuchte auf 100 RH% zu bringen. Des Weiteren wurden die Spannung der Sensorknotenbatterien gemessen und verschiedene Batteriesätze eingelegt. Zur Referenzmessung der Lichtwerte lag leider kein passendes Messgerät vor. Es wurde daher der Lichtsensor mit der Hand abgedeckt, in einen abgedunkelten Raum bei eingeschalteter Deckenbeleuchtung gelegt und in direktes Sonnenlicht gehalten.

Es zeigte sich, dass der Sensorknoten die Temperatur im Vergleich zu dem USB-Stick um ca. 2,5 °C zu hoch maß. Die Luftfeuchte wich durchschnittlich um 5 RH% im Bereich zwischen 55 RH% und 80 RH% ab. Luftfeuchten unter 55 RH% waren nicht bei den Tests vorhanden, und über 80 RH% war die Abweichung unspezifisch. Es zeigte sich jedoch, dass sowohl der Sensorknoten als auch der USB-Stick irgendwann 99 RH% erreichten. Ferner konnte beobachtet werden, dass der Sensorknoten um den Faktor 2 träger war als der USB-Stick. Es dauerte nach dem Aufstellen des Sensorknotens zehn Minuten, bis sich die Messwerte für Temperatur und Luftfeuchte auf einen Wert eingependelt hatten. Die Spannug der Batterien wurde mit den Sensorknoten zwischen 2,6 V und 2,9 V gemessen. Mit einem Spannungsmesser wurden Spannungen zwischen 2,8 V und 3,0 V gemessen. Bedenkt man, dass die mit dem Spannungsmesser gemessenen Werte ohne Belastung gemessen wurden, können die mit dem Sensorknoten gemessenen Werte als korrekt angesehen werden. Die Lichtwerte betrugen bei der Abdeckung mit der Hand nahezu 0, und der Wert bei Sonneneinstrahlung war etwa um den Faktor 5 höher als der bei Zimmerbeleuchtung. Da die Lichtmessung nur als grobe Referenz für eine Anwesenheit von Menschen genommen werden soll, wird das Testergebnis als ausreichend bewertet.

Test der Zeitsynchroniation Der Test der Zeitsynchronisation sollte zeigen, ob sie eine globale Zeit herstellt und ob die Genauigkeit ausreichend ist. Hierzu sind in TinyOS bereits Testprogramme vorhanden. Es wurde auf acht Sensorknoten das Programm FtspLpl geschrieben, welches das FTSP zusammen mit dem LPL bereitstellt. Sobald diese Sensorknoten eine beliebige Nachricht erhalten, erfassen sie sofort die aktuelle globale Zeit und senden sie an die Basisstation, welche an einem Laptop angeschlossen ist. Ein Sensorknoten wurde mit dem RadioCountToLeds-Beispielprogramm von TinyOS programmiert. Dieses sendet in Abständen von einer Sekunde eine Nachricht an alle benachbarten Sensorknoten. So erhalten alle FTSP-Sensorknoten gleichzeitig diese Pakete und sollten

somit dieselbe Zeit an die Basisstation senden. Am PC wurden die empfangenen Zeitinformationen mit dem Serial-Listener von TinyOS betrachtet.

Die acht Sensorknoten wurden im Haus so verteilt, dass nicht alle Sensorknoten direkt miteinander kommunizieren konnten. Anschließend wurde mit dem Laptop samt Basisstation und dem RadioCountToLeds-Sensorknoten an der Kette der FTSP-Sensorknoten entlanggegangen. In einem zweiten Test wurde in der Mitte der Synchronisationskette ein Sensorknoten entfernt, die Zeiten in beiden Teilbereichen gemessen und anschließend der Sensorknoten wieder hinzugefügt.

Zunächst wurde gemessen, dass es zwischen zehn und 30 Sekunden dauerte, bis die initiale Synchronisation hergestellt war. Danach wiesen alle Sensorknoten eine einheitliche globale Zeit auf, die maximal um 50 Millisekunden und im Durchschnitt um zehn Millisekunden voneinander abwich. Nachdem das Synchronisationsnetz in zwei Teile geteilt wurde, drifteten die globalen Zeiten deutlich auseinander. Es war nach 30 Minuten eine Abweichung von über 30 Sekunden festzustellen. Nachdem die Netze wieder verbunden worden waren, synchronisierten sie sich wieder.

Test des LPL Der Test des LPL sollte zeigen, ob keine Pakete verloren gehen. Der LPL wies in einigen Versionen von TinyOS aus dem CVS Fehler auf. Er ist auch der Grund, warum TinyOS genau in der Version vom 16.07.2009 genutzt werden muss.

Hierzu wurden die Beispielanwendungen RadioCountToLeds und BaseStation um den LPL erweitert. Der Sensorknoten mit RadioCountToLeds sendete in periodischen Abständen von 100 Millisekunden ein Paket an die Basisstation, worin jeweils eine Sequenznummer mitgeschickt wird. So kann am PC überprüft werden, ob Pakete verlorengegangen sind. Dieser Test konnte erfolgreich durchgeführt werden. Der eine Sensorknoten sendete alle Pakete erfolgreich an die Basisstation.

Test des Interessensystems Es wurden mehrere Tests durchgeführt, um das Interessensystem zu testen. Ausgangsbedingung war jeweils ein aufgestelltes System bestehend aus neun Sensorknoten und der einen Basisstation. Auf dem PC kam der Debugserver zum Einsatz.

Test der Interessenverbreitung Dieser Test sollte zeigen, ob alle Sensorknoten ein an das Sensornetzwerk versendetes Interesse auch empfangen. Hierzu wurde ein Interesse nach der Temperatur, welches im Abstand von einer Sekunde bedient werden sollte, an das Sensornetzwerk versendet.

Es stellte sich heraus, dass durchschnittlich in acht von zehn Fällen alle Sensorknoten das Interesse erhielten. In 10% der Fälle erhielt einer der Sensorknoten das Interesse nicht, und in den restlichen 10% erhielten mehrere Sensorknoten das Interesse nicht. Es ist zu vermuten, dass in den Fehlerfällen Kollisionen so häufig vorkamen, dass die maximale Anzahl an Versendeversuchen überschritten wurde. Der Wert von 80% wird jedoch als recht gut angesehen, zumal auch andere Forscher mit diesem Problem kämpfen. Als Beispiel kann das im Abschnitt 3.2 vorgestellte System zur Vulkanüberwachung genannt werden [WALJ+06].

Test des Start-Stop-Timers Dieser Test sollte zeigen, ob der *Start-Stop-Timer* die Interessen korrekt startet und wieder beendet. Hierzu wurden zwei Interessen an einen Sensorknoten des Sensornetzwerks versendet. Das erste lieferte die Temperatur und das zweite die Luftfeuchte. Die Start- und Endzeitpunkte lagen in der Zukunft und unterschieden sich um drei bzw. zehn Sekunden.

Das Resultat war, dass der erste Wert für die Temperatur einen um ca. drei Sekunden abweichenden Zeitstempel zu dem der Luftfeuchte aufwies. Die letzte Messung der Luftfeuchte wies einen um ca. zehn Sekunden abweichenden Zeitstempel gegenüber der letzten Temperaturmessung aus. Der *Start-Stop-Timer* funktioniert damit korrekt.

Test des Interessentimers Dieser Test sollte zeigen, ob der *Interessentimer* die Interessen in der richtigen Messauflösung bedient. Hierzu wurden Interessen mit verschiedenen Messauflösungen an das Sensornetzwerk gesendet.

Es ergab sich, dass die Zeitstempel zweier aufeinanderfolgender Messungen die gewünsche Differenz aufwiesen. Des Weiteren war zu beobachten, dass die Messauflösung bei einer Messung aller vier Werte maximal 100 Millisekunden betragen durfte, damit die Sensordatenerfassung das Interesse noch in der gewünschten Messauflösung bedienen konnte.

Test der Parallelität Dieser Test sollte zeigen, dass auch mehrere periodische Interessen parallel auf einem Sensorknoten korrekt abgearbeitet werden. Hierzu wurden vier Interessen mit gleicher Startzeit zu verschiedenen Werten an das Sensornetzwerk gesendet. Die Messauflösungen betrugen zwei, vier, acht und 16 Sekunden.

Als Ergebnis blieb festzuhalten, dass alle Interessen auch in der vorgesehenen Messauflösung bedient wurden. In den Fällen, in denen Messungen zum gleichen Zeitpunkt nötig waren, wichen die Zeitstempel um bis zu einer halben Sekunde voneinander ab.

Test der einmaligen Interessen Dieser Test sollte zeigen, ob die einmaligen Interessen korrekt bedient werden. Dazu wurde ein einmaliges Interesse zum Setzen der LEDs in verschiedenen Variationen an das Sensornetzwerk gesendet. Es kam heraus, dass der Status der LEDs immer dem Wert entsprechend geändert wurde. Die einmaligen Interessen und damit auch die Aktorenansteuerung funktionieren entsprechend dem Entwurf.

Test des Routingprotokolls Es wurden mehrere Tests durchgeführt, um das Routingprotokoll zu testen. Ausgangsbasis war wieder ein Sensornetzwerk wie bei den Tests des Interessensystems. Hierbei wurde jedoch darauf geachtet, dass mindestens ein Sensorknoten keine direkte Verbindung zur Basisstation besaß.

Test der Multi-Hop-Kommunikation Dieser Test sollte zeigen, ob auch in Multi-Hop-Netzwerken die Daten korrekt über mehrere Sensorknoten ihren Weg zur Basisstation finden. Dazu wurde ein Interesse nach der Sensorknotenadresse des besten Gradienten an alle Sensorknoten versendet. Damit konnte zum einen gezeigt werden, dass die Interessenantwort auch bei der Basisstation ankommt, und zum anderen, dass die Sensorknoten mindestens über einen anderen Sensorknoten die Daten an die Basisstation senden.

Es zeigte sich, dass alle Antworten bei der Basisstation angekamen und einige von ihnen nicht die Basisstation als besten Gradienten eingetragen hatten.

Test der Robustheit gegenüber Ausfällen Dieser Test sollte zeigen, wie robust das Sensornetzwerk gegenüber Ausfällen von Sensorknoten ist. Zunächst wurde ein periodisches Interesse an den nicht direkt mit der Basisstation in Verbindung stehenden Sensorknoten gesendet. Danach wurde der Sensorknoten, welcher der beste Gradient dieses Sensorknotens war, entfernt. Anschließend wurde ein neuer Sensorknoten mit einer anderen Adresse an dessen Stelle gelegt.

Das Ergebnis war, dass erst nach dem Versenden eines weiteren Interesses der neue Sensorknoten den alten verdrängen konnte. Das liegt dadran, dass die Gradiententabelle lediglich bei neuen Interessen bereinigt wird, solange das maximale Gradientenalter nicht erreicht ist. Als Verbesserung könnte man periodisch einen Timer feuern lassen, der die Gradiententabelle aufräumt. Um dies sicher erledigen zu können, müsste jedoch ein Handshake mit den Gradienten in der Tabelle durchgeführt werden. Diese Option wurde zwar in Betracht gezogen, aber letztendlich für zu aufwändig bei der gegebenen Restlaufzeit des Projekts angesehen. Das entworfene Sensornetzwerk ist damit nur bedingt vor Ausfällen geschützt, begegnet dem jedoch, indem es die Interessen vom Server in periodischen Abständen immer wieder neu versendet.

Test des Sendeverhaltens von Datenpaketen Dieser Test sollte zeigen, ob die *slowQueue* eine definierte Anzahl an Datenpaketen ansammelt, bevor sie gesendet werden. Hierzu wurde die Anzahl der anzusammelnden Pakete auf 20 gesetzt und ein Interesse nach der Temperatur mit einer Messauflösung von einer Sekunde an ein Sensornetzwerk mit einem Sensorknoten versendet.

Resultat war, dass ca. alle 20 Sekunden 20 Datenpakete auf einmal eintrafen.

Das PC-Programm wurde mittels Rapid Prototyping umgesetzt und die einzelnen Funktionen direkt nach ihrer Implementierung getestet. Es fand daher keine getrennte Testphase statt, und die Tests werden daher nicht so ausführlich dokumentiert wie im Sensornetzwerk. Es soll an dieser Stelle nur gesagt werden, dass keine Fehler bekannt sind. Die Komponente mit der seriellen Schnittstelle konnte beiläufig bei der Benutzung des Debugservers während der Tests des Sensornetzwerks getestet werden. Zudem wurde der Inhalt der Datenbank nach einem Interesse für alle vier Werte mit einer Messauflösung von einem Wert pro Sekunde über einen Zeitraum von 10 Minuten überprüft. Es waren 599 Werte in der Datenbank vorhanden. Es fehlte somit lediglich ein Wert. Also kann das Versenden der Interessen bis zur Aufnahme der Ergebnisse in die Datenbank ebenfalls als getestet angesehen werden.
Im Folgenden soll das entstandene System zur Erfassung eines Temperaturprofils in Wohn- und Büroräumen bewertet werden. Dazu werden zunächst, wie bereits in der Einleitung dieses Kapitels angekündigt, die einzelnen Anforderungen aus Abschnitt 2.2 durchgegangen.

E1 Datenerfassung: Diese Anforderung kann im vollen Umfang als erfüllt angesehen werden. Es werden Umgebungstemperatur, Luftfeuchte, Lichtwerte und dadrüber hinaus sogar noch die Batteriespannung erfasst. Es wird eine Messauflösung von 0,1 Sekunde erreicht, was um den Faktor 10 besser als der geforderte Wert ist.

E2 Laufzeit der Sensorknoten: Eine Laufzeit von einem Jahr war gefordert. Es ist schwierig, diese Anforderung genau zu bewerten, weil die Laufzeit nicht so einfach berechnet werden kann. Die Messauflösung und die Anzahl der Sensorknoten im Sensornetzwerk haben einen großen Einfluss auf das Datenaufkommen und damit auch den Energiebedarf. Das oberste Gebot war jedoch bei der gesamten Entwicklung, so schonend wie möglich mit den Ressourcen umzugehen. Es wird ein energieeffizientes Betriebssystem, Routingprotokoll und MAC-Protokoll verwendet. Die Laufzeit der Sensorknoten kann damit als maximiert angesehen werden, weswegen diese Anforderung als erfüllt betrachtet wird.

E3 Zeitsynchronisation: Die Sensorknoten synchronisieren sich mit einer durchschnittlichen Genauigkeit von zehn Millisekunden, was um den Faktor 25 besser ist als der geforderte Wert. Diese Anforderung ist damit erfüllt.

E4 Selbstorganisation der Sensorknoten: Die Sensorknoten organisieren sich nach dem Aufstellen autonom zu einem Sensornetzwerk. Neu hinzukommende Sensorknoten integrieren sich mit dem nächsten Interesse selbstständig in das Sensornetzwerk.

Kritischer ist der Ausfall eines Sensorknotens. Dieser wird erst nach dem Versenden eines Interesses kompensiert. Der Server stellt jedoch sicher, dass Interessen in regelmäßigen Abständen gesendet werden. Zudem sind die Sensorknoten in der Lage, einen Fehler bei der Kommunikation zu erkennen und die Daten vorzuhalten, bis eine neue Verbindung zur Basisstation hergestellt wurde. Des Weiteren wurde extra darauf geachtet, dass ein multipfadfähiges Routingprotokoll verwendet wird. Diese Anforderung wird daher insgesamt als erfüllt angesehen, wobei hier sicher noch Verbesserungspotential vorhanden ist.

E5 Datenspeicherung: Die Daten werden gespeichert, und die Anforderung kann damit als erfüllt angesehen werden.

E6 Aktoransteuerung: Die einmaligen Interessen können für die Ansteuerung der Aktoren genutzt werden, was auch anhand der LEDs umgesetzt wurde. Diese Anforderung ist damit ebenfalls erfüllt.

E7 Datenvisualisierung: Dem Benutzer werden die Messdaten in einem Zeit-Wert-Diagramm präsentiert, und die Anforderung ist somit erfüllt.

W1 Komplexe Visualisierung: Es ist eine Anwendung entstanden, in welcher der Benutzer Ebenen definieren und Sensoren positionieren kann. Diese ist übersichtlich aufgebaut und intuitiv zu bedienen. Dem Benutzer werden die Messdaten in einem Zeit-Werte-Diagramm präsentiert, wodurch er auf einen Blick die Daten überschauen kann und Trends in den Daten gut erkennbar sind. Diese Anforderung ist daher erfüllt.

W2 Trennung zwischen Benutzerschnittstelle und Sensornetzkontrolle: Das PC-Programm besteht aus einem Server und einem Client-Programm. Die Anforderung ist erfüllt.

W3 Komplexe Auswertung: Diese Anforderung konnte nicht erfüllt werden. Der Umfang des Projekts war mehr als ausreichend, und daher wurde nicht mehr mit dem Themenbereich der Auswertung begonnen. Die Entwicklung einer Regelung ist ein Themenbereich für sich, weswegen man sich bewusst dafür entschied, diesen Bereich nicht bloß oberflächlich anzuschneiden. Der Fokus wurde auf die Entwicklung des Datenerfassungssystems gelegt, damit ein möglichst mächtiges System für Anschlussforschungen zur Verfügung steht.

Wie nun feststeht, wurden bis auf eine alle Anforderungen erfüllt. Der Schwerpunkt der Arbeit lag auf der Entwicklung des Sensornetzwerks. Dies ist daher auch deutlich mächtiger als das PC-Programm. Hierbei kann man auch beim PC-Programm mehr Funktionen auf Server- als auf Clientseite finden. Der Server kann die Möglichkeiten des Sensornetzwerks in vollem Unfang nutzen. Das Client-Programm unterstützt keine Interessen, die nur einzelne Sensorknoten bedienen können. Diese Funktion könnte aber leicht hinzugefügt werden und wurde lediglich aus Zeitgründen weggelassen, da diese Funktion nicht von der Anforderungsdefinition vorgegeben wurde.
Verbesserungen beim Sensornetzwerk können neben dem bereits erwähnten Aufräumen der Gradiententabelle insbesondere bei der Sicherheit der Übertragung an alle benachbarten Sensorknoten stattfinden. Diese beiden Punkte könnten verbessert werden, indem der Sensorknoten regelmäßig alle seine Nachbarn auf Erreichbarkeit prüft und eine Tabelle mit seinen Nachbarn vorhält. Bei der Interessenausbreitung könnte dann statt eines Broadcasts ein gezielter Weiterversand der Interessen an einzelne Sensorknoten erfolgen. Dadurch kann der Empfang der Pakete bestätigt werden, und nicht mit einem Acknowledgement antwortende Sensorknoten erhalten das Interesse erneut. Außerdem könnte dann bei jeder Aktualisierung der Nachbarschaftstabelle die Gradiententabelle aufgeräumt werden. Eine solche Tabelle wurde nicht eingeführt, da sie bereits in TinyOS für die Zukunft geplant ist und dann eingesetzt werden kann, wenn das Betriebssystem um diese Funktion erweitert wurde.
Eine weitere Verbesserung des Sensornetzwerks wäre die Definition eines generischen Paketformats, welches für mehrere verschiedene Aufgaben eingesetzt werden kann. In dem hier vorgestellten Sensornetzwerk werden z. B. die Gradienten schon mithilfe der Informationsnachrichten gesendet, die eigentlich für Fehler- und Statusmeldungen vorgesehen sind. Die Definition eines generischen Paketformats ist jedoch nicht einfach, da zur Kompilierzeit der Aufbau des Datenbereichs eines Pakets fest angegeben werden muss. Zudem unterscheidet der Server eintreffende Pakete anhand der Größe des Datenbereichs. Dadurch kann Bandbreite gespart werden, indem nicht ein Pakettypfeld in jedes Paket aufgenommen werden muss. Positiv ist jedoch zu erwähnen, dass der Kanal für die Informationspakete neutral gegenüber dem Paketformat ist. Somit ist das Hinzufügen neuer Nachrichtentypen für den Datenaustausch von der Quelle zur Basisstation ohne weiteres möglich.
Was noch aussteht, ist ein Test des Sensornetzwerks über einen längeren Zeitraum. Das System wurde maximal sechs Stunden am Stück betrieben, für Langzeittests blieb gegen Ende der Bearbeitungszeit kein Raum mehr. Es soll jedoch auch erwähnt werden, dass andere, bereits im produktiven Einsatz befindliche Sensornetzwerke auch kurze Aufstellungszeiten aufweisen. Das Sensornetzwerk zur Vulkanüberwachung wurde insgesamt nur 19 Tage ausgebracht, und selbst während dieser Zeit kam es zu diversen Fehlern [WALJ+06].
Abschließend soll erwähnt werden, dass das entwickelte Sensornetzwerk nicht nur in Wohn- und Büroräumen eingesetzt werden kann. Alle Komponenten des Sensornetzwerks, also Betriebssystem, Routing-

protokoll, Zeitsynchronisationsprotokoll, Sensordatenerfassung und die Anwendung stellen keinerlei Bedingungen an den Aufstellungsort. Es muss lediglich sichergestellt sein, dass die Basisstation an ein Gerät mit Netzwerkzugang angeschlossen ist. Dies kann auch ein kleines eingebettetes System mit einer Linux-Distribution sein. So kann als Plattform für den Server auch ein Router oder NAS-Gerät genutzt werden. Aufgrund der Plattformunabhängigkeit können Server und Client-Programm sowohl unter Windows als auch unter GNU/Linux eingesetzt werden.

7 Fazit und Ausblick

Sensornetzwerke nehmen einen immer wichtigeren Stellenwert in unserer Gesellschaft ein. In den letzten Jahren sind diverse Sensornetzwerke für verschiedenste Anwendungsfälle entstanden. Viele davon befinden sich jedoch noch im Teststadium. Die überwiegende Mehrheit der während dieses Projekts gelesenen Veröffentlichungen zu Sensornetzwerken im Feldeinsatz wurde jedoch nicht für den Einsatz in Gebäuden entworfen. Es ist jedoch besonders interessant, Daten über den Temperaturverlauf innerhalb von Gebäuden zu sammeln. Für konventionelle Warmwasserheizungen kann auf Grundlage dieser Daten eine bessere Regelstrategie gefunden werden, und die modernen Warmluftheizungen setzen eine Temperaturerfassung sogar voraus. Gerade letztere gewinnen aufgrund der Forderung nach Energieeffizienz und Niedrigenergiehäusern einen immer größeren Stellenwert.

In dieser Arbeit ist ein solches Sensornetzwerk zur Erfassung eines Temperaturprofils innerhalb von Wohn- und Büroräumen entstanden. Dieses ist zudem auch in der Lage, ein Luftfeuchtigkeits- und Lichtprofil zu erstellen, womit auch das Klima und die Nutzung der Räume besser untersucht werden können. Das vorgestellte Sensornetzwerk besitzt eine Messauflösung von 100 Millisekunden und versieht alle Messwerte mit einem Zeitstempel. Weiterhin ist ein PC-Programm entstanden, mit dem der Benutzer das Sensornetzwerk ohne große Kenntnisse über dessen Funktionsweise betreiben kann. Als Ergebnis werden ihm die erfassten Daten in einem Diagramm präsentiert. Möchte man die reinen Messdaten haben, so kann man auch direkt auf die Datenbank zugreifen. Somit sind alle Möglichkeiten zur Weiterverarbeitung der Daten gegeben.

Das vorgestellte Sensornetzwerk ist auch gut erweiterbar. Es stehen noch mehrere Interessen zur Definition zur Verfügung und über herausgeführte Pins des Mikrocontrollers bei dem verwendeten Sensorknoten können weitere Sensoren angeschlossen werden. Dieses Sensornetzwerk kann damit allgemein für die Sammlung von Umweltdaten genutzt werden. Darüber hinaus können leicht weitere Paketformate hinzugefügt werden, um auch andere Daten übertragen zu können. Wird die Menge der Daten, die zu senden sind, zu groß, so kann das Routingprotokoll auch um eine Datenaggregation erweitert werden.

Neben der Datenerfassung zur Heizungsregelung kann das Sensornetzwerk auch gut für Forschungszwecke genutzt werden. Es wurde ein eigenes Routingprotokoll implementiert, welches es wert ist, weiter untersucht zu werden. Es wird bereits die Senderichtung der Sensorknoten mit dem Interesse nach dem besten Gradienten implementiert. So kann bereits jetzt die Baumstruktur der Netzwerkknoten zusammengesetzt werden. Darüber hinaus wäre es interessant zu untersuchen, wie sich das Routingprotokoll in größeren Sensornetzwerken verhält. Die bei der Bearbeitung gelesenen Veröffentlichungen zu Praxiseinsätzen umfassten selten mehr als 100 Sensorknoten, meistens sogar deutlich weniger. Der Abteilung *Systemsoftware und Verteilte Systeme* stehen 130 Sensorknoten zur Verfügung. In den gelesenen Veröffentlichungen wurde bei größeren Sensornetzwerken die Sicherheit der Paketübertragung betrachtet. Gerade in großen Netzen kann es häufig zu Kollisionen kommen, während gleichzeitig ein hoher Prozentsatz an erfolgreich von der Quelle zur Senke übertragenen Paketen erstrebenswert ist. Hierbei kann der Debugserver zum direkten Zugriff auf das Netzwerk genutzt und für entsprechende Untersuchungen erweitert werden.

Bei der Entwicklung des vorgestellten Sensornetzwerks wurde eine ingenieurmäßige Vorgehensweise an den Tag gelegt. Zunächst wurden die Anforderungen festgelegt, um die Aufgabe besser zu verstehen und mit dem Auftraggeber eine gemeinsame Basis bezüglich der umzusetzenden Funktionen zu haben. Danach wurde ein Entwurf erstellt und implementiert. Abschließend wurde das entstandene System anhand der Anforderungen evaluiert.

Bei der Bearbeitung zeigte sich, dass viel Literatur gelesen werden musste. Die Mechanismen in Sensornetzwerken unterscheiden sich aufgrund der Ressourcenbeschränkungen erheblich von herkömmlichen PC-Programmen. Daher wurden ausführlich die möglichen Alternativen beschrieben, um eine Auswahl zu treffen. Der Entwurf des Sensornetzwerks umfasste in nicht unerheblichem Maße das Verständnis der ein-

zusetzenden Technologien, die dann bereits im Falle des Betriebssystems mitsamt Gerätetreibern für die Sensoren und der Zeitsysnchronisation fertig implementiert vorlagen. Viel Arbeit musste jedoch mit der praktischen Nutzung dieser Software verbracht werden. Die Sensorknoten besitzen nur wenige direkt zugängliche Ein- und Ausgabegeräte in Form von drei LEDs und einem Knopf. Es war nicht immer einfach, Fehler zu lokalisieren. Zwar gibt es auch Simulationsumgebungen, aber in der realen Welt mit im gleichen Frequenzband operierenden WLANs und anderen Störquellen verhält sich die Software ganz anders. Möchte man eine neue Version testen, so muss man die Software immer umständlich auf jeden einzelnen Sensorknoten aufspielen und diese dann wieder, je nach Testfall, im gesamten Haus verteilen.
Dennoch konnten alle bis auf eine Anforderung erfüllt werden. Die nicht erfüllte Anforderung war jedoch nur wünschenswert und für den Fall gedacht, dass gegen Ende der Projektlaufzeit noch Zeit übrig geblieben wäre. Im Nachhinein stellte sich jedoch heraus, dass die gesteckten Ziele bereits sehr ambitioniert waren. Es sind keine großen Fehler in dem System bekannt, so dass insgesamt ein positives Fazit gezogen werden kann. Abschließend wird hiermit die entstandene Software unter die GNU General Public License (GPL) gestellt.

Literaturverzeichnis

[AKK04] AL-KARAKI, J. N. ; KAMAL, A. E.: Routing techniques in wireless sensor networks: a survey. In: *IEEE Wireless Communications* 11 (2004), December, Nr. 6, S. 6–28

[ASSC02] AKYILDIZ, Ian F. ; SU, Weilian ; SANKARASUBRAMANIAM, Yogi ; CAYIRCI, Erdal: Wireless sensor networks: a survey. In: *Computer networks* 38 (2002), Nr. 4, S. 393–422

[Bac09] BACHMAIER, Sebastian A.: UML 2.0 for modeling TinyOS components. In: *GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze". Hamburg* (2009)

[BCD+05] BHATTI, Shah ; CARLSON, James ; DAI, Hui ; DENG, Jing ; ROSE, Jeff ; SHETH, Anmol ; SHUCKER, Brian ; GRUENWALD, Charles ; TORGERSON, Adam ; HAN, Richard: MANTIS OS: an embedded multithreaded operating system for wireless micro sensor platforms. In: *Mob. Netw. Appl.* 10 (2005), Nr. 4, S. 563–579

[BCE+06] BONNET, Philippe ; CULLER, David ; ESTRIN, Deborah ; GOVINDAN, Ramesh ; HORTON, Mike ; KANG, Jeonghoon ; LEVIS, Philip ; NACHMAN, Lama ; STANKOVIC, Jack ; SZEWCZYK, Rob ; WELSH, Matt ; WOLISZ, Adam: TEP 120 TinyOS Alliance Structure. (2006)

[BKM04] BALDUS, Heribert ; KLABUNDE, Karin ; MÜSCH, Guido: Reliable Set-Up of Medical Body-Sensor Networks. In: *Wireless Sensor Networks* (2004), S. 353–363

[Bou09] BOUKERCHE, Azzedine: *Algorithms and Protocols for Wireless Sensor Networks*. 1. Wiley, 2009

[BU07] BRINKSCHULTE, Uwe ; UNGERER, Theo: *Mikrocontroller und Mikroprozessoren*. 2. Springer, 2007

[CES04] CULLER, David E. ; ESTRIN, Deborah ; SRIVASTAVA, Mani B.: Guest Editors' Introduction: Overview of Sensor Networks. In: *IEEE Computer* 37 (2004), Nr. 8, S. 41–49

[CI05] CHAKRABARTY, Krishnendu ; IYENGAR, S.S.: *Scalable Infrastructure for Distributed Sensor Networks*. 1. Springer, 2005

[CTL+08] CHANG, Ho-lin ; TIAN, Jr-ben ; LAI, Tsung-Te ; CHU, Hao-Hua ; HUANG, Polly: Spinning beacons for precise indoor localization. In: *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, ACM, 2008, S. 127–140

[DGV04] DUNKELS, Adam ; GRONVALL, Bjorn ; VOIGT, Thiemo: Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In: *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*. Washington, DC, USA : IEEE Computer Society, 2004, S. 455–462

[EGE02] ELSON, Jeremy ; GIROD, Lewis ; ESTRIN, Deborah: Fine-grained network time synchronization using reference broadcasts. In: *SIGOPS Oper. Syst. Rev.* 36 (2002), Nr. SI, S. 147–163

[Fira] FIRMA BUDERUS: `http://www.buderus.de`. – Online–Ressource, Abruf: 22.03.2010

[Firb] FIRMA HAMAMATSU: http://sales.hamamatsu.com/assets/pdf/parts_S/S1087_etc.pdf. – Online–Ressource, Abruf: 23.05.2010. – Si photodiode S1087/S1133 series

[Firc] FIRMA MAXFOR: http://www.maxfor.co.kr/eng/en_sub5_1_1.html. – Online–Ressource, Abruf: 23.03.2010. – Technische Daten des verwendeten Sensorknotens

[Fird] FIRMA SENSIRION: http://octopart.com/sht11-sensirion-910236#datasheets. – Online–Ressource, Abruf: 23.05.2010. – Datasheet SHT1x (SHT10, SHT11, SHT15)

[Fire] FIRMA TROLLTECH: *Signals and Slots*. http://doc.trolltech.com/4.6/signalsandslots.html. – Online–Ressource, Abruf: 08.05.2010

[GKS03] GANERIWAL, Saurabh ; KUMAR, Ram ; SRIVASTAVA, Mani B.: Timing-sync protocol for sensor networks. In: *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, ACM, 2003, S. 138–149

[GL] GREENSTEIN, Ben ; LEVIS, Philip: TEP 113 Serial Communication.

[GvHS05] GANERIWAL, Saurabh ; ČAPKUN, Srdjan ; HAN, Chih-Chieh ; SRIVASTAVA, Mani B.: Secure time synchronization service for sensor networks. In: *WiSe '05: Proceedings of the 4th ACM workshop on Wireless security*, ACM, 2005, S. 97–106

[HHSH06] HARTUNG, Carl ; HAN, Richard ; SEIELSTAD, Carl ; HOLBROOK, Saxon: FireWxNet: a multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In: *Proceedings of the 4th international conference on Mobile systems, applications and services*, 2006, S. 28–41

[IGE+03] INTANAGONWIWAT, Chalermek ; GOVINDAN, Ramesh ; ESTRIN, Deborah ; HEIDEMANN, John ; SILVA, Fabio: Directed diffusion for wireless sensor networking. In: *IEEE/ACM Trans. Netw.* 11 (2003), Nr. 1, S. 2–16

[KA08] KWON, YoungMin ; AGHA, Gul: Passive Localization: Large Size Sensor Network Localization Based on Environmental Events. In: *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*, IEEE Computer Society, 2008, S. 3–14

[KPC+07] KIM, Sukun ; PAKZAD, Shamim ; CULLER, David ; DEMMEL, James ; FENVES, Gregory ; GLASER, Steven ; TURON, Martin: Health monitoring of civil infrastructures using wireless sensor networks. In: *Proceedings of the 6th international conference on Information processing in sensor networks*, ACM Press, 2007, S. 254–263

[Lev] LEVIS, Philip: TEP 111 message_t.

[LG09] LEVIS, Philip ; GAY, David: *TinyOS Programming*. 1. Cambridge University Press, 2009

[LS] LEVIS, Philip ; SHARP, Cory: TEP 106 Schedulers and Tasks.

[MHK] MOSS, David ; HUI, Jonathan ; KLUES, Kevin: TEP 105 Low Power Listening.

[MHLC07] MOSS, David ; HUI, Jonathan ; LEVIS, Philip ; CHOI, Jung I.: TEP 126 CC2420 Radio Stack. (2007)

[MKSL04] MARÓTI, Miklós ; KUSY, Branislav ; SIMON, Gyula ; LÉDECZI, Ákos: The flooding time synchronization protocol. In: *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, ACM, 2004, S. 39–49

[MLV09] MOZUMDAR, Mohammad Mostafizur R. ; LAVAGNO, Luciano ; VANZAGO, Laura: A comparison of software platforms for wireless sensor networks: MANTIS, TinyOS, and ZigBee. In: *ACM Trans. Embed. Comput. Syst.* 8 (2009), Nr. 2, S. 1–23

[MS08] MAROTI, Miklos ; SALLAI, Janos: TEP 133 Packet-level time synchronization. (2008)

[RKJK03] REDDY, Adi M. ; KUMAR, AVU P. ; JANAKIRAM, D. ; KUMAR, G. A.: Operating Systems for Wireless Sensor Networks: A Survey. (2003)

[SGS+06] SOUSA, João B. ; GONÇALVES, G. ; SOUSA, A. ; PINTO, J. ; LEBRES, P.: Pilot Experiment of an Early Warning Fire Detection System. In: *3rd European Workshop on Wireless Sensor Networks*. Switzerland, 2006

[SLT+07] SZEWCZYK, Robert ; LEVIS, Philip ; TURON, Martin ; NACHMAN, Lama ; BUONADONNA, Philip ; HANDZISKI, Vlado: TEP 112 Microcontroller Power Management. (2007)

[SSS07] STOLERU, Radu ; STANKOVIC, John A. ; SON, Sang H.: Robust node localization for wireless sensor networks. In: *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors*, ACM, 2007. – ISBN 978–1–59593–694–3, S. 48–52

[Sta08] STANKOVIC, John A.: Wireless Sensor Networks. In: *IEEE Computer* 41 (2008), Nr. 10, S. 92–95

[Sto05] STOJMENOVIĆ, Ivan: *Handbook of Sensor Networks*. 1. Wiley, 2005

[Tan92] TANNENBAUM, Andrew S.: *Modern Operating Systems*. 1. Prentice-Hall, 1992

[The] THEODOR HEIMEIER METALLWERK GMBH: *Auslegung von Thermostatventilen*. `http://heimeier.com/de/index.asp?art=wissenswertes&id=135`. – Online–Ressource, Abruf: 29.04.2010

[The05] THEEL, Oliver: Skriptum zur Vorlesung Betriebssysteme 1. (2005)

[Unb] UNBEKANNT: *TOSThreads Tutorial*. `http://docs.tinyos.net/index.php/TOSThreads_Tutorial#The_TOSThreads_library`. – Online–Ressource, Abruf: 29.04.2010

[VDMC08] VERDONE, Roberto ; DARDARI, Davide ; MAZZINI, Gianluca ; CONTI, Andrea: *Wireless Sensor and Actuator Networks*. 1. Elsevier, 2008

[WALJ+06] WERNER-ALLEN, Geoff ; LORINCZ, Konrad ; JOHNSON, Jeff ; LEES, Jonathan ; WELSH, Matt: Fidelity and yield in a volcano monitoring sensor network. In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, USENIX Association, 2006, S. 27–43

[WALW+06] WERNER-ALLEN, Geoffrey ; LORINCZ, Konrad ; WELSH, Matt ; MARCILLO, Omar ; JOHNSON, Jeff ; RUIZ, Mario ; LEES, Jonathan: Deploying a Wireless Sensor Network on an Active Volcano. In: *IEEE Internet Computing* 10 (2006), Nr. 2, S. 18–25

[Wik10] WIKIPEDIA: *List of wireless sensor nodes — Wikipedia, The Free Encyclopedia*. Version: 2010. `http://en.wikipedia.org/w/index.php?title=List_of_wireless_sensor_nodes&oldid=354054449`. – Online–Ressource, Abruf: 7.04.2010

[WXR+04] WHANG, Daniel H. ; XU, Ning ; RANGWALA, Sumit ; CHINTALAPUDI, Krishna ; GOVINDAN, Ramesh ; WALLACE, John W.: Development of an Embedded Networked Sensing System for Structural Health Monitoring. In: *Proceedings of the International Workshop on Smart Materials and Structures Technology*, Center for Embedded Network Sensing, 2004

[XOL+07] XU, Yurong ; OUYANG, Yi ; LE, Zhengyi ; FORD, James ; MAKEDON, Fillia: Mobile anchor-free localization for wireless sensor networks. In: *DCOSS'07: Proceedings of the 3rd IEEE international conference on Distributed computing in sensor systems*, Springer-Verlag, 2007, S. 96–109

[ZH09] ZHONG, Ziguo ; HE, Tian: Achieving range-free localization beyond connectivity. In: *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, ACM, 2009, S. 281–294

Anhang

A Inhaltsverzeichnis der beiliegenden CD

/Bin/ Enthält die entstandenen Programme in gebauter Form

/Quellen Enthält alle Quelldateien der entstandenden Programme

- **/Debugserver/** Enthält die Quellen des Debugservers
- **/PC-Programm/Client-Programm/** Enthält die Quellen des Client-Programms
- **/PC-Programm/Server/** Enthält die Quellen des Servers
- **/Sensornetzwerk/Basisstation/** Enthält die Quellen der Basisstation
- **/Sensornetzwerk/Sensorknoten/** Enthält die Quellen der Sensorknoten

/TinyOS Enthält die ausgecheckte Fassung des TinyOS-Repositorys mit Entwicklungsstand vom 16.07.2009.

- **/app/** Enthält die TinyOS beiliegenden Beispielanwendungen
- **/doc/txt/** Enthält die TinyOS Enhancement Proposals (TEP).
- **/support/sdk/sf/sfsource.c** Die Implementierung des seriellen Protokollstapels von TinyOS für PC-Anwendungen.
- **/tos/** Enthält alle Quellen von TinyOS.
- **/tos/lib/ftsp/** Enthält die Quellen des Flooding-Time-Synchronization-Protokolls.

/doc Enthält alle während der Projektlaufzeit entstandenen Textdokumente

- **/Benutzerhandbuch.pdf** Das Benutzerhandbuch für das System zur Erfassung eines Temperaturprofils innerhalb von Wohn- und Büroräumen.
- **/Einstiegsvortrag.pdf** Die Folien des zu Beginn der Projektlaufzeit gehaltenen Einstiegsvortrags
- **/Proposal.pdf** Die zu Beginn der Projektlaufzeit abgegebene Arbeitsvereinbarung
- **/Diplomarbeit.pdf** Die schriftliche Ausarbeitung zu dem Projekt (dieses Dokument)