FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Sensornetzwerke im Bauingenieurwesen

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Filip Kovacevic
Matrikelnummer 1227213

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Ao.Univ.Prof.Dr. Wolfgang Kastner

Wien, 22. April 2015

Filip Kovacevic	Wolfgang Kastner

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Sensor Networks in Constructional Engineering

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Business Informatics

by

Filip Kovacevic
Registration Number 1227213

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof.Dr. Wolfgang Kastner

Vienna, 22nd April, 2015

Filip Kovacevic

Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Filip Kovacevic
Meidlinger Hauptstraße 70/6

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 22. April 2015

Filip Kovacevic

Danksagung

Ich möchte an erster Stelle meinem Trainingspartner und Freund Aleksandar Radoevski danken, welcher mir diese Arbeit ermöglicht und die Motivation dazu verliehen hat. Weiters spreche ich meinen Dank meinem Betreuer und Professor Ao.Univ.Prof.Dr. Wolfgang Kastner aus, welcher mich bei dieser Arbeit sehr unterstützt, die Arbeit gründlichst gelesen und wertvolle Kritik geliefert hat. Ein drittes Dankeschön geht an meinen Freund Johannes Hochsteger, welcher mir besonders viel Hoffnung und Kraft für diese Arbeit geschenkt hat.

Kurzfassung

Sensornetzwerke sind bei der Überwachung von zivilen Einrichtungen immer häufiger im Einsatz. In solchen Netzwerken ist ein Zusammenspiel sämtlicher Komponenten wichtig, um ein möglichst stabiles, effizientes und sicheres System zu gewährleisten. Einerseits sind es mikroelektromechanische Systeme (MEMS), welche als Teil von einem Netzwerk-/Sensorknoten effizient arbeiten müssen, da deren Energiekapazitäten in den meisten Umgebungen begrenzt sind. MEMS sind mit Sensoren für die eigentliche Messung ausgestattet. Schon bei den Sensoren kann auf eine effiziente Lösung gesetzt werden, da diese schon ein altes Forschungsgebiete sind und zwischen einer Vielzahl von Messmethoden gewählt werden kann. Insbesondere bei der Überwachung von zivilen Einrichtungen ist das Messverfahren entscheidend, da es nicht ein Verfahren gibt, welches für jedes Einsatzgebiet am geeignetsten ist. Das Konfigurieren und Warten der Sensorknoten stellt auch oft eine Herausforderung dar. Probleme bei der Kommunikation können auftreten und Signale fehlerhaft übermittelt werden oder erst gar nicht ankommen. Beim Kommunizieren muss verstärkt auf ausgefallene Sensorgeräte geachtet werden. Auch Sensornetzwerke bleiben nicht vom Thema Sicherheit verschont und müssen sogar noch durchdachtere Protokolle implementieren, welche die Energieressourcen der einzelnen Knoten berücksichtigen und trotzdem einen gewissen Grad an Sicherheit liefern. Weiters müssen auch darüberliegende Anwendungen ihre Leistung erbringen und das System steuern. Ausgefallene Knoten müssen erkannt und angezeigt werden. Die Knoten müssen von der Basisstation aus wartbar sein. Bei der Versendung von Softwarepaketen und Konfigurationsfiles ist ebenfalls der Energieverbrauch und die Speicherkapazität der Knoten zu berücksichtigen. Anwendungsprogrammierer haben es oft leichter, wenn sie mit Services einer Middleware arbeiten können, anstatt sich mit den Hardware-Befehlen des Betriebssystems der Sensorknoten auseinandersetzten zu müssen. Diese Arbeit baut schrittweise auf und bahnt sich ihren Weg von der Funktionsweise der Sensoren und MEMS, über den Aufbau von und Probleme bei Sensornetzwerken, bis hin zum Einsatz von Sensornetzwerken im Structural Health Monitoring.

Inhaltsverzeichnis

Abbildungsverzeichnis

KAPITEL 1

Einführung und Motivation

Sensoren stellen eine Verbindung zwischen der physischen und digitalen Welt dar, indem sie Daten wie z.B. Licht, Temperatur und Vibrationen aus ihrer unmittelbaren Umgebung erfassen und weiterleiten. Dabei werden diese im Zuge eines Transformationsprozesses digitalisiert, abgespeichert und können dann zur Weiterverarbeitung verwendet werden. Sensornetzwerke nutzen solche Sensoren als Teil von sogenannten mikroelektromechanischen Systemen (kurz MEMS) für die Lösung komplexer Aufgaben im Bereich des Monitorings. Aufgrund der vielfältigen Einsatzmöglichkeiten der Sensoren können auch Sensornetzwerke verschiedenartig eingesetzt werden. Beispielsweise bestehen Reifendruckkontrollsysteme in Fahrzeugen aus drahtlosen Sensoren, welche Informationen zum Reifendruck an den Kontrollbildschirm des Fahrzeugs senden. [12] Weiters werden biomedizinische Sensornetzwerke eingesetzt, um Patienten außerhalb der Krankenhäuser zu überwachen und im Notfall rechtzeitig einen Alarm auszusenden. [13] Auch das Militär, welches die treibende Kraft für Sensornetzwerke war („Distributed Sensor Network“ von DARPA 1980), findet für diese große Verwendung.[14] (S. 3-8)

1.1 Ziel und Struktur der Arbeit

Die Arbeit ist in einen theoretischen und einen praktischen Teil gegliedert, wobei sich der theoretische Teil allgemein mit ausgewählten Technologien und Herausforderungen von Sensoren, mikroelektromechanischen Systemen (MEMS) und Sensornetzwerken beschäftigt und in weiterer Folge gesondert das Baukonstrukt-Monitoring behandelt. Der praktische Teil stellt eine Software-Lösung für ein spezielles Problem eines bereits bestehenden Systems im „Bauplatten-Monitoring“ vor.

1.1.1 Theoretischer Teil

Diese Arbeit befasst sich im Allgemeinen mit Sensoren und MEMS sowie deren Anwendung in Sensornetzwerken. Um die Funktionsweise der Sensornetzwerke besser verstehen

zu können, gehen wir im Kapitel MEMS - Mikroelektromechanische Systeme zuerst auf die wichtigsten Grundbestandteile ein.
Bei der Implementierung von Sensornetzwerken stößt man meistens auf klassische Herausforderungen, denen man auch in anderen Typen von Netzwerken begegnet. Dazu gehören Hürden wie das Power-Management oder die Zeitsynchronisation. Auch die Wahl der Architektur und des Betriebssystems für die digitale Verarbeitung spielen eine wichtige Rolle. Auf diese genannten Punkte wird im Kapitel Drahtlose Sensornetzwerke, unter Berücksichtigung des derzeitigen Standes der Technik, näher eingegangen.
Im Speziellen wird dann im Kapitel Structural Health Monitoring (SHM) auf Anwendungsmöglichkeiten und Technologien der Smartdust-Systeme im Baubereich abgezielt. Jedes Baukonstrukt ist mit einem gewissen Risiko verbunden, Schaden durch Naturkatastrophen, gezielten Anschlägen oder Unfällen davonzutragen. Sensornetzwerke sollen dabei helfen, Bauten zu überwachen, um damit einerseits Schaden zu vermeiden und andererseits schnell auf eingetretenen Schaden reagieren zu können. Optische Glasfasersensoren werden den klassischen elektronischen Sensoren gegenübergestellt. Verschiedene Methoden zur Schadenserkennung (Beschleunigungsmesser, Dehnungsmessstreifen, Eigenfrequenzen, Schwingungsformen) werden ebenfalls beschrieben. Außerdem wird ein „state-of-the-art“-Prototyp und eine im Einsatz befindliche Komplettlösungen im SHM-Bereich vorgestellt.

1.1.2 Praktischer Teil - Problemlösung

Im praktischen Teil der Arbeit geht es um eine Erweiterung eines bereits vorhandenen Testsystems. Das Institut für Bauingenieurwesen an der TU Wien stellt diverse Bauplatten her, welche, bevor sie zum Einsatz kommen, getestet werden müssen. Eine Art von Test besteht darin, die Bauplatten stichprobenartig mit Sensoren, welche die Temperatur und Luftfeuchtigkeit messen, auszustatten und alle paar Stunden mit Hilfe des bestehenden Skripts die gemessenen Werte abzulesen und in einer Datei abzuspeichern.
Aufgrund der unbefriedigenden Übersicht der Messwerte gab es den Wunsch einer grafischen Veranschaulichung der Messreihe. Die Hauptanforderung an die Software-Lösung ist die Abbildung einer physischen Bauplatte mit den positionierten Messsensoren in Form eines 2D-Modells. Das Modell soll die Temperatur-Messwerte farblich darstellen und dabei konform mit der physischen Bauplatte sein. Das heißt, dass die Positionen der Sensoren auf der Bauplatte mit den Positionen im Modell übereinstimmen müssen. Die Feuchtigkeitsmesswerte werden als Prozentzahl in der Mitte des modellierten Messbereichs angezeigt. Die Implementierung soll eine Übersicht über die Messdaten liefern. Im Kapitel Implementierung wird das bestehende System näher erläutert und die neue Lösung vorgestellt.

KAPITEL 2

MEMS - Mikroelektromechanische Systeme

Mit fortschreitender Entwicklung im Bereich der Mikro- und Nanotechnologie hat auch das Potential an mikroelektromechanischen Systemen (MEMS) in verschiedenen Gebieten zugenommen. Weltraum-, Militär- aber auch industrielle Systeme profitieren von dieser Technologie. Desweiteren sind MEMS auch in Geräten zur medizinischen Diagnose, Satellitenkommunkationssystemen, Mobiltelefonen, Luftüberwachungs- und Grenzüberwachungssystemen im Einsatz. [15] Nicht zu vergessen sind auch Bausicherheitssysteme, in denen MEMS eine große Rolle spielen. Letzteren werden wir uns später widmen.

2.1 Was ist ein Sensor?

Um den Begriff MEMS definieren zu können, ist es zuerst notwendig, den Begriff „Sensor“ zu erklären. Ein Sensor ist eine Komponente, welche einen analogen oder digitalen Output als Antwort auf ein physikalisches, chemisches oder biologisches Phänomen liefert. Dabei gibt es mehrere Operationen, welche auf den Input ausgeführt werden. U.a. kann die Eingabe verstärkt, kompensiert, gefiltert oder normalisiert werden. Eine Sensor kann auch als ein Wandler (engl. Transducer), welcher oben beschriebene Phänomene in ein bestimmtes Signal (elektrisches, optisches, ...) konvertiert, gesehen werden. Dabei sollte man zwischen dem Begriff „Sensor“ und „Sensorgerät“ (engl. Sensor device) unterscheiden. Oft wird ersteres als Synonym für letzteres verwendet.
In der Praxis kann man ein Sensorgerät auf folgende Bestandteile herunterbrechen:

1. sensibles Element (meistens ein eingeschweißtes Membran (engl. welded Diaphragm))

2. primärer Messumformer (engl. Transducer), welcher ein sensibles Element oder eine Gruppe von sensiblen Elementen beinhaltet

3. Transducer, welcher aus mehreren separaten in Serie geschaltenen Transducern besteht (z.B. ein primärer Transducer und ein Verstärker)

4. isolierte Einheit in einem Spezialgehäuse, welche eines oder mehrere der Komponenten (1)-(3) in einer beliebigen Kombination beinhaltet

5. weitere isolierte Einheiten mit zusätzlichen Transducern (Verstärkern, AD-Wandler, Mikroprozessor, Mikrocontroller, usw)

[16] (S. 5-6), [17], [18], [16] (S. 27)

2.1.1 Funktionsweise

Es gibt kein allgemeines Messprinzip, welches alle möglichen Sensoren anwenden. Viele Prinzipien bedienen sich jedoch der Spannungserzeugung, welche gemessen werden kann. Abhängig von den Bauelementen eines Sensors gibt es mehrere Möglichkeiten, ein eingehendes Signal umzuwandeln. Sensoren mit elastischen Elementen können einerseits die Variation im Widerstand, in der Kapazität oder in der Induktivität nutzen, um das Inputsignal zu transformieren und eine resultierende Spannung zu bekommen. Weiters sind auch Umwandlungen unter Verwendung des piezoelektrischen Effekts oder mithilfe von Lichtdetektoren möglich. Unterschiedliche Temperaturen auf einem Leiter erzeugen ebenfalls eine messbare Spannung. Folgende Punkte sollen ein paar dieser Prinzipien erklären.

Messung anhand des Widerstands

Abbildung 2.1 zeigt ein Sensorgerät mit einer Zelle als sensibles Element, einem Potentiometer und einem Kontaktfinger, welcher mit einer Zelle (könnte auch eine eingeschweißte Membran sein) und dem Potentiometer verbunden ist. Wird Druck auf die Zelle ausgeübt, so verschiebt sich der Kontaktfinger und es entsteht eine messbare Spannung. Für ein ungeladenes Potentiometer mit Gesamtwiderstand R_n, Quellspannung V_s, Spannung zwischen dem Kontaktfinger und eines seiner Enden V_m und der Spannung zwischen dem Kotanktfinger und dem Ende des Potentiometers $R(x)$ gilt:

$$V_m = \frac{V_s \times R(x)}{R_n}$$

Gibt es ein Verhältnis zwischen

- dem Druck p, welcher gemessen werden soll und der Deformation des sensiblen Elementes;
- der Deformation des sensiblen Elementes und der Verschiebung x des Kontaktfingers;
- der Verschiebung des Kontakfingers und dem Widerstand $R(x)$;

so kann man sagen, dass

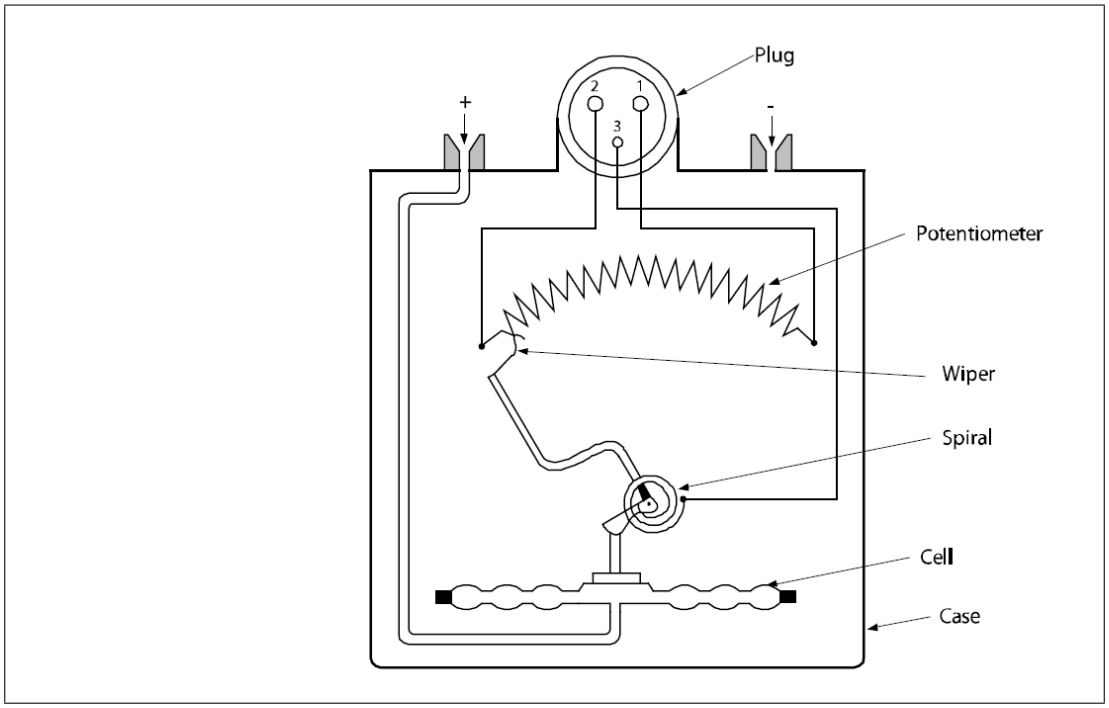


Abbildung 2.1: Potentiometer [1] (S. 14)

$$V_m = k \times V_s \times p$$

gilt, wobei k die charakteristische Konstante vom Sensorgerät ist. Drucksensoren verwenden u.a. dieses Prinzip. [1]

Messung mithilfe von Lichtdetektoren

Lichtdetektoren reagieren je nach Typ verschiedenartig auf das eintretende Licht. Ein **Photowiderstand** ist ein Lichtdetektor, dessen Widerstand sich beim Eintreten des Lichts ändert. Analysen haben ergeben, dass ein einzelnes Photon um die 900 Elektronen für das Leiten freilässt, weshalb ein Photowiderstand als ein Photomultiplier gesehen werden kann und deswegen ein sehr sensibles Gerät ist.
Photodioden sind eine weitere Art von Lichtdetektoren. Wird deren pn-Übergang in Vorwärtsrichtung betrieben, erhöht sich der Strom gering im Vergleich zum Dunkelstrom. Wird der pn-Übergang umgekehrt vorgespannt, so ist die Erhöhung des Stroms durchaus bemerkbar. Aufeinandertreffende Photonen erzeugen Defektelektronenpaare in der P^+-Region. Dementsprechend fließen die erzeugten Löcher zum negativen Terminal, was zu bedeuten hat, dass ein Photoelektronenstrom fließt. Somit ist die Umwandlung von Licht zum elektrischen Strom erfolgt. Diese beiden Prinzipien findet man in manchen optischen Sensoren wieder. [1] (S. 54-55)

Messung anhand der Thermoelektrizität

Wenn ein Streifen oder ein Ring eines homogenen Leiters erhitzt wird, bleibt die Konzentration der freien Elektronen nicht konstant an jeder Stelle des Materials. Die freien Elektronen suchen sich den niedrigsten Energiepunkt und diffundieren zum kühleren Teil. Der wärmere Teil wird im Vergleich zum kälteren Teil positiv geladen. Die dabei entstehende thermale Spannung (Seebeck-Spannung) erzeugt ein elektrisches Feld, welches dem Ungleichgewicht der Ströme entgegenwirkt. Wenn ein Paar aus Leitern (auch Thermoelement genannt), bestehend aus unterschiedlichem Material, verbunden wird, ein Knotenpunkt die Temperatur T1 und der zweite die Temperatur T2 hat, kann ein dazwischengeschaltenes Voltmeter elektromotorische Kraft ablesen. Dies wird der Seebeck-Effekt genannt. Die Spannung hängt also sowohl von den Temperaturen, als auch von den unterschiedlichen Metallen (Unterschied derer Seebeck-Koeffizienten) ab. Für ein Thermoelement gilt folgende Gleichung:

$$E = C_1 \times (T_1 T_2) C \times ((T_1)^2 (T_2)^2)$$

wobei

- E: Gesamtspannung
- T_1,T_2: absolute Temperaturen der Knotenpunkte
- C_1,C_2: Thermoelektrische Materialkonstanten

sind. [1] (S. 363-364)

2.1.2 Akustische Wellen-Sensoren

Akustische Wellen-Sensoren sind für viele aufkommende industrielle Produkte aufgrund ihrer Verlässlichkeit und Sensibilität vom großen Nutzen. Am häufigsten wird in der Telekommunikationsindustrie von ihnen Gebrauch gemacht. Eingesetzt werden diese Sensoren in Bandpassfiltern in Telefonen und Basisstationen. Außerdem wird der wachsende Markt für akustische Wellen-Sensoren auch durch die Verwendung dieser in Fahrzeugapplikationen, medizinischen Applikationen, weiters auch in industriellen und kommerziellen Applikationen, für Feuchtigkeits-, Temperatur und Dampfmessungen, begründet.
Der Begriff „akustische Wellen“ ist hier etwas irreführend, weil die Wellen, welche in dem Sensor erzeugt werden, sich nicht auf die Klang- oder Tonerzeugung beziehen, obwohl der Begriff „Akustik“ dies impliziert. Diese Wellen beziehen sich auf Belastungen oder Deformierungen des Mediums, über welches sich diese ausbreiten.
Für die Erzeugung akustischer Wellen wird meistens vom piezoelektrischen Effekt Gebrauch gemacht. Jedoch können akustische Wellen-Sensoren generell in drei Gruppen eingeteilt werden, wobei der piezoelektrische Mechanismus nicht bei jeder Gruppe von Vorteil sein muss.

1. Sensoren, bei welchen akustische Wellen, die sich über ein elastisches Medium ausbreiten, den Bereich der Messgröße bestimmen. Beispiele hierfür sind Beschleuni-

gungsmesser, Mikrofone und Tonabnehmer. Der piezoelektrische Effekt ist in dieser Gruppe nicht zwingend, obwohl er doch oft eingesetzt wird.

2. Sensoren, welche akustische Wellen in einem Medium, das sie umgibt, über eine Distanz (typischerweise länger als einige Wellenlängen) senden und empfangen, um die Eigenschaften des Mediums zu erkennen. Ein Beispiel hierfür sind Ultraschall-Transducer für akustische Inspektionen, Monitoring und Abbildungen von Luft, festem Material oder Flüssigkeiten. Meistens wird der piezoelektrische Mechanismus in dieser Gruppe von Geräten wegen seiner Effizienz und Umkehrbarkeit eingesetzt, jedoch auch nicht immer zwingend.

3. Sensoren, welche selber ein akustisches oder elektro-akustisches Feld besitzen, über welches man den Grad der Interaktion der akustischen Schwingungen mit dem umgebenden Medium oder die Eigenschaften des Mediums messen kann. Für diese Art von Sensoren ist der piezoelektrische Effekt von großer Bedeutung. Abhängig von der Konfiguration des Sensors kann dieser eine große Anzahl physikalischer Messgrößen messen, wie z.B. Kraft, Druck, Temperatur, Dichte und Zähflüssigkeit von umgebenden Flüssigkeiten.

[19][20]

2.1.3 Biomedizinische Sensoren

Biosensoren sind analytische Geräte, welche aus einem biologischen molekularen Erkennungselement und einem in engen Kontakt stehenden Transducer bestehen. Die Kombination aus Erkennungselementen wie Enzyme, Nukleinsäuren, Abwehrstoffe, Gewebe oder sogar ganze Zellen und elektromechanischen, optischen oder thermometrischen Transducern ermöglicht eine Vielzahl neuer Methoden in der Biomedizin. [21] U.a. werden biomedizinische Sensoren in der Diagnostik, Arzneimittellieferung, Neuralprothetik, Gewebezüchtung und in der minimalinvasiven Chirurgie eingesetzt. Um der ewig steigenden Nachfrage nach hochqualitativer medizinischer Versorgung entgegenzukommen, werden diese Sensoren als gute Lösung empfunden. Bewährte biomedizinische Systeme wie das EKG- oder Augeninnendruck-Monitoring, aber auch neurale Recording-Systeme kommen ohne diese Mikromessgeräte nicht aus. Biomedizinische Sensorgeräte haben in diesen Systemen die Funktion, Daten zu sammeln und diese weiterzuleiten. Weiters ist es auch möglich, sie als Implantate sowohl bei Tieren als auch bei Menschen zu verwenden. Mithilfe einer Spritznadel können extrem miniaturisierte Systeme in Patienten injiziert werden, um so die Gesundheitsüberwachung besser betreiben zu können. Diese können daher manchmal auch lebensrettend sein. Sie bilden außerdem die Basis für Hauspflege oder medizinische Fernversorgung. [2] (S. 1-2), [22]

Beispiel

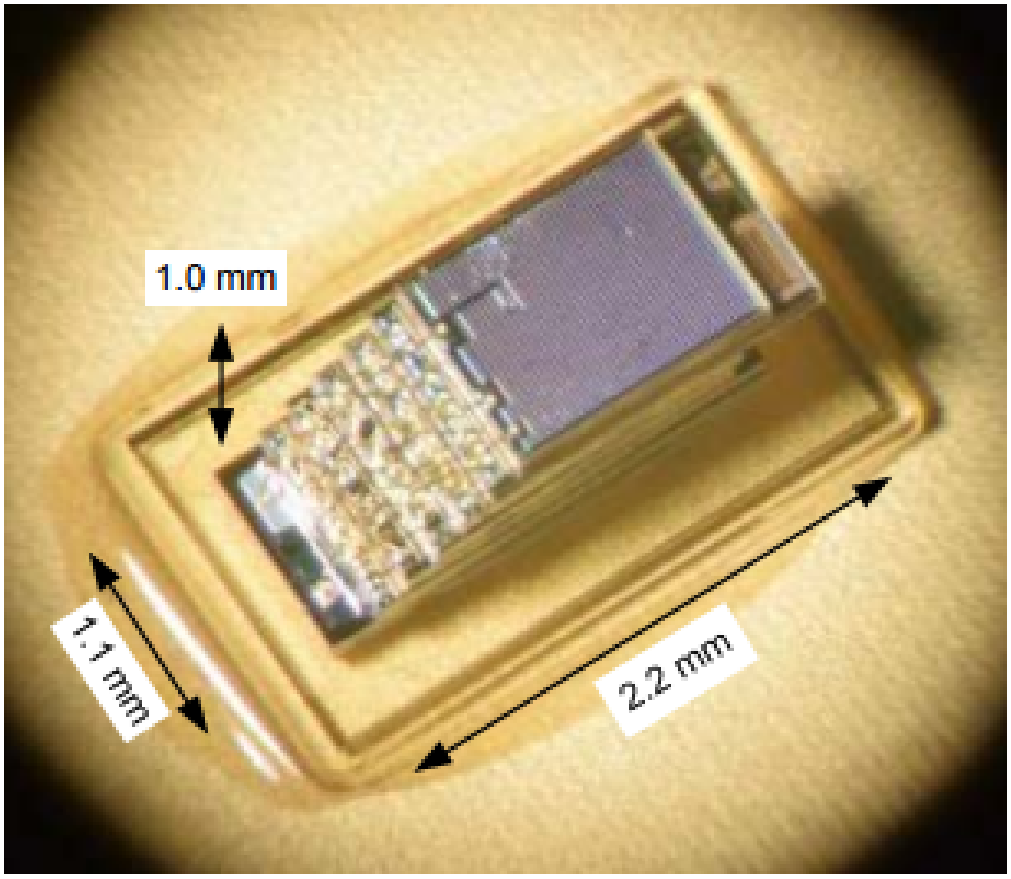
Abbildung 2.2 zeigt ein mehrschichtiges biomedizinisches Sensorgerät, in welchem ein „Switched Capacitor Energy Harvester“ integriert ist. Der „Switched-Capacitor Energy Harvester“ [23] verbessert die Energieeffizienz, indem er den Overhead bei der Takterzeugung und Pegelumsetzung entfernt, um den Schaltkondensator zu betreiben. Gezeigt wird eine aus mehreren Schichten bestehende „Sensing Platform“. Die Kommunikation zwischen den Schichten funktioniert über ein kabelgebundenes Bussystem. Dieses ermöglicht es einzelne Schichten leicht auszutauschen, hinzuzufügen oder zu entfernen. Dies ermöglicht wiederum ein anwendungsspezifisches biomedizinisches Sensorgerät.
Typisch in diesem Stack ist auch der „Capacitive-to-Digital Converter“ (CDC). Kapazitive Sensoren sind in Schwachstromapplikationen weit verbreitet, da für die Signalauslesung ein Ruhestrom benötigt wird, der Null ist. [2]

2.1.4 Glasfasersensoren

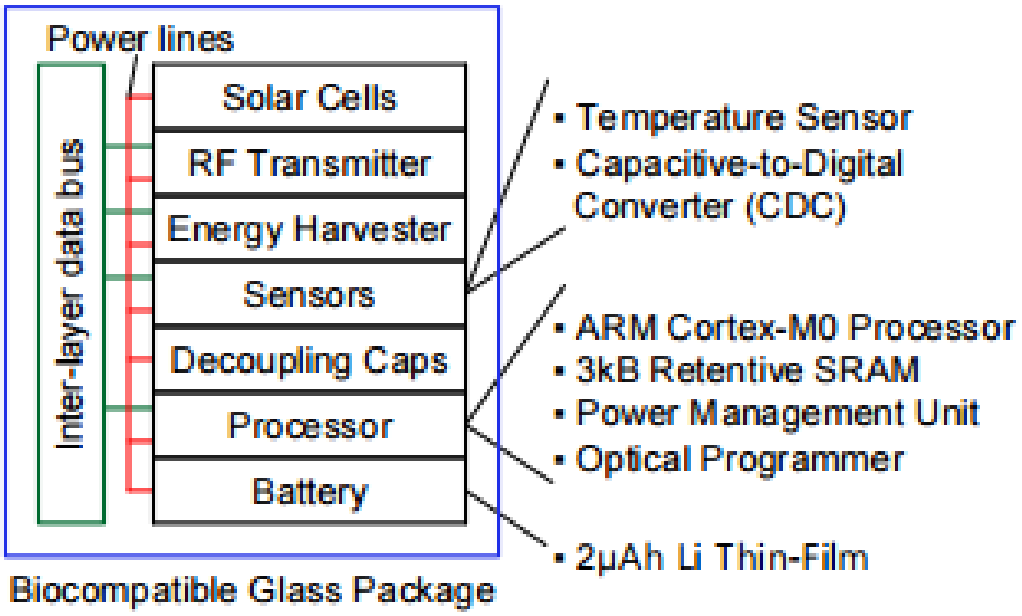
Glasfasersensoren werden meistens grob in zwei Klassen eingeteilt - die extrinischen (oder hybrid) und intrinischen (oder „all-fiber“) Sensoren.
In der ersteren Klasse wird eine Glasfaser durch eine Blackbox geführt. Durch diese Glasfaser führt ein Lichtstrahl, auf welchen Informationen aufgeprägt werden, welche die Blackbox durch Umgebungsveränderungen aufnimmt. Die Informationsaufprägung kann in Form von Veränderung der Phase, der Helligkeit, der Amplitude, der Frequenz des Lichtstrahls oder mit anderen Methoden (z.B. Polarisierung) erzielt werden. Eine weitere Glasfaser leitet den modifizierten Lichtstrahl schließlich zu einem elektronischen und/oder optischen Prozessor.
Im Gegensatz zu den extrinischen Sensoren verwenden intrinische Sensoren keine Blackbox oder Messregion. Stattdessen werden die Informationen aus der Umgebung auf den Lichtstrahl aufgetragen, während dieser sich in der Glasfaser befindet. Bei All-Fiber Sensoren findet also die Messung in der Glasfaser selber statt.[3] (S. 1-10), [24] (S. 3, 135)

Beispiel

Beide Glasfasersensorklassen haben sehr viele Unterklassen. Eine Unterkategorie bildet der oft einfach aufgebaute Hybridsensor mit **Helligkeitsmodulation**. Abbildung 2.3 zeigt solch einen Sensor mit sich zwei gegenüberstehenden Glasfasern. Licht wird durch eine Glasfaser geschickt und tritt gebrochen wieder aus. Der Brechungswinkel ist von der Differenz des Brechungsindex, zwischen dem Mantel und dem Kern der Glasfaser, abhängig. Die zweite Glasfaser nimmt das Licht anschließend auf. Wie viel Licht aufgenommen wird, hängt von dem Akzeptanz-Winkel und vorallem von der Distanz d zwischen den beiden Lichtfasern ab. Je näher sie sich sind, desto intensiver ist das eintretende Licht in der zweiten Glasfaser. Die Helligkeit kann also durch die Veränderung von d moduliert werden.
In der Praxis wird beispielsweise ein flexibler Spiegel im Sensor angebracht, um die Distanz zwischen den beiden Glasfasern zu regulieren. Wird Druck auf den Spiegel ausgeübt,



(a)



(b)

Abbildung 2.2: Multilayered Sensing System Platform [2] (S. 2)

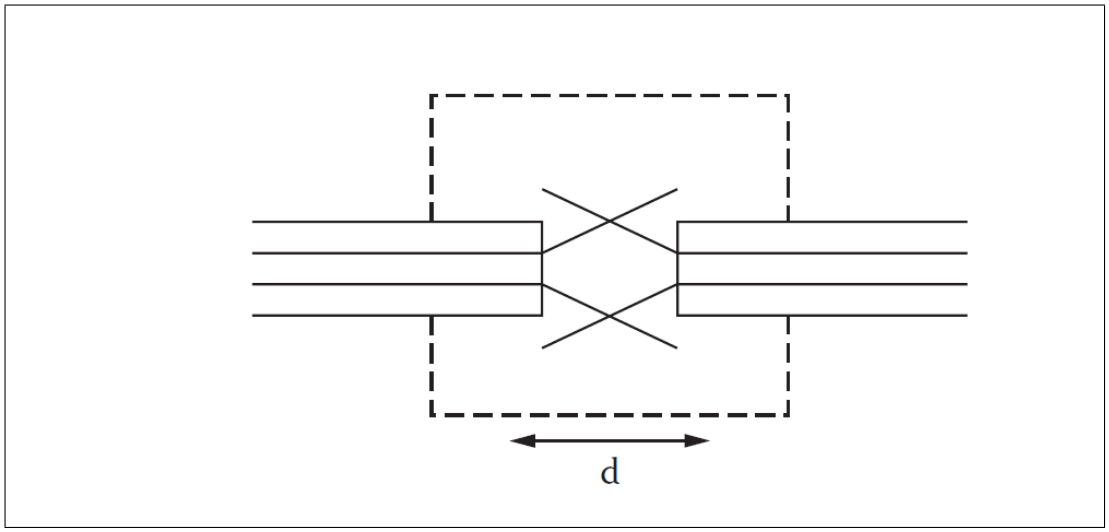


Abbildung 2.3: Hybridsensor mit Helligkeitsmodulation: Theoriemodell [3] (S. 3)

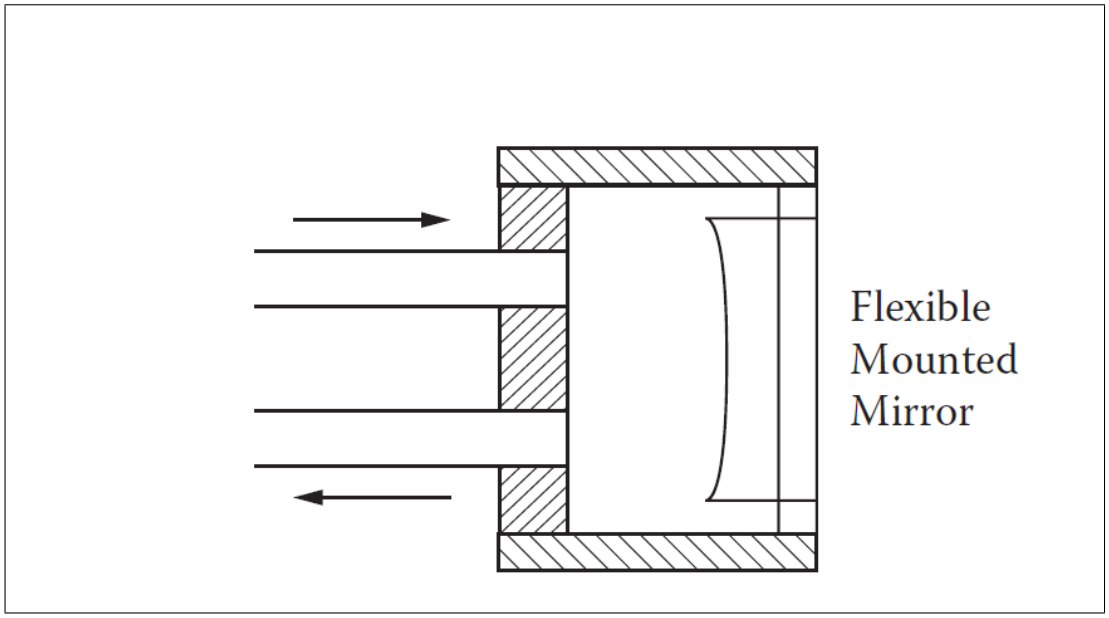


Abbildung 2.4: Hybridsensor mit Helligkeitsmodulation: Praxismodell [3] (S. 4)

so verschiebt sich dieser und damit auch die Distanz, welche das Licht vom Austritt aus der ersten Glasfaser bis zum Eintritt in die zweite Glasfaser zurücklegen muss. Abbildung 2.4 zeigt dieses Geschehen.

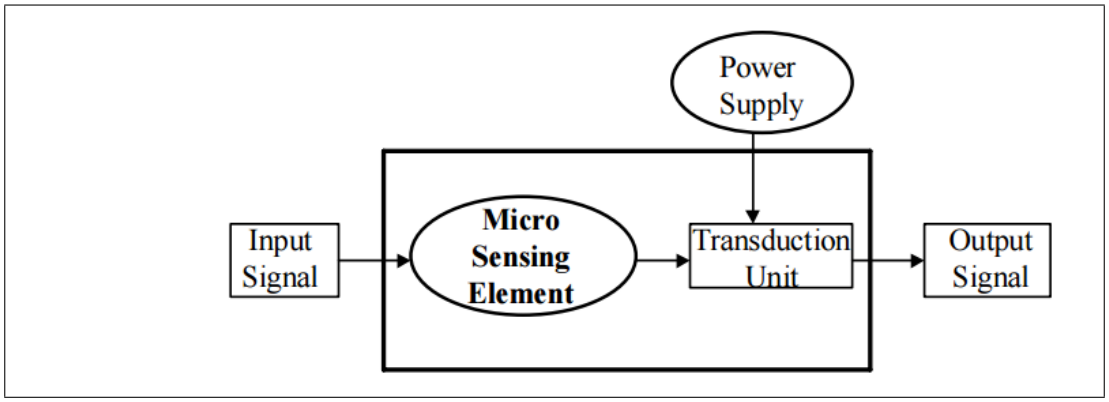


Abbildung 2.5: MEMS as a Sensor [4] (S. 2)

2.2 Was ist ein MEMS?

MEMS ist die Integrierung von Mikrokomponenten auf einem einzelnen Chip, welcher es dem Mikrosystem ermöglicht, seine Umgebung wahrzunehmen (bzw. physikalische Größen zu erkennen) und zu kontrollieren. Diese Komponenten beinhalten typischerweise mikroelektronische integrierte Schaltkreise (engl. Integrated Circuits - ICs - das „Hirn“ vom System), **Sensoren** (Erkennungs- und Aufspürungsfunktion - „das Nervensystem“) und Aktuatoren (die „Hände“ und „Arme“). Mikrofabrizierungstechnologien - ähnlich wie bei den ICs - werden verwendet, um die Komponenten auf dem Chip zu integrieren. [25] Manche MEMS bestehen dabei aus beweglichen Teilen, ebenfalls im Mikrometer- oder gar Nanometerbereich. Andere MEMS haben gar keine beweglichen Teile und können aus Plastik, Glas oder aus dielektrischem Material bestehen. Die oben erwähnten Komponenten können in der Praxis speziell folgende sein:

- Mikrosensoren (akustische Wellen-Sensoren, biomedizinische Sensoren, chemische Sensoren, Trägheitssensoren, optische Sensoren, Drucksensoren, Thermalsensoren, usw.)
- Mikroaktuatoren (Ventile, Pumpen und Mikrofluide; elektrische und optische Relais und Switches; Zangen und Pinzetten)
- Mikroelektronik (ICs, Kondensatoren, Induktoren)
- resonante Mikrostrukturen (Cantilever, Membrane, ...)

Abbildung 2.5 zeigt ein MEMS als einen Mikrosensor und die funktionale Beziehung zwischen dem Sensor und der Transduktionseinheit. Das eingehende Signal wird von der sensiblen Einheit erfasst und zur Transduktionseinheit weitergeleitet. Diese wandelt schließlich das eingehende Signal in das gewünschte (je nach Transducer) Output-Signal um. Der Transducer wird dabei mit Strom versorgt (siehe [15] S. 3, [4] (S. 1-3), [26]). In diesem Fall sieht man, wie sich die beiden Begriffe „MEMS“ und „Mikrosensorgerät“ überlappen.

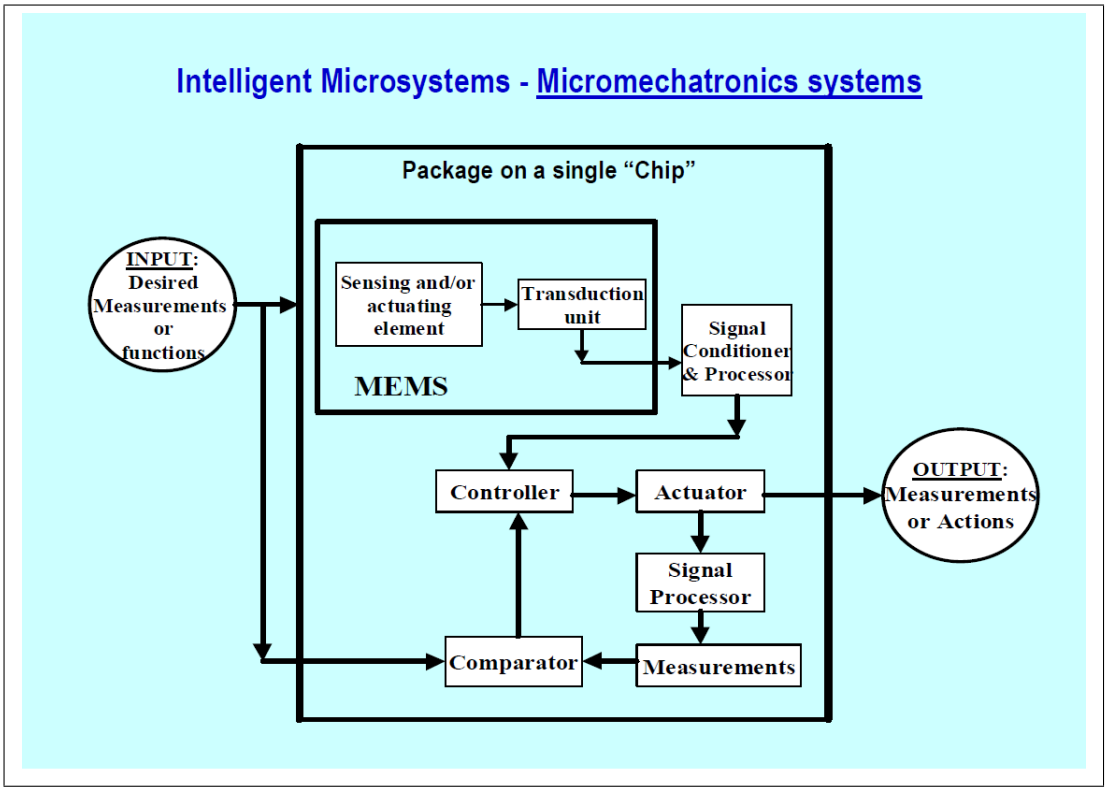


Abbildung 2.6: Intelligent Microsystems [4] (S. 5)

In der deutschen Literatur wird der Begriff „Mikrosystem“ als Synonym für MEMS verwendet. Jedoch klingt ersterer Begriff viel mehr wie ein Überbegriff vom letzteren. Tai-Ran Hsu unterscheidet in seinem Buch (siehe [4] S. 5) zwischen den beiden Begriffen und stellt MEMS als eine integrierte Einheit eines Mikrosystems dar, wie man in Abbildung 2.6 erkennen kann. Betrachtet man diese Abbildung etwas näher, so sieht man, dass die eingezeichneten Elemente der Definition eines MEMS entsprechen. Ein Mikrosensor, ein Aktuator und sämtliche Mikroelektroniken (Controller, Signal Processor, Comparator) sind vorhanden. Das integrierte MEMS selber enthält hier aber keine Prozesseinheit, d.h., kein „Hirn“, wie weiter oben definiert.

2.2.1 Vorteile von Miniaturisierung

Besonders im 21. Jhd. haben Computersysteme und -technologien einen Aufschwung erlebt. Die treibende Kraft dafür waren immer kleiner werdende Komponeten. Minituarisierung ist die Kunst Objekte zu verkleinern jedoch dabei seine Charakteristiken zu erhalten oder gar zu verbessern. Aus der technischen Perspektive sind die daraus resultierenden Vorteile offensichtlich:

- Kleinere Systeme bewegen sich aufgrund ihrer kleineren Masse und damit auch

geringeren mechanischen Trägheit schneller als größere Systeme.

- Kleinere Systeme reagieren auch viel schneller auf thermale Bedingungen wegen ihrer geringeren thermischen Trägheit. Der Wärmeverlust bleibt auch gering.
- Es treten weniger Probleme im Bezug auf Vibrationen auf, weil die Vibrationen eines Systems umgekehrt proportional zu seiner Masse sind. Kleinere System haben viel geringere Eigenfrequenzen.
- Miniaturisierte Systeme verbrauchen viel weniger Platz und sind deswegen auch in der Medizin und Arztpraxis von Nutzen, weil bestimmte Geräte Mikrosysteme erfordern.
- Aufgrund ihrer Präzision werden Mikrosysteme auch gerne in Satelliten und der Weltraumtechnik eingesetzt. Auch in der Telekommunikation spielt Genauigkeit eine Rolle, weshalb auch in dieser Industrie immer kleiner werdende Systeme von Bedeutung sind

[4] (S. 15-16)

2.3 Unterarten von MEMS

Aufgrund der unzähligen Einsatzgebiete haben sich auch spezielle Unterarten dieser Mikrosysteme heraus entwickelt.
RF-MEMS (Radio frequency MEMS) nutzen Radiowellen im Submillimeterbereich und elektromechanische Aktoren (z.B. scratch drive actuator oder auch „SDA“), um überragende aktive und passive RF-Geräte wie Switches, Phasenschieber oder auch Kapazitätsdioden zu ermöglichen. [15] (S. 2), [27]
IR-MEMS (Infrarot MEMS) arbeiten mit Infrarotsensoren, welche abhängig vom Typ (Quantum- oder Thermalsensoren) im Zuge der Infrarotstrahlenmessung Photonen zu elektrischen Trägern umleiten oder diese absorbieren. [28]
Eine weitere Unterart bilden die MOEMS - optische MEMS - welche eine Fusion aus Halbleiter-Mikrobearbeitung und optischen Elementen sind. Diese spielen eine große Rolle beim Design von optischen Lautstärkereglern und anderen optischen Komponenten sowie auch „lab on a chip“-Konzepten in der Biomedizin. [25], [15] (S. 4)
Speziell für biomedizinische Anwendungen wurden sogenannte BioMEMS entworfen. U.a. helfen BioMEMS dabei Aufgaben im Bereich der Genetik (DNA Mikrofelder), Zellbiologie (cytoskelettale Veränderungen erkennen), Biochemie (Oberflächenplasmonenresonanz), Neurowissenschaften (mikroelektrische Felder) und Physiologie (implantierbare Sensoren) leichter zu lösen. [29]

2.3.1 Radio frequency MEMS

Der Begriff RF-MEMS bezieht sich auf das Design und die Produktion von MEMS für ICs im Radiowellenbereich. MEMS-Geräte in RF-MEMS werden dabei für die Ingangsetzung von separaten RF-Komponenten wie Drehkondensatoren, Switches oder Filtern

verwendet. Vijay K. Varadan *et al.* unterscheiden dabei zwischen drei verschiedenen Architekturkategorien:

1. RF extrinisch: Die MEMS-Struktur befindet sich außerhalb des RF-Schaltkreises und steuert dabei andere Komponenten im RF-Schaltkreis (z.B. eine einstellbare Mikrostreifenübertragungsleitung)

2. RF intrinisch: Die MEMS-Struktur befindet sich innerhalb des RF-Schaltkreises und übernimmt sowohl die Steuerungs- als auch die Schaltkreisfunktion (z.B. MEMS-Cantilever als elektrostatischer Mikro-Switch)

3. RF reaktiv: Die MEMS-Struktur befindet sich innerhalb des RF-Schaltkreises und besitzt eine dämpfende RF-Funktion (z.B. kapazitiv gekoppelte einstellbare Filter und Resonatoren)

[30] (S. 1-2)

Die wahrscheinlich am meisten untersuchte RF-MEMS-Komponente ist der RF-Switch (= Schalter). Einen Switch kann man einfach als ein Gerät, welches einen Stromkreis unterbricht oder schließt, definieren. Dies mag einfach klingen, jedoch kann ein Schalter in einem RF-Signalweg z.B. zum Übersprechen führen. Der erste Ansatz von Petersen (1979) Silikon basierende Mikrofabrikation mit der Herangehensweise, welche man in mechanischen Relais vorfindet, zu verschmelzen, konnte außerdem höchstens auf einer Frequenz von 200kHz operieren. Erst 12 Jahre später stellte Larson einen Switch vor, welcher es bis auf 45 GHz schaffte. Ein Switch hat weiters aufgrund seiner mechanischen Operationen auch eine begrenzte Lebenszeit. Beispielsweise werden in einem Relais die Kontaktpunkte mit der Zeit abgenutzt. [30] (S. 109)
Eine typische Anwendung eines RF-Schalters ist z.B. eine Antenne, bei der zwischen Sender und Empfänger geschalten wird. In Kommunikationssystemen werden Switches für die digitale Modulation verwendet. Ein eingehendes Signal wird vom Switch, welcher als ein Gate dient, entweder gestoppt oder durchgelassen, um als Output eine bestimmte Signalwellenform zu erhalten (siehe [30] S. 109). In der Telekommunikation werden RF-Schalter meistens für das Signal-routing und für die Steuerung des Verstärkungsgrades in Verstärkern verwendet. Je nach Frequenzbereich, Energieverbrauch und Geschwindigket werden andere Technologien für Switches verwendet. Beispielsweise leisten elektromechanische Switches wie Relais gute Arbeit bei niedrigen Frequenzen und in „High Power“-Anwendungen. Sie schalten aber nicht so schnell wie elektronische solid-state Switches (z.B. Pin-Dioden, GaAs FETs, ...). Diese sind wiederum schlecht in Sachen Power-Handling. Eine Technologie, welche die Vorteile sowohl von elektromechanischen als auch von solid-state Switches nutzt, basiert auf dem Prinzip der MEMS. [30] (115-125)
MEMS-Switches umfassen einerseits einen Antriebsabschnitt (engl. actuation) und ein elektrisches Element, welches über das Antriebsschema (elektrostatisch, magnetostatisch, piezoelektrisch oder thermal), die geometrische Konfiguration (vertikaler/horizontaler

Antrieb, Cantilever, Membran, usw.) oder die elektrische Konfiguration (ohm'scher Kontakt oder Kondensatorswitch, serielle oder parallele Schaltung) kategorisiert werden kann. Unabhängig vom Design bieten MEMS-Switches generell einige Vorteile gegenüber Solid-State-Geräten und elektromagnetischen Switches. Einerseits können sie besser in Serien gefertigt werden, aufgrund der hoch repetitiven Nanofabrizierungstechniken. Was viel wichtiger ist und auch zur erhöhten Forschung der RF-MEMS-Switches geführt hat, waren die höhere Isolierung im Sperrzustand für höhere Frequenzen, die geringere Einfügungsdämpfung, ein höheres Energie-Handling und ein kleinerer Energieverbrauch. Letztere Eigenschaft wird bei RF-MEMS am häufigsten mit dem elektrostatischen Ansatz erzielt. [30] (S. 125), [16] (S. 474), [15] (S. 175)

2.3.2 Infrarot MEMS

Infrarotsensoren nutzen die Vorteile der MEMS-Technologie, um ganze Messsysteme auf einen einzelnen Chip zu bringen und deren Masse zu verringern und damit die Performance zu erhöhen. So schafften es Ulas Adiyan *et al. 2015* einen MEMS-Detektor zu produzieren, welcher einen Pixelpitch von 35-μm besitzt, der bis dato kleinste Pixelpitch in der IR-MEMS-Literatur. [31]
Allgemein gibt es zwei Arten von Infrarotsensoren - Quantum- und Thermalsensoren. Quantumsensoren bestehen aus Halbleitermaterialien, welche die IR-Photonen zu elektrischen Trägern weiterleiten. Eine Kühlung der Quantumsensoren ist erforderlich um das thermische Rauschen zu unterdrücken und um die energiearme infrarote Stralung zu messen. Im Vergleich brauchen die Thermalsensoren keine Kühlung. Diese bestehen aus Mikro-/Nano-strukturen und besitzen einen thermisch gut isolierten Detektor (Pixel), welcher die infrarote Strahlung in Hitze umwandelt. Gemessen wir dann thermoelektromotive Kraft, die sich aus den Temperaturunterschieden in den Kontaktpunkten zweier unterschiedlicher Metalle ergibt. Deswegen ist die Empfindlichkeit eines Thermalsensors prinzipiell auch wellenlängenunabhängig.
Die Umwandlung der Temperatur in ein elektrisches Signal hängt nun vom Sensortyp ab. Wird ein thermoelektrischer Wandler eingesetzt, so tritt der bereits erwähnte Seebeck-Effekt in Kraft. Weiters kann die Temperatur auch genutzt werden, um ein elektrisches Signal zu modulieren. Im letzteren Fall spricht man von parametrischen Sensoren. Parameter sind u.a. der elektrische Widerstand (Bolometer) oder der Druck, welche von der Temperatur beeinflusst werden können.
Infrarotsensoren spielen eine große Rolle in der Thermografie. Diese Verfahren findet Anwendung u.a. in Wärmebildkameras, Nachtsichtgeräten und auch in biomedizinsichen Applikationen. Diese Anwendungen verlangen nicht nur eine geringe rauschäquivalente Temperaturdifferenz (engl. NETD, < 250 mK für hohe Sensibilität), sondern auch kostengünstige, energiesparende und leichte Systeme. Ungekühlte IR-Systeme können diese Anforderungen erfüllen. Ein ausschlaggebendes Element ist hierbei der Detektor. Man spricht auch von einem thermomechanischen MEMS-Detektor. Ein mögliches Prinzip, um hohe Genauigkeit bei der Messung zu erzielen, ist die Verwendung eines Torsionsresonators bestehend aus zwei unterschiedlichen Materialien (engl. bimaterial). Bei Absorption von IR-Photonen biegt sich der Resonator, aufgrund der unterschiedlichen Ausdehnungskoeffi-

zienten der Materialien, nach oben. Die Biegung verursacht einen starken Federungseffekt in den Drehstäben, welcher die Resonanzfrequenz erhöht. Weil eine präzise Messung der Frequenzverschiebung möglich ist, kann eine hohe Genauigkeit erzielt werden. Die Verschiebung selber kann mit elektrischen oder optischen Methoden gemessen werden. [28] [32]
Um ein paar Arbeitsvorgänge eines IR-MEMS vorzuzeigen, wird als Beispiel der IR-MEMS-Temperatursensor TMP006 von Texas Instruments gezeigt. Dieser MEMS-Sensor hat die Eigenschaft Temperaturen zwischen $-40°C$ und $125°C$ zu messen, ohne dabei in Kontakt mit dem Messobjekt zu sein. Für die Spannungserzeugung wird ein Thermoelement (Seebeck-Effekt) genutzt. Die Thermosäule auf dem Chip absorbiert passive Infrarotstrahlung zwischen $4\mu m$ und $16\mu m$. Die Spannungsänderung in der Thermosäule wird mit einem on-chip-die über eine I2C und SMBus kompatible Schnittstelle digitalisiert und zu einem externen Prozessor weitergeleitet. Dieser kann dann aus den zur Verfügung gestellten Daten die Objekttemperatur berechnen.
Zur Berechnung werden die beiden Parameter Die-Temperatur (Würfel-Temperatur) T_{Die} und Sensorspannung V_{Sensor} benötigt. Im Idealfall ergibt sich zwischen V_{Sensor} und der Objekttemperatur T_{OBJ} folgendes Verhältnis

$$T_{OBJ} = \sqrt[4]{T_{Die}^4 + \frac{V_{Sensor}}{\epsilon\sigma}}$$

Zur Speicherung von Konfigurationen, Temperaturmessergebnissen und Sensorspannungsmessungen wird ein 16bit Register verwendet. Das Temperaturregister verwendet dabei die Zweierkomplementdarstellung (mit Vorzeichen), mit LSB = $1/32°C = 0.03125$, abgespeichert. Die Die-Temperatur ist eine binäre 14bit-Zahl mit Vorzeichen. Um aus dem binären Wert einen physikalsichen Temperaturwert in °C zu erhalten, müssen die letzten beiden LBSs nach rechts verschoben und dann durch 32 dividiert werden. Folgende Tabelle zeigt drei Beispiele. [33]

TEMPERATURE °C	DIGITAL OUTPUT (BINARY)	SHIFTED HEX
0.03125	0000 0000 0000 0100	0001
25	0000 1100 1000 0000	0320
–0.03125	1111 1111 1111 1100	FFFF

2.3.3 BioMEMS

Obwohl BioMEMS derzeit ein populäres Thema sind, liegen ihre Wurzeln weit zurück. Schon 1958 wurde der erste Herzschrittmacher gebaut. Endoskope wurden bereits im 19. Jhd. eingesetzt. Ein Beispiel für ein aktuelles BioMEMS-Endoskop ist ein selbstständiges Kapselendoskop für die Untersuchung des Magen-Darm-Traktes. Die allerersten Mikrosysteme wurden jedoch für den Gebrauch in der Neurowissenschaft entwickelt, um Körperfunktionalitäten, Stoffwechselstörungen und biologische Prozesse auf zellulärer Ebene zu untersuchen. Bis zur Kommerzialisierung des ersten MEMS-Gerätes hat es aber einige Jahre gedauert.
Am häufigsten eingesetzt werden die jüngsten BioMEMS im klinischen Monitoring - bei

der Messung von Glukose und anderen den Stoffwechsel betreffenden Parametern. Die GlucoWatch, als Beispiel, stellt eine gute Ergänzung zu üblichen Blutzuckermessgeräten mit Lanzette dar. Glukose wird dabei beim Tragen dieses Gerätes von der anliegenden Haut entnommen (Umgekehrte Iontophorese). Zwei Sets von Biosensoren und Iontophorese-Elektroden sind dabei im Einsatz. Im Vergleich zu elektrophysiologischen Parametern (z.B. EKG oder Pulsfrequenz) können metabolische Parameter nur in vivo genauestens gemessen werden.[21]
Für die Glukosemessung gibt es mehrere Methoden. Glukose kann u.a. lichttechnisch, amperometrisch oder mithilfe einer elektrischen Mikrowaage gemessen werden. Weiters werden auch die PH-sensible Methode und die Methode der molekulären Prägung (MIP) verwendet. Als beste Methode hat sich die MIP-Methode herauskristallisiert, da die anderen Methoden viele Nachteile aufweisen. Beispielsweise sind lichttechnische Sensoren mit Power-Management-Problemen und Signalverlusten verbunden, da das MEMS vom Gewebe umschlossen wird. Amperometrische MEMS haben eine schwache Sensibilität aufgrund ihrer geringen Stromausgabe und lange Antwortzeiten. PH-sensible ISFET-Sensoren weisen aufgrund der geringen Dissoziationskonstante ebenfalls eine geringe Empfindlichkeit auf. Die MIP-Methode ist eine sehr mächtige Methode für Glukosemessung. Die MIP-Mikrosensoren bestehen aus einer Reaktionskammer, Mikroelektroden und Mikronadeln. Die Hauptvorteile dieser Methode sind die Linearität, die hohe Empfindlichkeit und schnelle Antwortzeiten im Vergleich zu den anderen. [34]
Ein immer wieder aufkommender Begriff, wenn man von MEMS in der Medizin, Biomedizin und Biochemie spricht, ist das Konzept des Chiplabors (engl. lab on a chip). Chiplabors sind MEMS, welche die Automatisierung von typischen Prozeduren aus einem Laboratorium heraus ermöglichen. Man spricht dabei auch von mikrofluidischen Systemen, da sich geringste Mengen einer Flüssigkeit auf einem einzelnen Chip analysieren lassen. Chiplabore bieten viele Vorteile gegenüber konventionellen Labors wie bessere Performance, Portabilität, Verlässlichkeit und geringere Kosten. Die meisten kommerziellen mikrofluidischen Systeme werden mit der CMOS-Technologie umgesetzt. [35][36][37]
Chiplabors bestehen aus drei Hauptbestandteilen; Sensoren, Aktuatoren und Mikroelektronik (siehe MEMS). Die Sensoren sind für die Messung von elektrischen, optischen magnetischen oder thermischen Eigenschaften der Probe zuständig. Um die bio-elektrischen Aktivitäten von biologische Zellen nicht-invasiv zu messen, können diese mit Mikroelektroden kapazitiv gekoppelt werden. Um die Verbreitung von elektrischen Signalen zu beobachten und eine interzelluläre Kommunikation zu ermöglichen, ist die Implementierung einer Reihe von Mikroelektroden und Multiplexschaltungen notwendig. Die Aktuatoren erzeugen mithilfe von elektromagnetischen Feldern entweder mechanische oder elektrische Kraft, um Objekte wie DNA oder biologische Zellen in der Flüssigkeit zu steuern. Die Elektronik kümmert sich um die Verstärkung und Aufbereitung des Output-Signals, sowie die Störsignalreduktion. Sie steht in Verbindung mit den Sensoren und Aktuatoren und bietet eine Schnittstelle zur Kommunikation mit den Computern. Diese bedient sich auch der etablierten CMOS-Technologie.[36]

KAPITEL 3

Drahtlose Sensornetzwerke

MEMS, wie wir sie im vorherigen Kapitel besprochen haben, zusammen mit VLSI (Very Large Scale Intergration) und drahtlosen Netzwerken, leisten einen großen Beitrag zu der Verbreitung von verteilten Sensorsystemen. Beispielsweise erlauben es die Entwicklungen in der Halbleitertechnologie immer leistungsstärkere Prozessoren zu produzieren. Die Miniaturisierung von Sensoren treibt deren Kosten und Energieverbrauch weiter herunter. Während die Verteidigungs- und Raumfahrtsysteme den Markt weiterhin dominieren, wird immer mehr Fokus auf Systeme im Bereich Bau-Monitoring gelegt, um zivile Einrichtungen (z.B. Brücken oder Tunnel), nationale Energieversorungsnetze und Pipeline-Infrastrukturen zu schützen. Netzwerke von hunderten von Sensorknoten sind bereits bei der Überwachung großer geographischer Gebiete im Einsatz, um Umweltverschmutzung und Überschwemmungen zu modellieren und vorherzusagen, Informationen über den „Gesundheitszustand“ von Brücken zu sammeln und den Verbrauch von Wasser, Düngemitteln und Pestiziden zu kontrollieren, um eine bessere Ernte zu erhalten. [14]
Ein Sensornetzwork wird dazu eingesetzt, um Ereignisse oder Phänomene aufzuspüren, Daten zu sammeln und diese an Interessenten weiterzuleiten. Basismerkmale eines Sensornetzwerkes sind:

- Fähigkeit zur Selbstorganisation
- Broadcast-Kommunikation und Multihop-Routing
- Kompakter Einsatz und Kooperation der Sensorknoten
- Häufige Topologieänderungen aufgrund von Abnutzung und Knotenausfällen
- Einschränkungen in Lebensdauer, Übertragungsenergie, Rechenleistung und Speicher

Besonders die letzten drei Merkmale machen die Sensornetzwerk von anderen ad-hoc- und vermaschten Netzen unterscheidbar. [38] Diese und weitere Herausforderungen sowie allgemeine Kernpunkte im Zusammenhang mit (Sensor-)Netzwerken (Architektur,

Kommunikationsprotokolle, Zeitsynchronisation, Power-Management, Sicherheit, Betriebssysteme) werden in diesem Kapitel erläutert. Die Begriffe *drahtloses Sensornetzwerk* und *WSN* werden dabei abwechselnd und synonym genutzt.

3.1 Knotenarchitektur

Im vorherigen Kapitel sind wir schon näher auf die Hardware und Interaktion zwischen den Elementen von verschiedenen Sensoren und MEMS eingegangen. MEMS-Sensoren sind ein fixer Bestandteil eines Sensorknotens. Letzterer beinhaltet aber noch weitere Komponenten. In diesem Abschnitt wollen wir die Architektur von Sensorknoten und das Zusammenspiel der verschiedenen Elemente erläutern.

3.1.1 Komponenten eines Knotens

Ein Knoten besteht aus folgenden 4 Subsystemen: Sensor-Subsystem, Prozessor-Subsystem (Speicher- und Recheneinheit), Kommunikations-Subsystem (intern und Radio) und dem Energieversorgungs-Subsystem. Letzters wird im Kapitel Power-Management beschrieben. Ein Knotendesigner hat eine große Anzahl an Möglichkeiten, wie er diese Subsysteme baut und zusammenfügt. Das Sensor-Subsystem beinhaltet MEMS-basierte Sensoren, Aktuatoren und Analog/Digital-Wandler. Es hat die Aufgabe physische Phänomene zu erkennen und diese in digitale Daten umzuwandeln. Die Daten werden über eine Kommunikationsschnittstelle (I^2C, SPI, ...) zum Prozessor-Subsystem weitergeleitet, welches das zentrale Element des Knotens ist. Die Wahl eines Prozessors entscheidet über das Trade-Off zwischen Flexibilität und Effizienz. Zur Auswahl stehen mehrere Optionen für einen Prozessor: Mikrocontroller, Digitaler Signalprozessor (DSP), Anwendungsspezifische integrierte Schaltung (ASIC) und Field Programmable Gate Arrays (FPGA). Das Kommunikations-Subsystem umfasst weitere Prozessoren (Co-Prozessoren) und ein Funkgerät für die Kommunikation nach außen. Dieses Subsystem verbraucht normalerweise die meiste Energie. Deshalb sollte man bei der Implementierung auch eine Regulierung des Energieverbrauchs miteinbeziehen. Das Energieversorgungs-Subsystem stellt für alle anderen Subsysteme Energie bereit. Unter Einsatz eines DC-DC-Wandlers wird sichergestellt, dass jedes Subsystem die richtige Menge an Energie bekommt. [38] (S. 238-240), [14] (S. 47)

3.1.2 Sensor-Subsystem

Ein Sensor-Subsystem ist Teil eines Sensorknotens und besteht aus einem odere mehreren (MEMS-)Sensoren sowie einem oder mehreren Analog/Digital-Wandlern und beinhaltet weiters einen Multiplexmechanismus, um diese zu schalten. Analog/Digital-Wandler spielen hierbei eine wichtige Rolle. Weil der Output von den meisten MEMS-Sensoren analog ist (elektrische Spannung oder Licht), muss dieser Output zuerst digitalisiert werden. Einerseits muss das Signal quantifiziert werden (kontinuierliche Werte wie Zeit und Stärke in diskrete Werte umgewandelt werden). Beispielsweise kann ein 24-Bit A/D-Wandler $2^{24} = 16,777,216$ unterschiedliche diskrete Werte annehmen. Außerdem muss

auch die Samplingrate bestimmt werden. Hier genügt es jedoch nicht, diese allein durch die Nyquist-Grenze (doppelte Signalfrequenz) festzulegen, da aufgrund von Störgeräuschen Überabtastung (=Oversampling) erforderlich ist. Bei der Wahl eines A/D-Wandlers wird empfohlen, über die überwachten Prozesse oder Aktivitäten Bescheid zu wissen. Ein Temperatursensor, der bei einer Breite von $0 - 100°C$ einen Temperaturunterschied von $0.5°C$ messen muss, kommt mit einem 8-bit-Wandler aus ($\frac{1}{0.5} \times 100 = 200$, $2^8 = 256$). Bei einem Unterschied von 0.0625 ist allerdings ein 11-bit-Wandler erforderlich ($\frac{1}{0.0625} \times 100 = 1600$, $2^{11} = 2048$). [14] (S. 48-51)

3.1.3 Prozessor-Subsystem

Das Prozessor-Subsystem verbindet alle anderen Systeme und noch weitere Peripheriegeräte. Dessen Hauptaufgabe ist es, Operationen, welche das Messen, die Kommunikation und Selbstorganisation betreffen, auszuführen. Es besteht u.a. aus einem Prozessor, einem permanenten Speicher für die Befehle und einem aktiven Speicher (Daten werden nur für wenige Sekunden gespeichert) für die gemessenen Daten und einer Clock. Die Prozessorarchitektur kann nach den drei Basisarchitekturen - Von Neumann, Harvard und SHARC - entworfen werden. Bei der Wahl des Prozessors sollte die Komplexität der Aufgaben der Knoten berücksichtigt werden. Hat ein Knoten eine einfache Tätigkeit zu verrichten, welche sich nie ändert, dann sollte man einen DSP oder FPGA wählen. Beide sind im Energieverbrauch sehr effizient, die Implementierung ist jedoch meistens komplex und teuer. DSPs haben im Vergleich zu analogen Signalprozessoren den Vorteil, dass sie weitaus einfachere Komponenten benötigen. Ein DSP ist darauf spezialisiert, hochkomplexe mathematische Operationen mit einer hohen Effizienz auszuführen und Echtzeit-Performance zu bieten. Die meisten kommerziellen DSPs implementieren die Harvard-Architektur. FPGAs sind etwas flexibler als DSPs, da sie paralleles Rechnen unterstützen. Außerdem ist deren Rechengeschwindigkeit vom Anwendungsentwickler einstellbar, was zu mehr Kontrolle führt. Für komplizierte Aufgaben bedient man sich eines Mikrocontrollers. Diese lassen sich leicht programmieren und bieten daher auch mehr Flexibilität. Die Aufgaben eines Knotens können somit leicht modifiziert werden. Hinzu kommt noch, dass Mikrocontroller kostengünstig sind und daher oft bevorzugt werden. Jedoch sind diese nicht so schnell und effizient wie DSPs oder FPGAs. [14] (S. 41-57)

3.1.4 Kommunikations-Subsystem

So wie der richtige Prozessortyp für das Design eines Knotens wichtig ist, so ist auch die Art der Verbindung der Subkomponenten mit dem Prozessor-Subsystem wichtig. Eine schnelle und effiziente Datenübertragung zwischen den Komponenten ist entscheidend für die Gesamteffizienz des Netzwerks. So ist z.B. eine parallele Datenübertragung schneller als eine serielle, eine paralleler Bus benötigt jedoch mehr Platz. Aufgrund der Größe der Knoten werden parallele Busse nicht eingesetzt. Die Auswahl besteht oft zwischen seriellen Schnittstellen wie USB, GPIO (general purpose I/O), SDIO (secure data I/O), I^2C (inter-integrated circuit) und SPI (serial peripheral interface). Letztere zwei sind

jedoch die am häufigsten eingesetzten Schnittstellen.
SPI ist ein vollduplex synchroner serieller Bus, bei dem es nur einen Master (standardmäßig der Mikrocontroller) und mehrere Slaves gibt. Daten können gleichzeitig zwischen Master und Slave(s) kommuniziert werden. Dies wird durch die 4-Pin-Architektur ermöglicht. Die Pins MOSI (master out/slave in) und MISO (master in/slave out) werden für die Datenübertragung, der SCLK-Pin für das Clock-Signal und der SS-Pin (Slave select) für die Slave-Auswahl verwendet. Jede Datenübertragung wird vom Master eingeleitet. Die Clock vom Master muss an die Clock des langsamsten Slaves angepasst werden.
I^2C hingegen ist ein halbduplex synchroner serieller Bus, bei dem es mehrere Master geben kann. Statt 4 Pins werden nur zwei Pins - Taktleitung (SCL) und Datenleitung (SDA) - verwendet. Da es keinen SS-Port gibt, muss jedes Gerät, das I^2C verwendet, eine eindeutige Adresse für die Kommunikation mit einem Gerät besitzen. Die Clocks der Master-Geräte müssen synchronisiert werden, da jeder eine eigene Uhr hat. Ansonsten könnte ein langsamer Slave seine Adresse auf der Datenleitung finden und ein schnellerer Master Daten an ein drittes Gerät schicken. [14] (S. 58-62)

3.1.5 Prototypen

Realisierungen von den besprochenen Subsystemen werden nun anhand von zwei am Markt erhältlichen drahtlosen Netzwerkknoten vorgestellt. Beide Knoten sind „state-of-the-art“-Knoten (2014 und 2011).

WiSens

WiSens-Knoten verwenden pro Knoten einen Mikrocontroller (Texas Instruments' MSP430) für lokale Operationen (Prozessor-Subsystem) und ein Funkmodul (Texas Instruments' CC2520) für die Kommunkation nach außen. Das Funkmodul implementiert den „IEEE 802.15.4“-Standard. Anwendungsentwickler können sich mit dieser Plattform auf die Anwendung konzentrieren und brauchen sich nicht um das Hardware-Design oder die Funk-Performance sorgen. Wie Abbildung 3.1 zeigt, besteht zwischen dem Mikrocontroller und dem Radiomodul sowohl eine GPIO- als auch ein SPI-Schnittstelle. Die Sensoren jedoch können jedoch weiters auch über I^2C oder UART erreicht werden. Bevor ein WiSens-Knoten ein Netzwerk betritt, muss dessen EEPROM mit einer 64-bit-ID versehen werden. Der Anwender selbst muss dafür sorgen, dass jeder Knoten im Netzwerk eine UID hat. Die Antenne kann entweder als externes Modul oder auf dem PCB-Substrat angebracht werden. WiSens-Knoten bieten aufgrund der modularen Bauweise mehr Flexibilität. Basis-, Funk- und Sensormodul können verwendet werden, um einen Knoten zu bauen, je nachdem welche Funktion der Knoten übernehmen soll. Das Sensor-Subsystem besteht hier aus zwei auf der Basisplatte bereits integrierten Sensoren und einer eigenen Sensorplatte, welche die Möglichkeit bietet noch 8 weitere Sensoren anzuschließen. Abbildung 3.2 veranschaulicht diese Modularität. [5]

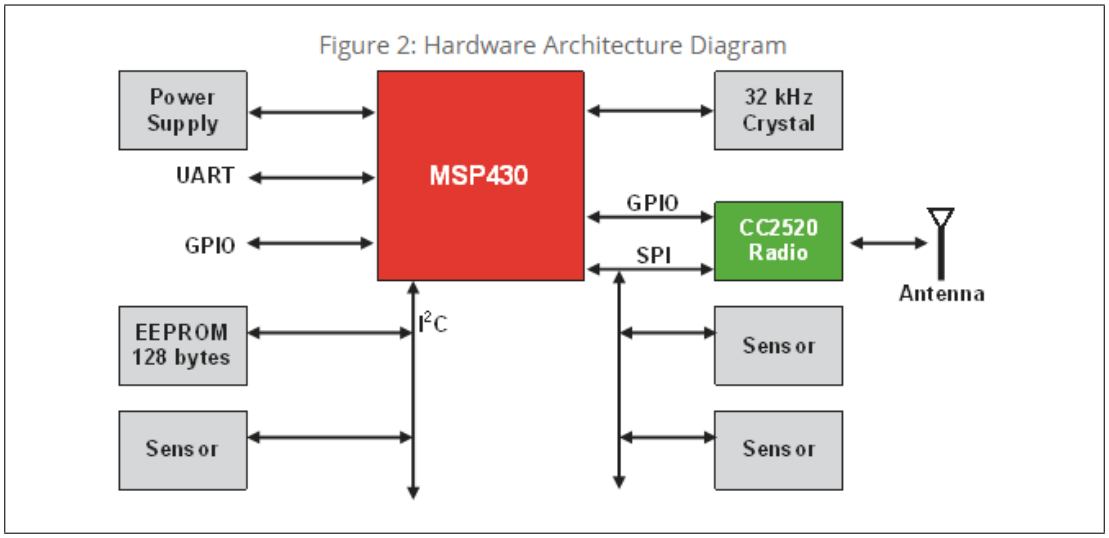


Abbildung 3.1: Hardware Architecture Diagram [5]

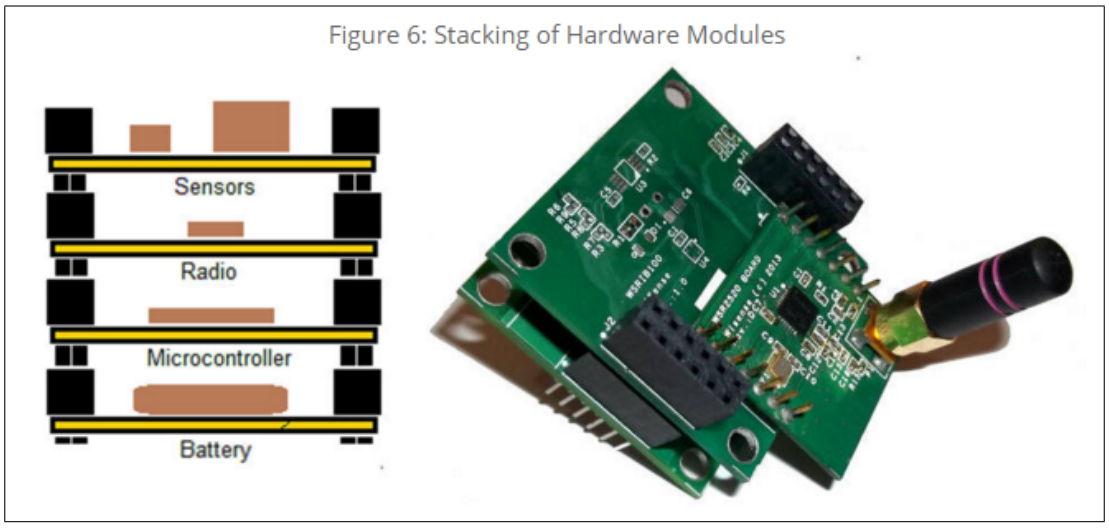


Abbildung 3.2: Knotenmodule [5]

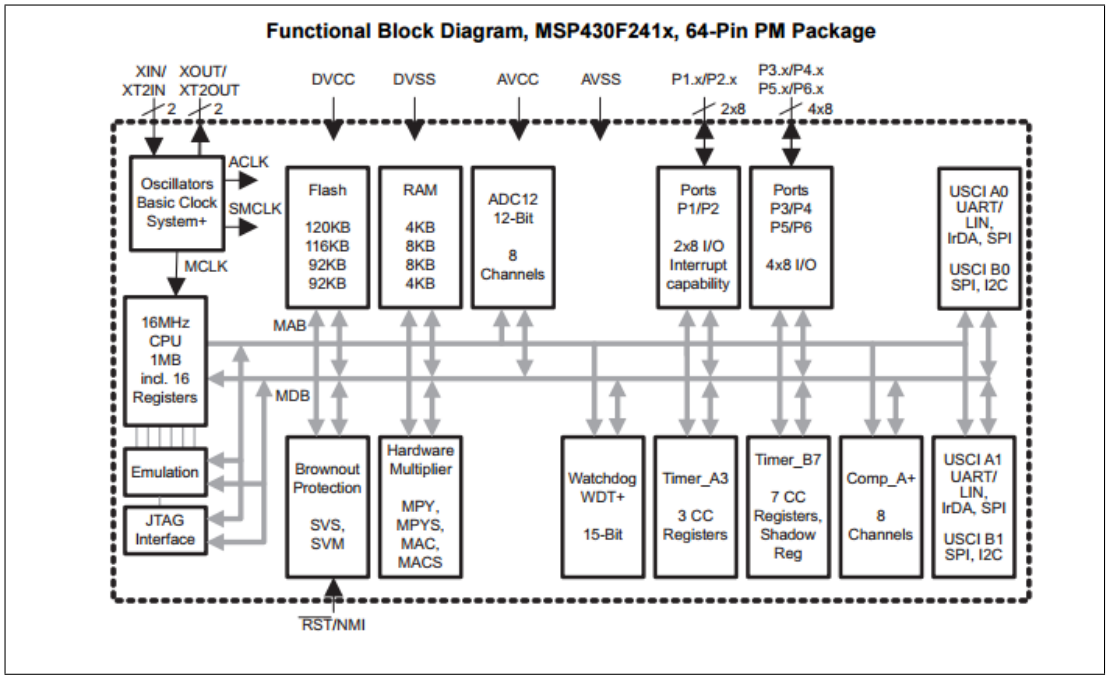


Abbildung 3.3: Functional block diagram msp430F241x [6]

XM1000

Das XM1000 Knotenmodul umfasst Temperatur-, Feuchtigkeits- und Lichtsensoren und ist mit der open-source Plattform TelosB kompatibel. Der Mikrocontroller ist ebenfalls von Texas Instruments (TI MSP430F2618) und TinyOS 2.x kompatibel. TinyOS ist ein weit verbreitetes Betriebssystem unter drahtlosen Netzwerkknoten, auf welches später noch eingegangen wird. Außerdem beinhaltet das Modul noch einen 2.4 GHz RF-Transceiver (CC2420 RF) basierend auf ZigBee. Abbildung 3.3 zeigt das funktionale Blockdiagramm des Mikrocontrollers. Die CPU hat eine 16-bit RISC-Architektur und 16 Register mit verkürzter Ausführungszeit für die Instruktionen. Das Befehlsset beinhaltet 51 Befehle mit drei verschiedenen Formaten und 7 Adressmodi. Der Flash-Speicher kann über den JTAG-Port, den Bootstrap-Loader oder „in-system“ von der CPU programmiert werden. Da Selbstregulierung ein wichtiger Punkt in drahtlosen Netzwerken ist, ist der Mikrocontroller auch mit einem Watchdog-Timer-Modul (WDT) ausgestattet. Tritt ein Software-Fehler auf, so wird ein kontrollierter Neustart vom System durchgeführt. Auch dem Energieproblem tritt der MSP430 mit verschiedenen Low-Power-Modi, in denen die CPU deaktiviert wird, entgegen. Je nach Modus sind die drei integrierten Clocks (Auxiliary clock, Main clock, Sub-Main Clock) aktiv oder inaktiv. [6]

3.2 Kommunikationsprotokolle

Bevor wir über Protokolle sprechen, wollen wir zuerst ein Schichtenmodell präsentieren, welches in drahtlosen Sensornetzwerken verwendet wird. Leider gibt es kein einheitliches Modell, welches für alle Sensornetzwerke gilt, da ein drahtloses Sensornetzwerk je nach Einsatz und Zweck variiert. Jedoch sind die meisten in der Literatur ([14] (S. 95-161), [38] (S. 312-325), [39], [40]) erwähnten Modelle ähnlich. Jedes Modell besitzt zumindest die folgenden 4 Schichten: Physikalische Schicht, Sicherungsschicht, Vermittlungsschicht und Anwendungsschicht. Die Transportschicht wird in manchen Modellen nicht erwähnt (z.B. ZigBee), welche im Folgenden aber dennoch erläutert wird.

3.2.1 Physikalische Schicht

Einer der erstrebenswerten Effekte eines drahtlosen Sensornetzwerk ist die Kommunikation über einen drahtlosen Link. Knoten sollen in einer gewissen Grenze beliebig positioniert werden können, mit dem Ziel maximale Abdeckung und Konnektivität zu erreichen. Die physikalische Ebene liefert uns die Grundlage dafür. Im Abschnitt Sensor-Subsystem wurden schon die Grundlagen der Signalübertragung angeschnitten.
Ein gesampeltes diskretes Signal wird zunächst in einen binären Datenfluss umgewandelt. Dieser Vorgang nennt sich **Quellencodierung**. Es ist wichtig, eine effiziente Codierungstechnik zu implementieren, damit die Bandbreite und Anforderungen an die Signalstärke zufriedengestellt werden. Eine Möglichkeit dies zu tun, ist ein Wahrscheinlichkeitsmodell von der Informationsquelle zu erstellen, damit die Länge jedes Informationssymbols von dessen Auftrittswahrscheinlichkeit abhängt. Der nächste Schritt ist die **Kanalkodierung** mit dem Ziel, das übertragene Signal von Störgeräuschen zu schützen und ggf. bei einem Fehler die originalen Daten wieder herzustellen. Weiters folgt die **Modulation** des Signals, bei der das Basisbandsignal in ein Bandpasssignal umgewandelt wird. Modulation ist vorallem deswegen wichtig, weil das Signal mit kürzeren Antennen gesendet und empfangen werden soll (je kürzer die Wellenlänge, desto kürzer die benötigte Antenne). Schließlich wird das elektrische Signal **verstärkt**, in ein elektromagnetisches Signal umgewandelt und über einen drahtlosen Link zum Zielknoten **gesendet**. Auf der Empfängerseite findet im Prinzip der umgekehrte Prozess statt, um die Nachricht aus der elektromagnetischen Welle zu erhalten. [14] (S. 95-123)

3.2.2 Sicherungsschicht

Ein drahtloses Medium soll von mehreren Knoten für die Informationsübertragung genutzt werden. Daher ist ein Mechanismus erforderlich, welcher den Zugriff auf das Medium regelt. Prinzipiell besteht die Sicherungsschicht aus zwei Unterschichten (LLC und MAC). Wir werden uns hier nur der MAC-Schicht widmen.
Die meisten MAC-Protokolle werden entworfen, um Fairness zu erzielen. Jeder Knoten soll die selben Ressourcen bekommen und keiner soll bevorzugt oder gesondert behandelt werden. In WSNs kooperieren alle Knoten, um einen bestimmten Nutzen zu erzielen. Daher ist Fairness eine weniger wichtige Eigenschaft im Vergleich zu geringer Latenz

oder hoher Zuverlässigkeit. Die wohl wichtigste Anforderung an MAC-Protokolle ist die Energieeffizienz. Eine häufige Technik, um diese zu erzielen, ist die Verwendung von unterschiedlichen Betriebsmodi (active, idle, sleep). Skalierbarkeit ist ebenso wichtig, Ressourcen sollten sparsam eingesetzt, um Overhead zu vermeiden. CDMA-basierte MAC-Protokolle müssen vielleicht eine große Menge an Codes cachen und machen sich damit ungeeignet für Sensorgeräte mit beschränkten Ressourcen. Anpassungsfähigkeit und Berechenbarkeit sind weitere Eigenschaften, die MAC-Protokolle in drahtlosen Sensornetzwerken besitzen sollten.
Um diesen Anforderungen gerecht zu werden, wurden für WSNs eigene Protokolle entwickelt, die auf bereits bestehenden Netzwerkprotokollen (TDMA, FDMA, Token passing, CSMA, ALOHA,...) basieren. Das TRAMA-Protokoll (Traffic-Adaptive Medium Access) ist ein Beispiel für ein konfliktfreies MAC-Protokoll, welches auf dem beliebten TDMA-Protokoll basiert. Im Vergleich zielt TRAMA aber auf mehr Netzwerkdurchsatz und Energieeffizienz ab. Es verwendet ein verteiltes Wahlsystem basierend auf der Information über den Datenverkehr bei jedem Knoten, um zu entscheiden, wann ein Knoten senden darf. Dies hilft zu verhindern, dass kein Slot an einen Knoten ohne Datenverkehr angeschlossen wird (mehr Netzwerkdurchsatz) und erlaubt den Knoten zu bestimmen, wann sie in den idle-Modus schalten sollen (mehr Energieeffizienz). Spezielle Security MAC-Protokolle sind u.a. TinySec, LLSP, SPINS und C-Sec. Diese zielen ebenso auf geringen Energieverbrauch, aber auch auf gerigen Speicherverbrauch und Flexibilität ab. AlMheiri *et al. 2013* stellt bei dem C-Sec-Protokoll die meiste Sicherheit, im Vergleich zu den anderen genannten Protokollen, fest. [41], [14] (S. 125-157)

3.2.3 Vermittlungsschicht

Die Hauptverantwortung in der Vermittlungsschicht liegt in der Bestimmung eines Weges vom Absenderknoten zum Adressaten (Basisstation bzw. Gateway). Diese Aufgabe kennen wir unter dem Begriff *Routing*. Drahtlose Sensornetzwerke verlangen aufgrund von Ressourcenmangel und Unzuverlässigkeit des Mediums wieder nach speziellen Protokollen und Lösungen. Hier gibt es wieder Kategorien, in welche Routing-Protokolle eingeteilt werden können. Diese können einerseits nach der Netzwerkorganisation (flach-, hierarchisch- und Orts-basiert) unterschieden werden. Außerdem unterscheidet man zwischen reaktivem, proaktivem und hybridem Routing. Protokolloperationen (verhandlungs-, multi-path-, query- QoS- und Kohärenz-basiert) sind ein weiteres Merkmal für die Klassifizierung von Routing-Protokollen. Sendet ein Knoten Informationen zu einem oder mehreren Adressaten, so bezeichnet man dies als knotenzentrisch (engl. node-centric). Datenzentrisch wird das Routing dann genannt, wenn Empfänger nach bestimmten Kriterien ausgewählt werden. Z.B. verlangt eine Basisstation nach Temperaturmessungen und nur die Knoten, die diese Information besitzen, senden eine Antwort zurück.
Sensor Protocols for Information via Negotiaton (SPIN) ist eine Familie von verhandlungsbasierten, datenzentrischen und zeitgesteuerte Flooding-Protokollen, welche speziell für drahtlose Sensornetzwerke konzipiert wurden. SPIN basiert auf zwei Schlüsselverfahren, um die Probleme im klassischen Flooding zu beseitigen. Um Implosion und Überlappung zu vermeiden, verhandeln SPIN-Knoten mit ihren Nachbarn. Um das Res-

sourcenproblem zu lösen, verwendet jeder SPIN-Knoten einen Ressourcenmanager, um den Ressourcenverbrauch aufzuzeichnen, damit Routing und Kommunikation anhand der Ressourcenverfügbarkeit angepasst werden können.
SPIN-PP ist das erste Mitglied der SPIN-Familie. Dessen Knoten haben drei mögliche Nachrichten, die sie senden können: **ADV** (der Sender dieser Nachricht hat neue Daten zur Verfügung), **REQ** (der Sender dieser Nachricht, teilt mit, dass er neue Daten braucht), **DATA** (die angekündigten Daten).
Beispiel-Szenario: Knoten A sendet ADV an Knoten B. Knoten B weiß jetzt, dass Knoten A neue Daten hat und sendet ein REQ zurück. Knoten A weiß jetzt, dass Knoten B die neuen Daten braucht und sendet schließlich die DATA-Nachricht.
Weitere Mitglieder der SPIN-Familie sind SPIN-EC, SPIN-BC und SPIN-RL. [14] (S. 163-197)

3.2.4 Transportschicht

Die Notwendigkeit einer Transportschicht in einem WSN wird von einigen Experten in Frage gestellt, da der Verlust von Daten, welche von den Sensorknoten zu der Basisstation (Hinweg) fließen, toleriert wird. Für die Basisstation sind die kollektiven Datenflüsse wichtiger als die einzelnen Paketübermittlungen oder ein Handshake wie beim klassischen TCP. Dennoch ist bei manchen Applikationen eine gewisse Genauigkeit bei der Ereigniserkennung wichtig. Dies erfordert eine zuverlässige Kommunikation am Hinweg. Außerdem bedarf es je nach Größe und Dichte des Sensornetzwerks auch einer Staukontrolle, da bei einem erkannten Event mehrere Knoten gleichzeitig senden und somit ein gewisser Datenverkehr entsteht. Ein guter Staukontrollmechanismus kann auch dabei helfen Energie zu sparen, indem er je nach verlangten Genauigkeitsgrad die Anzahl der sendenden Knoten regelt. Das klassische TCP wäre in solch einem Netzwerk aufgrund des strikten End-to-End-Prinzips und der damit verbundenen hohen Ressourcenverschwendung nicht passend. Das ESRT-Protokoll [42] erfüllt alle oben genannten Anforderungen und erfordert kein Zwischenspeichern von Informationen. ESRT verlangt nicht einmal nach einer Identifikation des Knotens, die Event-ID genügt. Die ESRT-Algorithmen laufen außerdem hauptsächlich auf der Basisstation und befreit ressourcenbeschränkte Sensorknoten von großem Aufwand. Daten, welche von der Basisstation zu den einzelnen Knoten fließen (Rückweg), sind hauptsächlich OS-binaries, Aufgaben, Code-Änderungen und Konfigurationsfiles. Die Übertragung dieser erfordert hingegen eine 100%-ige Genauigkeit und somit auch andere (TCP-ähnliche) Protokolle, wie z.B. das PSFQ-Protokoll (pump slowly, fetch quickly), welches die Pakete langsam ins Netzwerk injiziert, aber bei einem Paketverlust eine agressive „Hop-by-Hop-Recovery“ durchführt. [38] (S. 318-320)

3.2.5 Anwendungsschicht

Auf der Applikationseben gibt es eine Vielzahl von Aufgaben, welche festgelegt werden müssen und ein gutes Management erfordern. Dazu wurde das *Sensor Management Protocol* (SMP) vorgeschlagen, über welches Systemadministratoren mit dem Sensornetzwerk kommunizieren. Das SMP macht die Hardware und Software der darunterliegenden

Schichten transparent und bietet Software-Operationen für die Erledigung folgender Aufgaben:

- Attributbasiertes Naming und positionsbasierte Adressierung (Sensorknoten haben keine globale ID)
- Positionsaustausch
- Synchronisierung der Sensorknoten
- Ein- und Ausschalten der Sensorknoten
- Statusabfrage und Netzwerk (Re)Konfiguration
- Sicherheit beim Datenaustausch

Synchronisierung spielt sehr wohl auch auf der Applikationsschicht eine Rolle. Standardprotokolle wie Bluetooth oder ZigBee verhindern normalerweise den Zugriff auf tieferliegende Schichten und beeinträchtigen somit die Genauigkeit. Álvaro Marco *et al.* stellen in [43] das *Multihop Broadcast Synchronisation Protocol* vor ein *receiver-to-receiver*-Schema, welches genaue Synchronisierung von Multihop-Netzwerken erlaubt und den Datenverkehr dabei minimal hält. Zeitsynchronisierung wird im nächsten Abschnitt noch genauer besprochen. Weitere Protokolle auf der Applikationseben sind das TADAP und das SQDDP (siehe [38] S. 313-315). Diese werden jedoch hier nicht näher erläutert.

3.3 Zeitsynchronisation

In einem drahtlosen Sensornetzwerk hat jeder Knoten seine eigene Clock. Eine Clock, die für jeden Knoten gleich tickt, kann es praktisch nicht geben, da die Oszillation von vielen Parametern, wie z.B. der Dicke des Quarzkristalls aber auch der Umgebungsfeuchtigkeit und -temperatur abhängt. Sensorknoten brauchen jedoch einen Zeitmaßstab, um Kausalbeziehungen von Events zu identifizieren, redundante Daten zu vermeiden und allgemein Operationen in einem WSN zu erleichtern. Aufgrund der einzigartigen und bereits mehrmals erwähnten Herausforderungen von drahtlosen Sensornetzwerken eignen sich viele für kabelgebundene Netzwerke bestehende Protokolle nicht für billige und drahtlose Sensorknoten. In diesem Abschnitt werden einige Grundlagen, Probleme und Methoden erläutert und daraufhin ein „state-of-the-art“-Protokoll vorgestellt.

3.3.1 Grundlagen

Eine typische Clock besteht aus einem Quarzkristall und einem Counter, welcher sich mit jeder Oszillation des Kristalls verringert. Erreicht der Counter den Wert 0, so wird dieser zurückgesetzt und eine Unterbrechung erzeugt. Jede Unterbrechung (oder *clock tick*) führt zu einer Erhöhung einer Software-Clock, welche von Applikationen über eine API gelesen werden kann. Folgende Begriffe sind für das Verständnis von Clocks und Zeitsynchronisation wichtig:

- Zeitversatz: Der zeitliche Unterschied zweier Clocks.

- Taktfrequenz: Die Frequenz, mit welcher eine Clock voranschreitet.
- Taktversatz: Die Zeitdifferenz zwischen dem Eintreffen einer Taktflanke am ersten zu betrachtenden Element und dem Zeitpunkt des Eintreffens an einem zweiten Element.
- Driftrate: Die Rate, mit der zwei Clocks auseinanderdriften.
- Genauigkeit: Der maximale Zeitversatz zwischen einer Clock eines beliebigen Knotens und der Clock des Referenzknotens.
- Präzision: Der Zeitversatz zwischen zwei beliebigen Clocks im Netzwerk

Außerdem wird zwischen interner und externer Zeitsynchronisation differenziert. Bei externer Synchronisation werden alle Clocks auf eine Referenzclock abgestimmt. Diese Referenzclock ist ein Echtzeitstandard wie z.B. UTC. Bei interner Synchronisation sind alle Clocks aufeinander abgestimmt ohne einen externen Referenzwert zu erhalten. [14] (S. 229-231)

3.3.2 Gründe für Zeitsynchronisation

Für das Reporting von Aktivitäten und Events von beobachteten Objekten sind zeitlich synchronisierte Sensorknoten obligatorisch, um gewisse Fragen beantworten zu können. Will man beispielsweise die Anzahl der bewegenden Objekte ab einem gewissen Zeitraum wissen, so könnten nicht-synchronisierte Sensoren eine Fehlinformation liefern. Auch um die Geschwindigkeit eines Objektes zu messen, wenn mehrere Sensoren tätig sind, ist eine Synchronisation notwendig, da die zeitliche Differenz, der Zeitstempel $C_1(t_1)$ und $C_2(t_2)$ zwischen zwei Messpunkten, der Echtzeitdifferenz $t_1 - t_2$ entsprechen muss. Dies ist auch eine wichtige Anforderung für die Datenfusion, bei der es darum geht, Daten von diversen Sensoren zu sammeln, die für die Beobachtung desselben Events zuständig sind. Zeitsynchronisation ist ebenso in verteilten Systemen im Allgemeinen für viele Anwendungen und Algorithmen notwendig. Beispiele dafür sind Kommunikationsprotokolle (z.B. at-most-once-Nachrichtenübertragung), Sicherheit (z.B. limitierte Verwendung von bestimmten Schlüsseln), Datenkonsistenz (z.B. Cache-Konsistenz und Konsistenz von replizierten Daten) und viele weitere. [14] (S. 231-232)

3.3.3 Herausforderung in WSNs

Clock-Drifts müssen in drahtlosen Sensornetzwerken oft besonders berücksichtigt werden, da Sensoren in Umgebungen platziert werden, wo Temperatur, Feuchtigkeit und Druck einen großen Einfluss haben können. Variationen von 3ppm (1ppm $\approx$ eine Sekunde alle 12 Tage) in kontrollierten Umgebungen, wo sich lediglich die Raumtemperatur etwas ändert, können bei billigen Sensoren im Outdoorbereich viel höher sein.
Zeitsynchronisation sollte so effizient wie möglich erfolgen, da, wie schon bekannt, Sensoren batteriebetrieben sind. Eine minimale Anzahl an Nachrichten zur Synchronisation wird angestrebt.
Da sich die Topologie von WSNs ändern kann und manchmal unvorhersehbar ist, können

Latenzprobleme auftreten und zur asynchronen Kommunikation führen, sowie Sync-Nachrichten aufgrund der Reichweite nicht ankommen.
Verzögerungen beim Generieren der Sync-Nachricht so wie beim Zugriff auf den MAC-Layer, um die Nachricht zu senden, vergrößern sich, je schwächer die Prozessorleistung der Sensoren ist.
Je nach Applikation gibt es auch unterschiedliche Anforderungen an die Genauigkeit und Präzision. Für simple Objektbeobachtung müssen die Clocks nur intern synchronisiert werden. Demnach werden keine speziellen Anforderungen an die Genauigkeit gestellt. Bei Anwendungen hingegen, welche den Fußverkehr in öffentlichen Gebieten zu einer bestimmten Zeit messen, ist es wichtig eine genaue externe Synchronisation zu erzielen. [14] (S.232-233)

3.3.4 Herangehensweisen

Es gibt mehrere Möglichkeiten, zwei oder mehrere Knoten zu synchronisieren. Zunächst unterscheidet man zwischen unilateralem und bilateralem Nachrichtenaustausch. Ersterer benötigt lediglich eine Nachricht zur Synchronisation. Knoten A sendet eine Nachricht mit seinem Zeitstempel an Knoten B. Nach dem Empfang der Nachricht erstellt B einen eigenen Zeitstempel und berechnet sich aus der Differenz zum Ersten den Offset: $(t_2 - t_1) = D + \delta$.
D ist dabei die Propagationszeit und wird oft als Konstante angenommen oder ignoriert, da sie unbedeutend klein ist. In der bilateralen Synchronisation sendet B eine Nachricht an den Absender zurück. Diese beinhaltet t_1, t_2 so wie die Zeit t_3, an der die Nachricht gesendet wird. A ist nun in der Lage sich ebenfalls den Offset zu berechnen und sich außerdem D herzuleiten.
Beide Verfahren fallen auch unter die *Sender-Receiver-Synchronisation*. Im Gegensatz dazu wird beim *Receiver-Receiver-Prinzip* eine Nachricht an mehrere Knoten versendet, ohne dabei einen Zeitstempel mitzuschicken. Hingegen werden die Zeitstempel beim Empfangen der Nachricht generiert und zwecks Synchronisation mit den Nachbarn ausgetauscht. Dieses Verfahren hat den Vorteil, dass der kritische Pfad viel kürzer ist, weil sämtliche Verzögerungen, wie die Generierung der Nachricht, das Senden der Nachricht zur Netzwerkschnittstelle und der Zugriff auf den physischen Kanal keinen Einfluss auf die Synchronisation haben. Das RBS-Protokoll ist ein Beispiel für dieses Prinzip. [14] (S. 234-237)

3.3.5 Sticking Heartbeat Aperture Resynchronization Protocol

Das SHARP-Protokoll ist ein „state-of-the-art“-Protokoll (2015). Gegenüber dem, im Jahr 2011 vorgestellten, SISP-Protokoll, zeigt es sich in vielen Punkten (Payload, Anzahl der benötigten Nachrichten, Echtzeit-Synchronisation) vorteilhafter.[44] SISP bewährte sich gegen das in WSNs beliebte RBS-Protokoll (Reference Broadcast Synchronisation Protocol).
SHARP verwendet sogenannte „heartbeats“ für die Synchronisation. Ein Masterknoten sendet dabei einen *heartbeat* an alle Slave-Knoten nach einer Zeitperiode λ. Die

Intervalllänge λ des Netzwerks kann dabei beliebig angepasst werden, um den Netzwerkanforderungen gerecht zu werden. Spezifisch für SHARP ist, dass jeder Slave-Knoten eine Apertur/Öffnung (heartbeat aperture) hat, in welchem ein *heartbeat* auftreten sollte. Dieses Intervall ist abhängig von der Driftrate der Slave-Clock. Wird ein *heartbeat*-Signal empfangen, so endet die derzeitige Apertur und die Slave-Clock wird an die Master-Clock angepasst, im Grunde $n\lambda$, wobei n die Anzahl der bereits vergangenen Aperturen seit der ersten Synchronisation ist.
SHARP ist auch gegen verlorene *heartbeats* resistent. Slave-Knoten können die Anzahl der leergebliebenen Aperturen berechnen, wo eine Synchronisation hätte stattfinden sollen. Die Größe jeder nächsten Apertur wird auch angepasst, da angeommen wird, dass sich der Taktversatz mit jedem nicht erhaltenen *heartbeat* erhöht. SHARP ist energiesparend, da pro Knoten und pro Zeitperiode nur ein Signal übertragen werden muss. Außerdem ermöglicht die Existenz der *heartbeats* die Verwendung von TDMA. Die Aperturen können leicht in einen TDMA-Zeitplan eingebaut werden. [45]

3.4 Power-Management

Energie ist in einem drahtlosen Sensornetzwerk von großer Bedeutung, da aufgrund komplexer Aufgaben der Sensorgeräte wie Beobachtung, Kalkulation, Kommunikation und Selbst-Management viel Energie verbraucht wird, diese jedoch nur begrenzt verfügbar ist. WSNs erfordern Power-Management aber nicht nur aus diesem Grund. Idealerweise besteht ein WSN aus vielen verteilten und oft unzugänglichen Sensorknoten, so dass das Austauschen von Batterien manchmal unmöglich ist. Grobe Netzwerkfehler können auftreten, wenn gewisse Knoten aufgrund eines Ausfalls nicht mehr im Betrieb sind. Power-Management beschäftigt sich deswegen nicht nur mit der Entwicklung von effizienten Protokollen sondern auch mit der Identifizierung von verschwenderischen Aktivitäten. Beispielsweise könnten Knoten vorzeitig ausfallen, weil sie gewissen Datenverkehr ständig überhören und das Kommunikationssubsystem deswegen länger operieren muss. Die Ursache dafür sind oft schlechte Hardware- und/oder Softwarekonfigurationen.
Sogenannte Dynamische Power-Management Strategien (DPM) werden eingesetzt, um den Energieverbrauch im Knoten selber (lokal) oder im gesamten Netzwerk (global) minimal zu halten. Im Folgenden werden wir einige DPM-Strategien betrachten und erklären.

3.4.1 Dynamisches Power-Management

Bevor eine DPM-Strategie angewandt werden kann, sollte zunächst das Netzwerkdesign stehen und die Knoten konfiguriert sein. Erst dann kann eine DPM-Strategie dynamisch die kostengünstigsten Operationsbedingungen bestimmen. Anforderungen an das DPM sind dabei die Anwendung selber, die Netzwerktopologie und die Rate, mit der die Aufgaben der verschiedenen Subsysteme ankommen. Einteilen lassen sich DPM-Strategien in drei Kategorien:

1. Dynamische Operationsmodi

2. Dynamische Skalierung
3. Energy Harvesting

Dynamische Operationsmodi

Dynamische Operationsmodi findet man vorallem im Prozessor-Subsystem vor. Generell hat man n unterscheidbare Powerlevel-Modi und x Hardwarekomponenten. Demnach kann es $n \times x$ Modi geben. Klarerweise sind nicht alle Modi sinnvoll bzw. brauchbar. Bei dieser Strategie sind vorallem die Transition und die Zeitverzögerung beim Übergang von einem Modus in einen anderen zu beachten. Angenommen wir haben die beiden Modi *on* und *off*. Der Übergang von *off* zu *on* verursacht gewisse Kosten. Diese Kosten sind nur dann akzeptabel, wenn die Energie die im *off*-Modus gespart wird, groß genug ist. *Chiasserini and Rao 2003*[46] stellen dabei folgende Ungleichung auf:

- $t_{off} \geq max(0, \frac{(P_{on} - P_{off,on})t_{off,on}}{P_{on} - P_{off}})$

t_{off}:	Zeit im *off*-Modus
$t_{off,on}$:	Zeit während der Transition
P_{off}:	Energieverbrauch im *off*-Modus
$P_{off,on}$:	Energieverbrauch während der Transition
P_{on}:	Energieverbrauch im *on*-Modus

Energie kann hier nun auf drei verschiedene Arten gespart werden:

1. Durch die Erhöhung der Differenz zwischen P_{on} und P_{off}
2. Durch die Verringerung der Transitionszeit $t_{off,on}$
3. Durch die Erhöhung der Dauer des Zustandes t (t_{off})

Die meisten IEEE 802.15.4-tauglichen Funkgeräte haben neben dem aktiven Modus drei verschiedene Schlafmodi. Im tiefsten Schlafmodus (M3) ist sowohl der Oszillator als auch der Spannungsregulator deaktiviert. Dadurch wird in diesem Modus auch am wenigsten Strom verbraucht. Jedoch braucht man für die Reaktivierung der Komponenten auch sehr viel Zeit und Energie. Analog dazu braucht der oberflächlichste Schlafmodus (M1) wenig Energie, um in den aktiven Zustand zu schalten, jedoch viel Energie, um in diesem Zustand zu verweilen. Ein adaptives Funkgerät kann den Schlafmodus anhand der aktuellen Netzwerkbedingungen wählen. (Raja Jurdak *et al* 2010) [47]. Dazu überwacht eine Basisstation den Netzwerkverkehr und ermittelt anhand der Netzwerkauslastung den richtigen Schlafmodus, welcher dann an die Knoten propagiert wird. Im Allgemeinen bedeutet viel Verkehr einen leichten Schlafmodus, weil das ständige Schalten in den aktiven Zustand sonst zu viel Energie kosten würde. Bei wenig Verkehr schlafen die Knoten durchschnittlich länger, weswegen man diese in einen tiefen Schlafmodus schickt. Für die Entscheidung, welcher Schlafmodus für welchen Verkehr der geeignetste ist, ist

eine genauere Betrachtung und Analyse der energieverbrauchenden Komponenten nötig. [14] (S. 216-222)

Dynamische Skalierung

Dynamische Spannungsskalierung und *dynamische Frequenzskalierung* sind zwei Techniken die in Verbindung miteinander eingesetzt werden. Ein Sensor bzw. das Prozessor-Subsystem eines Sensorknotens verrichtet manchmal eine Aufgabe schneller, als man für diese Zeit eingeplant hat. Für die verbleibende Zeit wird der Prozessor trotzdem noch mit derselben Spannung und Frequenz betrieben, obwohl dies nicht nötig wäre. Im Zuge der dynamischen Skalierung wird die Aufgabe auf die eingeplante Zeit ausgeweitet. Dies erzielt man mit der Senkung der Prozessorfrequenz gefolgt von der Senkung der Versorgungsspannung.
Transistoren sind der Hauptbestandteil von Prozessoren. Das Schalten von Transistoren kostet Energie und die Menge der Energie hängt von vielen Faktoren ab, u.a. von der Betriebsfrequenz und der Versorgungsspannung. Es lässt sich zeigen, dass die Betriebsfrequenz in linearem Zusammenhang und die Versorgungsspannung in quadratischem Zusammenhang mit der (Schalt-)Energie stehen. [48] Jedoch können diese beiden Parameter nicht unter ein bestimmtes Limit gesenkt werden. Meind und Swanson (1972) fanden dabei heraus, dass das Limit der Versorgungsspannung für die CMOS-Logik bei $Vdd, limit = 48mV$ für 300K liegt. [14] (S. 219-222), [38] (S. 499-500)

Energy Harvesting

Die oben angeführten Techniken verlängern durch sämtliche Optimierungen die Lebensspanne der Sensoren, jedoch wird die Lebensspanne immer endlich bleiben. *Energy Harvesting* bedient sich Techniken, welche die Umgebungsenergie oder vom Menschen erzeugte Energie nutzen, um diese in elektrische Energie umzuwandeln und die Sensoren dadurch möglicherweise fortwährend zu erhalten. Die Herausforderung liegt dabei in der Abschätzung der Periodizität und der Menge der erntbaren Energie, um vorzeitige (vor dem nächsten Ladezyklus) Erschöpfung zu verhindern.
Ein typisches *Energy Harvesting*-System besteht aus einer Energiequelle, einer Architektur bestehend aus Umwandlungsmechanismen und einer Speichereinheit.
Bei den Architekturen unterscheidet man grob zwischen *Harvest-Use* und *Harvest-Store-Use*. Erstere Architektur wandelt die Energie nur bei Gebrauch um. Der Power-Output des Systems wird dabei knapp über dem Betriebspunkt gehalten. Variationen in der Erntekapazität können dabei zu Oszillationen zwischen dem Ein- und dem Auszustand führen. Beispiele für solche Systeme sind Keyboards, welche Piezokristalle verwenden, die beim Betätigen einer Taste verformt werden und dabei elektrische Energie erzeugen, die groß genug ist, um z.B. ein Radiosignal zu senden. *Harvest-Store-Use* verwendet hingegen einen Zwischenspeicher. Energie wird bei jeder Möglichkeit geerntet und für den späteren Gebrauch gespeichert. Solarenergie ist dabei eine beliebte Energiequelle. Bei Tag wird Solarenergie geerntet und bei Nacht wird die gespeicherte Energie u.a. verwendet, um den Sensorknoten zu betreiben.

Energy Source	Characteristics	Amount of Energy Available	Harvesting Technology	Conversion Efficiency	Amount of Energy Harvested
Solar[25], [26], [27], [28]	Ambient, Uncontrollable, Predictable	$100mW/cm^2$	Solar Cells	15%	$15mW/cm^2$
Wind[28]	Ambient, Uncontrollable, Predictable	-	Anemometer	-	1200mWh/day
Finger motion[22], [24]	Active human power, Fully controllable	19mW	Piezoelectric	11%	2.1mW
Footfalls[22], [24]	Active human power, Fully controllable	67W	Piezoelectric	7.5%	5W
Vibrations in indoor environments[29]	Ambient, Uncontrollable, Unpredictable	-	Electromagnetic Induction	-	$0.2mW/cm^2$
Exhalation[24]	Passive human power, Uncontrollable, Unpredictable	1W	Breath masks	40%	0.4W
Breathing[24]	Passive human power, Uncontrollable, Unpredictable	0.83W	Ratchet-flywheel	50%	0.42W
Blood Pressure[24]	Passive human power, Uncontrollable, Unpredictable	0.93W	Micro-generator	40%	0.37W

Abbildung 3.4: Energy Sources [7]

Energiequellen können kontrollierbar/unkontrollierbar und/oder vorhersehbar/nicht vorhersehrbar sein. Vorhersehbare, aber nicht kontrollierbare Energiequellen, wie Solarenergie sind dabei gut für *Harvest-Store-Use*-Systeme geeignet. Ein Vorhersagemodell kann dabei verwendet werden, um die Erntezeiten festzulegen. Für *Harvest-Use*-Systeme eignet ausschließlich kontrollierbare Energie wie z.B. Fingerbewegungen oder Schritte. Abbildung 3.4 zeigt diverse Energiequellen mit ihren Eigenschaften.
Bei der Speicherung von Energien werden entweder diverse aufladbare Batterien (SLA, NiCd, NiMH, Li-ion) oder Superkondensatoren verwendet. NiMH- und Lithiumbatterien kommen für Sensorknoten am meisten in Frage. Lithiumbatterien haben eine hohe Ausgangsspannung, Energiedichte, Effizienz und eine moderate Selbstentladungsrate. Jedoch benötigen sie einen hochpulsierenden Ladestrom. NimH-Batterien hingegen brauchen keine komplexe Ladeschaltung. Dafür leiden sie unter dem Memory-Effekt (Kapazitätsverlust bei sehr häufiger Teilentladung). Superkondensatoren haben viele Vorteile gegenüber Batterien. Sie entladen sich viel schneller, haben eine geringe Gewicht/Energie-Dichte (5Wh/kg im Vergleich zu 100Wh/kg bei NiMH-Batterien) und leiden auch nicht unter dem Memory-Effekt. Theoretisch haben Superkondensatoren unendlich viele Ladezyklen und können daher beliebig oft aufgeladen werden. [7]

3.5 Sicherheit

So wie jedes andere Netzwerk kann auch ein drahtloses Sensornetzwerk das Ziel von einer Vielzahl von Attacken sein. Die einzigartigen Eigenschaften eines WSNs verlangen auch in Sachen Sicherheit nach eigenen Lösungen. Die Sicherheit ist in solchen Netzwerken von großer Bedeutung, da das Fehlverhalten eines drahtlosen Sensorsystems z.B. in Kriegsgebieten, bei der Gebäudeüberwachung oder in Katastrophengebieten oft kritisch

sein kann. Dieser Abschnitt gibt einen Überblick über die Sicherheitsbelangen eines WSNs und stellt daraufhin mögliche Lösungen und Protokolle vor.

3.5.1 Sicherheitsherausforderungen

Entfernte Standorte. WSNs werden oft in Umgebungen eingesetzt, in denen sie Naturgewalten oder physischen Attacken ausgesetzt sind. Hinzu kommt, dass sie manchmal physisch gar nicht erreicht werden oder beobachtet werden können. Sensorgeräte könnten somit entweder durch Gewalteinwirkung zerstört oder durch einen Angreifer modifiziert werden. Ausfall durch Gewalteinwirkung lässt sich vielleicht nicht vermeiden, jedoch gibt es verschiedene Wege, um sich vor Modifizierung zu schützen.
Datenintegrität. In vielen Netzwerken geht es oft um die Gewährleistung, dass eine gesendete Nachricht noch immer dieselbe ist, wenn sie beim Empfänger ankommt. Auch WSNs müssen dies sicherstellen. Spezifisch aber für drahtlose Sensornetzwerke ist die Integrität von Datenaggregationen. Wird von gesammelten Datenwerten z.B. der Durchschnitt berechnet und weitergeleitet, so könnte ein Angreifer mit der Abänderung eines einzigen Datenwertes den Output bestimmen. Darauf sollte man zusätzlich bei der Integrierung von Sicherheitsmechanismen achten.
Vertraulichkeit. Daten dürfen nur von authoritisierten Knoten gelesen und modifiziert werden. Dies sollte zunächst durch einen effizienten Verschlüsselungsmechanismus bewerkstelligt werden. Das heißt, dass nur die Knoten, die den Schlüssel haben, die Daten auch entschlüssel können. Gelangt der Schlüssel an einen feindlichen Knoten, so hat dieser dieselben Möglichkeiten. Viele Algorithmen stellen deswegen die Vertrauenswürdigkeit (Authentizität) der Knoten fest, denen die Daten übermittelt werden sollen.
Privatsphäre. Aufgrund der Kommunikation über öffentliche Links können Angreifer leicht den Netzwerkverkehr verfolgen. Ohne Verschlüsselung sind die Daten leichte Beute. Angreifer könnten aber auch den Verkehr analysieren, um wichtige Knoten eines Netzwerkes (z.B. Basisstationen) zu identifizieren, um so noch mehr Daten zu kompromittieren. Oft muss es auch ein Trade-Off zwischen Privatsphäre und Datenintegrität geben. Beides zur selben Zeit zu gewährleisten, ist schwierig. Übliche Algorithmen zum Schutz der Privatsphäre vor anderen Netzwerkteilnehmern verhindern oft den Zugriff auf Informationen benachbarter Knoten, um die Datenintegrität zu bestätigen.
Anwendungsspezifische Architekturen. Um Ressourcen zu sparen, sollte jede Architektur eines WSNs anwendungsspezifisch sein. Fast jeder Aspekt eines WSNs kann im Bezug auf Energieverbrauch optimiert werden. Dies verlangt genauso nach anwendungsspezifischen Sicherheitsmechanismen und -protokollen. Ein Architekturdesigner muss dabei die wichtigen und in Frage kommenden Bedrohungen kennen und sich dann für die richtige Abwehr, mit Rücksicht auf den Energieverbrauch, gegen diese Bedrohungen entscheiden. [14] (S. 269), [38] (S. 585), [49]

3.5.2 Angriffe und Abwehrmechanismen

Angriffe können mit verschiedenen Zielen ausgeführt werden. Oft will man mit Angriffen einen Systemabsturz oder eine Dienstverweigerung des Servers erreichen. Sogenannte

DoS-Attacken können diesem Zweck dienen. Ein anderes Motiv ist die Ergatterung von privaten Daten. Mehrere Angriffsmethoden, die diese Ziele abdecken, und jeweilige Schutzmechanismen, werden im Folgenden erläutert.

DoS-Attacken

Eine Denial-of-Service-Attacke hat in einem WSN das Ziel, die Ressourcen eines Sensorknotens aufzubrauchen. Ein Angriff kann dabei auf verschiedenen Netzwerkschichten erfolgen. *Jamming* ist ein Beispiel für einen DoS-Angriff in der Bitübertragungsschicht. Dabei sendet der Angreifer ein Störsignal auf derselben Frequenz wie der zu störende Sender und verhindert somit die Kommunikation zwischen diesem Knoten und anderen im Netzwerk. Ist der Störsender selber ein Sensorknoten, so wäre das ständige Senden des Störsignals zu energieraubend. Deswegen sendet der Angreiferknoten nur dann, wenn er eine Übertragung mitbekommt.
Schützen kann man sich vor einer *Jamming*-Attacke, indem man eine größere Bandbreite zum Kommunizieren verwendet, als nötig wäre und dabei zwischen den Frequenzen wechselt (Frequenzspreizung). Der Sender und Empfänger müssen bei einer solchen Maßnahme koordiniert werden.
Auf der Sicherungsschicht kann der Angreifer auf Kollisionen setzen, um die Ressourcen eines oderer mehrerer Knoten aufzubrauchen. Schlüsselelemente, die zur Checksummengenerierung führen, können verändert werden, was zur Paketverwerfung und unnötig längerem Zugriff auf das Medium führt.
Ein Verteidigung gegen diese Attacke ist schwierig. Eine Möglichkeit jedoch ist ein Fehlerkorrekturverfahren zu verwenden. Dieses kostet aber zusätzliche Energie und ein Angreifer kann immer mehr Daten beschädigen, als das Verfahren korrigieren kann.
Blackhole- und *Sinkhole*-Attacken sind Angriffe, die in der Netzwerkschicht ausgeführt werden können. Bei beiden geht es darum, einen Knoten in die Route einzuschleusen und entweder bestimmte (*selective forwarding*) oder alle Pakete zu verwerfen, die diesen Knoten passieren wollen. Bei einem *Sinkhole*-Angriff möchte man den Knoten so positionieren, dass möglichst viele Daten angezogen werden. Ein „Wurmloch“ kann dabei helfen möglichst viel Aufmerksamkeit auf einen Knoten zu lenken. Dabei geben zwei Angreiferknoten (z.B. zwei Laptops mit starkem Funksender) an, einen schnellen Kommunikationsweg mit niedriger Latenz zu besitzen und nur einen Hop von der Quelle entfernt zu sein, was für viele Routing-Algorithmen attraktiv wirkt.
Ein gutes Abwehrprinzip beruht auf dem geographischen Routing. Jeder Knoten trifft für die Weiterleitung der Daten eine unabhängige Entscheidung basierend auf der physischen Position des Nachbars. Außerdem kann für das gesamte Netzwerk eine bestimmte Verbindungsqualität festgelegt werden. Kommunikationspartner können dann mittels *end-to-end*-Verifikation eine Abweichung leicht feststellen und ein systematische Re-Routing starten.
Die Transportschicht kümmert sich um das Management von *end-to-end*-Verbindungen in einem Netzwerk. Die *Flooding*-Attacke nützt die Tatsache aus, dass viele Transportprotokolle (z.B. TCP) Zustandsinformation speichern. Ein Angreifer könnte immer wieder neue Verbindungsanfragen schicken und somit den Zielknoten mit immer mehr Informationen

überschwemmen, bis dessen Ressourcen erschöpfen.
Eine Möglichkeit sich dagegen zu verteidigen, ist die Verwendung von Puzzles. Ein Knoten muss ein gewisses Puzzle per Brute-Force lösen, bevor es auf verbindungsbezogene Ressourcen zugreifen kann. Dieses Puzzle ist skalierbar. Ein Knoten, der sich angegriffen fühlt, könnte den Schwierigkeitsgrad des Puzzles erhöhen, so dass der Angreifer möglicherweise ewig braucht, um das Puzzle zu lösen. In einem WSN sollte man damit jedoch aufgrund der Ressourcenknappheit vorsichtig umgehen. [49], [14] (S. 270-278), [38] (S. 611-616)

Angriff auf die Privatsphähre

Gesammelte Informationen in einem WSN sind auch oft dem Risiko eines Angriffs ausgesetzt. Ohne Verschlüsselung und anderen Sicherheitsmechanismen wird das Abhören und die Verkehrsanalyse dem Angreifer leicht gemacht. Symmetrische Verschlüsselungsverfahren sind für drahtlose Sensornetzwerke augrund ihrer Ressourceneffizienz meistens die attraktivere Lösung, obwohl es auch RSA- und ECC-Lösungen für ressourcenbeschränkte Sensorknoten gibt. Der größte Nachteil von symmetrischen Verschlüsselungsverfahren ist die Verbreitung der Schlüssel.
Für kabellose Sensornetzwerke gibt es beispielsweise den PIKE-Ansatz (Peer Intermediaries for Key Establishment). Sensorknoten werden dabei als vertrauenswürdige Vermittler gewählt und jeder Knoten teilt von Anfang an ein Schlüsselpaar mit $O(\sqrt{n})$ anderen Knoten, wobei n die Anzahl der Knoten im gesamten Netzwerk ist. Vorallem aber ist die Schlüsselverteilung so gewählt, dass für je zwei Knoten A und B ein Knoten C existiert, welcher jeweils ein Schlüsselpaar mit beiden teilt. Somit können A und B über C kommunizieren. Alle Knoten haben eine ID der Form (x,y), $x, y \in 0, 1, 2, ...\sqrt{n} - 1$. Jeder Knoten teilt jeweils ein Schlüsselpaar mit allen Nachbarn derselben Zeile und Spalte.
Angenommen, A = (1,0) und B = (3,2) wollen miteinander Daten austauschen. Da sie keine direkten Nachbarn sind, müssen sie über einen weiteren Knoten, z.B. C = (1,2), kommunizieren, welcher ein Kreuzpunkt der beiden Knoten A und B ist. Abbildung 3.5a veranschaulicht dieses Beispiel. [14] (S. 274-275)
Dieser Ansatz stößt auf seine Grenzen, wenn weitere Knoten und somit auch weitere Schlüsselpaare in das Netzwerk eingebunden werden sollen, während das Netzwerk schon in Betrieb ist. Eine Lösung dafür stellen Yumi Sakemi *et al.* (2015) mit ihrem neuen Schema SPIKE (Scalable Peer Intermediaries for Key Establishment) vor. Jeder Knoten hat genau zwei Schlüssel, nämlich K_x und L_y, wobei x und y Koordinatenwerte sind. Die Kommunikation funktioniert ähnlich wie in PIKE. Für zwei Knoten A und B, die keinen gemeinsamen Schlüssel teilen, reicht es einen Kreuzpunkt zu finden (siehe Abbildung 3.5b) , welcher K_x mit A und L_y mit B teilt. Beim Hinzufügen neuer Knoten ins Netzwerk müssen die bestehenden Knoten keine neuen Schlüsselpaare speichern. Für die neuen Knoten werden lediglich neue Schlüsselpaare nach diesem Koordinatenschema generiert und die Matrix somit erweitert. [8]

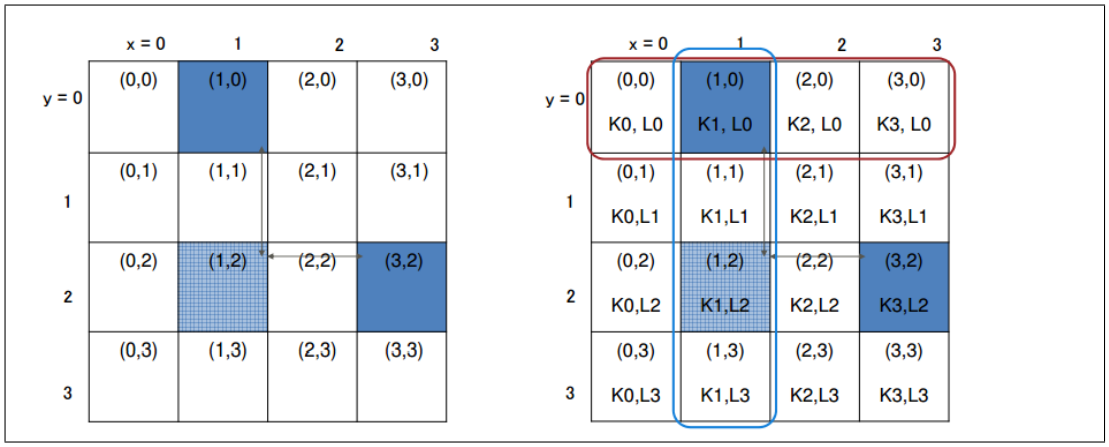


Abbildung 3.5: a) PIKE Key Sharing Matrix b) SPIKE Key Sharing Matrix [8]

3.5.3 IEEE 802.15.4

Der *IEEE 802.15.4*-Standard ist eine beliebte Wahl, wenn es um die Sicherheit in einem WSN geht. Authentifizierung, Vertraulichkeit, Datenintegrität und Schutz vor Replay-Attacken werden vom Standard unterstützt. 8 verschiedene Sicherheitsmodi stehen dabei zur Auswahl, um für jede Anwendung den richtigen Trade-Off zwischen Sicherheit und Energieverbrauch zu finden. Der erste Modus bietet gar keine Sicherheit. Alle anderen verwenden zunächst die AES-Verschlüsselung. Für die reine Verschlüsselung wird im Counter Mode (CTR) operiert. Für die reine Authentifizierung wird eine Kombination aus AES, CBC und MAC (32-, 64- oder 128-bit MAC) verwendet. Will man beides haben, so gibt es eine „AES - CCM“-Implementierung als Möglichkeit. Der CCM-Modus ist dabei eine Kombination aus dem CTR zur Verschlüsselung und dem CBC-MAC zur Integritätssicherung. Der MAC kann wieder zwischen 32 und 128 bit gewählt werden (siehe [14] S. 280-281). IEEE 802.15.4 basiert neben den Operationsmodi auch auf zwei weiteren wichtigen Elementen: den *Auxiliary Security Header* (ASH) und den *Security Procedures*. Der ASH wird bei der Datenübertragung neben dem Standard-MAC-Header eingefügt. Dieser erhält Informationen wie den ausgewählten Sicherheitsmodus, den Frame-Counter-Wert für Anti-Replay-Services und den KeyIdMode für die Art der Schlüsselerwerbung. [50] Durch die Übertragung und des Management des ASH entstehen Overheads, die letztendlich auch zu Performanceeinbußen führen. Zusätzlicher Speicherplatz ist außerdem für die *Security Procedures*, die Datenstrukturen und die Operationen zur Verwaltung der Datenstrukturen notwendig. (R. Daidone; Open-Source-Projekt 2011) [51]

3.6 Betriebssysteme

3.6.1 TinyOS

TinyOs ist eine ereignisgesteuerte, weit verbreitete, umfassend dokumentierte und toolunterstützte Laufzeitumgebung in drahtlosen Sensornetzwerken. Seine kompakte Architektur macht es für viele Applikationen geeignet. Es besteht aus einem Scheduler und einem Set von Komponenten, welche miteinander über gut definierte Schnittstellen verbunden werden können. Eine Komponente kann entweder ein *Modul* oder eine *Konfiguration* sein. Eine Konfiguration beschreibt, wie die Module miteinander verbunden werden. Module sind grundlegende Baublöcke eines TinyOS-Programms. TinyOS unterstützt keine klare Trennung zwischen dem Betriebssystem und dem Anwendungsprogramm.
Eine Komponente ist vergleichbar mit einem Objekt in einer OO-Programmiersprache, da es einen Zustand speichert und über Schnittstellen mit anderen Komponenten kommuniziert. Es besteht aus folgenden Elementen:

- Frame
- Command Handlers
- Event Handlers
- Tasks (Aufgaben)

Komponenten sind hiearchisch aufgebaut und verwenden *Commands* (Befehle) und *Events*, um miteinander zu kommunizieren. Höhergestellte Komponenten geben darunterliegenden Komponenten Befehle und letztere signalisieren ersteren auftretende Events. Daraus ergibt sich, dass höhergestellte Komponenten *Event Handler* und darunterliegende Komponenten *Command Handler* implementieren müssen. Genauer gesagt, werden *Tasks* verwendet, um diese Kommunikation zu ermöglichen. *Tasks* können aber auch andere *Tasks* (inkl. sich selber) in den Scheduler schicken. Ein gutes Beispiel ist ein Task, der für das Lesen von Paketen im Kommunikationssubsystem verantwortlich ist. Dieser Task wird sich selber solange aufrufen, bis alle Pakete gelesen wurden. Ein wichtiges Merkmal von *Tasks* ist, dass sie von anderen Tasks nicht unterbrochen werden können (sehr wohl aber von Events) und bis zum Ende ausgeführt werden müssen. Auf diese Art wird die Nebenläufigkeit in TinyOS unterstützt. *Tasks* werden nach dem FIFO-Prinzip ausgeführt. Generell ist TinyOS etwas inflexibel und weniger anpassungsfähig. Für die Ressourcenallokation verwendet es einen statischen Speicher. Ein statischer Speicher hat den Vorteil, dass zusätzlicher Overhead, welcher beim dynamischen Speicher entstehen würde, nicht entsteht, wenn die Anforderungen an den Speicher schon vor der Übersetzung bekannt sind. Auf der anderen Seite unterstützt ein statischer Speicher während der Laufzeit keine Adaption. Da es keine klare Trennung zwischen dem Betriebssystem und der Anwendung gibt, können Komponenten ohne weiteres nicht einfach ausgetauscht werden, was die weniger vorhandene Anpassungsfähigkeit untermalt. [14] (S. 75-78)

3.6.2 LiteOS

LiteOS ist ein Thread-basiertes Betriebssystem und unterstützt eine klare Trennung zwischen der Systemebene und den Applikationen, die darauf laufen. Im weiteren Vergleich zum TinyOS müssen beim LiteOS keine fix gelieferten Komponenten miteinander verbunden werden, um eine Applikation zu erstellen. Die Entwicklung von Bausteinen und deren Interaktionen miteinander bleibt ganz dem User überlassen.
Die Architektur des LiteOS besteht aus drei Subsystemen: dem Kernel, dem Dateisystem (LiteFS) und der Nutzerumgebung inklusive Shell.
Die Basis stellt der **Kernel** dar. Dieser besteht aus einem *Scheduler*, einem Set von *System Calls* und einem *binary installer*. Der Scheduler implementiert sowohl prioritätenbasiertes Scheduling als auch das Rundlauf-Verfahren (engl. Round Robin). System Calls können verwendet werden, um einen entfernten Benutzer auf lokale Daten zugreifen zu lassen. Diese Technik kennt man auch unter dem Begriff RPC (Remote System Calls). Die Trennung zwischen Kernel und Applikation wird von sogennanten *call-gates* unterstützt. Sie sind der einzige Weg, über welchen User auf Systemressourcen zugreifen können.
Weiters wird auch *dynamic loading* und *un-loading* unterstützt, wenn es um die Umprogrammierung von Benutzeranwendungen geht. Zwei Verfahren werden beim *dynamic loading* unterschieden. Im ersten wird angenommen, dass der Sourcecode am Betriebssystem existiert. In diesem Fall wird das Programm mit neuen Speichereinstellungen rekompiliert und alle Pointer und Referenzen zu der alten Version werden dabei umgeleitet. Ist der Sourcecode nicht vorhanden, so wird ein „differential patching“-Mechanismus angewandt. Informationen zur Umplatzierung werden direkt in die Binärdateien kodiert, indem differenzielle Patches eingefügt werden.
LiteFS ist ein verteiltes Dateisystem und ein wesentlicher Bestandteil von LiteOS. Es verwendet einen drahtlosen „Node mounting“-Mechanismus, welcher Sensorknoten ermöglicht, sich selber an das Root-Dateisystem der nächsten Basisstation zu binden. Somit kann das ganze Netzwerk wie ein hierarchisches und verteiltes Dateisystem gesehen werden. Ein Nutzer kann somit auf eine entfernte Datei zugreifen, als ob sie lokal verfügbar wäre. Für die Speicherung der Dateistruktur wird ein EEPROM verwendet und für die Speicherung der Daten ein Flash-Speicher. Allokationsinformation zu beiden Speichern werden im RAM gespeichert, sowie auch ein File-Handler, welcher erlaubt, bis zu acht Dateien gleichzeitig offen zu haben. Die genaue Zusammensetzung des EEPROMs und des Flash-Speichers wird hier nicht erläutert.
Die **Shell** stellt UNIX-ähnliche Befehle zur Verfügung und läuft auf der Basisstation. Mit der Shell können Knoten beispielsweise umprogrammiert werden. Wichtig zu erwähnen ist, dass ein Knoten eine zustandslose Komponente ist. Ein Knoten antwortet nur auf übersetzte Nachrichten von der Shell, welche sehr leicht zu parsen sind. Dieses asymmetrische Design erlaubt es nicht nur den Code-Footprint des Kernels auf jedem Knoten zu reduzieren, sondern auch die Shell mit komplexeren Befehlen (z.B. Security) zu erweitern.
Die Befehle werden in fünf Kategorien eingeteilt, diese werden durch folgende Tabelle erläutert:

Befehl	Beschreibung
File Commands	Navigation durch das Dateisystem. Bearbeiten von Dateien und Verzeichnissen
Process Commands	Verwaltung von Threads (Erstellen, Pausieren und Killen)
Debugging Commands	Aufsetzen einer Debugging-Umgebung, um Code zu debuggen
Environment Commands	Verwaltung der OS-Umgebung, Anzeigen des Interaktionsverlaufs, Referenzen für Befehle
Device Commands	Direkter Zugriff auf die Hardware (z.B. Sensor oder Funk)

[52], [14] (S. 85-87)

KAPITEL 4

Structural Health Monitoring

Structural Health Monitoring (SHM) ist eine Methode, mit welcher man den Zustand von Baukonstrukten wie z.B. Brücken, Gebäuden oder Straßen analysiert, um Schäden ausfindig zu machen und potentielle irreversiblen Schäden (z.B. Zusammenbruch einer Brücke) zu vermeiden. SHM-Systeme, welche auf WSNs basieren, verwenden dazu viele Sensorknoten, welche über das gesamte Baukonstrukt verteilt sind und bautypische Daten sammeln, die sie in weiterer Folge an die Basisstation weiterleiten. WSNs gehören zu sogenannten globalen Inspektionstechniken im SHM, welche darauf abzielen, Schäden zu erkennen, die das ganze Baukonstrukt betreffen. Globale Inspektionstechniken kann man als eine inverse Herangehensweise betrachten, das heißt, der „Gesundheitsstatus“ der Konstruktion wird auf Basis seiner Reaktion auf einen externen Stimulus bestimmt. Dieser Stimulus kann natürlich (z.B. Erdbeben, Wind) oder künstlich (z.B. Schlaghammer) herbeigeführt werden. Modale Parameter wie Eigenfrequenzen oder Eigenschwingungsformen werden dabei zur Analyse herangezogen.
Der WSN-Ansatz ist besonders attraktiv, da eine ähnliche Qualität wie bei konventionellen (verkabelten) Systemen erreicht werden kann und zusätzliche Kosten aufgrund von leichteren Wartungsmöglichkeiten eingespart werden. Die Wartung kommt auch nicht mit anderen Operationen in Konflikt, da diese drahtlos durchgeführt werden kann. Außerdem können die kleinen Sensoren an Stellen eingesetzt werden, die für größere und verdrahtete Geräte nicht erreichbar sind.

4.1 Messung von Strukturveränderungen

Seismische Aktivitäten in großen Bauonstrukten umfassen sehr tiefe (Eigen-)Frequenzen (unter 10 Hz). Ein beliebtes Einsatzinstrument, um deren Schwingungen zu messen, ist der Beschleunigungssensor (vgl. „Golden Gate Bridge“-Projekt [11]). Belastungen und Verformungen werden meistens mittels Dehnungsmessstreifen erfasst. Diese sind auch in Kombination mit drahtlosen Sensoren zu finden [53]. Kipp- und piezoelektrische Sensoren finden ebenfalls Anwendung im SHM, auf diese wird hier jedoch nicht eingangen werden.

4.1.1 MEMS-Beschleunigungsmesser

MEMS-Beschleunigungsmesser sind Feder-Masse-Systeme, bei denen die „Federn“ nur wenige μm breite Silicium-Stege sind und die Masse auch aus Silicium hergestellt ist. Die Messung erfolgt kapazitiv, indem bei Beschleunigung die Masse bewegt wird und dabei anliegende Kondensatoren verändert werden. Die Kapazität wird dabei in Spannung umgewandelt, welche direkt proportional zur Beschleunigung ist. [54] [55] Beschleunigungssensoren können sowohl die statische (Gravitation) als auch die dynamische (Bewegung oder Vibration) Beschleunigung messen. [56] Abgesehen von MEMS-Beschleunigungssensoren gibt es auch noch weitere, nicht auf MEMS beruhende, Beschleunigungssensoren. Diese bleiben der MEMS-Technologie jedoch oft unterlegen. [54] In WSNs sind Beschleunigungssensoren oft Teil von Sensor-Boards, welche mehrere Sensoren, eine Core Unit (z.B. Mikrocontroller) und einen Transmitter für die Kommunikation zwischen anderen Boards (Knoten) und der Basisstation, besitzen.[57] Manchmal ist auch ein Thermometer angebracht, um die Temperatur vom Board zwecks Kühlung zu messen. [11] Da beim Messvorgang oft Störgeräusche mitaufgenommen werden können, werden die Sensoren mit sehr hohen Frequenzen überabgetastet.

4.1.2 Dehnungsmessstreifen

Dehnungsmessstreifen sind im *Structural Heal Monitoring* weit verbreitet. Neben der Tatsache, dass man sie günstig erwerben kann, sind sie auch leicht anzubringen und empfindlich genug, um einen potentiellen Zusammenbruch einer (zivilen) Einrichtung zu erkennen. Oftmals werden DMS-Streifen verdrahtet oder in Verbinung mit single-hop drahtlosen Knoten eingesetzt. Der Nachteil dabei ist, dass die Anzahl der Sensoren limitiert ist und damit die Erweiterbarkeit des Systems beeinträchtigt wird. Multihop-Systeme, wie das von Haksoo Choi *et al.* spezifizierte, funktionieren einwandfrei und liefern Messwerte, die den Werten von Industriestandardgeräten gleichen. Das Prinzip dabei ist die Verwendung von Sterntopologien mit Sensorboards inklusive DMS-Streifen an den Enden und einem Sammelknoten in der Mitte. Der zentrale Knoten sammelt sämtliche Daten von allen lokalen Teilnehmern und übermittelt diese an die Basisstation. Die Übermittlung kann dabei über mehrere Hops erfolgen.
Ein Problem beim Interpretieren der Daten ist, dass nach Einsatz eines Analog-Digital-Wandlers die Integrität aufgrund der Störgeräusche beim Rohsignal verloren gehen kann. Viele Sensorboards verwenden dabei einen Tiefpassfilter, um die Störgeräusche nicht mitaufzunehmen. [53][11] Eine weitere Herausforderung ist die verlustloste Übermittlung der Daten an die Basisstation. Das Protokoll in [11] vorgestellte Protokoll löst dieses Problem mit klassischen ACK-Signalen und einem Split der Pakete. Datenpakete werden in mehreren Runden gesendet, das letzte Paket der Runde fragt nach einer Bestätigung. Bei einem Fehler werden nur die Pakete der Runde gesendet, in welcher die Pakete verloren gegangen sind.

4.2 Schadenserkennung mittels Eigenfrequenzen

Diese Technik berechnet die Korrelation zwischen der gemessenen und der prognostizierten Eigenfrequenz (Hypothese), um den Schaden zu bestimmten. Die Parametervektoren, welche für die Evaluierung der Korrelationskoeffizienten verwendet werden, bestehen aus der Rate der ersten n Eigenfrequenzänderungen aufgrund eines Schadens in der Struktur. Diese Änderungen lassen sich als $\Delta\omega = (\omega_h - \omega_d)$ darstellen, wobei ω_h und ω_d die Eigenfrequenzvektoren der unbeschädigten und beschädigten Strukturelemente kennzeichnen. Um den Vektor $\delta\omega$ der prognostizierten Eigenfrequenzen aufzustellen, wird ein analytisches Modell verwendet. Dieser Vektor wird verwendet, um die Schadensstellen und das Schadensausmaß abzuleiten.
Mit diesen Vektoren lässt sich der Grad der Korrelation auf mehrere Arten schätzen. Die einfachste Möglichkeit ist, den Winkel zwischen ωh und ωd zu berechnen. Eine weitere Möglichkeit besteht darin, die lineare Korrelation zwischen den Eigenfrequenz-Variationsvektoren zu finden. Folgende Gleichung führt zu diesem Ziel:

$$C_j = \frac{\Delta\omega^T \delta\omega_j}{|\Delta\omega||\delta\omega_j|}$$

Dabei ist j (1,2,...,r) ein Indikator für die prognostizierte Schadensstelle.
Eine ähnliche und beliebte Gleichung ist das *damage localization assurance criterion* (DLAC). Diese Methode vergleicht zwei Eigenfrequenz-Variationsvektoren (den auf den Messungen basierenden Vektor und den Vektor der j-ten Hypothese aus dem analytischen Modell) um den Grad der Korrelation zwischen den Parametervektoren zu bestimmten. Das DLAC wird wie folgt beschrieben:

$$DLAC_j = \frac{|\Delta\omega^T \delta\omega_j|^2}{|\Delta\omega^T \Delta\omega_j||\delta\omega_j^T \delta\omega_j|}$$

Hat die Struktur mehrere oder eine ungewisse Anzahl an Defekten, so ist das DLAC nicht mehr ausreichend. In diesem Fall bedient man sich des MDLAC (Multiple DLAC), welches aus einer Sensitivitätsmatrix des analytischen Modells hergeleitet werden kann (siehe. [14] S. 19-20) Die Erläuterung dessen würde jedoch den Rahmen dieser Arbeit sprengen.

4.3 Schadenserkennung mittels Eigenschwingungsformen

Schadenserkennung mittels Eigenschwingungsformen wird meistens dann eingesetzt, wenn der Schaden an mehreren Stellen auftritt (Multiple damage detection). Das MDLAC, welches auf Eigenfrequenzen basiert, erweist sich als ineffizient, weil sämtliche Kombinationen von Schadensvariablen, welche diesen maximieren, evaluiert werden müssen. Im Gegensatz zu Eigenfrequenzen, bieten Eigenschwingungsformen räumliche Informationen für einen gegeben Freiheitsgrad. Andererseits können auf Eigenschwingungsformen basierende Schadenserkennungstechniken das Schadensausmaß weniger genau bestimmten. Ein weiteres beliebtes Kriterium für die Identifikation ist das *Modal assurance criterion* (MAC). Dieses

Kriterium bildet das Skalarprodukt zwischen zwei normierten Eigenvektoren nach der Vorschrift:

$$DLAC_j = \frac{|\phi_1 \phi_2|^2}{|\phi_1 \phi_1||\phi_2 \phi_2|}$$

Das MAC wird immer zwischen Paaren von Schwingungsformen, nämlich der beschädigten und unbeschädigten, bestimmt. Somit ist die Anzahl der verfügbaren MAC-Werte gleich der Anzahl der gemessenen Schwingungsformen. Für die Variationen der identifizierten und prognostizierten Schwingungsformen werden Vektoren gebildet. Dadurch kann man wie bei den Eigenfrequenzen wieder die lineare Korrelation schätzen. [14] (S. 19-20)

4.4 Erkennung von fehlerhaften Sensorsignalen

Viele Fehler, die mit der Zeit in SHM-Sensoren auftreten, haben direkten Einfluss auf die Schadenserkennung. So können Schäden in der Struktur nicht erkannt (falsch negativ) oder unbeschädigte Stellen als Schaden (falsch positiv) gemeldet werden. Prominente Schemen (z.B decision fusion, value fusion, heartbeat reception) für fehlertolerante Ereigniserkennung sind keine optimale Lösung für SHM-Systeme, da diese komplett verschiedene Ansätze zur Schadenserkennung verwenden. Hingegen haben Algorithmen wie NFMC [58] und SPEM [59] weitaus mehr Erfolg, aber auch deren Ressourcennutzung ist verbesserungswürdig. Unter Berücksichtigung der Probleme der zivilen Baueinrichtungen und auch der Herausforderungen der WSNs stellten Md Zakirul Alam Bhuiyan *et al.* eine Lösung vor, welche sie mit soeben genannten Algorithmen verglichen haben. Die Auswertung zeigte, dass diese dabei in Energieverbrauch, Fehlererkennungsfähigkeit und Fehlererkennungsgenauigkeit übertroffen wurden.[60] Die Idee dahinter wird nun im Folgenden beschrieben.

4.4.1 Mutual Information Independence

Transinformation (engl. Mutual Information) ist eine Größe aus der Informationstheorie, die die Stärke des statistischen Zusammenhangs zweier Zufallsgrößen angibt. In diesem praktischen Fall wird der MII-Wert verwendet, um die statistische Abhängigkeit zwischen zwei Sensorsignalen unterschiedlicher Sensoren zu quantifizieren. Die Basisidee beruht auf der Messung der Informationen eines Sensors über einen anderen Sensor in einer Signalmenge in D (:= Submenge von Sensoren), durch die Funktion $\omega(y_i, y_j, C)$. y_i und y_j sind dabei zwei Signale und C das Korrelationsmodell. Ein auf der Normalverteilung basierendes Korrelationsmodell wird für die Messung der Korrelation zweier Signale gewählt. Man kann zeigen, dass sich ω ändert, sobald ein Fehler in einem Sensor auftritt, weil das fehlerhafte Signal aus der Signalmenge ausgeschlossen wird. Genauer gesagt, ist diese Änderung eine Reduktion, die sich aus einer ω-Matrix für die berechneten MII-Werte möglicher Sensor-Output-Kombinationen erkennen lässt. [60]

4.4.2 Centralized fault detection under localized processing (CFD-LP)

Bei der zentralisierten Fehlererkennung ist die Basisstation für diese zuständig. In jedem Entscheidungszyklus entscheidet die Basisstation anhand von k zuletzt erhaltenen Signalen. Dabei wird der MII-Wert (Mutual Information Independence) für jedes Signal berechnet und das Signal mit der maximalen Unabhängigkeit für die Fehlererkennung ausgewählt. Der zentralisierte Ansatz ist für ressourcenbeschränkte und große WSNs nicht geeignet, da eine große Menge an Daten an die Basisstation gesendet werden muss. Deswegen sollten die Daten in jedem Sensorgerät vor dem Absenden reduziert werden. [60]

4.4.3 Distributed fault detection under localized data processing (DFD-LP)

Die Fehlererkennung kann auch verteilt erfolgen. Dabei entscheidet jeder Sensorknoten, anhand der gesammelten Informationen von seinen Nachbarn, ob dieser fehlerhaft ist oder nicht. Wird ein bestimmter MII-Wert überschritten, so stuft sich der betroffene Sensorknoten selber als fehlerhaft ein und kommuniziert das neue Set an fehlerhaften Sensoren an seine Nachbarn. Die verteilte Methode erfordert nur die Synchronisierung der Nachbarn. Ein weiterer Vorteil ist, dass die Erkennung viel schneller erfolgt, da die Sensoren nur auf Signale aus ihrer unmittelbaren Umgebung warten müssen (ein Hop). Außerdem muss eine Fehlermeldung nicht an die Basisstation übermittelt werden, was Energiekosten spart. [60]

4.5 Rekonstruierung von fehlerhaften Sensorsignalen

Verlustbehaftete Übertragung ist ein häufiges Problem für Überwachungssysteme basierend auf WSNs. Verlässliche Kommunikationsprotokolle, welche verloren gegangene Datenpakete erneut senden, sind eine Möglichkeit, um dieses Problem zu beheben. Jedoch basieren diese Protokolle auf bidirektionaler Kommunikation (NACK/ACK) und können aufgrund des erhöhten Datenverkehrs ineffizient für WSNs sein, da diese mit Energieproblemen zu kämpfen haben. Eine alternative Herangehensweise ist es, den Datenverlust bis zu einem gewissen Grad zu erlauben und verloren gegangene Daten mithilfe von Algorithmen zu rekonstruieren. *Kalman Filter* und *Compressive Sensing* (CS) sind solche Alternativen und werden in diesem Kapitel erläutert.

4.5.1 Rekonstruierung mittels Kalman Filter

Der Kalman Filter ist ein Satz mathematischer Gleichungen, mit dessen Hilfe man bei Vorliegen fehlerhafter Beobachtungen Rückschlüsse auf den Zustanden vieler (dynamischer) Systeme schließen kann. Dieser basiert auf eine Zustandsraummodellierung des Systems, welche wiederum sämtliche Beziehungen der Zustandsgrößen, Eingangsgrößen und Ausgangsgrößen in Form von Matrizen und Vektoren dargestellt.

Der Systemzustand wird anhand des gewichteten Durschnitts von dem tatsächlich gemessenen Wert und dem prognostizierten Systemzustand für die aktuelle Zeitperiode geschätzt. Die Gewichtung erfolgt anhand von Unsicherheiten in jeder Zeitperiode. Je niedriger die Unsicherheiten, desto höher die Gewichtung (z.B. Kalman gain). Um die Unsicherheiten zu bestimmten, bedient man sich der Kovarianz-Matrizen, welche für die beiden wichtigen Parameter - Messrauschen und Prozessrauschen - erstellt werden.
Eine hohe Messrausch-Kovarianz bedeutet, dass die Differenz zwischen den gemessenen und geschätzten Siganlwerten klein ist und die originalen Signale (auch die fehlerhaften Signale) wiederhergestellt werden können. Trifft ein Sensorsignal nicht auf das modellierte System zu und wurde auch noch fälschlicherweise eine niedrige Kovarianz angenommen, so können die Signale nicht rekonstruiert werden, da oben erwähnte Differenz zu groß wäre. [60]

4.5.2 Rekonstruierung mittels Compressive Sensing und Random Demodulator

Die Grundidee von *Compressive Sensing* beruht auf Transformierung und Komprimierung des Signals. Statt das rohe Signal, welches der Sensor gemessen hat, zu senden, wird dieses auf eine zufällig generierte Matrix projiziert. Diese Matrix erlaubt eine Komprimierung des Signals bis zu einem gewissen Grad. Das heißt, dass aus dem transformierten Signal das rohe Signal wieder gewonnen werden kann, solange die Verlustrate eine bestimmte Grenze nicht übersteigt. Abbildung 4.1 veranschaulicht diesen Vorgang.
In der Theorie und in Simulationen wurde dieses Prinzip schon erfolgreich angewandt. In der Praxis ist CS nicht leicht umzusetzen, da das Speichern der Matrizen den Speicher vieler Sensorknoten ausschöpft. Größere Matrizen sind aber vom Vorteil, da diese einen größeren Datenverlust erlauben. Zilong Zou *et al.* verwenden eine Methode namens *Random Demodulator* (RD), um die Herausforderungen der Speicherbeschränkung und des Energieverbrauchs zu bewältigen. Diese Methode ist nicht neu, die Ergebnisse der Operationen (Demodulation, Low-pass filtering und Sampling) müssen aber in Form einer Matrix dargestellt werden. Der Vorteil von RD ist, dass die Nyquist-Rate unterschritten werden kann. Sei B die Bandbreitenlimit von einem Signal und K die Gesamtanzahl der Frequenzkomponenten, dann wird bloß eine Samplingrate von $O(Klog(B/W))$ benötigt, welche deutlich geringer als die Nyquist-Rate von $2W$ ist. Demnach wird auch eine deutlich kleinere Matrix und damit eine höhere Verlusttoleranz im Vergleich zur CS-Methode benötigt. Diese Methode wurde erfolgreich auf einem *Imote2* getestet. [9]

4.6 Middleware

SHM-Anwendungsprogrammierer - meistens Bauingenieure - sind generell nicht mit dem Betriebssystem eines WSNs (z.B. TinyOS) vertraut und haben daher auch wenig Bezug zu den Komplexitäten (Energiemanagement, drahtlose Kommunkation, ...) eines WSNs. Eine Middleware, welche zwischen der Anwendungs- und WSN-Systemschicht liegt, kann eingesetzt werden, um komplexe Hardwarebefehle zu abstrahieren, Netzwerk-

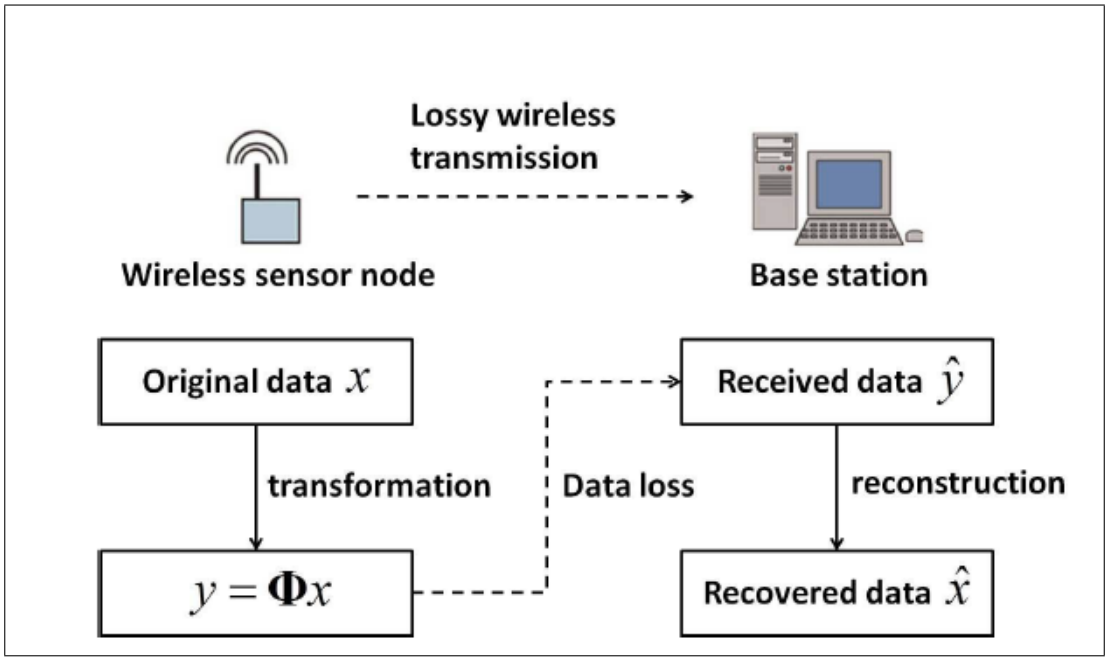


Abbildung 4.1: CS-based data loss recovery for wireless smart sensors [9]

ressourcen optimal zu managen und verschiedenartigen Anforderungen an die Anwendung/Anwendungsschicht zu genügen.
Es gab bereits viele Versuche eine Middleware für WSNs zu entwickeln, welche mit allen Herausforderungen umgehen kann. Dieses Gebiet lässt jedoch noch genug Platz für weitere Forschungen. Immer populärer wird dabei der „Serviceorientierte Architektur"-Ansatz (SOA) aufgrund der Möglichkeit, Dienste voneinander unabhängig einzusetzen, diese außerdem beliebig zu gestalten und die Middleware um neue Dienste zu erweitern. Im Weiteren werden einige Middleware-Ansätze und ein „state-of-the-art"-Middlewarekonzept basierend auf SOA (Yuvraj Sahni 2016 *et al.*) beschrieben. [10]

4.6.1 Middleware-Ansätze

Datenbank-Ansatz: Dieser Ansatz sieht das WSN als eine Datenbank an und bietet Anwendungen die Möglichkeit, Daten per SQL von den Sensorknoten herauszulesen. Der Hauptnachteil dieser Lösung ist, dass es keine Unterstützung für Raum-Zeit-Beziehungen zwischen Ereignissen bietet. Außerdem können die Daten nicht in Echtzeit wiedergegeben werden und die Ergebnisse sind nur approximativ. *TinyDB*, *SINA*, *TinyLIME*, *DSWare* sind Beispiele für Middleware-Architekturen basierend auf diesem Ansatz. [10]

Nachrichtenorientierter Ansatz: Der nachrichtenorientierte Ansatz verwendet das „publish/subscribe"-Kommunikationsmodell, um asynchrone Kommunikation zwischen Sender und Empfänger bereitzustellen. Dieser Ansatz ist für WSNs, welche ereignisbasiertes Monitoring benötigen, sehr effizient. Einige Beispiele dafür sind *TinyDDS*,

PS-QUASAR und *Mires*. [10]

Modulare Ansatz: Der modulare Ansatz basiert auf der Tatsache, dass WSNs eine dynamische Netzwerktopologie haben und anpassungsfähige Applikationen benötigen, um mit Fehlern klarzukommen. Die Anpassung einer Anwendung kann auf verschiedene Arten durchgeführt werden. Eine Möglichkeit ist es, eine monolithische Software zu schreiben, welche sich an sämtliche Szenarien anpasst, die zum Zeitpunkt der Entwicklung bedacht wurden. Das Problem dabei ist, dass so eine adaptive Software schwierig zu entwickeln ist und meistens nicht ohne Fehler kommt. Weiters können auch in Zukunft Szenarien auftreten, die man vorher nicht kannte. Dies erfordert meist große Updates, welche drahtlos kommuniziert werden müssen und aufgrund deren Größe oft fehleranfällig sind. Der Hauptgedanke hinter dem modularen Ansatz ist es, diese Updates in Teilen anzuliefern und so keine großen Codestücke senden zu müssen. Zwei häufig eingesetzte Middlewares sind *Impala* und *Agilla* [10] [61]

Anwendungsorientierter Ansatz: Beim anwendungsorientierten Ansatz sind Applikation und Netzwerk eng aneinander gekoppelt. Applikationen spezifizieren ihre Anforderungen in Form von QoS. Das darunterliegende Netzwerk wird dann von der Middleware dem Bedarf entsprechend adjustiert. Die Middleware muss „wissen“, welche Sensorknoten für die Anforderungen am geeignetsten sind und dementsprechend einen Link von den jeweiligen Knoten zu der Applikation erstellen. Der Nachteil dieses Ansatzes ist, dass die Middleware nicht generisch ist. Beispiele dafür sind *MiLan* und *MidFusion* [62] [63]

Serviceorientierte Architektur (SOA): Die oben beschrieben Ansätze reichen aus, wenn WSNs einfache Probleme, wie leichtere Softwareinstallation oder Datenaggregation, lösen sollen. Die heutigen WSNs erfordern jedoch komplexere Funktionen und unterstützen teilweise mehrere Applikationen gleichzeitig. Eine Anwendung ist nichts anderes, als eine Menge von Services, welche in einer bestimmten Art und Weise miteinander verlinkt sind. Das SOA-Prinzip beruht auf schwach gekoppelten und voneinander unabhängigen Services, welche für verschiedene Funktionalitäten und Applikationen wiederverwendet werden können. Es ist nicht notwendig, die inneren Details eines Services zu wissen, um mit einer neuen Anforderungen klarzukommen.
Der größte Nachteil dieses Ansatzes ist, dass traditionelle Webservice-Standards wie SOAP, WSDL und UDDI nicht eingesetzt werden können, da diese nicht für ressourcenbeschränkte Netzwerke geeignet sind. Viele Techniken (*Byte sequence messaging format*, *Simple service and task description language*, *WSN-SOA-Protokoll*) wurden bereits erfolgreich eingesetzt, um dieses Problem zu bewältigen. Dennoch gibt es auch weitere Herausforderungen (QoS-Versorgungsprozesse, Dynamische Service-Bindung, Gleichzeitiges Deployen von mehreren Anwendungen, Auswahl des richtigen Algorithmus für die Schadenserkennung, ...) in WSNs für SHMs, welche in aktuellen SOA-basierten Middlewares noch nicht gemeistert wurden. [10]

4.6.2 MidSHM

MidSHM ist eine auf dem SOA-Prinzip basierende Middleware, welche eine Vielzahl von Bedingungen an WSNs im SHM berücksichtigt. Eine Implementierung gibt es bisher noch nicht, aber dafür eine genaue Beschreibung der Architektur. MidSHM operiert nach

dem Grundsatz „Separation of concern“ und trennt dabei verschiedene Module, welche auf drei Schichten (*Network Management Layer*, *Task Management Layer*, *Application Management Layer*) verteilt sind. Abbildung 4.2 veranschaulicht dir Architektur von MidSHM.
Der *Network Management Layer* ist hauptsächlich für Netzwerk- und Hardwareangelegenheiten verantwortlich. Der *Task Management Layer* greift auf verschiedene Dienste zu und führt diese unter Betrachtung der vorhandenen Ressourcen aus. Auf der obersten Schicht (*Application Management Layer*) werden domänenspezifische Probleme gehandhabt und verschiedene Services angeboten, welche von der Applikation verlangt werden.
Jede Schicht besteht aus verschiedenen Diensten. Diese Dienste werden von Anwendungsentwicklern mittels des „*Service Consumer*“- und „*Service Producer*“-Moduls verwaltet. Das „*Data Management*“-Modul ist u.a. für die Datenaggregation und dem Filtern von Daten für die „In-Network“-Prozessierung zuständig. Die „In-Network“-Prozessierung wird auch durch das Vorhandensein aller Services im Netzwerk selber erleichtert.
Das Routing Service im entsprechenden Block bietet Routing-Protokolle und die Möglichkeit für das Netzwerk sich selbst zu konfigurieren.
Die Fehlerkontrolle und Festlegung der Fehlertoleranz erfolgt im „*Fault Tolerance*“-Block.
QoS-Anforderungen von Anwendungen werden vom *QoS Interpretation Service* interpretiert, während Anforderungen vom Netzwerk vom *Network Query Service* berücksichtigt werden. Beide Dienste leiten diese Anforderungen an den *Task management layer* für die weitere Prozessierung weiter.
Ressourcenoptimierung ist ein breites Thema, welches den Energieverbauch, die Speicherkapazität, Rechnungszeiten und die Übermittlung berücksichtigt. Das *Resource Management Service* ist in Kombination mit anderen Modulen (*Data Management*, *Power Management*, *Dynamic Memory Management*) an dieser Aufgabe beteiligt.
MidSHM bietet ein Orts- und Zeitbewusstsein bezüglich der Datenmessung, indem das *Location Awareness Service* und das *Time Synchronization Service* eingesetzt werden. [10]

4.7 Anwendungsbeispiel für ein WSN-System im SHM - Golden Gate Bridge

Ein sehr bekanntes Beispiel einer WSN-Anwendung für SHM ist das „Golden Gate Bridge“-WSN, welches von Sukun Kim *et al.* (2007) entworfen, implementiert und getestet wurde. Das 46-hop-System besteht aus 64 Sensorknoten, welche sich über die Südseite der Brücke erstrecken. Für dieses Projekt wurde eine spezielle Software-Architektur und ein eigenes Sensor-Board entworfen, um den Anforderungen von WSNs im SHM gerecht zu werden.

4.7.1 Anforderungen

In diesem Anwendungsbereich ist es einerseits wichtig, die Sampling-Rate so zu wählen, dass die Zuverlässigkeit der akquirierten Daten gewährleistet wird. Zeitsynchronisation, zuverlässige Übermittlung der Befehle und zuverlässige Datensammlung, sowie eine

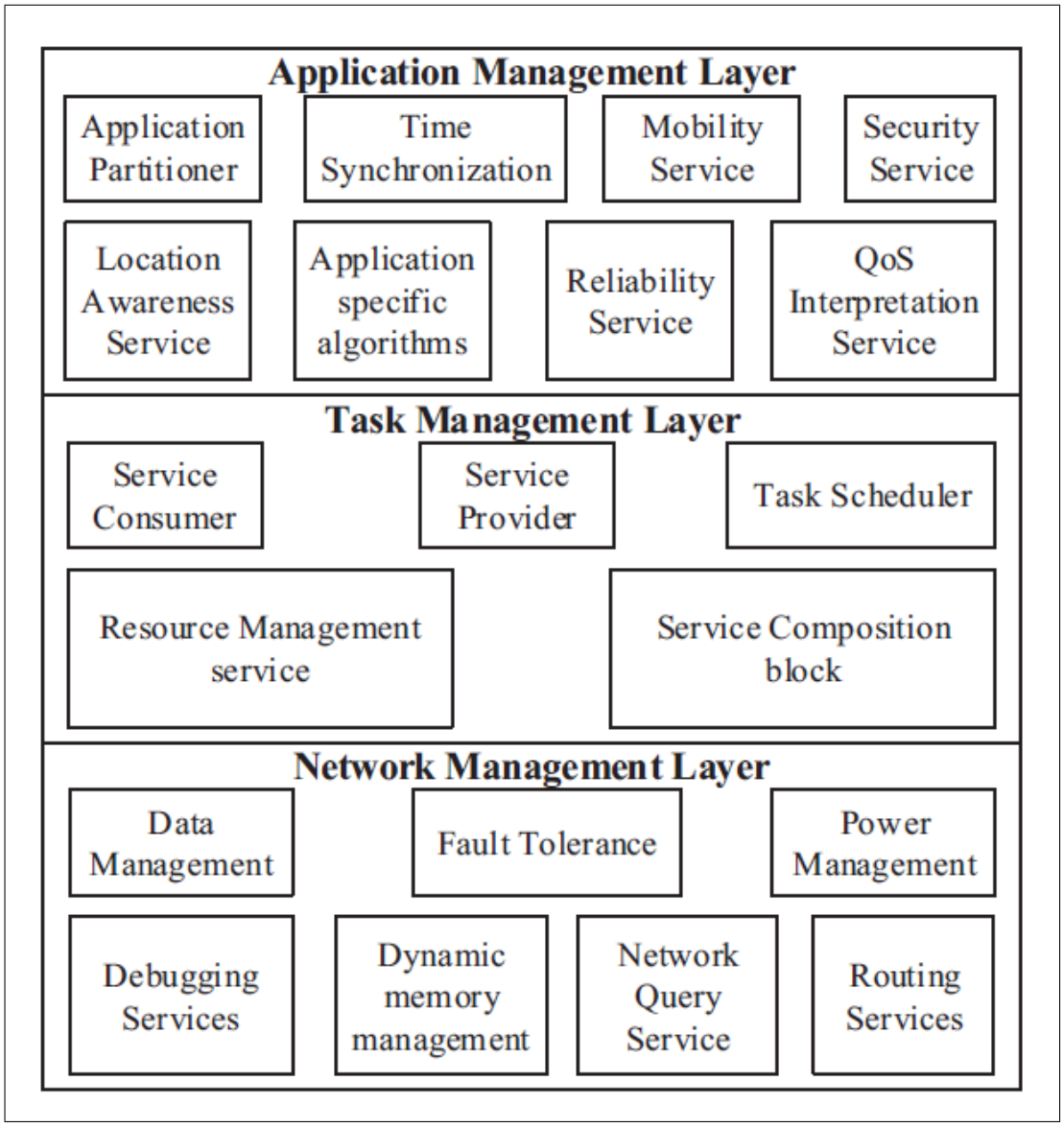


Abbildung 4.2: Proposed Middleware Architecture[10]

stabile Operation des Multi-Hop-Netzes sind in weiterer Folge für ein fehlerfreies Funktionieren der Anwendung essentiell. Um diese Anforderungen zu realisieren, wird die Datensammlung in folgende drei Phasen unterteilt:

1. **Datenakquisition**: Samplet die Vibrationsdaten der Struktur und loggt diese ins EEPROM.

2. **Datensammlung**: Transferiert die Daten verlässlich zur Basisstation.

3. **Datenprozessierung & Feedback**: Analysiert die Daten, ermittelt den Gesundheitszustand der Struktur und sendet, falls nötig, ein Feedback an die Knoten.

Eine speziell entworfene Software wird dabei in diesen Phasen eingesetzt. [64]

4.7.2 Software-Architektur

Abbildung 4.3 zeigt die Software-Architektur. Die Auswahl der Software-Komponenten, die in der Abbildung gezeigt werden, basiert auf den speziellen Anforderungen dieses Projekts und sind in *TinyOS* integriert. Ein Kommunikationsservice mit einer geringen Latenz wird benötigt, damit die Befehle nicht erst ankommen, wenn sie nicht mehr benötigt werden. Diesem Zweck dient das Broadcast-Service vom *TinyOS*. In der Praxis kann dieses Service durch Sendewiederholungen eine Verlässlichkeit von 100% erreichen. *MintRoute* ist ein Multi-Hop-Routing-Protokoll, welches Konvergenz-Routing unterstützt und in diesem Projekt für „Information Reply“ (vergleiche ICMP-Pakettyp 16) verwendet wurde. Für die Datenansammlung wurde das STRAW-Service (Scalable Thin and Rapid Amassment Without loss) implementiert, welches eine Schicht über Broadcast und MintRoute liegt. Die Datenübertragung wird in STRAW immer vom Empfänger initiiert. Der Empfänger ist in dem Fall der PC/Laptop und der Sender ein Knoten. Verloren gegangene Pakete werden vom Empfänger selektiv erneut angefordert. Bei der Wahl des Protokolls zur Zeitsynchronisation wurde das Hauptaugenmerk auf den Jitter gelegt. Dieser durfte eine Grenze von 250μs bei einer Sampling-Rate von 200Hz nicht überschreiten. FTSP (Flooding Time Synchronization Protocol) erfüllte diese Anforderungen (67μs Fehler) und wurde daher für die Zeitsynchronisation eingesetzt. BufferedLog ermöglicht hochfrequentes Sampling und lightweight Logging. Geloggt wird in den Flash-Speicher des *Motes*.
Structural Health Monitoring Toolkit (Sentri) ist ein Anwendungsschichtprogramm und steuert alle beschriebenen Komponenten. Sentri ist wie ein RPC-Server aufgebaut. Für jede Operation wird ein Befehl von der Basisstation zu einem Knoten gesendet. [11]

4.7.3 Sensor-Board

Abbildung 4.4a zeigt eine Übersicht der Hardware als ein Blockdiagramm. Das Board hat vier unabhängige Kanäle mit jeweils zwei Beschleunigungsmessern. Jeder Beschleunigungsmesser hört zwei Richtungen ab (vertikal und transversal). Das Thermometer misst für Kühlzwecke die Temperatur der Sensoren. Der 1221L-Accelerometer ist für die vom Wind und Straßenverkehr verursachten Schwingungen zuständig, während der

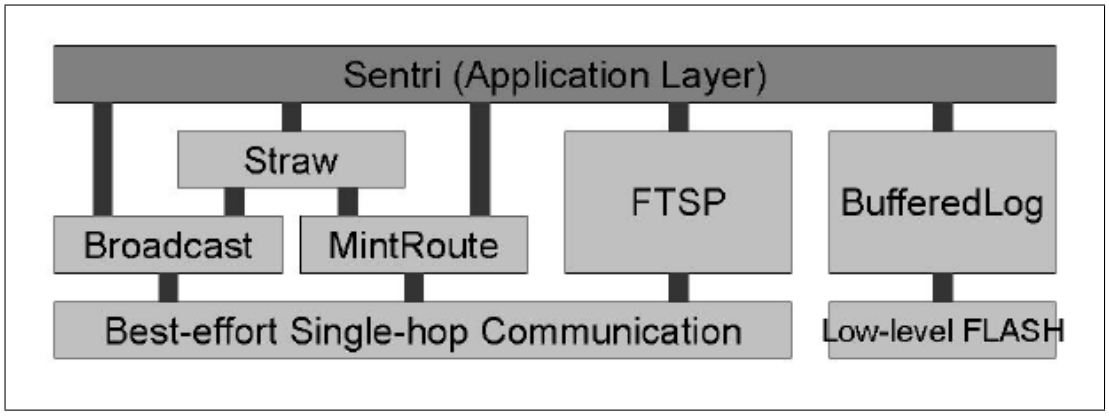


Abbildung 4.3: Overall Software Architecture [11]

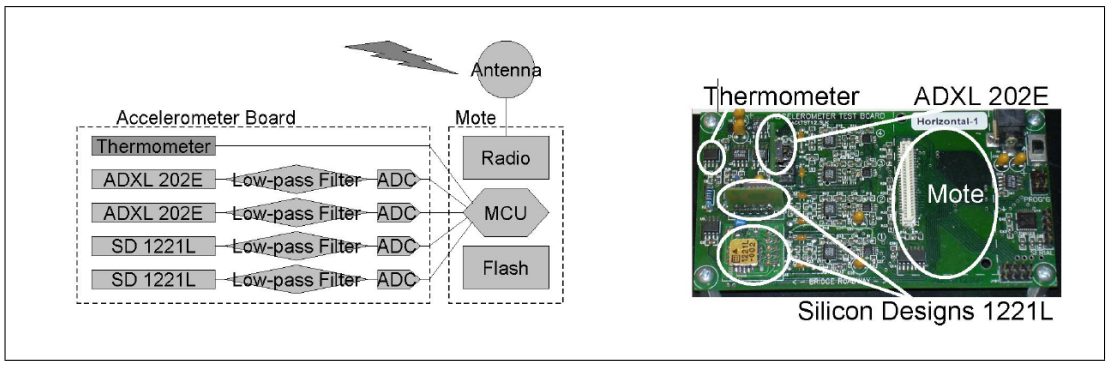


Abbildung 4.4: a) Hardware Block Diagram b) Accelerometer Board [11]

ADXL202E stärkere Erschütterungen (z.B. vom Erdbeben) misst. Die analogen Signale, die die Sensoren als Output liefern, werden durch einen Tiefpassfilter und weiters durch einen 16-bit-ADC gesendet, bevor die Daten dann in den Flash-Speicher geloggt und anschließend per Funk versendet werden. Als Mote hat man sich für den MicaZ statt Mica2 entschieden, da dieser den „IEEE 802.15.4“-Standard verwendet. Abbildung 4.4b zeigt das eigentliche Sensor-Board. [11]

KAPITEL 5

Implementierung

5.1 Einführung

Auf dem Bauingenieurinstitut der Technischen Universität Wien hat man sich im Zuge eines Projekts mit der Messung von Temperatur und Feuchtigkeit von speziellen Bauplatten befasst. Dazu wurden Sensoren auf den einzelnen Testplatten angebracht und mithilfe speziell entworfener Hardware und Software die Werte periodisch ausgelesen. Die Persistierung der Messdaten in reiner Textform war eine unbefriedigende Lösung und führte zum Wunsch, eine Anwendung mit der Datenvisualisierung als Kernfunktion zu entwickeln. Dieses Kapitel befasst sich im Weiteren mit Konzepten, Methodiken und Algorithmen der entworfenen und implementierten Softwarelösung.

5.2 Anforderungen

Neben der Visualisierung der Daten soll die Wunschanwendung auch eine Reihe von weiteren Funktionen bieten.

5.2.1 Funktionale Anforderungen

Einerseits sollen neue Projekte erstellt und konfiguriert werden können. Ein Projekt ist dabei eine Abbildung eines realen Bauplatten-Setups. Dieses beinhaltet daher das Hinzufügen von Bauplatten sowie das Belegen der Bauplatten mit Sensoren, die Auswahl der Messperiode sowie weitere übliche CRUD-Funktionen.
Weiters ist eine grafische Abbildung der erstellen Bauplatten samt Sensoren für das geöffnete Projekt erwünscht. Nach einer Messung müssen die Messwerte dann in diesen Bauplattengrafiken angezeigt werden. Werden mehrere Messungen in einem Projekt durchgeführt, so sollen die einzelnen Messzeitpunkte in einer Liste abgespeichert werden. Bei der Auswahl eines Zeitpunktes werden dann die entsprechenden Messwerte in den

Bauplattengrafiken angezeigt. Ein gestarteter periodischer Messvorgang soll auch jederzeit manuell gestoppt werden können.

5.2.2 Nicht-Funktionale Anforderungen

An das Projekt wurden auch einige nicht-funktionale Anforderungen gestellt. Der verwendete Multiplexer als Bridge zwischen Mikrocontroller und Bauplatten soll durch jeden anderen Multiplexer ersetzbar sein, welcher eine gleichartige Kommunikation mit bis zu 64 Sensoren unterstützt. Das System soll verlässlich periodische Messungen durchführen können. Diese sollen solange laufen, bis es zu einem manuellen Abbruch kommt (oder theoretisch der Festplattenspeicher ausgeht). Wird die Anwendung geschlossen, so soll bei deren Neustart der Scheduler die Messperioden/Messzeitpunkte wiederherstellen können. Die Software soll außerdem für jemanden mit Basiswissen über das Hardware-Setup intuitiv bedienbar sein.

5.3 Methodologie

Dieser Abschnitt beschreibt nun die verwendeten Methoden, Konzepte, Sprachen, Datenmodelle und Algorithmen in diesem Projekt. Die Hardware sowie deren Treiber zur Kommunikation wurden vom Vorprojekt übernommen und ins Projekt integriert. Der bestehende Algorithmus zur Messung wurde nur teilweise wiederverwendet, da einige Änderungen notwendig waren. Alle weiteren Software-Features wurden speziell für die Software neu entwickelt.

5.3.1 Hardware-Setup

Abbildung 5.1 zeigt das verwendete Hardware-Setup zum Testen der Applikation. Der speziell entworfene Multiplexer kann per VGA bis zu sieben Platten mit dem Arduino 2560 MEGA Mikrocontroller verbinden. Jeder Eingang ünterstützt dabei bis zu 9 Sensoren. Die Kommunikation verläuft seriell. Insgesamt können also bis zu 7x9 = 63 Sensoren und ein weiterer Sensor, welcher am Multiplexer selber zwecks Raumtemperaturmessung angebracht ist, angesprochen werden. Zum Testen wurde eine Bauplatte mit vier Sensoren belegt und an den Multiplexer angeschlossen. Der Mikrocontroller ist per USB mit dem Computer verbunden.

5.3.2 Treiber

Der Hardware-Treiber besteht aus drei „C++“-Files, welche am Arduino installiert wurden. Diese sind für die eigentliche Auslesung der Messdaten verantwortlich. Über die serielle Schnittstelle kann man mit einem Befehl den Messvorgang starten. Für die Befehleingabe kann man beispielsweise den *Serial Monitor* vom Arduino oder Putty verwenden. Es folgt eine Abfrage aller 64 Kanäle. Kanäle mit einem Sensor am Ende liefern echte Messwerte für Temperatur und Feuchtigkeit, alle anderen Kanäle liefern

Abbildung 5.1: Hardware Setup

Default-Werte. Mit einem weiteren Befehl besteht auch die Möglichkeit einen Multiplexer-Test durchzuführen, um Informationen über dessen Status zu erhalten.
Der Treiber führt lediglich Schreibbefehle über die serielle Schnittstelle aus. Zum Auslesen der Daten ist dann in weiterer Folge das Python-Skript zuständig. Code-Teile davon wurden in das neue Software-Projekt übernommen.

5.3.3 Datenmodell

Im ersten Projekt wurden die Daten noch in einem simplen .txt-File gespeichert. Der Vorteil dabei war, dass lediglich ein Zeichenkettenformat aber kein Datenmodell benötigt wurde. Als Nachteil aber hat sich die schlechte Lesbarkeit entpuppt. In der neuen Lösung werden die Daten serialisiert in Projekt-Files mit der eigenen Endung „sht“ gespeichert und beim Laden des Projekts wieder deserialisiert. Der Vorteil dieser Lösung ist, dass keine Datenbank benötigt wird. Dennoch ist das serialisierte Objekt eine Kapselung von weiteren Objekten und primitiven Datentypen, welches als Basis ein Datenmodell braucht. Abbildung 5.2 ist eine Darstellung des verwendeten Datenmodells in Form eines Klassendiagramms.
Objekte der Klasse *Project* kapseln einserseits die Projektkonfiguration und andererseits

sämtlich durchgeführte Messungen. Eine Projektkonfiguration ist simpel und beinhaltet den Projektnamen, die erstellen Platten, die Messperiode und den Speicherort. Eine Platte wird grafisch als ein Rechteck dargestellt. Deswegen sind die beiden Attribute *width* und *height* essentiell. Weiters wird noch der Name, die ID und eine Liste von Sensoren gespeichert, die auf der Platte liegen sollen. Sensoren werden grafisch als Quadrate innerhalb der Platten dargestellt. Die Größe dieser Quadrate ist nicht veränderbar. Sie soll den Platzverbrauch des Sensors repräsentieren. Die Sensor-ID gleicht der „Channel-ID“ (= die Nummer des Kanals: von 1 bis 64). Sensoren haben auch eine x- und eine y-Koordinate. Damit ist ihre Position auf der Platte festgelegt.
Eine Messung erfolgt zu einem bestimmten Zeitpunkt und liefert für jeden der 64 Kanäle einen Temperatur- und einen Feuchtigkeitswert. Diese Daten werden in Objekten der Klasse *Readout* gespeichert.
Erfolgt eine Messung, so müssen die ausgelesenen Daten mit den Plattengrafiken gemappt werden, damit in den Sensorkacheln auch die Werte der zugehörigen Sensoren angezeigt werden. Dieses Mapping wird im Abschnitt Algorithmen gezeigt.

5.3.4 Konzepte und Methoden

Datenpersistierung: Über die Methode der Datenpersistierung wurde bereits im vorherigen Abschnitt kurz gesprochen. Anfängliche Überlegungen haben sich jedoch mit einer lightweight SQL-Datenbank für Python (die Sprache in der die Software geschrieben wurde) beschäftigt. Der Verzicht auf eine Datenbank war jedoch eine wichtige Entscheidung, da dadurch das Entwickeln erleichtert und keine unnötigen Zusatzkomponenten gebraucht wurden.

GUI-Framework: Auch beim GUI-Framework wurde zunächst mit Pythons standardmäßiger GUI-Bibliothek *TKinter* experimentiert. Diese erwies sich jedoch als unreif, was zu einem Wechsel zur mächtigen Qt-Bibliothek führte. Qts Signal-Slot-Konzept wurde dann auch im Zuge des Projekts mehrfach eingesetzt, um einen ereignisgesteuerten Programmfluss zu ermöglichen.

Projektstruktur: Die Architektur des Projekts basiert auf dem bekannten MVC-Prinzip. Die Modellklassen wurden alle bereits in Abbildung 5.2 gezeigt und erläutert. Jede Controller-Klasse steht mit einer View-Klasse in Verbindung. Ein Controller „weiß“ von einem View aber nicht umgekehrt. Manche View-Klassen haben jedoch keine eigene Controller-Klasse, da diese Erweiterungen von Qts Standardklassen sind und gewisse Events und Methoden überschreiben bzw. anpassen. Die Geschäftslogik (die Kernfunktionen des Programms) ist in einer separaten Modellklasse implementiert, welche eine Erweiterung des ursprünglichen Skripts ist.

Verlässlichkeit: Damit die Messungen verlässlich periodisch ausgeführt werden, wurde ein Scheduler implementiert. Dieser führt Messungen entsprechend der gewählten Periode in einem eigenen Thread aus (dies ist eine wichtige Änderung im Vergleich zum ursprünglichen Skript). Dadurch wird das Programm nicht blockiert und es können währenddessen andere Aktionen durchgeführt werden. Die laufenden Messungen können durch keine anderen Aktionen im Programm (außer der Stop-Funktion) behindert werden und das Programm liefert somit die Messwerte verlässlich.

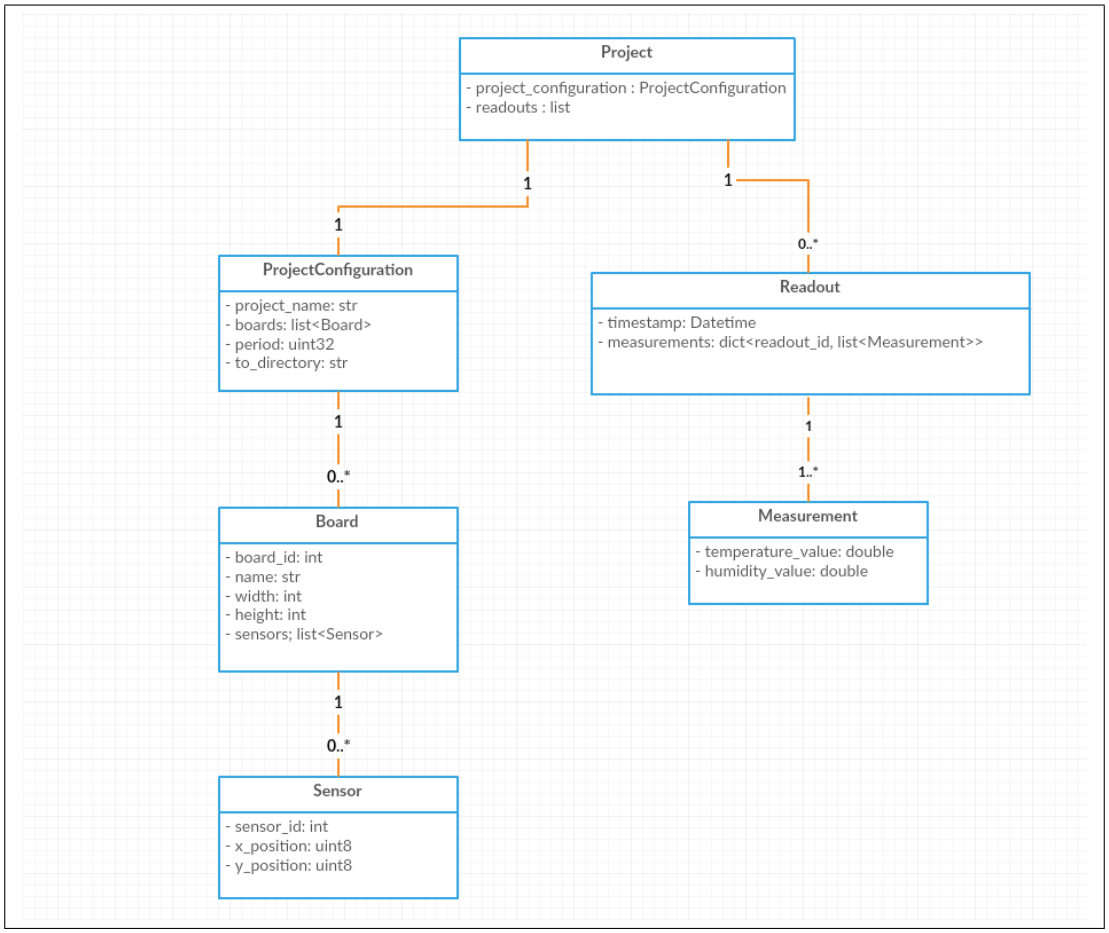


Abbildung 5.2: Klassendiagramm

Robustheit: Unmittelbar nach der Messung wird das Projekt gespeichert, sprich, das *Project*-Objekt wird nach jeder Messung serialisiert. Somit ist keine manuelle Speicherung notwendig und ein Ausfall des Programms/Systems führt zu keinen Datenverlusten. Bei einem Neustart der Anwendung besteht im Falle eines ungewollt abgebrochenen Messvorgangs die Möglichkeit, den Messvorgang fortzusetzen und die nächste Messung zum nächstgeplanten Zeitpunkt durchzuführen.

5.3.5 Algorithmen

Projekt Laden: Folgender Algorithmus in Abbildung 5.3 zeigt, wie ein Projekt geladen wird. Abhängig davon, ob das Projekt bereits konfigurierte Bauplatten hat, kann ein Messvorgang gestartet werden oder auch nicht. Beim wiederholten Laden muss sichergestellt werden, dass der Container, indem die graphischen Bauplatten angezeigt werden, geleert wird. Pro konfigurierte Bauplatte im zuvor erstellten Projekt wird eine graphische Abbildung erstellt. Pro Bauplatte werden ebenfalls graphische Abbildungen der Sensoren

dieser Platte erstellt.
Messung: Das Diagramm in Abbildung 5.4 zeigt einen Messvorgang vom Start bis zur Speicherung des Projekts. Ein Event-Listener in der Klasse *MainController* startet das Skript, sobald über die grafische Benutzeroberfläche der Messvorgang gestartet wird. Das Skript sendet zunächst einen Befehl an den Mikrocontroller, welcher dann über den „gemounteten“ Hardware-Treiber die Sensoren zum Messen auffordert und anschließend die Werte ausliest. Für die Kanäle ohne Sensor wird einfach der Default-Wert 998 zurückgegeben. Die ausgelesenen Werte werden als String formatiert und auf den seriellen Monitor geschrieben. Die Strings, die das Python-Skript anschließend sequentiell ausliest, müssen dann entsprechend geparst werden, um die Temperatur- und Feuchtigkeitswerte zu bekommen. Die Daten werden während dem Messvorgang in entsprechende Objekte verpackt. Ist der Messvorgang fertig, so wird der *MainController* benachrichtigt, welcher abschließend eine Methode startet, um das Projekt zu speichern (die Serialisierung findet hier statt).
Mapping: In diesem Algorithmus geht es um das Mapping zwischen den ausgelesenen Werten und den Sensoren auf den grafischen Bauplatten. Die Vorarbeit findet bereits beim Laden des Projekts statt. Dort werden die nötigen Labels (Platzhalter für die Temperatur- und Feuchtigkeitswerte) schon erstellt und in eine *Dictionary* mit der entsprechend konfigurierten Sensor-ID gespeichert. Wählt man über die grafische Oberfläche eine Messung/einen Messzeitpunkt aus, so müssen diese Labels vorerst bereinigt werden. Dies ist deswegen wichtig, weil ein bestehendes Board auch nach erfolgter Messung mit weiteren Sensoren belegt werden kann. Beim Auswählen einer „alten“ Auslesung bleiben die zu dem Zeitpunkt noch nicht verwendeten Sensor-Labels nach dem erfolgten Mapping einfach leer. Schließlich werden die Messungen der ausgewählten Auslesung durchlaufen und zu den entsprechenden Sensor-Labels gemappt. Abbildung 5.5 zeigt diesen simplen Algorithmus.

5.4 Auslieferung

Das Bauingenieurinstitut der Technischen Universität bekommt sowohl die auf dem Windows Betriebssystem ausführbare Software (exe-File) als auch den Python-Code samt verwendete Libraries überreicht. Sämtliche für dieses Projekt vom Institut zur Verfügung gestellte Hardware wird auch zurückgegeben. Folgende Software-Elemente sind in der Übergabe inkludiert:

- Gesamter Python code mit folgenden libraries:
  - PyQt 5.6
  - pyserial 2.7
  - setuptools 18.2
  - pip 7.1.2
  - numpy 1.11.1
- C++11 drivers: HTU21D.cpp, HTU21D.h, TemperatureHumidity Arduino File
- Arduino Software 1.6.9

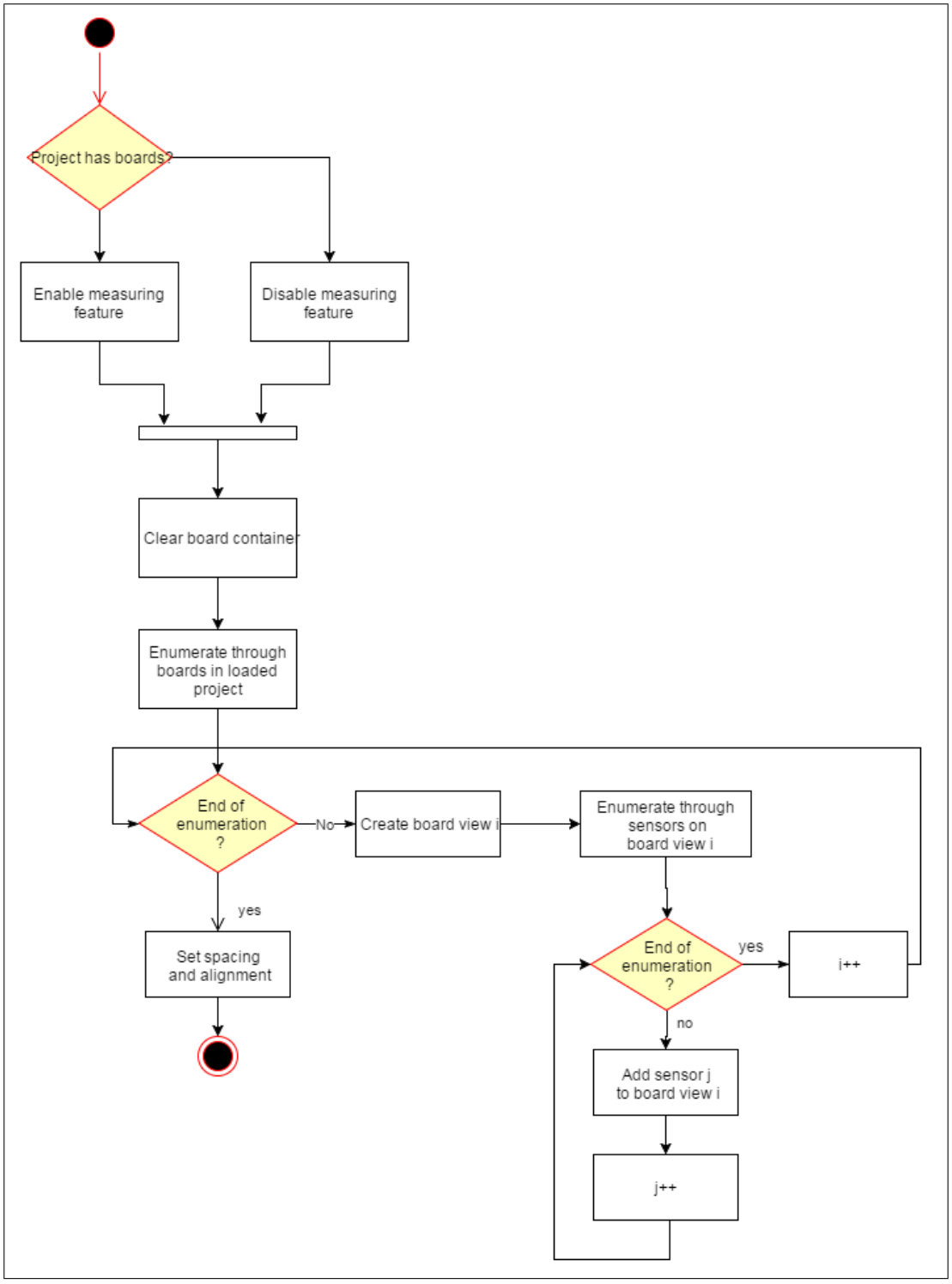


Abbildung 5.3: Load project

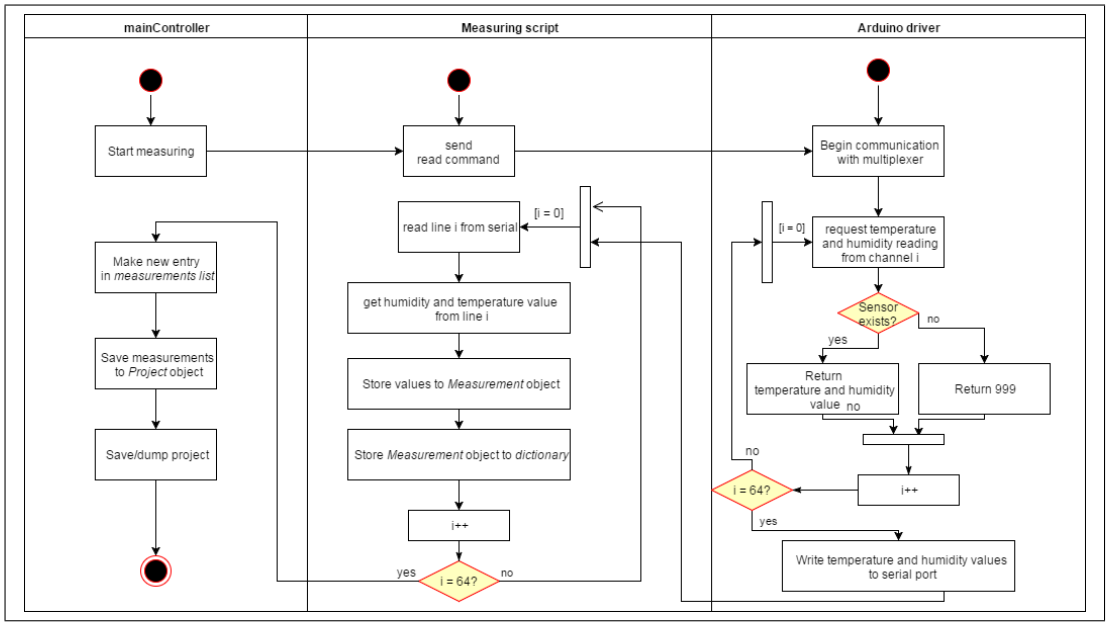


Abbildung 5.4: Activities from measuring start to saving project

Folgende Hardware-Elemente sind in der Übergabe inkludiert:

- Multiplexer (7 VGA outputs x 9 sensors support)
- Bauplatte mit vier Sensoren und VGA-Input
- Arduino 2560 MEGA Microcontroller
- USB-Kabel für die Verbindung zwischen Arduino und PC

Die Software wird auf dem Institutsrechner mit entsprechender Hardware getestet, auf dem die Anwenndung auch für zukünftige Projekte eingesetzt werden soll.

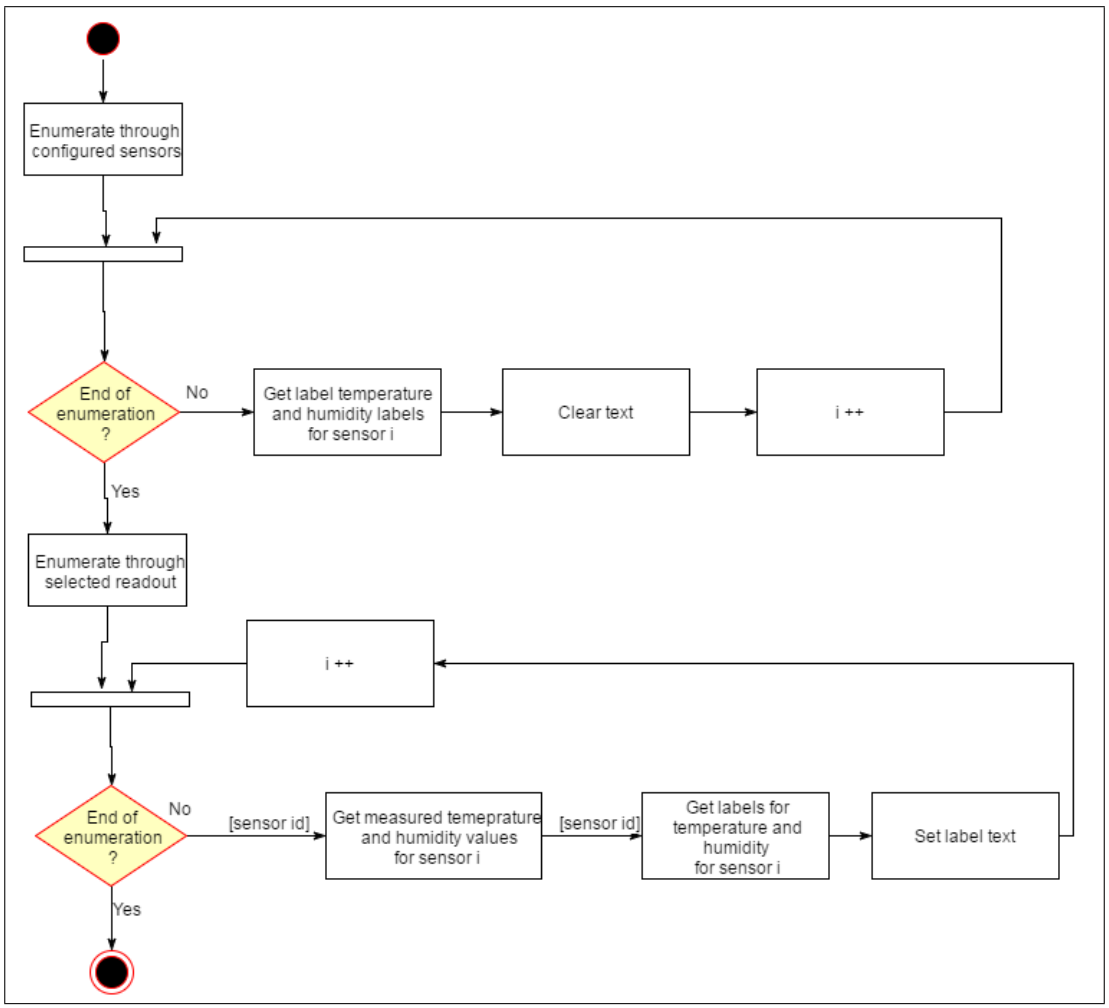


Abbildung 5.5: Mapping between readout values and configured sensors

KAPITEL 6

Kritische Reflexion und Zusammenfassung

6.1 Kritische Reflexion

Diese Arbeit hat sich u.a. ausführlich mit dem Thema MEMS befasst, jedoch die Nanoelektromechanischensysteme (NEMS), welche im Prinzip eine weitere Miniaturisierung der MEMS sind, zur gänze ausgelassen. Schon 1987 gab es ein *call for papers* [65] für beide Systeme - MEMS und NEMS. Logisch ist es, dass die Zukunft mit fortschreitender Verfeinerung mehr auf Nanotechnologie und weniger auf Mikrotechnologie setzen wird. Heute wird in beiden Gebieten noch stark geforscht, vorallem weil die Einsatzgebiete nicht unbedingt dieselben sind (vgl. [66]). In der Literatur werden die Begriffe MEMS, Sensorgerät und Sensor häufig vermischt. Diese Arbeit hat versucht die Begriffe klar zu trennen und richtig einzusetzen.
Im Kapitel Structural Health Monitoring wurden im Vergleich zu erfolgreichen Büchern wie [67], [14] und [38] weniger Anwendungsbeispiele genannt, da sich diese Arbeit einerseits mehr mit der Theorie und den Grundlagen befasst und andererseits speziell auf den SHM-Bereich abzielt. Dafür wurde das bedeutsame Thema „Middlewares“ erläutert und die verschiedenen Ansätze beschrieben, welche in den genannten allumfassenden Büchern nicht vorkommen.
Die erläuterten Algorithmen zum praktischen Teil sowie das Datenmodell sind zwar noch weiter optimierbar, aber sie erfüllen ihren Zweck, da sich mit der gelieferten Performance auf jeden Fall arbeiten lässt. Falls zukünftige Projekte extrem kurze Messintervalle erfordern, könnte man den Messvorgang programmiertechnisch noch weiter beschleunigen, indem man den Hardware-Treiber direkt bearbeitet.

6.2 Zusammenfassung

Sensoren bzw. Sensorgeräte sind Hauptkomponenten von Sensornetzwerken. Ein eingehendes Signal kann dabei auf verschiedene Arten gemessen werden (z.B. anhand des Widerstands, mithilfe von Lichtdetektoren, anhand der Thermoelektrizität), wonach man Sensoren kategorisieren kann. Sensoren kann man aber auch auf viele weitere Arten unterscheiden, z.B. aufgrund des eingesetzten Effekts (Piezoelektrischer Effekt, Optischer Effekt) oder des Einsatzzweckes (Biomedizinische Sensoren, Beschleunigungssensor). Eine Bündelung von Sensoren und weiteren Mikrokomponenten (Aktuatoren, ICs, Membranen) auf einem Chip wird dann unter dem Begriff MEMS verbreitet, welche aufgrund ihrer Vielfältigkeit in unzähligen Gebieten im Einsatz. MEMS leisten auch einen großen Beitrag zu der Verbreitung von verteilten Sensorsystemen. Sensornetzwerke sind einer Vielzahl von Herausforderungen ausgesetzt. Neben den Problemen, welche IT-Rechnernetzwerke auch haben, haben Sensornetzwerke u.a. mit der Einsparung von Energieressourcen zu kämpfen. Dies führt zur Folge, dass sämtliche Protokolle und Methoden nicht einfach adaptiert werden können sondern oft für die Anwendung maßgeschneidert werden müssen. An neuen Methoden und Protokollen wird daher ständig geforscht. Die Überwachung von zivilen Baueinrichtungen, bekannt als Structural Health Monitoring (SHM), ist immer mehr ein beliebtes Einsatzgebiet für Sensornetzwerke. SHM wiederum ist ein eigenes Forschungsgebiet, indem verschiedene Schadenserkennungs- und Fehlererkennungstechniken sowie Methoden zur Rekonstruierung von fehlerhaften Signalen untersucht werden. Diverse Middlewares werden ebenso entwickelt, um SHM-Anwendungsprogrammierern bestimmte Services zur Verfügung zu stellen und somit die Entwicklung zu erleichtern. Auch wenn es noch keine Sensornetzwerke ohne Nachteile gibt und diese oft mit Trade-Offs zwischen Rechenzeit, Sicherheit und Energieverbrauch verbunden sind, steht fest, dass die Forschung immer weiter voranschreitet und mit der Zeit immer effizientere Lösungen kommen und immer mehr Probleme behoben werden.

Literaturverzeichnis

[1] Alois Tipek Pavel Ripka. *Modern Sensor Handbook*, chapter 1.4.2.1.1. ISTE, 2007.

[2] Inhee Lee; Yejoong Kim; Suyoung Bang; Gyouho Kim; Hyunsoo Ha; Yen-Po Chen; Dongsuk Jeon; Seokhyun Jeong; Wanyeong Jung; Mohammad Hassan Ghaed; Zhiyoong Foo; Yoonmyung Lee; Jae-Yoon Sim; Dennis Sylvester and David Blaauw1. *Circuit Techniques for Miniaturized Biomedical Sensors*, chapter Introduction. 2014.

[3] Shizhuo Yin Francis T.S. Yu. *Fiber Optic Sensors*, chapter Basic Concepts and Intensity-Based Fiber Optic Sensors. CRC Press, 2 edition, 2008.

[4] Tai-Ran Hsu. *MEMS and Microsystems: Design, Manufacture, and Nanoscale Engineering*. John Wiley Sons, 2008.

[5] http://www.wisense.in/api/html/SystemOverview.html.

[6] http://www.advanticsys.com/shop/documents/1322654976_MSP430F2618.pdf.

[7] S. Sudevalayam and P. Kulkarni. Energy harvesting sensor nodes: Survey and implications. *IEEE Communications Surveys Tutorials*, 13(3):443–461, Third 2011.

[8] Y. Sakemi, M. Takenaka, and T. Izu. Spike: Scalable peer intermediaries for key establishment in sensor networks. In *Network-Based Information Systems (NBiS), 2015 18th International Conference on*, pages 634–639, Sept 2015.

[9] X. Sun Y. Yu Y. Bao, H. Li and J. Ou. Compressive sampling based data loss recovery for wireless sensor networks used in civil structural health monitoring. volume 12, pages 78–95, 2013.

[10] Y. Sahni, J. Cao, and X. Liu. Midshm: A flexible middleware for shm application based on service-oriented architecture. In *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 126–135, March 2016.

[11] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *2007 6th International Symposium on Information Processing in Sensor Networks*, pages 254–263, April 2007.

[12] T. KvisterØy E. Westby, M. Löhndorf and E. Halvorsen. Evaluation of energy harvesting concepts for tire pressure monitoring systems. 2007.

[13] Mohd Adib Sarijari Nurhija Mahalin Rozeha A. Rashid, Mohd Rozaini Abd Rahim. Design and implementation ofwireless biomedical sensor networks for ecg home health monitoring. 2008.

[14] Christian Poellabauer Waltenegus Dargie. *Fundamentals of Wireless Sensor Networks*. Wiley, 2010.

[15] Ph.D. A.R. Jha. *MEMS and Nanotechnology-Based Sensors and Devices for Communications, Medical and Aerospace Applications*. CRC Press, 2008.

[16] Stoyan Nihitianov. *Smart sensors and MEMS Intelligent devices and microsystems for industrial applications*. Woodhead Publishing, 2014.

[17] R. Taymanov and K. Sapozhnikova. Problems of terminology in the field of measuring instruments with elements of artificial intelligence. *Sensors Transducers*, 102:52, 2009.

[18] Mihaela Duta and Manus Henry. The fusion of redundant seva measurement. *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY*, 13:173, 2005.

[19] Vittorio Ferrari and Ralf Lucklum. *Piezoelectric Transducers and Applications*, chapter Overview of Acoustic-Wave Microsensors. Springer Verlag, 2008.

[20] Tai-Ran Hsu. Mems and microsystems: Design, manufacture, and nanoscale engineering. 2008.

[21] Gerald A. Urban. *BioMEMS*, pages 1–16. Springer, 2006.

[22] Rosario Morello. *Use of TEDS to Improve Performances of Smart Biomedical Sensors and Instrumentation*, chapter Introduction. 2015.

[23] MI USA ; Sechang Oh ; Suyoung Bang ; Yoonmyung Lee Wanyeong Jung; Univ. of Michigan, Ann Arbor. A 3nw fully integrated energy harvester based on self-oscillating switched-capacitor dc-dc converter. 2014.

[24] JR. ERIC UDD, WILLIAM B. SPILLMAN. *Fiber Optic Sensors: An Introduction to Engineers and Scientists*, chapter THE EMERGENCE OF FIBER OPTICSENSOR TECHNOLOGY. Wiley, 2 edition, 2011.

[25] Mark A Mentzer. *Applied optics fundamentals and device applications : Nano, MOEMS, and biotechnology*, chapter MEMS, MOEMS, Nano, and Bionanotechnologies, page 321. CRC Press, 2011.

[26] Nitin S. Kale. Mems: Design, fabrication; their applications as chemical biosensors. *IEEE Computer Society*, page 8, 2015.

[27] S Lucyszyn. Review of radio frequency microelectromechanical systems technology. *Science, Measurement and Technology, IEE Proceedings*, 151(2):93–103, 2004.

[28] Toyota Japan ; Kumagai S. ; Yamashita I. ; Uraoka Y. Jeong, J.-H. ; Toyota Technol. Inst. Vibrational ir mems sensor: Application of torsion-bars tension-enhanced by bio-nano crystallization for highly sensitive detection. *Optical MEMS and Nanophotonics (OMN), 2014 International Conference on*, 2014.

[29] W.C. ; Mantilla B. ; Ivanov D. Perez-Castillejos, R. Hunter. Biomems summer bioengineering institute: Integrating engineering and biology education through biomems design, fabrication, and test. *Integrated STEM Education Conference (ISEC), 2012 IEEE 2nd*, page 1, 2012.

[30] K.A. Jose Vijay K. Varadan, K.J. Vinoy. *RF MEMS and Their Applications*, chapter Introduction. Wiley, 2003.

[31] Onur Ferhanoglu Hamdi Torun Ulas Adiyan, Fehmi C¸ ivitc¸i. A 35-m pitch ir thermo-mechanical mems sensor with ac-coupled optical readout. 2015.

[32] Helmut Budzier Gerald Gerlach. *thermal infrared sensors*, chapter Thermal Infrared Sensors, pages 149–151. Wiley, 2011.

[33] Texas Instruments. *TMP006/B Infrared Thermopile Sensor in Chip-Scale Package*, 2015.

[34] Mir Majid Teymoori. Mems based medical microsensors. *2009 Second International Conference on Computer and Electrical Engineering*, page 159.

[35] Chen Liao and Shiyan Hu. Physical-level synthesis for digital lab-on-a-chip considering variation, contamination, and defect. 13, 2014.

[36] Yehya H. Ghallab and Yehea Ismail. Cmos based lab-on-a-chip: Applications, challenges and future trends. 2015.

[37] Gerald A. Urban. *BioMEMS*, chapter LAB-ON-A-CHIP SYSTEMS FOR CELLULAR ASSAYS, page 279. Springer, 2006.

[38] MOHAMMAD ILYAS and IMAD MAHGOUB, editors. *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems.* CRC Press LLC, 2005.

[39] PIERGIUSEPPE DI MARCO. *Protocol Design and Implementation for Wireless Sensor Networks*, chapter Wireless Sensor Networks: an Overview, page 8. KTH, vetenskap och konst, 2008.

[40] S.Ilango Sambasivan Harish.I. A protocol stack design and implementation of wireless sensor network for emerging application. 2013.

[41] AlQamzi H.S. AlMheiri, S.M. Data link layer security protocols in wireless sensor networks: A survey. 2013.

[42] Yogesh Sankarasubramaniam et al. Esrt: Event-to-sink reliable transport in wireless sensor. 2003.

[43] A. Marco, R. Casas, J. L. Sevillano Ramos, V. Coarasa, A. Asensio, and M. S. Obaidat. Synchronization of multihop wireless sensor networks at the application layer. *IEEE Wireless Communications*, 18(1):82–88, February 2011.

[44] A. Van Den Bossche, T. Val, and R. Dalce. Sisp: A lightweight synchronization protocol for wireless sensor networks. In *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1–4, Sept 2011.

[45] Santiago Gonzalez. Improving time synchronization protocols in wireless sensor networks. Master's thesis, 2015.

[46] C. F. Chiasserini and R. R. Rao. Improving energy saving in wireless systems by using dynamic power management. *IEEE Transactions on Wireless Communications*, 2(5):1090–1100, Sept 2003.

[47] R. Jurdak, A. G. Ruzzelli, and G. M. P. O'Hare. Radio sleep mode optimization in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 9(7):955–968, July 2010.

[48] W. Dargie. Dynamic power management in wireless sensor networks: State-of-the-art. *IEEE Sensors Journal*, 12(5):1518–1528, May 2012.

[49] S. Ahmad Salehi, M. A. Razzaque, P. Naraei, and A. Farrokhtala. Security in wireless sensor networks: Issues and challanges. In *Space Science and Communication (IconSpace), 2013 IEEE International Conference on*, pages 356–360, July 2013.

[50] R. Daidone, G. Dini, and M. Tiloca. On experimentally evaluating the impact of security on ieee 802.15.4 networks. In *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, pages 1–6, June 2011.

[51] R. Daidone. Experimental evaluations of security impact on ieee 802.15.4 networks. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, pages 1–2, June 2011.

[52] Q. Cao, T. Abdelzaher, J. Stankovic, and T. He. The liteos operating system: Towards unix-like abstractions for wireless sensor networks. In *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, pages 233–244, April 2008.

[53] Haksoo Choi, Sukwon Choi, and Hojung Cha. Structural health monitoring system based on strain gauge enabled wireless sensor nodes. In *Networked Sensing Systems, 2008. INSS 2008. 5th International Conference on*, pages 211–214, June 2008.

[54] S. Awasthi and A. Joshi. Mems accelerometer based system for motion analysis. In *Electronics and Communication Systems (ICECS), 2015 2nd International Conference on*, pages 762–767, Feb 2015.

[55] C. Alippi, C. Galperti, and M. Zanchetta. Micro acoustic monitoring with mems accelerometers: towards a wsn implementation. In *Sensors, 2007 IEEE*, pages 966–969, Oct 2007.

[56] T. K. Sethuramalingam and A. Vimalajuliet. Design of mems based capacitive accelerometer. In *Mechanical and Electrical Technology (ICMET), 2010 2nd International Conference on*, pages 565–568, Sept 2010.

[57] A. Sabato, M. Q. Feng, Y. Fukuda, D. L. Carní, and G. Fortino. A novel wireless accelerometer board for measuring low-frequency and low-amplitude structural vibration. *IEEE Sensors Journal*, 16(9):2942–2949, May 2016.

[58] X. Liu, J. Cao, M. Bhuiyan, S. Lai, H. Wu, and G. Wang. Fault tolerant wsn-based structural health monitoring. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*, pages 37–48, June 2011.

[59] B. Li, D. Wang, F. Wang, and Y. Q. Ni. High quality sensor placement for shm systems: Refocusing on application demands. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9, March 2010.

[60] M. Z. A. Bhuiyan, J. Cao, G. Wang, and X. Liu. Energy-efficient and fault-tolerant structural health monitoring in wireless sensor networks. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, pages 301–310, Oct 2012.

[61] Ting Liu and Margaret Martonosi. Impala: A middleware system for managing autonomic, parallel sensor systems. *SIGPLAN Not.*, 38(10):107–118, June 2003.

[62] H. Alex, M. Kumar, and B. Shirazi. Midfusion: middleware for information fusion in sensor network applications. In *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004. Proceedings of the 2004*, pages 617–622, Dec 2004.

[63] A. L. Murphy and W. B. Heinzelman. Milan: Middleware linking applications and networks. Technical report, Rochester, NY, USA, 2002.

[64] Georgios Exarchakos (University of Surrey UK) Maozhen Li (Brunel University UK) Nick Antonopoulos (University of Derby, UK) and The Netherlands) Antonio Liotta (Technical University of Eindhoven. January 2010.

[65] http://ieeexplore.ieee.org/search/searchresult.jsp?queryText=nems&sortType=asc_p_Publication_Year.

[66] Longhai Li, Xiaojun Tian, Z. Dong, Lianqing Liu, O. Tabata, and W. J. Li. Manipulation of dna origami nanotubes in liquid using a programmable tapping mode afm. In *Nano/Micro Engineered and Molecular Systems (NEMS), 2013 8th IEEE International Conference on*, pages 56–59, April 2013.

[67] Xiangyu Wang Chimay J. Anumba. *Mobile and Pervasive Computing in Construction.* John Wiley Sons.