Analyse und Implementierung des Flooding Time Synchronization Protokolls auf einem IEEE 802.15.4 RF-SoC

Bachelor-Thesis von Anant Gupta
April 2011

Fachbereich Informatik
Fachgebiet Eingebettete Systeme und ihre Anwendungen

Analyse und Implementierung des Flooding Time Synchronization Protokolls auf einem IEEE 802.15.4 RF-SoC

Vorgelegte Bachelor-Thesis von Anant Gupta

1. Gutachten: Prof. Dr. Andreas Koch
2. Gutachten: Dipl.-Inform. Andreas Engel

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 5. April 2012

(Anant Gupta)

Inhaltsverzeichnis

1 Einleitung

Drahtlose Sensornetze (engl. Wireless Sensor Network, WSN) haben sich in den letzten Jahren als wichtiges Forschungsgebiet etabliert. Sie bestehen aus einer großen Anzahl von Sensorknoten, die zur drahtlosen Kommunikation und zur Datenverarbeitung fähig sind. Die Einsatzgebiete solcher Sensornetze reichen von militärischen Anwendungen zur Zielbeobachtung und -erfassung bis hin zur Überwachung von biologischen Habitaten oder der Erfassung des Zustandes eines Bauwerks. All diese Anwendungen haben dabei gemein, dass Messdaten an unterschiedlichen Stellen erhoben und diese dann zur Auswertung zusammengeführt werden müssen. Dafür muss die korrekte zeitliche Relation zwischen den erhobenen Messdaten sichergestellt werden.

Wie in den meisten verteilten Systemen, spielt die Zeitsynchronisation der einzelnen Teilnehmer also auch in drahtlosen Sensornetzen eine wichtige Rolle. Da die lokalen Zeitgeber der Sensorknoten nicht perfekt sind, können sie über einen längeren Zeitraum hinweg, ohne gezielte Gegenmaßnahmen, stark voneinander abweichen. Diese Problemstellung hat in der Vergangenheit viele Zeitsynchronisationsverfahren hervorgebracht. Drahtlose Sensornetze weisen Charakteristiken wie limitierte Ressourcen an Energie, Speicher oder Rechenleistung auf. Oft genügen jedoch die herkömmlichen Zeitsynchronsiationsverfahren diesen besonderen Eigenschaften nicht.

In dieser Arbeit wird das „Flooding Time Synchronization Protocol“ (FTSP) untersucht und auf einem IEEE 802.15.4 Radio-Frequency System-on-Chip (RF-SoC) implementiert. Es handelt sich dabei um ein speziell auf die Anwendung in drahtlosen Sensornetzen zugeschnittenes Protokoll zur Zeitsynchronisation. Kennzeichnend für dieses Protokoll sind vor allem die Erhebung der Zeitstempel auf der MAC-Ebene und die Kompensation der Drift der lokalen Zeitgeber. Durch ersteres werden Synchronisationsfehler vermieden, die durch die Verzögerungszeiten der überliegenden Schichten resultieren. Die Driftkompensation vermindert den Fehler, der aufgrund der Bauteiltoleranzen der Oszillatoren entsteht. Dieses Protokoll wurde erstmals in [23] vorgestellt und auf den Plattformen UCB Mica und Mica2 unter Einsatz des Betriebssystems TinyOS implementiert. In der hier vorliegenden Arbeit wird das FTSP unter Verwendung des ereignisgesteuerten Betriebsssystems Contiki auf den Plattformen CC2530 und CC2531 von Texas Instruments als Netzwerkprotokollschicht realisiert.

Die restliche Arbeit ist wie folgt aufgebaut:

Kapitel 2 gibt einen Überblick über die gängigen Zeitsynchronisationsverfahren. Dabei wird das FTSP detailliert beschrieben und die Gründe für die Auswahl dieses Protokolls dargelegt.

In Kapitel 3 wird die verwendete Entwicklungsumgebung, bestehend aus der Hardware, Software und zusätzlichen Hilfsprogrammen, kurz vorgestellt.

Die Details zur Implementierung des Protokolls werden in Kapitel 4 beschrieben. Unter anderem werden die Vorgänge beim Versenden und Empfangen von Synchronisationspaketen, die Details zur Etablierung eines globalen Zeitmaßes und der notwendige Ressourcenbedarf dargestellt. Anschließend werden die dem Nutzer verfügbaren Schnittstellen zur Nutzung des FTSPs als Netzwerkprotokollschicht erläutert.

Die genaue Inbetriebnahme dieser Protokollschicht wird anhand eines kurzen Beispielprogrammes in Kapitel 5 erklärt.

In Kapitel 6 wird die Implementierung auf Genauigkeit und Robustheit getestet. Es werden dazu Messreihen mit unterschiedlichen Konfigurationen der Timeline und der verwendeten Datentypen dargestellt.

Auf die Einschränkungen dieser Arbeit wird in Kapitel 7 eingegangen.

Zuletzt werden in Kapitel 8 die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick auf mögliche weiterführende Verbesserungen der FTSP-Implementierung gegeben.

2 Bestehende Lösungen

In diesem Abschnitt werden einige der gängigen Protokolle zur Zeitsynchronisation vorgestellt. Es werden die Gründe für die Auswahl des Flooding Time Synchronization Protokolls dargelegt und dieses im Abschnitt 2.2 detailliert erläutert.

Um das allgemeine Verständnis zur Zeitsynchronisation zu fördern, sind in der Tabelle 2.1 die Verzögerungszeiten, die bei der Übertragung einer Nachricht entstehen, kurz dargelegt. Diese müssen beim Auswerten der Zeitinformation berücksichtigt werden. Dabei orientiert sich diese Aufteilung an [23].

Zeit	Beschreibung
Sendezeit (nichtdetereministisch)	Zeit, die zwischen der Sendeanweisung auf der Applikationsschicht und dem Beginn des Sendevorgangs auf der MAC-Schicht vergeht. Diese Verzögerungszeit variiert mit der aktuellen Systemauslastung (Systemaufrufe, Interrupt-Handler).
Zugriffszeit (nichtdetereministisch)	Wartezeit bis zum Zugriff auf das Übertragungsmedium. Diese Zeit ist von der aktuellen Netzwerkauslastung abhängig.
Übertragungszeit (deterministisch)	Zeit, die auf der Bit-Übertragungsschicht zum Übertragen eines Pakets über Funk notwendig ist. Sie ist von der Datenübertragungsrate und der Länge des zu übermittelnden Paketes abhängig.
Ausbreitungszeit (deterministisch)	Zeit, in der das Paket vom Sender zum Empfänger unterwegs ist.
Empfangszeit (deterministisch)	Zeit, die vergeht, bis die empfangenen Bits zur MAC-Schicht gelangen. Sie entspricht der senderseitigen Übertragungszeit.
Bearbeitungszeit (nichtdetereministisch)	Zeit, die zum Aufbereiten der Nachricht für die überliegende Anwendung notwendig ist. Die Unsicherheitsquellen entsprechen denen der Sendezeit.

Tabelle 2.1: Verzögerungszeiten

Abbildung 2.1 stellt diese Gliederung noch einmal grafisch dar:

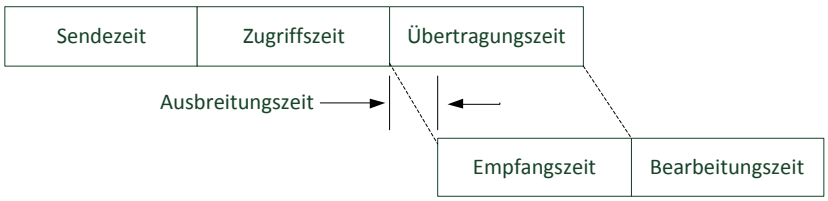

Abbildung 2.1: Verzögerungszeiten einer Nachricht (vgl. [23])

Als Gesamtübertragungszeit soll an dieser Stelle die Verzögerung bestehend aus der Zugriffs-, der Übertragungs- und der Ausbreitungszeit definiert werden.

2.1 Algorithmen zur Zeitsynchronisation

2.1.1 Network Time Protocol (NTP)

Eines der bekanntesten Protokolle zur Zeitsynchronisation ist das Network Time Protocol. Dieses wurde Anfang der 80er Jahre von David L. Mills an der Universität von Delaware entwickelt und findet seinen Einsatz in lokalen Netzwerken und dem Internet (vgl. [25]).

Das NTP geht von Zeit-Servern aus, die in einer baumartigen Hierarchie organisiert sind. Der primäre Zeit-Server (Stratum 1) synchronisiert seine lokale Uhr und Frequenz über externe Hardware-Uhren. Sekundäre Zeit-Server (Stratum 2) beziehen ihre lokale Systemzeit auf den primären Zeit-Server und Tertiäre (Stratum 3) wiederum auf die Sekundären, sodass die Ungenauigkeit mit Anzahl der zwischengeschalteten Zeit-Servern zunimmt.

Vereinfacht dargestellt läuft die Synchronisation von Clients zu Servern folgendermaßen ab: Der Client versendet ein NTP-Paket, das mit einem Zeitstempel $T1$ versehen ist, welches den Sendezeitpunkt darstellt. Ein Zeit-Server beantwortet ein solches Paket durch das Anhängen zwei weiterer Zeitstempel $T2$ und $T3$, die jeweils den Empfangs- und Sendezeitpunkt darstellen. Beim Empfang dieser Nachricht erhebt der Client wieder einen Zeitstempel $T4$ und ermittelt aus den vier vorhandenen Zeitstempeln die Paketumlaufzeit (RTT) und den Versatz ($Offset$), um den die lokale Systemzeit von der Uhr des Zeit-Servers abweicht.

$$Offset = \frac{(T3 - T4) + (T2 - T1)}{2} \tag{2.1}$$

$$RTT = (T4 - T1) - (T3 - T2) \tag{2.2}$$

Anhand Gleichung 2.1 erkennt man, dass die Berechnung des $Offset$ von symmetrischen Gesamtübertragungszeiten ausgeht. Unsymmetrische Gesamtübertragungszeiten gehen als Fehler in diese Berechnung ein. In verkabelten Netzwerken kann angenommen werden, dass $RTT \ll Offset$ gilt, sodass dieser Fehler klein ausfällt. In Funknetzwerken kommen jedoch weitere Störquellen hinzu, wodurch sich stark unsymmetrische Übertragungswege ergeben können. Um die Genauigkeit zu erhöhen, verwaltet jeder Client eine Liste der letzten n empfangenen NTP-Nachrichten. Der aktuelle $Offset$ berechnet sich aus der Nachricht mit dem kleinsten RTT-Wert. Um die Güte der Zeit-Server zu ermitteln, wird aus diesen Paketen ein gewichteter Fehler, die Dispersion, berechnet. Man ermittelt dazu die Abweichungen der letzten $Offset$-Werte vom aktuellen $Offset$ und gewichtet die Differenzen in Abhängigkeit von den RTT-Werten. Kleine RTT-Werte bewirken eine stärkere Gewichtung als Größere. Um die Robustheit zu erhöhen, kommuniziert jeder Client mit mehreren Zeit-Servern gleichzeitig, sodass der Ausfall eines

Zeit-Servers keine große Auswirkung auf die Synchronisationsgenauigkeit hat. Aus der Menge der verfügbaren Zeit-Servern wird anhand bestimmter Kriterien die beste Quelle zur Synchronisation ausgewählt. Solche Kriterien sind zum Beispiel das Stratum, die RTT-Werte und die Dispersion der jeweiligen Zeit-Server.

Das NTP geht für seinen Einsatz davon aus, dass der Prozessor beliebig intensiv genutzt werden kann, ein ständiger Zugriff auf das Netzwerk möglich ist und man beliebig viele Nachrichten verschicken kann. In drahtlosen Sensornetzen ist jedoch die zur Verfügung stehende Energie eine limitierende Ressource, sodass die Grundannahmen von NTP nicht erfüllt werden können. Des Weiteren erreicht das NTP eine Genauigkeit im Millisekundenbereich (vgl. [23]), welche für viele WSN-Anwendungen nicht ausreicht. Die statische Struktur, die NTP voraussetzt, ist ein weiteres Problem in drahtlosen Sensornetzen. Das NTP geht von einer globalen Systemzeit aus, von denen die Uhren der restlichen Teilnehmer möglichst wenig abweichen sollen. Da jeder Netzwerkknoten mit einer Anzahl von Zeit-Servern verbunden ist, kann es zum Beispiel erfolgen, dass zwei nahe liegende Sensorknoten sich zu unterschiedlichen Zeit-Servern synchronisieren und damit zeitlich stark voneinander abweichen. Damit jedoch Daten in eine chronologische Reihenfolge gebracht werden können, spielen in drahtlosen Sensornetzen oft die Abweichungen von nahe liegenden Sensoren eine größere Rolle als die Abweichung von einer gemeinsamen globalen Zeit (vgl. [9]).

2.1.2 Precision Time Protocol (PTP)

Das Precision Time Protocol ist ein Standard zur Zeitsynchronisation, der seit 2002 als IEEE 1588 definiert ist (vgl. [1]). Im Gegensatz zum NTP liegt das Einsatzgebiet vom PTP in lokal begrenzten Netzwerken, die eine hohe Genauigkeit im Bereich von einigen Nanosekunden bis zu wenigen Mikrosekunden voraussetzen.

Ein PTP-Netz geht von miteinander kommunizierenden Uhren aus. Anhand des Best-Master-Clock-Algorithmus (BMC) bestimmt jeder Teilnehmer des Netzwerks den Knoten, welcher die Uhr mit der höchsten Präzision aufweist. Dazu werden Kriterien, wie zum Beispiel das Stratum, herangezogen. Der genaue Ablauf des BMC-Algorithmus soll an dieser Stelle nicht beschrieben werden, stattdessen wird auf [7] Abschnitt 7.6 verwiesen. Die Uhr mit der höchsten Präzision wird als Grandmaster-Clock bezeichnet und dient als Referenzzeitgeber für die restlichen Teilnehmer (Slaves). Durch den BMC-Algorithmus entkoppelt sich das PTP von einer festen Netzwerk-Topologie und ist somit tolerant gegenüber Ausfällen. Die Grundausführung dieses Protokolls unterscheidet zwei Arten von Uhren: Boundry-Clocks und Ordinary-Clocks. Ordinary-Clocks bezeichnen die Slaves eines Teilnetzwerks. Sie synchronisieren sich über die Referenzuhr des Teilnetzwerks. Eine Boundry-Clock ist die Referenzuhr eines solchen Teilnetzes. Sie sind für Ethernet-Netzwerke Switches mit geringen Latenzzeiten und stehen in Verbindung mit der Grandmaster-Clock. Die Kommunikation zwischen Slaves und Master erfolgt auf direktem Weg.

Die Synchronisation läuft ähnlich wie beim NTP ab. Die Grandmaster-Clock versendet periodisch in kurzen Zeitintervallen Synchronisationsnachrichten. Zum Sendezeitpunkt wird dazu ein Zeitstempel $T1$ erhoben. Dieser wird entweder direkt in die Synchronisationsnachricht (vgl. [26]) oder nach dem IEEE 1588 Standard (vgl. [1]) in einer zusätz-

lichen Follow-Up-Nachricht eingefügt und verschickt. Beim Empfang des Synchronisationspaketes wird ein weiterer Zeitstempel $T2$ von den Slaves erhoben. Zur Schätzung der Gesamtübertragungszeit werden von den Slaves Delay-Request-Nachrichten an den jeweiligen Master verschickt, die ebenfalls mit einem Zeistempel $T3$, der den Sendezeitpunkt darstellt, gekennzeichnet sind. Diese werden dann von dem Master durch die zum Empfangszeitpunkt mit $T4$ zeitgestempelte Delay-Response-Nachricht quittiert, sodass wieder vier Zeitstempel zur Verfügung stehen. Mit den Gleichungen 2.1 und 2.2 können dann RTT und $Offset$ berechnet werden. Genauso wie das NTP geht das PTP von symmetrischen Gesamtübertragungszeiten aus, sodass Abweichungen von dieser Annahme wieder als Fehler in die $Offset$-Berechnung einfließen.

Das PTP wurde für den Einsatz in Ethernet-Netzwerken entwickelt und eignet sich daher nur bedingt für drahtlose Sensornetze. Eine Kombination aus einem Ethernet-Netzwerk und einem WSN-Teilnetzwerk wurde in [4] erprobt. Um eine hohe Genauigkeit zu erreichen, wurden dazu die Synchronosationsnachrichten alle 100 ms periodisch verschickt. Dieses ist aufgrund der begrenzten Energieressourcen für die meisten Anwendungen, in denen drahtlose Sensornetze ihren Einsatz finden, ungeeignet.

2.1.3 Reference Broadcast Synchronization (RBS)

Das RBS-Protokoll wurde 2002 an der Universität von Kalifornien, USA entwickelt und beschreibt im Gegensatz zu den bisherigen Protokollen, die eine Sender-Empfänger-Synchronisation durchführen, eine Empfänger-Empfänger-Synchronisation (vgl. [8]). Dazu wird von einem bestimmten Netzwerkknoten eine Referenznachricht an die restlichen n Empfänger des Netzwerks verschickt. Man geht dabei idealisiert von der Annahme aus, dass diese Nachricht alle Empfänger gleichzeitig erreicht. Die Empfänger notieren den Empfang durch einen Zeitstempel T. In einem zweiten Schritt tauschen die Empfänger untereinander diese Zeitstempel aus, sodass der $Offset$ zwischen einem Empfänger i und einem Empfänger j folgendermaßen berechnet werden kann:

$$Offset[i,j] = (T_j - T_i)\,\forall i,j \in \{1...n\} \tag{2.3}$$

Ein bisher noch unbetrachteter Aspekt ist die Uhren-Drift. Es wurde davon ausgegangen, dass die Uhren der Teilnehmer mit gleicher Geschwindigkeit laufen. Diese Annahme erweist sich in der Realität als falsch, sodass durch die Offset-Korrektur zwar die Systemuhren zweier Knoten für einen Zeitpunkt abgeglichen werden können, aber mit fortschreitender Zeit die Uhren aufgrund der unterschiedlichen Taktgeschwindigkeit der Oszillatoren wieder auseinander driften. Um dieser Problematik entgegen zu wirken, wird statt Gleichung 2.3 ein Ansatz der linearen Regression gewählt, wodurch sich der lineare Zusammenhang zwischen $Offset$ und der lokalen Zeit angeben lässt. Dieses Verfahren wird ebenfalls vom FTSP eingesetzt und wird daher im Abschnitt 2.3.1 detailliert erklärt. Das RBS-Protokoll etabliert keine globale Systemzeit. Stattdessen kann ein Sensorknoten i zu jedem beliebigen Zeitpunkt den $Offset[i,j]$ zum Knoten j mithilfe der Regressionsgeraden errechnen. Dieses Verfahren sorgt für eine Synchronisation der Teilnehmer eines Netzwerks, in dem alle Sensorknoten sich gegenseitig direkt erreichen können. Um eine Multihop-Synchronisation zu ermöglichen, muss das gesamte

Netzwerk in Teilnetze unterteilt werden, in denen eine direkte Kommunikation möglich ist. Netzwerkteilnehmer, die am Rand dieser Netzwerk-Cluster angeordnet sind, beteiligen sich an der Synchronisation aller angrenzender Netzwerk-Cluster, sodass diese die notwendigen Informationen zur Zeitumwandlung von einem Cluster zum anderen kennen. Die Multihop-Synchronisation bewirkt jedoch, dass ein Umrechnungsfehler bei der Zeitumwandlung entsteht, der mit der Anzahl der Hops ansteigt. In [8] wurde gezeigt, dass dieser bei n Hops in der Größenordnung von $O(\sqrt{n})$ ansteigt.

Der Vorteil des RBS-Protokolls liegt darin, dass die die Sende- und Zugriffszeit als senderseitige nichtdeterministische Ungenauigkeitsquellen herausfallen. Dieses liegt daran, dass die Referenznachricht nicht mit einem Zeitstempel versehen wird und diese Verzögerungszeiten für alle Empfänger gleichgroß sind. Die Ausbreitungszeit ist für eine Genauigkeit im μs-Bereich vernachlässigbar klein. Für eine typische Reichweite von 30 m beträgt diese etwa 100 ns.

2.1.4 Timing-sync Protocol for Sensor Networks (TPSN)

Mit dem TPSN wurde 2003 von S. Ganeriwal, R. Kumar und M. Srivastava in [13] ein Protokoll vorgestellt, das eine kontinuierliche Zeitsynchronisation in Sensornetzen ermöglicht. Die Synchronisation lässt sich in zwei Schritten unterteilen. Im ersten Schritt wird ausgehend von einem Sensorknoten ein Spannbaum aufgebaut. Dieser Knoten, als Wurzel dieser Hierarchie, sendet ein Level-Discovery-Paket mit seinem eigenen Level aus, wobei das Level der Wurzel als 0 festgelegt ist. Die Empfänger des Pakets inkrementieren dieses um eins und übernehmen es als das eigene Level. Nach Ablauf eines zufälligen Zeitgebers senden diese wiederum selber ein Level-Discovery-Paket aus, sodass schrittweise ein Spannbaum erstellt werden kann, in dem jeder Kindknoten seinen Vaterknoten und sein Level kennt. Sensorknoten, die aufgrund von Kollisionen nicht erreicht werden konnten, senden nach Ablauf eines zufälligen Zeitgebers ein Level-Request-Paket aus. Dieses wird von den anliegenden Nachbarknoten mit dem eigenen Level beantwortet. Der Versender des Level-Request-Pakets sucht sich aus den erhaltenen Antworten das Paket mit dem minimalen Level aus, inkrementiert dieses um eins und übernimmt es als das eigene Level.

Im zweiten Schritt wird von der Wurzel die Synchronisation durch ein Time-Sync-Paket initiiert. Die „Level 1“-Knoten verschicken beim Erhalt dieser Nachricht ein zeitgestempeltes Sync-Pulse-Paket, welches sowohl von der Wurzel als auch von den Kinderknoten registriert wird. Die Wurzel quittiert den Empfang mit einem Acknowledgement-Paket, sodass analog zum NTP wieder vier Zeitstempel zur Verfügung stehen. Anhand der Gleichungen 2.2 und 2.1 kann die Reaktionszeit des Netzwerks und der $Offset$ der eigenen Uhr zur Wurzel bestimmt werden (vgl. Abschnitt 2.1.1). Die Kinderknoten starten beim Empfang des Sync-Pulse-Paketes einen Zeitgeber und verschicken nach dessen Ablauf wiederum ihrerseits ein Sync-Pulse-Paket an den nun synchronisierten Vaterknoten, sodass über den vollständigen Spannbaum schrittweise eine Vater-Kind-Synchronisation entlang der Kanten erfolgt. Dadurch, dass ein Sync-Pulse-Paket sowohl vom Vater- als auch vom Kindknoten registriert wird, erspart man sich die zusätzliche Initiierung durch ein Time-Sync-Paket.

Im Gegensatz zum NTP und PTP werden zur Erreichung einer höheren Genauigkeit die Zeitstempel auf der MAC-Schicht erhoben. Als Ungenauigkeitsquellen verbleiben damit nur noch die Übertragungs- und Empfangszeiten. Die Ausbreitungszeit kann durch den ermittelten RTT-Wert bestimmt werden. In [13] wird behauptet, dass durch die Erhebung der Zeitstempel auf der MAC-Schicht das TPSN im Vergleich zu einer RBS-Implementierung eine doppelt so hohe Genauigkeit erreichen kann.

Da das TPSN eine baumartige Hierarchie nutzt, ist es nicht robust gegen dynamische Änderung der Netzwerktopologie. Des Weiteren erfolgt keine Drift-Kompensation, sodass es hinsichtlich der Genauigkeit beschränkt ist.

2.1.5 Flooding Time Synchronization Protocol (FTSP)

Mit dem FTSP wurde in [23] ein Synchronisationsprotokoll vorgestellt, dass sich durch geringen Kommunikationsoverhead und Robustheit auszeichnet. Es ermöglicht eine Sender-Empfänger-Zeitsynchronisation, die darauf basiert, dass sich der Netzwerkknoten mit kleinster ID als Wurzel erklärt und als Quelle für eine Referenzzeit dient. Dieser versendet in periodischen Zeitabständen Synchronisationspakete, die einen präzisen Zeitstempel zum Sendezeitpunkt enthalten. Die Empfänger dieser Nachricht erheben beim Erhalt dieser Nachricht ebenfalls einen genauen Zeitstempel, ermitteln den Versatz der eigenen lokalen Zeit zur globalen Zeit und führen anhand der letzten n Synchronisationsnachrichten eine lineare Regression zur Uhren-Drift-Kompensation durch, sodass sich aus der lokalen Zeit und den ermittelten Daten eine Schätzung der globalen Zeit angeben lässt. Die synchronisierten Empfänger versenden wiederum periodisch Schätzungen der globalen Zeit, sodass sich der Effekt des Flutens der Zeitinformation ins Netzwerk ergibt. Die genauen Details zur Zeitumwandlung, zur linearen Regression und zur Wurzelauswahl werden im Abschnitt 2.3 ausführlich erklärt.

2.2 Gründe für die Auswahl von FTSP

An dieser Stelle wird die Auswahl des FTSPs für die Zeitsynchronisation in drahtlosen Sensornetzen motiviert. In den Abschnitten 2.1.1 und 2.1.2 wurde bereits erklärt, warum sich die Synchronisationsprotokolle NTP und PTP für drahtlose Sensornetze nicht eignen. Einer der Gründe dafür ist der hohe Energieverbrauch. Vergleicht man das FTSP mit dem RBS-Protokoll und dem TPSN, so erkennt man, dass die Synchronisation zwischen zwei Teilnehmern über das FTSP nur auf dem Austausch einer Nachricht basiert. Das RBS-Protokoll hingegen benötigt mit der Referenznachricht und dem gegenseitigen Austauschen der Zeitstempel 1,5 Pakete pro Knoten zur Synchronisation. Das TPSN benötigt nach dem Aufbau des Spannbaums und der Initiierung durch ein Time-Sync-Paket zwei Synchronisationsnachrichten, sodass sowohl das RBS-Protokoll als auch das TPSN einen höheren Kommunikationsoverhead als das FTSP aufweisen.

Da eine Schätzung der globalen Zeit in das restliche Sensornetz geflutet wird, ist keine Initialisierungsphase zum Aufbau einer statischen Baumstruktur wie beim TPSN notwendig ist. Die Unabhängigkeit von einer statischen Hierarchie steigert die Robustheit gegenüber dynamischen Topologieänderungen. Der Algorithmus zur Wurzelauswahl er-

möglicht im Falle eines Ausfalls der Wurzel, dass sich automatisch eine neue Wurzel herausbildet.

Hinsichtlich der Genauigkeit lässt das FTSP eine höhere Präzision erwarten, da es durch die lineare Regression und der Erhebung der Zeitstempel auf der MAC-Schicht die Vorteile von TPSN und RBS vereint. In [13] behaupten die Autoren, dass die Implementierung von TPSN mit einem durchschnittlichen Fehler von 16,9 μs im Vergleich zu einer RBS-Implementierung mit einem durchschnittlichen Fehler von 29,1 μs etwa doppelt so genau ist. Die FTSP-Implementierung in [23] hingegen erreicht im Single-Hop-Fall einen durchschnittlichen Fehler von 1,48 μs. Bei dem Vergleich dieser Werte ist jedoch zu beachten, dass die Protokolle auf unterschiedlicher Hardware implementiert wurden, sodass ein direkter Vergleich zwischen den Implementierungen in [13] und [23] nur bedingt möglich ist.

Ein konkretes Anwendungsszenario für die Zeitsynchronisation mit FTSP ist die Überwachung mechanischer Strukturen mittels verteilter intelligenter Sensorknoten [21]. Dabei werden Schwingungen an verschiedenen Stellen der Struktur innerhalb bestimmter Zeitfenster gemessen und lokal ausgewertet, um Schädigungen an der Struktur frühzeitig erkennen zu können. Für eine konsistente Datenerhebung müssen die Messungen bei allen Sensorknoten zur selben Zeit stattfinden, was den Einsatz von Synchronisationsmechanismen voraussetzt. Da die Trigger-Bedingungen zum Start der Messfenster ausgehend von wenigen Wurzeln im gesamten Netzwerk verteilt werden müssen, eignet sich das FTSP-Prinzip des Flutens von Informationen besonders gut für diesen Einsatzzweck.

2.3 Detaillierte Beschreibung von FTSP

Das FTSP erzielt seine hohe Synchronisationsgenauigkeit durch die Erhebung der Zeitstempel auf der MAC-Schicht und durch die Ausführung einer linearen Regression zur Uhren-Drift-Kompensation. Die Generierung der Zeitstempel auf der MAC-Schicht hat den Vorteil, dass die sender- und empfängerseitigen nichtdeterministischen Ungenauigkeitsquellen vermieden werden. In [23] werden verschiedene Möglichkeiten zur Reduktion von Unsicherheiten bei der Erhebung von Zeitstempeln beschrieben. Diese umfassen das Mitteln über mehrere Zeitstempel sowie die Korrektur der Byte-Ausrichtung. Sie werden in dieser Arbeit nicht benötigt, da die Implementierung auf einem IEEE 802.15.4 Radio-Frequency System-on-Chip erfolgt, der hardwareseitig die Erhebung solcher Zeitstempel zum Sende-/ bzw. Empfangszeitpunkt des start-of-frame-delimiters (SFD) unterstützt.

Sensorknoten in drahtlosen Sensornetzen benutzen üblicherweise einen lokalen Oszillator als Taktgeber für ihre Systemzeit. Dieser Oszillator läuft mit einer bestimmten Frequenz f. Ein Zeitschritt in der Systemzeit kann dabei für zwei Sensorknoten der gleichen Baureihe aufgrund von Bauteiltoleranzen unterschiedlich lange dauern. Die Oszillatorfrequenzen variieren innerhalb den vom Hersteller angegebenen Tolerenzgebiet. Die Kristall-Oszillatoren der verwendeten Sensorknoten sind mit einer Frequenzgenauigkeit von ± 40 parts-per-million (ppm) charakterisiert. Bei einem Systemtakt von 32 MHz kann diese bereits zu mehr als $\pm$ 1 Takt Abweichung pro Sekunde führen.

Abbildung 2.2 stellt den Generierungszeitpunkt der Zeitstempel grafisch dar. Die verwendeten RF-SoCs ermöglichen, dass noch während des Sendevorgangs die Zeitstempel

an die Nachricht angehängt werden können (vgl. Abschnitt 4.3). Eine Alternative für Sensorknoten, die dieses nicht unterstützen, wäre die Nutzung von zusätzlichen Follow-Up-Nachrichten (vgl. Abschnitt 2.1.2).

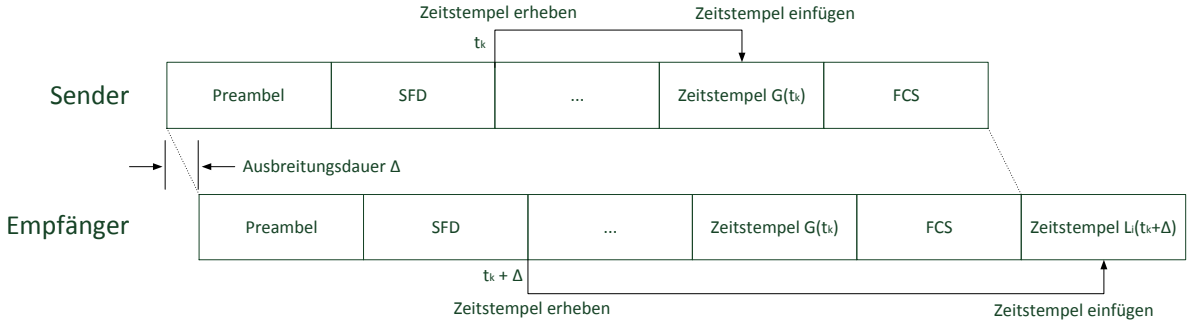

Abbildung 2.2: Generierung der Zeitstempel

Ein Referenzknoten für die globale Sytemzeit versendet in periodischen Abständen Synchronisationsnachrichten, die einen Zeitstempel $G(t_k)$ der globalen Zeit zum Sendezeitpunkt t_k enthalten. Dieser Referenzknoten wird im Folgenden als Wurzel mit dem Index r gekennzeichnet. Für einen Sensorknoten mit der ID i wird die Oszillatorfrequenz f_i als konstant angenommen. Veränderungen der Frequenz aufgrund von Alterung, Temperaturschwankungen oder Umweltbedingungen werden hier vernachlässigt. Der Einschaltzeitpunkt eines Sensorknotens i sei durch t_i^0 gekennzeichnet. Für die Wurzel sei dieser definiert durch $t_r^0 := 0$.

Die lokale Systemzeit $L_i(t)$ eines Knotens i zum Zeitpunkt t kann demnach beschrieben werden durch:

$$L_i(t) := \frac{t - t_i^0}{f_i} \tag{2.4}$$

Da sich die globale Systemzeit auf die lokale Systemzeit der Wurzel bezieht, kann diese folgendermaßen angegeben werden:

$$G(t) := L_r(t) = \frac{t}{f_r}. \tag{2.5}$$

Der Empfänger i eines Synchronisationspaketes kennzeichnet den Empfangszeitpunkt $t_k + \Delta$ in der eigenen Systemzeit $L_i(t_k + \Delta)$, wobei Δ die zu vernachlässigende Ausbreitungsdauer der Funkwellen beschreibt. Mit jedem Synchronisationspaket wird demnach das Informationspaar $(G(t_k), L_i(t_k))$ generiert, welches allerdings mit dem systematischen Fehler der vernachlässigten Ausbreitungszeit behaftet ist. Um einen Zusammenhang zwischen lokaler und globaler Zeit herzustellen, wird eine lineare Regression basierend auf einer Liste solcher Zeitstempelpaare durchgeführt. Dies wird im Folgenden näher beschrieben.

2.3.1 Clock Skew

Da sich die Einschaltzeitpunkte der Sensorknoten im Allgemeinen unterscheiden, weicht die lokale Systemzeit $L_i(t)$ des Sensorknoten i um einen Offset $O_i(t)$ von der globalen Zeit $G(t)$ ab. Dieser berechnet sich sich am Knoten i zum Zeitpunkt t zu

$$O_i(t) := G(t) - L_i(t). \tag{2.6}$$

$$\overset{2.5,\ 2.4}{=} \frac{t}{f_r} - \frac{t - t_i^0}{f_i} = \frac{f_i - f_r}{f_i f_r} \cdot t + \frac{t_i^0}{f_i} =: m_i \cdot t + b_i. \tag{2.7}$$

Man erkennt einen linearen Zusammenhang zwischen Offset und Zeit. Da f_i, f_r und t_i^0 unbekannte Konstanten sind, müssen die Parameter m_i und b_i mittels linearer Regression ermittelt werden, deren Grundlagen im nächsten Abschnitt beschrieben werden.

Mathematische Grundlagen zur linearen Regression

Die lineare Regression ermöglicht die Bestimmung einer Ausgleichsgeraden zwischen den Wertepaaren (x_i, y_i). Diese drückt dabei die Größe y in Abhängigkeit der Größe x über die Steigung m und den Achsenabschnitt b aus:

$$y(x) = m \cdot x + b \tag{2.8}$$

Da die Wertepaare um die Geradengleichung 2.8 verstreut liegen, gilt:

$$y_i = m \cdot x_i + b + \epsilon_i \tag{2.9}$$

ϵ_i beschreibt hier die Differenz $|y_i - y(x_i)|$. Gesucht werden dabei die Koeffizienten m und b, die diese Differenzen minimieren. Mithilfe der Methode der kleinsten Quadrate wird die Summe der Abstandsquadrate

$$S(m, b) = \sum_{i=1}^{n} (\epsilon_i)^2 = \sum_{i=1}^{n} (y_i - m \cdot x_i - b)^2 \tag{2.10}$$

minimiert. Gesucht wird also die Lösung des Problems

$$min_{(m,b)\in\mathbb{R}} S(m, b). \tag{2.11}$$

Nach einer Zwischenrechnung (siehe Anhang 9.1) erhält man als Lösung für 2.11 die Koeffizienten der Regressionsgeraden:

$$m = \frac{\sum_{i=1}^{n} (y_i - \overline{y})(x_i - \overline{x})}{\sum_{i=1}^{n} (x_i - \overline{x})^2} \tag{2.12}$$

$$b = \overline{y} - m \cdot \overline{x} \tag{2.13}$$

Setzt man 2.13 in 2.8 ein, so erhält man für die Regressionsgerade folgenden Zusammenhang:

$$y(x) = \overline{y} + m \cdot (x - \overline{x}) \tag{2.14}$$

Hierbei beschreiben $\overline{x}$ und $\overline{y}$ die arithmetischen Mittelwerte der Größen x bzw. y.

Lineare Regression zur Skew-Kompensation

Die Sensorknoten verfügen über keine genauen Informationen über den globalen Zeitpunkt t, sondern nur über ihre lokale Systemzeit $L_i(t)$. Ersetzt man t mit der lokalen Zeit, so erhält aus Gleichung 2.7 für den Offset folgenden Zusammenhang:

$$O_i(L_i(t)) = m_i \cdot L_i(t) + b_i. \tag{2.15}$$

Die Linearparameter m_i und b_i sind nun aus der Regressionstabelle $(L_i(t_k), O_i(t_k) = G(t_k) - L_i(t_k))_{k=1}^{N}$ der letzten N Synchronisationspakete nach der Methode kleinster Fehlerquadrate zu ermitteln. Mit den Formeln 2.13 und 2.12, lassen sich diese zu

$$m_i = \frac{\sum_{k=1}^{N}(O_i(t_k) - \overline{O_i})(L_i(t_k) - \overline{L_i})}{\sum_{k=1}^{N}(L_i(t_k) - \overline{L_i})^2} \tag{2.16}$$

$$b_i = \overline{O_i} - m_i \cdot \overline{L_i} \tag{2.17}$$

angeben. Hierbei bezeichnen $\overline{L_i}$ und $\overline{O_i}$ die arithmetischen Mittelwerte:

$$\overline{L_i} := \frac{1}{N}\sum_{k=1}^{N} L_i(t_k) \quad \text{bzw.} \quad \overline{O_i} := \frac{1}{N}\sum_{k=1}^{N} O_i(t_k) \tag{2.18}$$

2.3.2 Umrechnung zwischen globaler und lokaler Zeit

Hat ein Knoten i genügend Synchronisationspakete für die lineare Regression gesammelt, wird er als synchronisiert bezeichnet. Er kann dann jedem lokalen Zeitpunkt $L_i(t)$ einen Offset und damit eine Schätzung für die globale Zeit zuordnen:

$$G_i(t) := L_i(t) + O_i(L_i(t)) \overset{2.14}{=} L_i(t) + \overline{O_i} + m_i \cdot (L_i(t) - \overline{L_i}). \tag{2.19}$$

Dies wird beispielsweise verwendet, wenn synchronisierte Knoten ihrerseits Synchronisationspakete broadcasten, um die Zeitinformation weiter im Netzwerk zu verteilen.

Für das gleichzeitige Ausführen einer bestimmten Aufgabe durch alle Sensorknoten ist es außerdem notwendig, aus einer Schätzung für die globale Systemzeit eine entsprechende lokale Systemzeit zu berechnen. Nach dem Umstellen von Gleichung 2.19 nach $L_i(t)$ erhält man hierfür:

$$L_i(t) = \frac{G_i(t) - \overline{O_i} + m_i \cdot \overline{L_i}}{m_i + 1}. \tag{2.20}$$

2.3.3 Verfahren zur Auswahl der Wurzel

Da es keinen ausgezeichneten Netzwerkknoten als Referenzeitgeber gibt, muss zu Beginn der Netzwerksynchronisation ein Referenzknoten dynamisch ausgewählt werden. Das FTSP sieht vor, dass jedem Sensorknoten eine eindeutige Identifikation (ID) zugeordnet ist. Diese IDs werden vom Auswahlprozess ausgenutzt. Wenn ein Knoten nach dem Ablauf einer bestimmten Dauer von keiner Synchronisationsnachricht erreicht wurde, erklärt sich dieser zur Wurzel und beginnt mit dem Versenden von Synchronisationspaketen. Es ist daher möglich, dass mehrere Knoten sich zur Wurzel erklären. Um dieses Problem zu lösen, wird der Knoten mit der kleinsten ID als Wurzel ausgewählt. Dazu enthalten die Synchronisationspakete, neben dem Zeitstempel $G(t_k)$ die ID der Wurzel, zu der der Sender synchronisiert ist. Wird ein Sensorknoten durch ein Synchronisationspaket erreicht, das eine kleinere Wurzel-ID aufweist als die eigenverwaltete ID der Wurzel, so synchronisiert sich der Knoten zu dieser neuen Wurzel und gibt gegebenenfalls den Status als Wurzel auf. Die Details zur Umsetzung dieses Aufwahlverfahrens werden im Abschnitt 4.2 erklärt.

3 Entwicklungsumgebung

In diesem Abschnitt werden die benutzte Hardware, Software und Hilfsprogramme (Tools) kurz vorgestellt.

3.1 Hardware

Für die Implementierung des FTSPs werden die CC2530/1 RF-SoCs [18, 20] von Texas Instruments benutzt. Diese Chips basieren auf einem 8051-Mikrocontroller (MCU) und verfügen über 256 kB programmierbaren Flash-Speicher und einen 8 kB großen SRAM. Als Taktgeber dient ein 32 MHz Kristall-Oszillator mit einer maximalen Ungenauigkeit von ± 40 ppm. Die Funkkommunikation findet im ISM-Frequenzbereich von 2,4 GHz (2394 MHz - 2507 MHz) statt.

Für eine mobile Energieversorgung wird der CC2530 zusammen mit einem SmartRF05 Batterie-Board betrieben. Beim CC2531 handelt es sich hingegen um einen USB-Dongle, der über die USB-Schnittstelle eines Rechners mit Energie versorgt werden und Kommunikationsdaten austauschen kann.

Zum Programmieren der SoCs wird der CC Debugger von Texas Instruments benutzt. Die Möglichkeit zum Debuggen kann nicht genutzt werden, da diese nicht zum verwendeten Compiler kompatibel ist und für die Entwicklungsumgebung IAR Embedded Workbench vorgesehen ist.

3.2 Software

Als Basis für die unterste Netzwerk-Protokollschicht wird das Minimal-Radio-Frequency-Interface (MRFI) des Protokollstacks SimpliciTI [15] von Texas Instruments benutzt. Es handelt sich dabei um eine Programmschicht, die ausschließlich für den Funkbetrieb zuständig ist. Zum Verschicken der Nachrichten wird dabei das IEEE 802.15.4 Frame-Format (vgl. [2, 16]) nach Tabelle 3.1 verwendet.

Preambel	SFD	Länge	FCF	SeqNr.	Adressinform.	Payload	FCS
4 Bytes	1 Byte	1 Byte	2 Bytes	1 Byte	8 Bytes	0..n Byte(s)	2 Bytes

Tabelle 3.1: MRFI-Frame-Format

Die niederwertigen Bytes eines Feldes werden dabei zuerst verschickt. Die Kommunikation findet standardmäßig auf der 2425 MHz Frequenz statt. Weiterhin unterstützt das MRFI die Prüfung der Kanalfreiheit (CCA) zur Vermeidung von Kollisionen.

Als Betriebsystem kommt Contiki [5, 6] zum Einsatz. Es handelt sich um ein frei verfügbares ereignisgesteuertes Betriebssystem, das speziell für eingebettete Systeme mit geringen Speicherressourcen entwickelt wurde. Die wesentlichen Merkmale von Contiki sind der ereignisgesteuerte Kernel und die daraus resultierenden geringen Speicheranforderungen. Ein Prozess (Protothread) wird nur dann ausgeführt, wenn ein entsprechendes

Ereignis gesetzt wurde. In diesem Fall wird dieser Prozess dann vollständig durchlaufen. Dieses führt zu einem kompakteren Programm und benötigt weniger Speicher als ein Thread-basierter Kernel, der für jeden einzelnen Thread einen eigenen Stack verwalten muss.

3.3 Tools

Der Small-Device-C-Compiler (SDCC) [11] ist ein freier optimierender ANSI-C-Compiler, der auf 8-Bit-Mikrocontroller (Intel MCS-51, Dallas DS80C390, Freescale HC08, Zilog Z80) zugeschnitten ist. Unter anderem unterstützt dieser Compiler die 4-Byte Integer-Arithmetik und Fließkommazahlen nach dem IEEE 754 Standard mit einfacher Genauigkeit. In dieser Arbeit wird der SDCC in der Version 3.0.0 eingesetzt.

Als weitere Hilfsmittel werden die beiden von Texas Instruments entwickelten Programme SmartRF Flash Programmer [17] und SmartRF Protocol Packet Sniffer [19] eingesetzt. Der SmartRF Flash Programmer dient zum Überspielen des Programm-Codes auf die SoCs. Bei dem SmartRF Packet Sniffer handelt es sich um ein Programm, dass es ermöglicht, die Funkkommunikation zu beobachten. Dazu wird auf einem CC2531 USB-Dongle die SmartRF Packet Sniffer Firmware überspielt und der Funkverkehr aufgezeichnet und auf dem angeschlossenen Rechner ausgegeben. Dieses Programm unterstützt dabei verschiedene IEEE 802.15.4 basierte Funkprotokolle.

Eine Möglichkeit zur Anzeige der über die USB/UART-Schnittstelle ausgegebenen Daten bietet das serielle Terminal H-Term [14]. Bei dieser Arbeit wird H-Term zum Testen der Synchronisationsgenauigkeit und zur Fehlerbeseitigung benutzt.

4 FTSP Implementierung

4.1 Interaktion mit einer Timeline

Damit eine Zeitsynchronisation überhaupt umgesetzt werden kann, ist die Implementierung einer Timeline notwendig. Da die globale und lokale Zeit im FTSP getrennt verwaltet werden, muss die lokale Systemuhr zu keinem Zeitpunkt verändert werden. Im Wesentlichen muss die Timeline das Auslesen von 32-Bit breiten Zeitstempeln zum Sende- und Empfangszeitpunkts des SFD-Bytes ermöglichen.

Der genaue Entwurf der Timeline war nicht Aufgabe dieser Arbeit. Daher wird nur testweise eine Timeline durch einen Hardware-Zähler repräsentiert. Dazu wird der Timer 2 des CC2530/1 benutzt. Dies ist ein mit 32 MHz getakteter 40-Bit-Zähler, welcher die Repräsentation einer Timeline von insgesamt ca. 9,54 h bei einer Taktauflösung von 31,25 ns ermöglicht.

Da der verwendete Compiler, wie für 8-Bit-Architekturen üblich, nur 32-Bit-Arithmetik unterstützt, kann zwischen einer Timeline mit einer Laufzeit 9,54 h und einer Taktauflösung von etwa 8 μs oder einer Timeline mit einer Laufzeit von etwa 2,34 Minuten und einer Taktauflösung von 31,25 ns ausgewählt werden. Dazu werden entweder die oberen bzw. unteren vier Bytes des insgesamt 5-Byte-Zählers benutzt.

Timer 2 ermöglicht die hardwareseitige Erhebung von Zeitstempeln auf der MAC-Schicht. Es wird dazu automatisch der Zählerstand beim Empfangen oder Versenden des SFD-Bytes gespeichert. Eine weitere Problemstellung ist die Verwaltung des Zählerwertes im Schlafmodus. Viele WSN-Anwendungen sind nur kurzzeitig aktiv und verweilen sonst im Energiesparmodus. Timer 2 ermöglicht den synchronen Betrieb mit dem SleepTimer, wobei der Zähler des Timer 2 beim Aufwachen aus dem Energiesparmodus bis auf eine Restungenauigkeit von ± 1 Takt korrigiert wird (vgl. [16] Abschnitt 22.4). Die in dieser Arbeit verwendete Timeline ist im Modul `timeline` implementiert.

4.2 „heartBeats“-Konzept zur Auswahl der Wurzel

Wie bereits erwähnt, bietet das FTSP-Protokoll eine hohe Robustheit gegen Topologieänderungen innerhalb des Sensornetzes. Diese Robustheit wird durch einen Algorithmus zur dynamischen Wurzelauswahl ermöglicht. Jeder Sensorknoten verwaltet dazu eine eindeutige ID `myID`, die ID der aktuellen Wurzel `myRootID` und einen Zähler `heartBeats`. Dieser Zähler wird dabei periodisch von jedem Sensorknoten hochgezählt, solange keine Synchronisationspakete empfangen werden. Erreicht er einen bestimmten Schwellwert `ROOT_TIMEOUT`, so erklärt sich der Sensorknoten selber zur Wurzel (`myRootID = myID`) und versendet in periodischen Zeitabständen Synchronisationsnachrichten.

Eine Synchronisationsnachricht besteht neben dem Zeitstempel weiter aus der `nodeID`, der `rootID` und einer Sequenznummer `seqNum`. Mit `nodeID` wird die ID des Senders der Synchronisationsnachricht bezeichnet. Die Sequenznummer der Nachricht wird von der Wurzel mit jedem versendeten Synchronisationspaket hochgezählt. Da Synchronisati-

onsnachrichten in das Netz geflutet werden, kann es vorkommen, dass ein Synchronisationspaket einen Sensorknoten mehrfach erreicht. Der Sensorknoten kann anhand der `seqNum` alte Pakete verwerfen. Dazu verwaltet er mit `highestSeqNum` die Sequenznummer des letzten gültigen Synchronisationspaketes. Synchronisationsnachrichten werden nur berücksichtigt, sofern `seqNum` größer als `highestSeqNum` ist. Wird eine solche Nachricht empfangen, so wird `heartBeats` zurück auf null gesetzt. So wird sichergestellt, dass nach Ablauf einer bestimmten Anzahl von Synchornisationszyklen immer mindestens eine Wurzel im Sensornetz vorhanden ist. Es ist jedoch möglich, dass zu einem Zeitpunkt mehrere Wurzeln aktiv sind. Um zu gewährleisten, dass sich nach einer bestimmten Zeit genau eine Wurzel herausbildet, geben die Wurzeln ihren Status auf, sobald sie von einer Synchronisationsnachricht erreicht werden, die eine kleinere `rootID` aufweist als `myRootID`.

4.3 Programmablauf

In diesem Abschnitt wird der Empfangs- bzw. Sendevorgang eines Synchronisationspaketes beschrieben. Abbildung 4.1 zeigt dabei den logischen Programmablauf. Dieser wird im Folgenden genauer beschrieben.

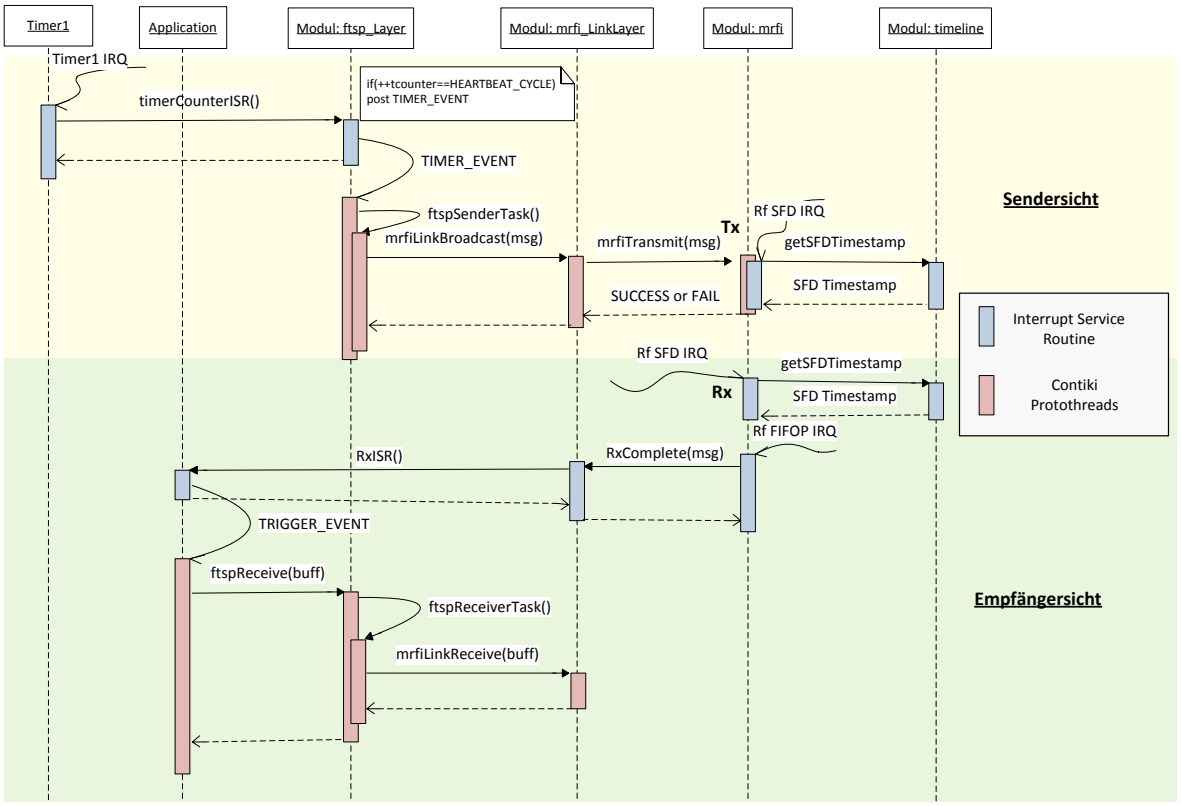

Abbildung 4.1: Empfänger- und Sendersicht

4.3.1 Sendersicht

Das FTSP sieht es vor, in periodischen Zeitabständen Synchronisationspakete ins Netzwerk zu fluten. Eine Periode wird durch den Zeitgeber Timer 1 bestimmt. Er ist als 32-kHz-Zeitgeber im Modul `timer_32k` implementiert und generiert etwa alle 0,25 Sekunden eine Unterbrechungsanforderung (IRQ). Die Unterbrechungsbehandlung (ISR) ruft dabei die Methode `timerCounterISR()` des Moduls `ftsp_Layer` auf. Diese wird bei der Initialisierung des Timer 1 als Funktionszeiger im Modul `timer_32k` registriert. In der Methode `timerCounterISR()` wird ein Zähler `tcounter` inkrementiert. Erreicht dieser den Wert `HEARTBEAT_CYCLE`, so wird ein Ereignis an die Event-Queue des Protothreads `ftspSyncSender` angehängt.

Beim Behandeln dieses Ereignisses wird die Methode `ftspSenderTask()` (siehe Programmauszug 4.13) aufgerufen. Diese ist zum einen für das Umsetzen des Algorithmus zur Auswahl der Wurzel (vgl. Abschnitt 4.2) und zum anderen für die Generierung von Synchronisationspaketen zuständig. In dieser Methode wird, falls der Sensorknoten sich bereits im synchronisierten Zustand befindet, ein Synchronisationspaket bestehend aus `myRootID`, `myID` und `highestSeqNum` generiert und über die Methode `sendMsg(ftspMsg*msg)` verschickt. Des Weiteren werden der Zähler `heartBeats` und, falls der gerade betrachtete Sensorknoten sich selbst für die Wurzel hält, die `highestSeqNum` inkrementiert. Ein solches Paket gelangt auf der untersten Schicht an das Modul `mrfi`, das für das eigentliche Versenden der Nachricht sorgt. Die Daten werden dabei über das `RFD`-Register in den `TXFIFO` geschrieben. Anschließend erfolgt die Überprüfung auf Kanalfreiheit. Wurde das Medium als frei erkannt, so wird der Sendevorgang begonnen. Beim Versenden des SFD-Bytes wird zum einen hardwareseitig ein Zeitstempel des Timer 2 generiert und zum anderen eine IRQ ausgelöst. In der korrespondierenden ISR wird dieser Zeitstempel ausgelesen, über die Methode `localToGlobal(uint32 *time)` in die globale Referenzzeit transformiert und noch während des Sendevorgangs an das Ende der Nachricht angehängt (siehe auch Abbildung 2.2). Abhängig von der verwendeten Timeline werden dabei entweder die unteren oder oberen vier Bytes des Timer 2 (vgl. Abschnitt 4.1) an die Nachricht angehängt. Für diese Funktionalität musste die Sendemethode `MRFI_Transmit(mrfiPacket_t *pPacket, uint8_t txType)` und die ISR des `mrfi`-Moduls modifiziert werden. In der Sendemethode wird zunächst die SFD-Unterbrechung deaktiviert um zu verhindern, dass während des CCA eine Unterbrechung ausgelöst wird. Des Weiteren wird das Längenfeld aus Tabelle 3.1 um die Anzahl der Bytes, die zusätzlich durch den Zeitstempel entstehen, vergrößert. Erst nach dem CCA-Algorithmus wird die SFD-Unterbrechung wieder aktiviert.

4.3.2 Empfängersicht

Geht eine Synchronisationsnachricht bei einem Sensorknoten ein, so wird beim Empfang des SFD-Bytes eine IRQ ausgelöst und in der entsprechenden ISR des `mrfi`-Moduls der Hardware-Zeitstempel des Timer 2 in einem Puffer `SFDTimeStamp` zwischengespeichert, welcher $L_i(t)$ im FTSP repräsentiert. Dieser Schritt ist notwendig, da der Zeitstempel erst nach dem Erhalt der vollständigen Nachricht in diese eingefügt wird (siehe Ab-

bildung 2.2). Der Zeitstempel wird hardwareseitig automatisch generiert und könnte zwischenzeitlich durch eine neue Nachricht überschrieben werden. Softwareseitig bleibt dieser bis zum erneuten Auslösen der ISR erhalten. Sofern Daten im `RXFIFO` vorhanden sind, werden diese in der ISR in einen weiteren Puffer `mrfiIncomingPacket` geschrieben. Nach dem Auslesen des `RXFIFO` wird dann der `SFDTimeStamp` ans Ende des Paketes angehängt. So wird sichergestellt, dass immer der korrekte Zeitstempel mit der empfangenen Nachricht verknüpft ist. Durch den angehängten Zeitstempel wird die Paketgröße verändert, daher wird das Längenfeld aus Tabelle 3.1 um vier inkrementiert. Nach erfolgreicher zyklischer Redundanzprüfung (CRC) wird dann die Methode `MRFI_RxCompleteISR()` des Moduls `mrfi_LinkLayer` aufgerufen. Diese führt eine Funktion der Anwendungsschicht aus, die zuvor über einen Funktionszeiger im Modul `mrfi_LinkLayer` registriert wurde. Diese Funktion kann zum Beispiel zum Setzen eines Ereignisses für einen Protothread genutzt werden, mit dem die Anwendungsschicht über den Empfang einer neuen Nachricht in Kenntnis gesetzt wird. Die weitere Verarbeitung einer Nachricht muss durch die Anwendungsschicht durch das Aufrufen von `ftspReceive(puffer)` des Moduls `ftsp_Layer` initiiert werden. Diese ruft wiederum die Methode `ftspReceiverTask()` auf, welche für das Verarbeiten einer eingehenden Nachricht zuständig ist.

Zunächst werden die Daten aus dem empfangenen Paket extrahiert. Ein FTSP-Paket ist im Nutzdatenbereich des IEEE 802.15.4 Formats (vgl. Tabelle 3.1) abgelegt und folgendermaßen aufgebaut:

rootID	nodeID	seqNum	PayloadSize	Payload	GlobalTime
1 Byte	1 Byte	2 Bytes	1 Byte	0..n Byte(s)	4 Bytes

Tabelle 4.1: FTSP-Frame-Format

Der empfängerseitige lokale Zeitstempel erweitert dabei dieses Format um weitere vier Bytes. Diese Informationen werden in einer C-Struktur `ftspMsg` abgelegt:

```
1 typedef struct{
2
3   uint8   rootID;                    // ID der Wurzel
4   uint8   nodeID;                    // ID des Senders
5   uint16  seqNum;                    // Sequenznummer
6   uint32  globalTime;                // Schätzung der globalen Zeit
7   uint32  localTime;                 // Empfänger: SFD Timestamp
8   uint8   payload[PAYLOADSIZE+1];    // Nutzdaten
9   uint8   validTime;                 // TRUE, wenn Zeitinformation gültig ist
10
11 } ftspMsg;
```

Programmauszug 4.1: ftspMsg

Anhand der `seqNum` und der `rootID` wird nun entschieden, ob das Paket zur Berechnung der globalen Zeit berücksichtigt wird:

```
1
2 void  ftspReceiverTask(){
3       //Details zur Extraktion der Informationen wurden hier ausgeblendet
4
5       if(plSize>0) dataRdy=TRUE;
6
7       if (msg->rootID < myRootID){
```

```
8           myRootID = msg->rootID;
9           highestSeqNum = msg->seqNum;
10          clearTable();
11        } else if ((myRootID == msg->rootID) && (msg->seqNum > highestSeqNum)) {
12          highestSeqNum = msg->seqNum;
13        } else return;
14
15        if (myRootID < myID) heartBeats = 0;
16
17        addNewEntry(msg);       //Eintrag in Regressiontabelle einfügen
18        linearRegression();     //Regression ausführen
19  }
```

Programmauszug 4.2: Auszug aus ftspReceiverTask()

In Zeile 7 wird überprüft, ob die Wurzel der erhaltenen Nachricht kleiner ist als die dem Sensorknoten bekannte Wurzel. In diesem Fall wird die neue Wurzel übernommen, die höchste Sequenznummer korrigiert und die Regressionstabelle geleert (Zeile 8-10). Handelt es sich lediglich um eine neue Synchronisationsnachricht der aktuellen Wurzel (Zeile 11), so wird nur die höchste Sequenznummer aktualisiert. Andernfalls wird die Nachricht verworfen (Zeile 13). Sofern der Sensorknoten nicht als Wurzel deklariert ist, wird der `heartBeats`-Zähler zurückgesetzt (Zeile 15). Anschließend werden die neuen Informationen in die Regressionstabelle eingefügt und die Berechnung der Linearparameter der Regressionsgerade angestoßen (Zeile 17 und 18). Die Details zur Verwaltung der Regressionstabelle werden im Abschnitt 4.4 erläutert. Am Ende von `ftspReceive(buff)` werden, sofern vorhanden, die Nutzdaten in die Variable `buff` kopiert und die Anzahl der kopierten Bytes an die Applikation zurückgegeben.

4.4 Verwaltung der Regressionstabelle

Die Regressionstabelle beinhaltet die letzten n Wertepaare (`localTime`,`offset`), wobei `localTime` und `offset` jeweils $L_i(t_k)$ und $O_i(t_k)$ aus Abschnitt 2.3.1 beschreiben. Ein Wertepaar ist dabei in einer C-Struktur `TableItem` zusammengefasst:

```
1 typedef struct{
2   uint32  localTime;  //Receiver: SFD Timestamp
3   int32   offset; // globalTime - localTime
4 } TableItem;
5
6 static XDATA    TableItem     table[MAX_ENTRIES];    //Regressionstabelle
```

Programmauszug 4.3: TableItem

Die Regressionstabelle `table` ist dabei als ein Array realisiert. Die maximale Tabellengröße wird durch den Konfigurationsparameter `MAX_ENTRIES` beschrieben. Sie wird als einfache Ringstruktur durch die Zeiger `first` und `last` verwaltet. Der Zeiger `first` zeigt dabei immer auf das älteste Element und `last` immer auf den nächsten freien Eintrag. Ist die Regressionstabelle voll, so wird immer der älteste Eintrag überschrieben. Der Zähler `numEntries` gibt die Anzahl der gespeicherten Einträge wieder.

In der Methode `addNewEntry(ftspMsg* msg)` werden im nicht synchronisierten Zustand (`numEntries` < `NUMENTRIES_LIMIT`) eingehende Nachrichten immer berücksichtigt. Im synchronisierten Zustand hingegen wird in der Methode `getError(ftspMsg* msg)` aus dem Zeitstempel `localTime` mittels linearer Regression die erwartete globale

Zeit berechnet und die Differenz zur empfangenen `globalTime` gebildet. Nur wenn die Differenz kleiner als `ENTRY_THROWOUT_LIMIT` ist, erfolgt ein neuer Eintrag in die Regressionstabelle. So wird verhindert, dass einzelne fehlerhafte Pakete in die Regressiontabelle aufgenommen werden. Erst nach `MAX_ERRORS` aufeinanderfolgenden Überschreitungen von `ENTRY_THROWOUT_LIMIT` wird die eigene Regressionstabelle als ungenau angenommen und geleert. Dazu müssen in der Methode `clearTable()` lediglich die Zeiger `first` und `last` und der Zähler `numEntries` auf null gesetzt werden.

4.5 Zeitumwandlung

Die Methoden zur Zeitumwandlung werden in dem Modul `ftsp_mrfi_interface` bereit gestellt. Eine Schätzung der globalen Zeit kann dabei mittels der Methode `localToGlobal(uint32 *time)` berechnet werden:

```
1 void  localToGlobal(uint32 *time){
2   *time += oAverage + (int32)(Skew * (int32)(*time - lAverage));
3 }
```

Programmauszug 4.4: Methode zur Berechnung der globalen Zeit (vgl. [24])

Als Eingangsparameter `*time` wird dabei ein Wert für die lokale Zeit erwartet, der in die globale Zeit umgewandelt werden soll. Die Variablen `int32 oAverage`, `uint32 lAverage` und `float Skew` werden dabei in der Methode `linearRegression()` bestimmt und entsprechen den Variablen $\overline{O_i}$, $\overline{L_i}$ und m_i aus Gleichung 2.19.

Die Bestimmung der lokalen Zeit aus der globalen Zeit nach Gleichung 2.20 ist wegen der notwendigen Division mit einem höheren Rechenaufwand verbunden. Um diesen zu umgehen, wird in [3] Abschnitt 4 die folgende Näherung vorgeschlagen:

$$L_i(t) \overset{2.19}{=} G_i(t) - \overline{O_i} - m_i \cdot (L_i(t) - \overline{L_i}) \approx G_i(t) - \overline{O_i} - m_i \cdot (G_i(t) - \overline{O_i} - \overline{L_i}). \quad (4.1)$$

Im Anhang 9.2 wird gezeigt, dass der systematische Fehler dieser Näherung für $m_i \to 0$ vernachlässigt werden kann. Die Berechnung der lokalen Zeit aus der globalen Zeit wird daher wie folgt realisiert:

```
1 void  globalToLocal(uint32 *time){
2   uint32 elt = *time - oAverage;  \\ estimated local time
3   *time = elt - (int32)(Skew * (int32)(elt - lAverage));
4 }
```

Programmauszug 4.5: Methode zur Berechnung der lokalen Zeit (vgl. [24])

4.6 Berechnung mit Fließkommazahlen

Zur Gewährleistung einer ausreichenden Genauigkeit wird der Parameter `Skew` als IEEE 754 Fließkommazahl mit einfacher Genauigkeit dargestellt. Ein alternativer Ansatz, der mit Einschränkungen ganz ohne Fließkommazahlen auskommt, wird im Abschnitt 4.7 vorgestellt. Fließkommazahlen werden abgesehen von den Methoden `localToGlobal(uint32 *time)` und `globalTolocal(uint32 *time)` auch in der Methode `lineareRegression()` eingesetzt:

```
1 void linearRegression(){
2
3   uint32 localAverage=0;
4   int32  offsetAverage=0;
5   char   i, end;
6   float  lsum=0.0, osum=0.0;
7
8   if(first<last)end=last;
9   else end=MAX_ENTRIES;
10
11  //Details zur Berechnung von localAverage und OffsetAverage wurden ausgeblendet
12
13  for(i = 0; i < end; i++) {
14      int32 a = table[i].localTime - localAverage;
15      int32 b = table[i].offset - offsetAverage;
16      lsum   += ((float)a) * ((float)a);
17      osum   += ((float)a) * ((float)b);
18  }
19
20  setLocalAverage(localAverage);
21  setOffsetAverage(offsetAverage);
22  if(lsum!=0) setSkew(osum/lsum);
23 }
```

Programmauszug 4.6: Ausschnitt aus der linearen Regression (vgl. [24])

In der `for`-Schleife von Zeile 13-18 werden Zähler und Nenner des Skews (vgl. Gleichung 2.16) einzeln berechnet. Die Variable `a` beschreibt dabei den Term $(L_i(t_k) - \overline{L_i})$ und `b` den Term $(O_i(t_k) - \overline{O_i})$. Problematisch sind hierbei die Multiplikationen $a \cdot b$ und $a \cdot a$. Sie können ohne weiteres bei einer reinen Ganzzahl-Umsetzung zu Bereichsüberläufen führen. Aus diesem Grunde erfolgt in Zeile 16 und 17 die Typumwandlung auf Fließkommazahlen, die einen wesentlich größeren Darstellungsbereich aufweisen als Ganzzahlen. In den Zeilen 20-22 werden die berechneten Werte dem Modul `ftsp_mrfi_interface` übergeben.

Bei der Berechnung der Mittelwerte wurde auf eine Fließkomma-Umsetzung zur Vermeidung von Bereichsüberläufen verzichtet und stattdessen der Ansatz aus [12, 24] übernommen.

Der arithmetische Mittelwert einer Zahlenfolge ist definiert als

$$\overline{x} = \frac{1}{n} \sum_{i=0}^{n-1} x_i. \tag{4.2}$$

Um die Gefahr eines Überlaufs zu verringern, muss die Division in die Summe gezogen werden. Demnach berechnet sich der Mittelwert von Ganzzahlen folgendermaßen:

$$\overline{x} = \sum_{i=0}^{n-1} \lfloor \frac{x_i}{n} \rfloor. \tag{4.3}$$

Wegen des wiederholten Abschneidens der Nachkommastellen ist dieses Vorgehen jedoch ungenau. Einen genaueren Mittelwert kann man durch die Berücksichtigung des Divisionsrestes mit Hilfe der Modulo-Zerlegung

$$x = n \cdot \lfloor \frac{x}{n} \rfloor + (x \% n) \tag{4.4}$$

berechnen:

$$\overline{x} \overset{4.2}{=} \frac{1}{n}\sum_{i=0}^{n-1} x_i \overset{4.4}{=} \frac{1}{n}\sum_{i=0}^{n-1}(n\cdot\lfloor\frac{x_i}{n}\rfloor+(x_i\%n)) = \sum_{i=0}^{n-1}\lfloor\frac{x_i}{n}\rfloor+\frac{1}{n}\sum_{i=0}^{n-1}(x_i\%n) \quad (4.5)$$

Die Berechnung der Mittelwerte im Programmauszug 4.7 setzt diese Gleichung in der `for`-Schleife um:

```
1   int32  localAverageRest  = 0;
2   int32  offsetAverageRest = 0;
3   uint32 localAverageTemp  = 0;
4   int32  offsetAverageTemp = 0;
5
6   localAverage  = table[first].localTime;
7   offsetAverage = table[first].offset;
8
9   for(i=0;i<end;i++){
10    localAverageTemp  += (table[i].localTime      - localAverage)  / numEntries;
11    localAverageRest  += (table[i].localTime      - localAverage)  % numEntries;
12    offsetAverageTemp += (int32)(table[i].offset  - offsetAverage) / numEntries;
13    offsetAverageRest += (table[i].offset         - offsetAverage) % numEntries;
14  }
15
16  localAverage  += localAverageTemp  + localAverageRest  / numEntries;
17  offsetAverage += offsetAverageTemp + offsetAverageRest / numEntries;
```

Programmauszug 4.7: Berechnung der Mittelwerte (vgl. [24])

Dieser Ansatz erbrachte eine höhere Genauigkeit als die Berechnung der Mittelwert in Fließkommazahlen und das spätere Umcasten auf Ganzzahl-Werte. Durch die Differenzenbildung wird die Gefahr von Überläufen reduziert. Ungenauigkeiten, die auf Berechnungen mit Fließkommazahlen zurückzuführen sind, werden im Abschnitt 7.1 aufgeführt.

4.7 Berechnung mit Ganzzahlen

Die Verwendung der Fließkomma-Bibliotheken erhöht den Bedarf an Programmspeicher und ggf. auch die notwendige Rechenzeit (vgl. Abschnitt 4.9). In diesem Abschnitt wird daher eine Implementierung vorgestellt, die ohne Fließkommazahlen auskommt.

Da Ganzzahlen keine Nachkommastellen darstellen können, werden Zähler und Nenner des gebrochen rationalen `Skew`-Parameters getrennt verwaltet. Problematisch bei einer Ganzzahl-Umsetzung sind vor allem Multiplikationen, die zu Bereichsüberläufen führen können. Um dieses zu verhindern, werden Skalierungsfaktoren eingeführt. Der Parameter `Skew` aus Programmauszug 4.6 lässt sich folgendermaßen darstellen:

$$Skew = \frac{a_0\cdot b_0+\ldots+a_{n-1}\cdot b_{n-1}}{a_0\cdot a_0+\ldots+a_{n-1}\cdot a_{n-1}} \quad (4.6)$$

Da $Skew$ ein Verhältnis beschreibt und dieses gleich bleibt, wenn Zähler und Nenner gleichermaßen durch einen Faktor c skaliert werden, lässt sich Gleichung 4.6 auch durch

$$Skew = \frac{\frac{1}{c}(a_0\cdot b_0+\ldots+a_{n-1}\cdot b_{n-1})}{\frac{1}{c}(a_0\cdot a_0+\ldots+a_{n-1}\cdot a_{n-1})} = \frac{\frac{a_0}{c}\cdot b_0+\ldots+\frac{a_{n-1}}{c}\cdot b_{n-1}}{\frac{a_0}{c}\cdot a_0+\ldots+\frac{a_{n-1}}{c}\cdot a_{n-1}} \quad (4.7)$$

beschreiben. Eine solche Skalierung kann effizient durch einen arithmetischen Rechtsshift um `SCALEFACTOR1` realisiert werden. Der Quelltext aus Programmauszug 4.6 wird folgendermaßen modifziert:

```
1   int32 zaehler=0;
2   uint32 nenner=0;
3   for(i = 0; i < end; i++) {
4     int32 a = table[i].localTime - localAverage;
5     int32 b = table[i].offset    - offsetAverage;
6
7     zaehler += (a>>SCALEFACTOR1) * b;
8     nenner  += (a>>SCALEFACTOR1) * a;
9   }
10
11  setLocalAverage(localAverage);
12  setOffsetAverage(offsetAverage);
13  setZaehler(zaehler);
14  setNenner(nenner);
```

Programmauszug 4.8: Auschnitt aus der linearen Regression als Integer-Umsetzung

Weiterhin verändern sich die Methoden zur Zeitumwandlung zu:

```
1 void  localToGlobal(uint32 *time){
2   *time += oAverage + ((int32) ((int32)(zaehler * (int32)(*time - lAverage))) / (
        int32)nenner);
3 }
```

Programmauszug 4.9: localToGlobal als Integer-Umsetzung

```
1 void  globalToLocal(uint32 *time){
2   uint32 elt = *time - oAverage; \\ estimated local time
3   *time = elt - ((int32)((zaehler * (int32)(elt - lAverage)))/(int32)nenner);
4 }
```

Programmauszug 4.10: globalToLocal als Integer-Umsetzung

Die Variablen `zaehler` und `nenner` werden in den jeweiligen `set`-Methoden durch einen weiteren Skalierungsfaktor `SCALEFACTOR2` geteilt:

```
1 void setZaehler(int32 z){
2   zaehler=z>>SCALEFACTOR2;
3 }
```

Programmauszug 4.11: Skalierung in setZaehler

```
1 void setNenner(uint32 n){
2   nenner=n>>SCALEFACTOR2;
3 }
```

Programmauszug 4.12: Skalierung in setNenner

Die Skalierungsgröße $SCALEFACTOR1$ sorgt dafür, dass das Risiko eines Bereichsüberlaufs bei der Multiplikation und anschließender Summation der Terme gesenkt wird. Allerdings führen zu große Werte von $SCALEFACTOR1$ dazu, dass die einzelnen Summanden des Zählers zu ungenau repräsentiert werden. Daher erfolgt durch $SCALEFACTOR2$ eine erneute Skalierung der Zähler- und Nennersumme um auch in den Methoden 4.9 und 4.10 die Gefahr der Bereichsüberläufe zu reduzieren, ohne die Genauigkeit zu stark einzuschränken.

4.8 FTSP als Netzwerkprotokollschicht

Das FTSP wurde in dieser Arbeit als Netzwerkprotokollschicht implementiert. Über diese Schicht ist es möglich, Nutzdaten zusammen mit den notwendigen Informationen zur Zeitsynchronisation in das Sensornetz zu fluten. Diese Umsetzung hat den Vorteil, dass die Netzwerkauslastung verringert wird und Energieressourcen durch Vermeidung von leeren Synchronisationsnachrichten geschont werden.

4.8.1 Schnittstellen

Die Tabellen 4.2 bis 4.4 geben einen Überblick über die Schnittstellen, die durch das Einbinden von ftsp_Layer.h verfügbar werden. Diese sind in den Modulen `ftsp_Layer`, `ftsp_mrfi_Layer` bzw. `mrfi_LinkLayer` implementiert.

`Schnittstelle`	Beschreibung

`void ftspInit(`
`uint16 myAddress,`
`uint16 myPanID,`
`uint8 myNodeID)`
	Initialisiert die FTSP-Schicht inklusive der darunter liegenden Schichten. Als Eingangsparameter werden die eigene Netzwerkadresse, die PAN-ID und die eigene Knoten-ID (`myID`) erwartet. Die ersten beiden Parameter werden dem Modul `mrfi` übermittelt und beschreiben die Quelle einer Nachricht. Weiterhin werden die Zeitgeber Timer 2 als Timeline und Timer 1 als Zeitgeber für eine Synchronisationsperiode gestartet.

`uint8 ftspReceive(` `uint8* msg)`	Kopiert die Nutzdaten einer empfangenen Nachricht in den Puffer `msg` und gibt die Anzahl der kopierten Bytes zurück. Erst durch den Aufruf dieser Methode erfolgt die Verarbeitung der Informationen zur Zeitsynchronisation.
`void ftspRxConnect(` `ISR_FUNC_PTR isr)`	Legt einen Zeiger auf eine Funktion ab, die nach einer vollständig empfangenen Nachricht aufgerufen werden soll. Diese Funktion wird direkt in der Unterbrechungsroutine aufgerufen und sollte möglichst einfach gestaltet sein wie etwa zum Setzen eines Ereignisses für ein Protothread. Diese Methode leitet dabei den Funktionszeiger über `mrfiLinkRxConnect` an die `mrfi_LinkLayer`-Schicht weiter.
`void ftspSend(` `uint8* msg, uint8 size)`	Broadcastet die Nutzdaten `msg` inklusive der Daten zur Zeitsynchronisation in das Sensornetz. Nachrichten, die über diese Schnittstelle versendet werden, enthalten als Zielinformationen die Adresse 0xFFFF und die PAN-ID `MRFI_LINK_PAN_ID`. Diese Präprozessordirektive kann im Modul `mrfi_LinkLayer` verändert werden und ist standardmäßig auf den Wert 0x2007 gesetzt.
`uint16` `getHighestSeqNum()`	Liefert die aktuelle Sequenznummer zurück.

`uint32 getLocalTime()`	Liefert einen Zeitstempel in lokaler Zeit zurück.
`uint8 getMyID()`	Liefert die ID des Sensorknotens zurück.
`uint8 getMyRootID()`	Liefert die vom Sensorknoten gesehene Wurzel zurück.
`uint8 getNumEntries()`	Liefert die aktuelle Anzahl der Einträge in der Regressionstabelle zurück.
`uint32 getSFDTimestamp()`	Liefert den lokalen Zeitstempel der letzten empfangenen Nachricht zurück.
`uint8 isSynchronized()`	Gibt an, ob sich der Netzwerkknoten im synchronisierten Zustand befindet.

Tabelle 4.2: FTSP-Schnittstellen

`Schnittstelle`	Beschreibung
`void globalToLocal( uint32 *time)`	Wandelt einen Wert aus der globalen Zeitskala in die lokale Zeitskala um.
`void localToGlobal( uint32 *time)`	Wandelt einen Wert aus der lokalen Zeitskala in die globale Zeitskala um.

Tabelle 4.3: ftsp_mrfi_Interface-Schnittstellen

Schnittstelle	Beschreibung
`uint8 mrfiLinkBroadcast( uint8 *pBuf, uint8 len, uint8 nRetrans)`	Über diese Methode können Nachrichten innerhalb der PAN-ID, die bei der Initialisierung festgelegt wurde, geflutet werden. Die Nachrichten werden stets mit der Zieladresse 0xFFFF verschickt. Diese ist hardwareseitig für Broadcast-Pakete reserviert.
`uint8 mrfiLinkDataRdy()`	Gibt *TRUE* zurück, falls ein neues Paket angekommen ist.
`void mrfiLinkInit( uint16 src, uint16 panID, uint8 mrfiChannel)`	Initialisiert die `mrfi`-Schicht und legt über den Eingangsparameter `mrfiChannel` den Kanal fest, auf dem die Kommunikation abläuft. Des Weiteren wird hier die Addresse des Sensorknotens, bestehend aus den Eingangsparametern `src` und `panID`, zur Filterung ankommender Pakete festgelegt. Diese Methode wird bereits durch die Schnittstelle `ftspInit` aufgerufen und sollte daher nur dann verwendet werden, wenn auf das Modul `ftsp_Layer` verzichtet wird.
`uint8 mrfiLinkRecv( uint8 *pBuf)`	Liest ankommende Daten aus dem Rx-Puffer aus.
`void mrfiLinkRxConnect( ISR_FUNC_PTR isr)`	Erhält als Eingangsparameter einen Funktionszeiger, der beim Empfang einer neuen Nachricht in der ISR aufgerufen wird. Die übergebene Funktion sollte daher möglichst schlank gestaltet sein.
`void mrfiLinkSrcMatch_enable( uint8* table, uint8 size)`	Aktiviert die Filterung von Paketen anhand der Quelladressen. Es werden nur Pakete akzeptiert, die als Eingangsparameter in Form eines Arrays übergeben werden.
`uint8 mrfiLinkTransmit( uint8 *pBuf, uint8 len, uint16 dst, uint16 panID, uint8 pktType, uint8 nRetrans)`	Die Methode dient zum Verschicken von Unicast-Nachrichten. Die Zieladresse des Empfängers muss hier als Eingangsparameter übergeben werden. Über `pktType` wird der Typ des Pakets festgelegt. Mit `nRetrans` wird die Anzahl der Sendeversuche angeben, die unternommen werden sollen, falls beim ersten Sendevorgang die CCA-Prüfung fehlschlägt.

Tabelle 4.4: mrfi_LinkLayer-Schnittstellen

4.8.2 Automatische Generierung von Synchronisationsnachrichten

Das FTSP setzt voraus, dass in bestimmten Zeitabständen Synchronisationspakete empfangen werden, damit die Parameter zur Bestimmung der globalen Zeit berechnet werden können. Daher ist es notwendig, dass solche Synchronisationsnachrichten auch dann automatisch verschickt werden, wenn keine Nutzdaten von der Anwendungsschicht anfallen. Dazu werden neben der `highestSeqNum` zwei weitere Variablen `lastSentSeqNum` und `lastSentRootID` verwaltet (vgl. [27]). Diese geben die letzte tatsächliche versendete Sequenznummer und die ID der Wurzel wieder. Leere Synchronisationsnachrichten wer-

den nur dann verschickt, wenn `myRootID` ≠ `lastSentRootID` oder `highestSeqNum` ≠ `lastSentSeqNum` gilt. Die erste Bedingung ist notwendig, da bei einem Wechsel der Wurzel `highestSeqNum` und `lastSentSeqNum` übereinstimmen könnten, obwohl der Sensorknoten noch kein Paket mit der neuen Wurzel verschickt hat. Durch die zweite Bedingung wird sichergestellt, dass keine reinen Synchronisationspakete generiert werden, falls die Synchronisationsinformationen bereits mit dem letzten Nutzdatenpaket versendet wurden. Wurde zum Beispiel zum Zeitpunkt i_0 ein neues Paket mit den Informationen zur Synchronisation empfangen, wird `highestSeqNum` erhöht und es gilt `highestSeqNum` ≠ `lastSentSeqNum`. Erfolgt nun seitens der Applikation ein Sendevorgang, so wird `lastSentSeqNum` zu `higestSeqNum` aktualisiert. Dies hat zur Folge, dass beim Ablaufen des Zeitgebers Timer 1 `lastSentSeqNum = highestSeqNum` gilt. Dann wird keine Nachricht automatisch generiert. Fallen jedoch nach dem Erhalt neuer Synchronisationsinformationen keine Nutzdaten an, sodass ein Sendevorgang seitens der Applikation ausbleibt, dann gilt nach dem Ablauf des Zeitgebers Timer 1 weiterhin `lastSentSeqNum` ≠ `highestSeqNum`. In diesem Fall wird dann ein Synchronisationspaket ohne Nutzdaten generiert:

```
1  void  ftspSenderTask(){
2    heartBeats++;
3    if(myRootID!=myID && heartBeats>= ROOT_TIMEOUT){
4      if(myRootID ==0xFF)
5          highestSeqNum=0;
6      myRootID=myID;
7    }
8    if(isSynchronized()){
9      if(!(highestSeqNum==lastSentSeqNum && myRootID==lastSentRootID)){
10       msg->rootID=myRootID;
11       msg->nodeID=myID;
12       msg->seqNum=highestSeqNum;
13       msg->payload[0]=0;
14
15       lastSentSeqNum=msg->seqNum;
16       lastSentRootID=msg->rootID;
17       sendMsg(msg);
18     }
19     if(myRootID==myID)
20         highestSeqNum++;
21   }
22 }
```

Programmauszug 4.13: Ausschnitt aus ftspSenderTask()

Die Implementierung des FTSP in [23] sieht es vor, dass Synchronisationsnachrichten nur dann verschickt werden, wenn ein Knoten sich im synchronisierten Zustand befindet. Da in der hier dargestellten Implementierung die Synchronisationsinformationen immer auch zusammen mit den Nutzdaten verschickt werden, ist es notwendig, dass der Empfänger weiß, ob der Sender eines Paketes zum Senden von Synchronisationsinformationen berechtigt war. Dazu wird das meist signifikante Bit des Feldes `nodeID` aus Tabelle 4.1 genau dann gesetzt, wenn der Sender synchronisiert ist. Dadurch beschränkt sich die maximale Anzahl der Sensorknoten auf insgesamt 128. Sollen mehr als 128 Sensorknoten eingesetzt werden, so muss das Feld erweitert oder ein zusätzliches Byte für dieses Bit spendiert werden. Diese Information wird beim Empfangen einer Nachricht in der Variable `validTime` aus Programmauszug 4.1 abgelegt. Nur wenn `validTime` 1 ist, werden eingehende Pakete zur Durchführung der linearen Regression berücksichtigt.

4.9 Ressourcenbedarf

4.9.1 Quelltextgröße

Anders als bei den gängigen Einzelplatzrechnern, sind die Speicherressourcen bei den Mikrocontrollern von Sensorknoten stark begrenzt. Die in dieser Arbeit verwendeten SoCs verfügen über einen 256 kB großen Flash-Speicher und einen 8 kB großen SRAM, daher ist eine sparsame Speicherbelegung notwendig.

Tabelle 4.5 gibt einen Überblick über den reservierten Speicher der Fließkomma- und Ganzzahl-Umsetzung. Die Unterschiede der Fließkomma- bzw. Ganzzahl-Implementierung machen sich vor allem in den Methoden `linearRegression` und `localToGlobal` bemerkbar, daher sind diese gesondert angegeben. Die Gesamtcodegröße beschreibt die Speicherbelegung eines lauffähigen Programms. Zur Ermittelung dieser Daten wurde der Ressourcenverbrauch der Applikation `ftsp_accuracy_query` (vgl. Abschnitt 6) analysiert.

Codegröße in Bytes					
	linearRegression	localToGlobal	Gesamtcodegröße		
			DATA	XDATA	CODE
Ganzzahl	1437	265	256	1326	19216
Fließkommazahl	1565	280	256	1326	20040

Tabelle 4.5: Speicherbelegung

Man erkennt, dass für die Fließkomma-Implementierung 824 Bytes zusätzlich spendiert werden müssen, welche im Wesentlichen auf die Fließkomma-Bibliotheken des Compilers zurückzuführen sind. Die in dieser Tabelle eingetragenen Speicherbereiche haben folgende Bedeutung: Der Bereich CODE ist für den Programmcode reserviert. Es handelt sich um einen 32 kB großen Speicherbereich, auf den nur lesend zugegriffen werden kann. DATA hingegen verweist auf einen 256 Bytes großen Bereich, der für häufig verwendete Variablen und den Programmstack reserviert ist. Mit XDATA wird ein 8 kB großer Speicherbereich angesprochen, dessen Zugriffszeit im Vergleich zu DATA vier- bis fünf-fach größer ist.

4.9.2 Ausführungszeiten

Neben der Quelltextgröße sind vor allem die Ausführungszeiten der Methoden `linearRegression()` und `localToglobal(uint32 *time)` interessant, da diese besonders intensiv beziehungsweise häufig die MCU beanspruchen. Zur Messung der Ausführungszeit wurde ein Zeitstempel des Timer 2 vor und nach Ausführung dieser Methoden erhoben und daraus die Differenz gebildet. Diese wurden bei der Fließkomma-Implementierungen für die Timeline mit einer Auflösung von 8 μs und 31,25 ns und bei der Integer-Implementierung für die 8 μs Auflösung gemessen. Die über 37 Messungen gemittelten Ausführungszeiten sind in Tabelle 4.6 eingetragen.

Implementierung	Auflösung	linearRegression	localToGlobal
Fließkommazahl	31,25 ns	3312 μs	83 μs
Fließkommazahl	8 μs	2886 μs	83 μs
Ganzzzahl	8 μs	2300 μs	149 μs

Tabelle 4.6: Ausführungszeiten

Vergleicht man die Ganzzahl- mit der Fließkomma-Implementierung, so ist zu erkennen, dass bei einer Zeitauflösung von 8 μs die lineare Regression ohne die Verwendung von Fließkommazahlen schneller und die Zeitumrechnung langsamer realisiert wird. Die lineare Regression wird nur beim Empfang von neuen Synchronisationsinformationen durchgeführt. Kritischer ist die Ausführungszeit der Zeitumwandlung. Diese kann von der Applikation, je nach anfallenden Nutzdaten, öfters ausgeführt werden, sodass der Vorteil der Fließkomma-Implementierung überwiegt.

4.9.3 Schätzung zur Leistungsaufnahme

Der Energieverbrauch ist stark von der Applikation abhängig, die die FTSP-Schicht nutzen soll. Hier werden daher nur Schätzungen für die energielastigen Operationen angegeben.

Die Leistung berechnet sich zu $P = U \cdot I$, wobei für die Versorgungspannung $U = 3V$ angenommen wird. Die Berechnung der linearen Regression beansprucht die MCU in der Fließkomma-Implementierung für etwa 3,31 ms. Für eine MCU-Aktivität ohne den Betrieb von sonstiger Peripherie ist im Datenblatt [20] die Stromstärke mit 6,5 mA angegeben. Der Energieverbrauch kann daher zu

$$E_{reg} = 6,5 \cdot 10^{-3} A \cdot 3\,V \cdot 3,3 \cdot 10^{-3} s = 65 \cdot 10^{-6} J$$

abgeschätzt werden.

Für den Sende- und Empfangsvorgang wird der Kommunikationsoverhead betrachtet, der durch die Verwendung des FTSP-Protokolls entsteht. Der Energieverbrauch berechnet sich unter der Verwendung der Funk-Einheit wie folgt:

$$E = \frac{\text{Anzahl der übertragenen Bits} \cdot \text{Leistungsaufnahme Radio}}{\text{Datenübertragungsrate}}. \tag{4.8}$$

Gemäß der Tabelle 4.1 besteht ein Synchronisationspaket ohne Nutzdaten aus 72 Bits. Bei einer Signaleingangsleistung von -100 dBm fließt gemäß dem Datenblatt [20] ein Strom I in der Größenordnung von etwa 24,3 mA. Bei einer Datenübertragungsrate von 250 kBit/s ergibt sich nach Gleichung 4.9.3 eine Leistungsaufnahme von etwa

$$E_{rx} = \frac{72\,Bits \cdot 24,3 \cdot 10^{-3} A \cdot 3\,V}{250 \cdot 10^{3}\,Bits/s} = 20,9952 \cdot 10^{-6} J.$$

Im Sendemodus hängt der Energieverbrauch vor allem von der Sendeleistung ab, wobei das `mrfi`-Modul standardmäßig mit 1 dBm sendet. Der typische Stromverbrauch für

diesen Wert ist in [20] zu 29 mA angegeben. Damit ergibt sich gemäß Gleichung 4.8 ein Energieverbrauch von etwa

$$E_{tx} = \frac{72\,Bits \cdot 29 \cdot 10^{-3}\,A \cdot 3\,V}{250 \cdot 10^{3}\,Bits/s} = 25,056 \cdot 10^{-6}\,J.$$

Bei diesen Schätzwerten ist zu beachten, dass die Kosten für die laufende Peripherie wie die Zeitgeber Timer 1 und Timer 2 nicht enthalten sind. Zusammen erhöhen sie die mittlere Leistungsaufnahme um etwa 540 μW (vgl. [20] und [18]).

4.10 Zusammenfassung

4.10.1 Überblick der Module

Abbildung 9.2 gibt einen Überblick über die verwendeten Module. Die Module zur Funkkommunikation sind im Ordner rf abgelegt. Die Headerdatei ftsp_Layer.h legt die nach außen sichtbaren Schnittstellen der FTSP-Schicht fest. In ihr sind auch diverse Parameter zu finden, über die sich die FTSP-Schicht konfigurieren lässt (vgl. Abschnitt 4.10.2). Die Datei ftsp_Layer.c enthält die eigentliche Implementierung der FTSP-Schicht.

Die Dateien ftsp_mrfi_interface.h und ftsp_mrfi_interface.c enthalten die Methoden zur Zeitumwandlung. Diese Schnittstellen werden dabei direkt in der Header-Datei ftsp_Layer.h eingebunden und sind damit ebenfalls für die Anwendung nach außen sichtbar.

Das Modul `mrfi_LinkLayer` ist als weitere Netzwerkschicht dem `ftsp_Layer` und der `mrfi`-Schicht zwischengeschaltet. Die FTSP-Schicht ist nur zum Versenden von Nachrichten an alle Teilnehmer gedacht. Soll eine gerichtete Kommunikation zwischen bestimmten Teilnehmer erfolgen, so kann die Schnittstelle `mrfiLinkTransmit(uint8*pBuf,uint8 len,uint16 dst,uint16 panID, uint8 pktType,uint8 nRetrans)` genutzt werden. Des Weiteren kann über `mrfiLinkRxConnect(ISR_FUNC_PTR isr)` eine Funktion aus der Applikation registriert werden, die beim vollständigen Empfangen einer Nachricht automatisch in der Unterbrechungsroutine aufgerufen wird. Diese Möglichkeit ist zum Setzen eines Ereignisses gedacht. Diese Schnittstellen sind ebenfalls für die Applikation sichtbar, da sie ebenfalls direkt in der Header-Datei ftsp_Layer.h eingebunden werden. Weiterhin kann diese Schicht um zusätzliche Pakettypen erweitert werden.

Die unterste Netzwerkschicht ist die `mrfi`-Schicht. Dieses Modul ist allein für die Funk-Kommunikation zuständig und enthält auch die Implementierung der entsprechenden `ISR`, in der die Zeitstempel zum Sende- und Empfangszeitpunkt des SFD-Bytes generiert werden.

Der Ordner hal enthält die Module der Hardwareabstraktionsschicht. Unter anderem befindet sich darin das Modul `timeline`. In diesem werden die Zeitgeber Timer 2 und SleepTimer konfiguriert. Timer 2 dient zur Repräsentation der Timeline. Des Weiteren enthält dieser Ordner die Dateien timer_32k.h und timer32_k.c, die zur Konfiguration

des Timer 1 als 32 kHz Zeitgeber dienen. Dieser wird in der FTSP-Schicht zur Definition eines Synchronisationszyklusses benutzt.

Der Ordner os enthält alle betriebssystemrelevanten Dateien. In dieser Implementierung werden lediglich die ereignisgesteuerten Contiki Protothreads benutzt.

4.10.2 Konfigurationsmöglichkeiten

In diesem Abschnitt werden die verschiedenen Konfigurationsparameter zusammengefasst.

`ENTRY_THROWOUT_LIMIT` ftsp_Layer.h	Maximal erlaubte Abweichung zwischen sender- und empfängerseitigem SFD-Zeitstempel in globaler Zeit. Nur Pakete, die eine kleinere Differenz aufweisen, werden in die Regressionstabelle aufgenommen.
`HEARTBEAT_CYCLE` ftsp_Layer.h	Länge eines Synchronisationsintervalls (LSB $\approx$ 0,25 s) bzw. Periode, mit der `heartBeats` hochgezählt und von der Wurzel eine neue Synchronisationsrunde gestartet wird.
`INTEGER_ESTIMATION` ftsp_mrfi_interface.h	Präprozessordirektive, die einkommentiert werden muss, damit von der Fließkomma-Umsetzung auf die Integer-Umsetzung umgeschaltet wird.
`_LOWER_BYTE_` ftsp_mrfi_interface.h	Präprozessordirektive, die einkommentiert werden muss, um die Timeline-Auflösung von 8 μs auf 31,25 ns umzuschalten. Falls kein Wert für `HEARTBEAT_CYCLE` eingetragen wurde, so wird bei der höheren Auflösung automatisch ein Synchronisationintervall von etwa 4 s eingestellt.
`MAX_ENTRIES` ftsp_Layer.h	Maximale Größe der Regressionstabelle.
`MAX_ERRORS` ftsp_Layer.h	Maximale Anzahl der Synchronisationspakete, deren Zeitstempel stark von der aktuellen Regressionsgeraden abweichen. Erst nach der Überschreitung dieses Schwellwertes wird die Regressionstabelle verworfen.
`MRFI_BASE_PAN_ID` mrfi_LinkLayer.h	PAN-ID der Basestation. Diese PAN-ID wird zum Versenden der REPLY-Pakete genutzt. (vgl. Abschnitt 6.1)

`MRFI_CHANNEL` ftsp_Layer.h	Frequenzkanal, auf dem die Nachricht verschickt werden soll. Standardmäßig ist `MRFI_CHANNEL` auf 0 gesetzt. Es darf dabei die Werte 0,1,2,3 annehmen. Sie stehen für die Frequenzen 2425 MHz, 2450 MHz, 2475 MHz, 2480 MHz. Soll auf einer anderen Frequenz innerhalb 2394 MHz und 2507 MHz gesendet werden, so muss die Tabelle `mrfiLogicalChanTable` im Modul `mrfi` erweitertet werden. Weitere Informationen dazu sind in [16] Abschnitt 23.5 zu finden.
`MRFI_LINK_DATA` mrfi_LinkLayer.h	Dient zur Kennzeichnung von FTSP-Paketen. Nur wenn ein Paket diese Kennung trägt, wird im Modul `mrfi_LinkLayer` die Anwendung über die Ankunft einer neuen Nachricht informiert.
`MRFI_LINK_PAN_ID` mrfi_LinkLayer.h	PAN-ID, unter der die FTSP-Kommunikation abläuft. Sie wird als Quell-PAN-ID der Sensorknoten eingetragen und ist standardmäßig auf 0x2007 gesetzt. Nachrichten, die über die Schnittstelle `mrfiLinkBroadcast` verschickt werden, tragen automatisch diese als Ziel-PAN-ID.
`NUMENTRIES_LIMIT` ftsp_Layer.h	Mindestanzahl an Einträgen in der Regressionstabelle, die zur Berechnung der Regressions-Parameter vorrausgesetzt werden.
`OFFSET_CORRECTION` ftsp_Layer.h	Experimenteller Parameter, der zur Verbesserung der Synchronisationsgenauigkeit dient. Er wird empfängerseitig in der Methode `ftspReceiverTask()` zur tatsächlich erhaltenen Schätzung der globalen Zeit hinzu addiert.
`PAYLOADSIZE` ftsp_Layer.h	Maximale Größe der Nutzdaten in Bytes, die über die FTSP-Schicht verschickt werden können.
`ROOT_TIMEOUT` ftsp_Layer.h	Maximale Anzahl der Synchronisationszyklen, in denen keine Sychronisationsnachricht erhalten wurde. Nach Ablauf dieser Zeit erklärt sich der betroffene Sensorknoten selbst zur Wurzel.
`SCALEFACTOR1` ftsp_Layer.h	Skalierungsfaktor zur Reduzierung der Gefahr von Bereichsüberläufen in der Methode `linearRegression` bei der Integer-Umsetzung.
`SCALEFACTOR2` ftsp_mrfi_interface.h	Skalierungsfaktor zur Reduzierung der Gefahr von Bereichsüberlaufen in den Methoden `localToGlobal` und `globalToLocal` bei der Integer-Umsetzung.

5 Beispielprogramm zur Nutzung von FTSP als Protokollschicht

In diesem Kapitel wird die Verwendung der FTSP-Netzwerkschicht anhand eines kurzen Beispielprogrammes vorgestellt. Diese Anwendung befindet sich im im Modul `ftsp_ProtocolTest`. Sie ermöglicht den Austausch von Nachrichten, wobei sich die Teilnehmer im Hintergrund synchronisieren. Zum Senden und Empfangen von Nachrichten werden hierbei Contiki-Protothreads eingesetzt. Zunächst wird hier kurz die Arbeitsweise der Protothreads erklärt.

Ein Protothread wird in der einfachsten Form nach Programmauszug 5.1 angelegt.

```
1 PROCESS(myThread,"myThread");
2 PROCINIT(&myThread);
3
4 PROCESS_THREAD(myThread, ev, data)
5 PROCESS_BEGIN()
6
7   //Initialisierungsanweisungen. Befehle, die hier stehen, werden nur beim ersten
        Aufruf des Protothreads ausgeführt.
8
9   while (TRUE) {
10    PROCESS_WAIT_EVENT_UNTIL(ev == TRIGGER_EVENT);     // Warten auf Ereignis
11    //weitere Anweisungen, die immer beim Eintreffen eines Ereignisses ausgeführt
          werden.
12  }
13
14 PROCESS_END()
```

Programmauszug 5.1: Protothread

In Zeile 1 wird der Protothread `myThread` deklariert und in Zeile 2 beim Betriebssystem registriert. Die tatsächliche Implementierung des Protothreads erfolgt in den Zeilen von 4 bis 14. Anweisungen, die vor der `while`-Schleife stehen, werden nur während der Initialisierung des Protothreads ausgeführt. Anweisungen innerhalb dieser Schleife werden hingegen zyklisch ausgeführt, bis der Protothread blockiert oder beendet wird. Die `PROCESS_WAIT_EVENT_UNTIL`-Anweisung in Zeile 10 stellt eine solche blockierende Anweisung dar. Es wird solange gewartet, bis ein Ereignis `TRIGGER_EVENT` an diesen Protothread gesendet wird. Bei einer solchen Warteanweisung kehrt die Protothread-Funktion zum Aufrufer, d.h. zur Thread-Verwaltung des Betriebssystems, zurück. Dabei ist zu beachten, dass Protothreads keinen eigenen Stack haben und deshalb lokale Variablen statisch oder global definiert werden müssen um ihren Wert über einen Kontextwechsel hinweg zu erhalten.

Wie bereits erwähnt, nutzt diese Anwendung zwei Protothreads zum Senden und Empfangen von Nachrichten. Im Modul `ftsp_Layer` wird ein weiterer Protothread zur Verwaltung der Dauer eines Synchronisationszyklusses benutzt. Da die Registrierung der Protothreads an genau einer Stelle erfolgen muss, sorgt die Anwendung dafür, dass der Protothread der FTSP-Schicht ebenfalls mit registriert wird (siehe Programmauszug 5.2). Dazu ist der Protothread `ftspSyncSender` in der Header-Datei ftsp_Layer.h als `extern` definiert und nach außen für die Applikation sichtbar.

Die Anwendung verschickt die eigene ID in periodischen Zeitabständen (siehe Programmauszug 5.3), welche durch den Timer 3 und der zugehörigen ISR (Programmauszug 5.4) per Event-Generierung vorgegeben werden.

```
1 PROCESS(Receiver,"Receiver");
2 PROCESS(Sender,"Sender");
3 PROCINIT(&Receiver, &ftspSyncSender, &Sender);
```

Programmauszug 5.2: Initialisierung der Protothreads

```
1 PROCESS_THREAD(Sender, ev, data)
2 PROCESS_BEGIN()
3   while (TRUE) {
4     PROCESS_WAIT_EVENT_UNTIL(ev == TRIGGER_EVENT);
5     ftspSend("ID 0\\n",5);
6   }
7 PROCESS_END()
```

Programmauszug 5.3: Sender

```
1 void T3ISR(){
2 static uint16 counter=0;
3 #ifdef _LOWER_BYTE_
4   if(++counter >= 4000){       //etwa 4 Sekunden
5 #else
6   if(++counter>=25000){        //etwa 25 Sekunden
7 #endif
8     counter=0;
9     process_post(&Sender, TRIGGER_EVENT, NULL);
10  }
11 }
```

Programmauszug 5.4: T3ISR()

Der Empfänger-Prototthread `Receiver` enthält neben den Befehlen zur Ausgabe der empfangenen Nachricht auch weitere Initialisierungsanweisungen.

```
1 PROCESS_THREAD(Receiver, ev, data)
2 PROCESS_BEGIN()
3   halTimer3ISRConnect(&T3ISR);
4   halTimer3Init();
5   halTimer3Start();
6
7   ftspRxConnect(&RxISR);
8   ftspInit(MYADDRESS+MYID,0x2007,MYID);
9
10  while (TRUE) {
11    PROCESS_WAIT_EVENT_UNTIL(ev == TRIGGER_EVENT);
12    size= ftspReceive(RxData);
13    if(size>0) printf("%s",RxData);
14    printf("\n");
15  }
16 PROCESS_END()
```

Programmauszug 5.5: Receiver Protothread

In Zeile 3 wird die Methode `T3ISR` im Modul `Timer3` als Funktionszeiger abgelegt. In den Zeilen 4-5 wird der Timer 3 initialisiert und gestartet. In Zeile 7 wird die Methode `RxISR` (siehe Programmauszug 5.6) im Modul `mrfi_LinkLayer` registriert. Diese wird nach einer vollständig empfangenen Nachricht aufgerufen und verschickt ein Ereignis an den `Receiver`-Thread. In Zeile 8 wird die FTSP-Schicht initialisiert. Durch den Aufruf von `ftspReceive(RxData)` wird die eingegangene Nachricht in der FTSP-Schicht

verarbeitet und die Informationen zur Synchronisation extrahiert. Falls Nutzdaten vorlagen, werden diese in den Puffer **RxDATA** kopiert und die Anzahl der kopierten Bytes zurückgegeben. In diesem Fall werden diese in Zeile 13 ausgegeben.

```
1 static void RxISR(void){
2     process_post(&Receiver, TRIGGER_EVENT, NULL);
3 }
```

Programmauszug 5.6: RxISR

6 Evaluation der FTSP-Implementierung

In diesem Kapitel wird die vorliegende Implementierung des FTSPs hinsichtlich der Synchronisationsgenauigkeit und Robustheit getestet. Zunächst erfolgt eine Beschreibung des Testverfahrens. Zum besseren Vergleich mit den Ergebnissen von [23] orientiert sich das Testverfahren an deren Ausführungen.

6.1 Beschreibung des Testverfahrens

Ausgehend von einem gesonderten Netzwerkknoten, der in [23] als Reference-Broadcaster bezeichnet wird, werden in zyklischen Abständen QUERY-Pakete in das Sensornetz gesendet. Ein QUERY-Paket erreicht dabei jeden Teilnehmer auf direktem Wege. Beim Empfangen des SFD-Bytes dieser Nachricht wird ein Zeitstempel in lokaler Zeit generiert. Dieser Zeitstempel wird in globale Zeit umgewandelt und in einem REPLY-Paket an einen weiteren gesonderten Netzwerkknoten, der als Base-Station bezeichnet wird, verschickt. Die Base-Station empfängt also von allen Teilnehmern eine Schätzung der globalen Zeit zu einem bestimmten Zeitpunkt. Über die paarweisen Abweichungen dieser globalen Zeitstempel kann dann eine Aussage bezüglich der Synchronisationsgenauigkeit getroffen werden.

Der Sensorknoten Reference-Broadcaster muss mit der Applikation `ftsp_accuracy_query` programmiert werden. Das Abfrage-Intervall kann durch die Präprozessordirektive `INTERVAL` festgelegt werden (LSB = 2,82123 μs). Standardmäßig ist das Intervall für die 8 μs-Timeline auf etwa 32,62 Sekunden und für die 31,25 ns-Timeline auf etwa 5 Sekunden eingestellt. Da QUERY-Nachrichten an Sensorknoten verschickt werden, die die FTSP-Schnittstelle zur Kommunikation nutzen, entspricht das Format einer QUERY-Nachricht dem normalen FTSP-Format aus Tabelle 4.1. Ein QUERY-Paket enthält als Nutzdaten die Kennung 0x77, die besagt, dass es sich um ein QUERY-Paket handelt und die aktuelle QUERY-Nummer, die die Nummer Abfragerunde angibt. Damit jedoch ein solches Paket nicht die Regressionstabelle beeinflusst, wird es immer mit ungültigen Zeitinformationen (`validTime`=0) verschickt.

Der Sensorknoten, der als Base-Station genutzt werden soll, muss mit der Applikation `ftsp_accuracy_base` programmiert werden. Sie dient lediglich zum Ausgeben der globalen Zeitstempel und ist daher möglichst einfach gehalten und arbeitet ohne den Contiki-Protothreads. Stattdessen wird ein Polling durch das Abfragen der Methode `mrfiLinkDataRdy()` des Moduls `mrfi_LinkLayer` betrieben. Gibt diese `TRUE` zurück, so wird die neu eingegangene Nachricht über die serielle Schnittstelle (38400 baud UART mit acht Daten-Bits, ein Stopp-Bit, kein Parity-Bit) ausgegeben. Der Aufbau einer REPLY-Nachricht kann aus Tabelle 6.1 entnommen werden. Die ersten drei Felder entsprechen jeweils der ID der Wurzel, der ID des Sensorknotens und dem Sequenzzähler `highestSeqNum`. Die aktuelle Anzahl der Regressionstabelleneinträge wird im Feld `numEntries` abgelegt. `SFDGlobal` und `SFDLocal` beinhalten den globalen bzw. lokalen Zeitstempel zum Empfangszeitpunkt der zugehörigen QUERY-Nachricht. Das

Feld `QUERYNum` enthält die aktuelle Nummer des QUERY-Pakets und `TxSFD` den genauen Sendezeitpunkt der REPLY-Nachricht in globaler Systemzeit.

myRootID	myID	highestSeqNum	numEntries	SFDGlobal	SFDLocal	QUERYNum	TxSFD
1 Byte	1 Byte	2 Bytes	1 Byte	4 Bytes	4 Bytes	2 Bytes	4 Bytes

Tabelle 6.1: REPLY-Paket

Ein empfangenes REPLY-Paket wird beginnend mit der QUERY-Nummer folgendermaßen von der Konsole ausgegeben:

$$QUERY...|myRoodID...|myID...|seqNr...|numEntries...|SFDGlobal:...|SFDLocal:...|TxSFD:...$$

Wichtig für die Aussage zur Genauigkeit ist das Feld SFDGlobal. Dieses sollte für alle Sensorknoten einer QUERY-Runde annähernd den selben Wert anzeigen, damit die Sensorknoten gut zueinander synchronisiert sind. Um die Base-Station nicht mit dem Empfang aller Synchronisationspakete zu überlasten, wird sie in einem Subnetz mit der PAN-ID 0x2010 betrieben, während das synchronisierte Netz mit 0x2007 gekennzeichnet ist.

Die restlichen Sensorknoten müssen mit der Applikation `ftsp_accuracytest` programmiert werden. Diese Applikation nutzt die Schnittstellen der FTSP-Schicht zur Kommunikation und zur Synchronisation. Zum Testen der Genauigkeit werden hierbei keine Nutzdaten verschickt, sondern die Synchronisationsnachrichten automatisch durch das Ablaufen des Zeitgebers Timer 1 generiert (vgl. Abschnitt 4.8.2).

Eine in das Sensornetz versendete Nachricht ist mit der Broadcast-Zieladresse 0xFFFF gekennzeichnet und kann daher von allen Teilnehmern mit derselben PAN-ID empfangen werden. Die Sensorknoten weisen im Allgemeinen auch bei niedrigster Sendeleistung eine Reichweite von mehreren Metern auf. Um dennoch eine Kommunikation über mehrere Hops zu erzwingen, wurde die Möglichkeit des Source-Address-Matchings ausgenutzt. Diese Funktionalität wird vom CC253x hardwareseitig unterstützt und wurde im Modul `mrfi_LinkLayer` über die Methode `mrfiLinkSrcMatch_enable(uint8 *table, uint8 size)` realisiert. Dabei können bis zu 24 Quelladressen bestehend aus 2-Bytes PAN-ID und 2-Bytes Kurzadresse registriert werden. Für diese Quelladressen ist im RAM ein bestimmter Bereich ab der Speicheradresse 0x6100 reserviert. Wird eine Nachricht empfangen, die als Quelladresse eines dieser Einträge enthält, so wird der entsprechende Index im Register `SRCRESINDEX` angezeigt. Andernfalls ist `SRCRESINDEX` mit dem Wert 0x3F belegt, was zum Verwerfen der Nachricht in `MRFI_RxCompleteISR()` führt. In `ftsp_accuracytest` wird über dieses Verfahren nur die Kommunikation zwischen zwei direkten Nachbarn bzw. zum Reference-Broadcaster erlaubt. Abhängig von der vergebenen Sender-ID werden in der Methode `initSourceTable()` unterschiedliche Adressen in die Quelladressentabelle geschrieben. Abbildung 6.1 zeigt die maximale Länge einer Kommunikationskette von sieben Hops bei acht Sensorknoten. Sollen mehr Sensorknoten getestet werden, so muss `initSourceTable()` entsprechend um diese erweitert werden. Abbildung 9.3 zeigt für acht Sensorknoten exemplarisch den Testaufbau.

Der Sensorknoten mit der kleinsten ID etabliert sich als Wurzel, in diesem Fall also die ID 0. Damit der Sensorknoten mit der nächst größeren ID den potentiellen Ausfall der Wurzel nicht sofort mitbekommt, wurde ID 1 bewusst vier Hops nach ID 0 eingetragen.

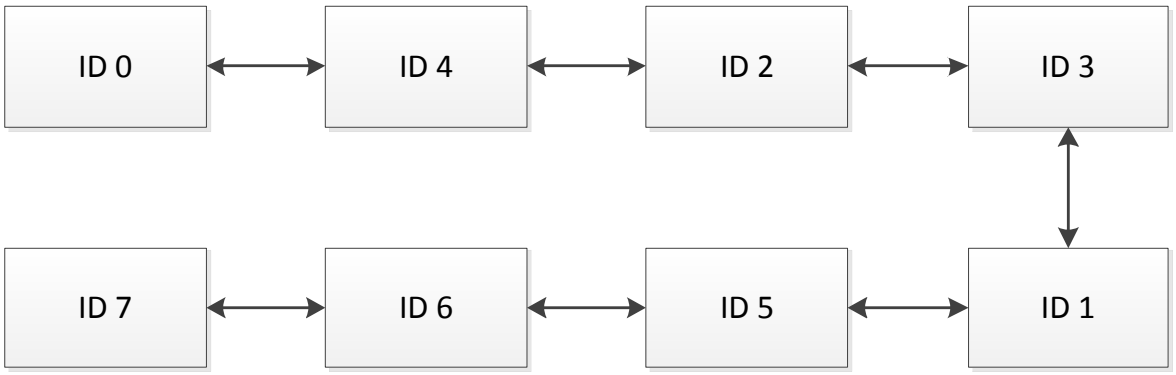

Abbildung 6.1: Multihop-Nachrichtenkette

Zum Senden der REPLY-Pakete wird nicht die FTSP-Schnittstelle genutzt, da diese nur zum Verteilen von Nachrichten innerhalb derselben PAN-ID gedacht ist. Stattdessen wird die Sendemethode der `mrfi_LinkLayer`-Schicht benutzt, die eine gerichtete Kommunikation auch zwischen verschiedenen Subnetzen ermöglicht.

6.2 Erreichte Genauigkeit

Zur Bestimmung der Genauigkeit werden in jeder QUERY-Runde aus allen empfangenen REPLY-Paketen jeweils paarweise der maximale und durchschnittliche Fehler über die globalen Zeitstempel `SFDGlobal` ermittelt und in Abhängigkeit der Hops-Anzahl ausgewertet. Im Single-Hop-Fall wird z.B. gemäß Abbildung 6.1 das Knotenpaar $(ID0, ID4)$ und für zwei Hops die Knotenpaare $\{(ID0, ID4), (ID0, ID2), (ID4, ID2)\}$ berücksichtigt. Wie in [23] wird hier also die zeitliche Abweichung zwischen den Sensorknoten statt einer absoluten Abweichung zu einer globalen Referenzzeit betrachtet.

Für alle Messungen gilt, wenn nicht anders spezifiziert, die Konfiguration nach Tabelle 6.2 (vgl. Abschnitt 4.10.2):

HEARTBEAT_CYCLE	120
MAX_ENTRIES	8
MAX_ERRORS	2
NUMENTRIES_LIMIT	3
OFFSET_CORRECTION	0
ENTRY_THROWOUT_LIMIT	400
ROOT_TIMEOUT	4
INTERVAL	0xB08D3D

Tabelle 6.2: Konfiguration

`OFFSET_CORRECTION` stellt hierbei einen experimentellen Parameter zur Verbesserung der Genauigkeit dar. Unter Berücksichtigung dieses Parameters wird eine Regression über die Wertepaare $(L_i(t_k), O_i(t_k) = G(t_k) + \text{OFFSET\_CORRECTION} - L_i(t_k))_{k=1}^{N}$ durchgeführt. Die Ursache, warum dieser Parameter zur Verbesserung der Synchronisationsgenauigkeit führt, konnte nicht eindeutig ermittelt werden. Da die Synchronisation

beim FTSP nur auf den Austausch einer Nachricht besteht, wird die Ausbreitungszeit bei der Synchronisation nicht berücksichtigt. Es lässt sich vermuten, dass der Korrekturfaktor die Kompensation dieser Zeit beinhaltet und daher zur Verbesserung führt. Die Zeiten, in der eine Nachricht in eine elektromagnetische Welle um- beziehungsweise wieder zurückgewandelt wird, sind im Datenblatt nicht näher spezifiziert und könnten ebenfalls durch diesen Korrekturfaktor ausgeglichen werden.

6.2.1 Timeline mit Auflösung von 8 μs

Um die Genauigkeit der Fließkomma-Implementierung beurteilen zu können, wurde diese zunächst mit einer Timeline-Auflösung von 8 μs getestet. Dazu wurden fünf Sensorknoten mit der ID von 0 bis 4 nach Abbildung 6.1 in Betrieb genommen, sodass eine Kommunikation über vier Hops erfolgte. Die Synchronisationsnachrichten wurden alle 31,454 Sekunden und die QUERY-Pakete alle 32,642 Sekunden versendet.

Das Resultat dieser Messung ist in den Abbildungen 9.4 bis 9.7 zu sehen. Die Sensorknoten wurden nacheinander eingeschaltet und waren etwa 42 Minuten in Betrieb.

Weiterhin ist den Diagrammen zu entnehmen, wie viele Sensorknoten in diesem Zeitraum synchronisiert waren. Die Schwankungen am Anfang der Synchronisationsphase ergeben sich durch die unterschiedlichen Einschaltzeitpunkte und der schrittweisen Synchronisation entlang der Kommunikationskette (vgl. Abbildung 6.1). Hier sei darauf hingewiesen, dass ein Knoten als synchronisiert gilt, wenn dieser mindestens `NUMENTRIES_LIMIT` Einträge in der Regressionstabelle besitzt oder selber eine Wurzel darstellt. Laut dieser Definition kann eine 100%-Synchronisation erreicht werden, obwohl sich die Sensorknoten auf unterschiedliche Wurzeln und damit auf unterschiedliche globale Zeiten beziehen. Abbildung 6.2 zeigt den Mittelwert des durchschnittlichen und maximalen Fehlers für den Zeitraum, in dem alle Knoten synchronisiert waren und sich auf dieselbe eindeutige Wurzel bezogen haben. Man erkennt, dass mit steigender Hop-Distanz sowohl der durchschnittliche als auch der maximale Fehler ansteigt.

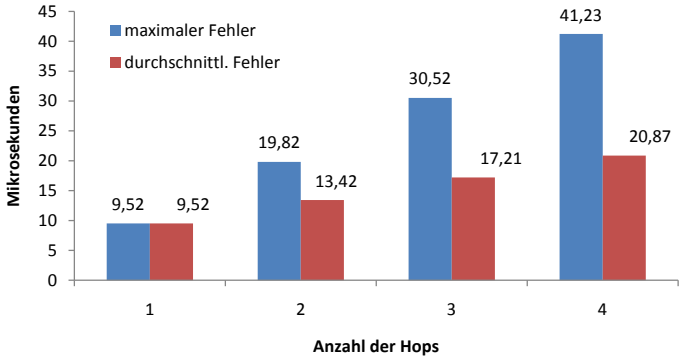

Abbildung 6.2: mittlerer paarweiser Fehler für 8 μs Timeline

Die Messung wurde für acht Sensorknoten, d.h. für sieben Hops wiederholt. Diesmal wurde für `OFFSET_CORRECTION` der Wert 1 benutzt. Für die verwendete Timeline bedeutet dies, dass 8 μs zur erhaltenen globalen Zeit hinzuaddiert wurden. Anhand den Abbildungen 9.8 bis 9.14 ist eine deutliche Verbesserung zu erkennen. In Abbil-

dung 6.3 sind die paarweisen Abweichungen für den Zeitraum eingetragen, in denen alle Sensorknoten synchronisiert waren und sich auf eine eindeutige Wurzel bezogen haben.

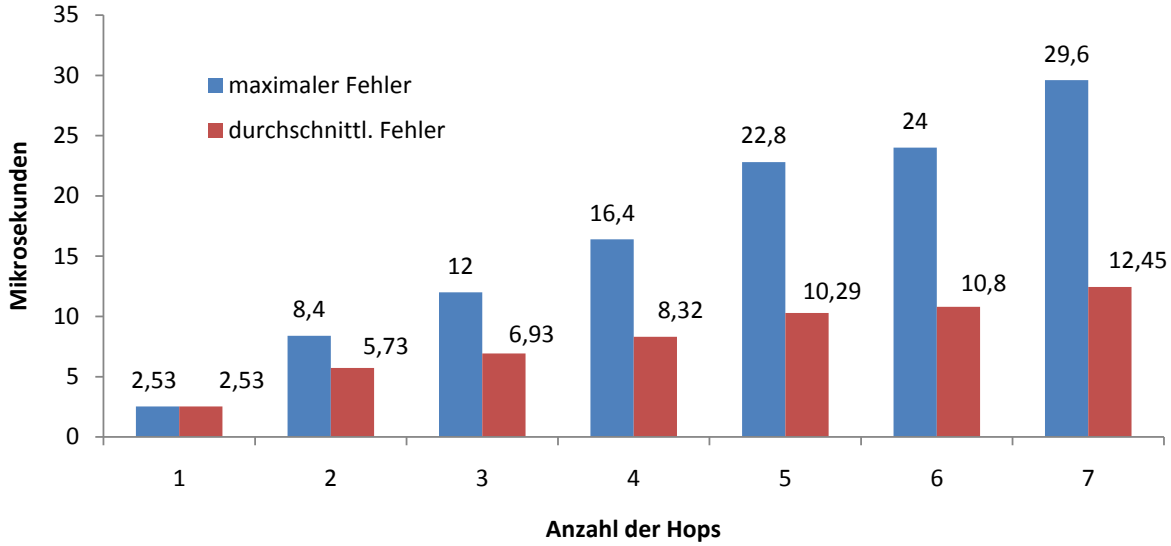

Abbildung 6.3: mittlerer paarweiser Fehler für 8 μs Timeline mit OFFSET_CORRECTION 1

6.2.2 Timeline mit Auflösung von 31,25 ns

Im vorherigen Abschnitt wurde die FTSP-Implementierung mit einer Auflösung von 8 μs pro Takt getestet. Dadurch wurde die mögliche Genauigkeit beschränkt. Eine höhere Auflösung der Timeline lässt eine höhere Genauigkeit vermuten. In dieser Messung wurde daher mit einer Auflösung von 31,25 ns pro Takt gearbeitet. Dadurch beschränkt sich der repräsentierbare Zeitraum auf etwa 2,34 Minuten. Um in diesen Zeitraum genügend Informationen sammeln zu können, wurden etwa alle 4 s Synchronisationsnachrichten ausgetauscht und alle 5 s QUERY-Pakete zur Genauigkeitsmessung verschickt. `HEARTBEAT_CYCLE` wurde dazu auf den Wert 15, `OFFSET_CORRECTION` auf 110 und `INTERVAL` auf 0xB08D3D gesetzt.

Die Abbildungen 9.15 bis 9.18 zeigen, dass hierbei eine Genauigkeit von unter einer Mikrosekunde möglich ist. Zur Messung wurden dabei fünf Sensorknoten eingesetzt, die nach Abbildung 6.1 in Verbindung standen, sodass eine Kommunikation über vier Hops erfolgte. Der Timeline-Zähler der einzelnen Sensorknoten hat zwischen den QUERY-Paketen 25, 26 und 52, 53 seinen Endwert erreicht und ist somit von vorne gestartet. Abbildung 6.4 zeigt die Mittelwerte der maximalen und durchschnittlichen Abweichungen in Abhängigkeit der Hop-Distanz für den Zeitraum, in dem alle Sensorknoten synchronisiert waren.

Die Autoren von [23] geben einen mittleren Fehler von 0,5 μs pro Hop an. Im Vergleich dazu erreicht die 8 μs-Timeline etwa 1,78 μs Abweichung pro Hop und die 31,25 ns Timeline etwa 20 ns Abweichung pro Hop. Demnach kann erst mit voller Timeline-Auflösung eine Verbesserung der Synchronisationsgenauigkeit gegenüber [23] erzielt werden.

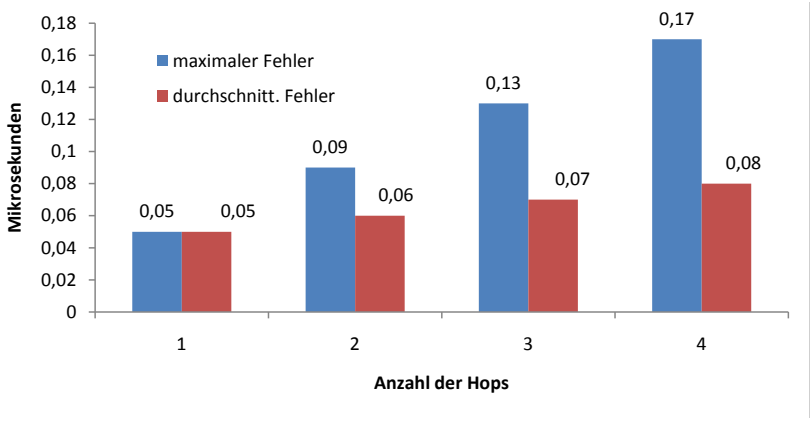

Abbildung 6.4: mittlerer Fehler für 31,25 ns Timeline mit OFFSET_CORRECTION 110

6.3 Vergleich mit Ganzzahl-Umsetzung

In dieser Messreihe wurde die Integer-Implementierung auf Genauigkeit getestet. Die Synchronisationsrate und das Intervall der QUERY-Pakete entsprechen dabei der Messung aus Abschnitt 6.2.1. Die Skalierungsfaktoren wurden dabei experimentell ermittelt (vgl. Abschnitt 7.2) und wie folgt gewählt: `SCALEFACTOR1 = 19` und `SCALEFACTOR2 = 8`. Es wurden acht Sensorknoten in Betrieb genommen, was nach Abbildung 6.1 eine Kommunikation über sieben Hops bedeutet. Die Ergebnisse dieser Messung sind in den Abbildungen 9.19 bis 9.25 dargestellt.

Abbildung 6.5 zeigt den Mittelwert für den paarweisen durchschnittlichen und maximalen Fehler in Abhängigkeit der Hop-Distanz für die Phase, in der alle Sensorknoten synchron waren und sich auf dieselbe Wurzel bezogen haben.

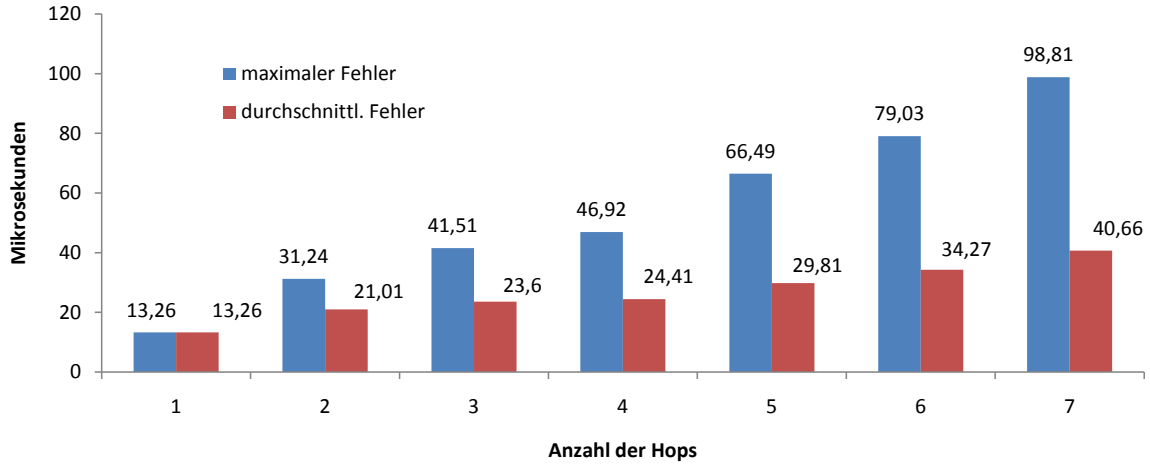

Abbildung 6.5: mittlerer Fehler für 8 μs Timeline - Integer-Implementierung

Vergleicht man Abbildung 6.5 mit 6.2, so erkennt man, dass durch den Einsatz von Fließkommazahlen eine höhere Genauigkeit erreicht werden kann. Durch die Fließkomma-Implementierung wurde für den Mittelwert des durchschnittlichen bzw. des maximalen Fehlers für vier Hops eine um 14 % bzw. eine um 12 % bessere Ge-

nauigkeit erreicht. Die höhere Ungenauigkeit der Integer-Implementierung lässt sich durch die zweifache Skalierung zur Vermeidung von Überläufen erklären. Wie sich in Abschnitt 7.2 zeigen wird, birgt die reine Integer-Implementierung weitere Nachteile.

6.4 Robustheit

Um die Implementierung auf Robustheit zu testen, wurde eine Messreihe gestartet, in der alle 31,454 s Synchronisationsnachrichten generiert wurden. Hierbei wurde die Fließkomma-Implementierung mit der 8 μs-Timeline eingesetzt. Der Abstand der QUERY-Pakete wurde zu 32,642 s gewählt. Des Weiteren wurde `OFFSET_CORRECTION` auf 1 gesetzt.

Zur Messung wurden fünf Sensorknoten (ID_0 bis ID_4) nach Abbildung 6.1 benutzt. Abbildung 6.6 zeigt hierbei den paarweisen maximalen und durchschnittlichen Fehler für vier Hops. In Abbildung 6.7 sind für die einzelnen Sensorknoten die Werte der Variable `myRootID` über die QUERY-Runde q aufgetragen.

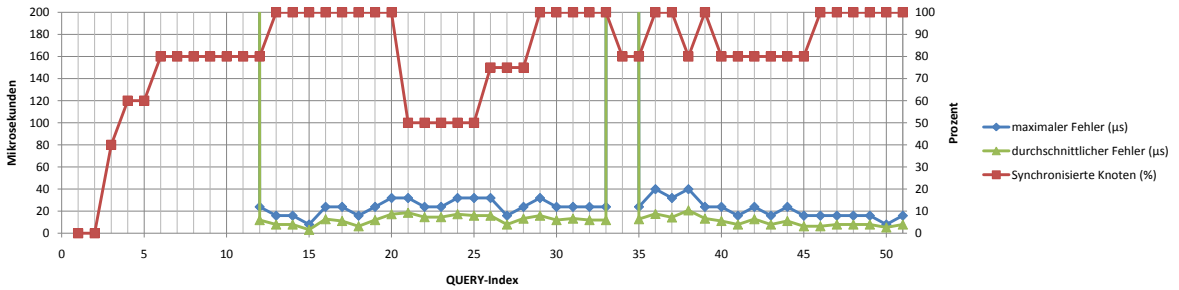

Abbildung 6.6: Paarweiser Fehler für 4 Hops

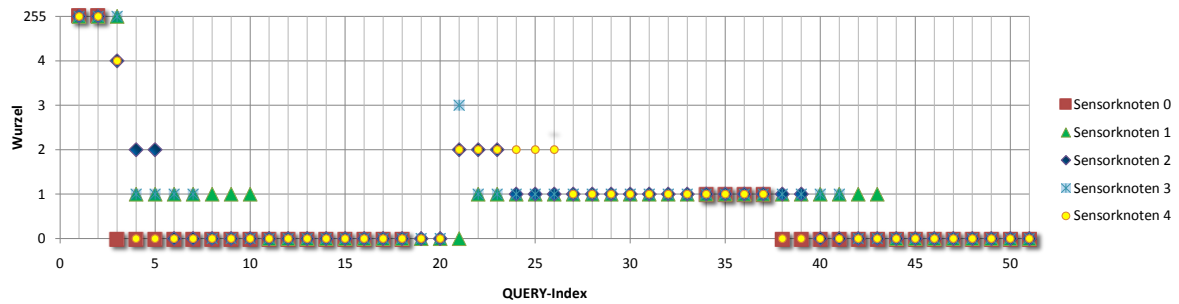

Abbildung 6.7: Wurzelauswahl

Die Sensorknoten wurden nacheinander eingeschaltet. Die QUERY-Runde $q = 0$ stellt dabei den Einschaltzeitpunkt des Reference-Broadcasters dar. Bei der Initialisierung wird die Variable `myRootID` aller Sensorknoten auf den Wert 255 gesetzt. Bis $q = 2$ ist kein Paketaustausch zu vernehmen, da kein Sensorknoten als synchronisiert gilt. In der 3. QUERY-Runde hat der `heartBeats`-Zähler von ID_0 und ID_4 bereits den Wert `ROOT_TIMEOUT` überschritten, sodass sich diese zur Wurzel erklärt haben und Synchronisationsnachrichten ins Netzwerk fluten können. Dadurch, dass ID_2 erst nach ID_4 eingeschaltet wurde und den Initialwert 255 als Wurzel verwaltet, empfängt dieser, noch bevor er sich zur Wurzel erklären konnte, ein Synchronisationspaket von ID_4, an dem

er sich synchronisiert. Da ID_1 und ID_3 von keiner Synchronisationsnachricht erreicht wurden und aufgrund der späteren Einschaltzeitpunkte den Wert `ROOT_TIMEOUT` nicht überschritten haben, verwalten sie zu diesem Zeitpunkt weiterhin den Initialwert 255 als Wurzel. In der 4. QUERY-Runde gibt es drei aktive Wurzeln. Bis $q = 6$ konnte ID_4 genügend Synchronisationspakete von der Wurzel 0 sammeln, um selbst Pakete an ID_2 versenden zu dürfen. ID_2 gibt daraufhin seinen Status als Wurzel auf. ID_3 synchronisiert sich, bis ID_2 zum Senden von Synchronisationspaketen berechtigt ist, weiterhin an den Nachrichten der Wurzel 1. Nach $q = 10$ gibt auch die Wurzel 1 ihren Status auf, da sie von einem Synchronisationspaket von ID_3 erreicht wurde. ID_0 hat sich also als eindeutige Wurzel etabliert.

Nach der 18. QUERY-Runde wurde ID_0 ausgeschaltet, um den Ausfall der Wurzel zu simulieren. Da nun keine Synchronisationsnachrichten mehr emittiert werden, wird der `heartBeats`-Zähler der einzelnen Sensorknoten bis zum Erreichen von `ROOT_TIMEOUT` inkrementiert. Zwischen $q = 20$ und $q = 21$ hat der `heartBeats`-Zähler der Sensorknoten 2, 3 und 4 den Wert `ROOT_TIMEOUT` überschritten, sodass sie sich zur Wurzel erklären und Synchronisationspakete verschicken dürfen. Dieser Zustand ist in Abbildung 6.7 jedoch nicht sichtbar, da ID_4 noch vor der nächsten QUERY-Runde von einem Synchronisationspaket von ID_2 erreicht wurde. Dieser Vorgang muss noch in dem Zeitraum erfolgt sein, während ID_3 den Sensorknoten 0 als Wurzel verwaltet und somit das Synchronisationspaket von ID_2 verworfen hat. Der `heartBeats`-Zähler von ID_3 erreicht erst danach den Wert `ROOT_TIMEOUT`. ID_1 verwaltet währenddessen weiterhin den Sensorknoten 0 als Wurzel, da er als letztes Glied der Kommunikationskette ein Sychronisationspaket mit der Wurzel 0 erhalten und `heartBeats` den Wert `ROOT_TIMEOUT` noch nicht erreicht hat. Bei $q = 22$ gibt es zwei Wurzeln, Sensorknoten 1 und 2. Knoten 1 setzt sich nach der 26. QUERY-Runde als eindeutige Wurzel durch.

Nach $q = 33$ wurde ID_0 wieder eingeschaltet. Dieser ist zunächst nicht synchronisiert, wodurch der maximale und der durchschnittliche Fehler stark ansteigen. Da sich ein neu eingeführter Knoten erst nach Ablauf des `heartBeats`-Zählers zur Wurzel erklärt, bleibt diesem Knoten noch genug Zeit um Synchronisiationpakete entgegen zu nehmen, sodass keine neue globale Zeit eingeführt wird. In der Zeit zwischen der 34. und 38. QUERY-Runde übernimmt ID_0 die Wurzel 1 und kann somit Werte in der Regressiontabelle speichern. Da jedoch die eigene ID kleiner als die der Wurzel ist, wird der `heartBeats`-Zähler nicht zurückgesetzt. Dieses führt dazu, dass sich ID_0 nach der 37. Runde zur Wurzel erklärt. ID_1 bekommt die neue Wurzel als letzter mit und behält bis $q = 43$ seinen Status als Wurzel bei. In Abbildung 6.6 erkennt man, dass während des Auswahlprozesses sich sowohl der durchschnittliche als auch maximale Fehler relativ stabil verhalten.

6.5 Synchronisationsintervall

Um die Abhängigkeit der Synchronisationsgenauigkeit vom Synchronisationsintervall zu untersuchen, wurde der absolute Fehler im Single-Hop-Fall bestimmt. Dazu wurden zwei Netzwerkknoten mit der 8 μs-Timeline eingesetzt, die Synchronisationsnachrichten in unterschiedlichen Zeitabständen verschickt und mit der Fließkomma-Implementierung verarbeitet. Weiterhin wurde `ROOT_TIMEOUT` zu 2 und `OFFSET_CORRECTION` zu 1 ge-

wählt. Die Konfigurationsparameter `HEARTBEAT_CYCLE` und `INTERVAL` sind der Tabelle 6.3 zu entnehmen. Abbildung 6.8 zeigt die Mittelwerte des absoluten Fehlers in Abhängigkeit des Synchronisationsintervalls für den Zeitraum, in dem die Sensorknoten synchronisiert waren. Die Messergebnisse lassen darauf schließen, dass die Wahl der Synchronisationsperiode keinen signifikanten Einfluss auf die Synchronisationsgenauigkeit hat.

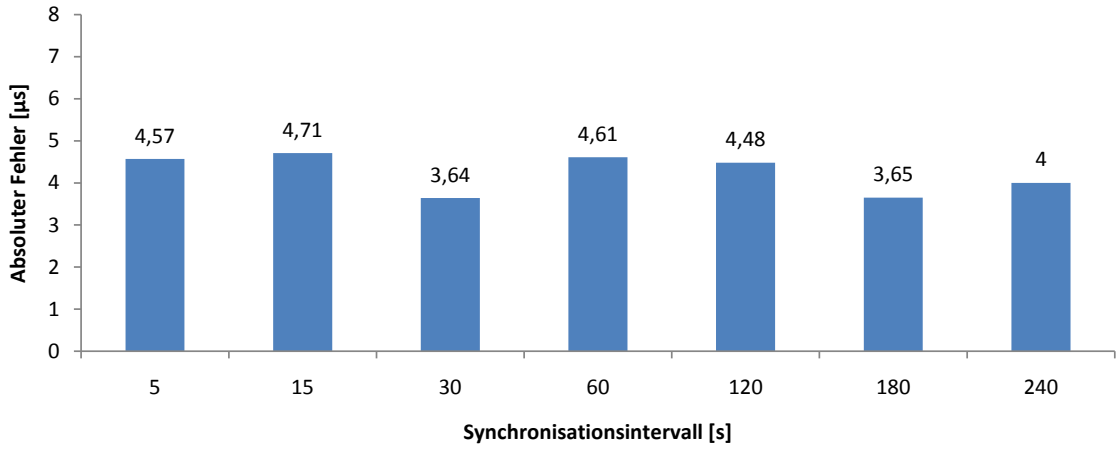

Abbildung 6.8: Absoluter Fehler (Single-Hop)

Synchronisationsintervall	HEARTBEAT_CYCLE	INTERVAL
5 s	19	0x1B0AF5
15 s	57	0x5120DF
30 s	114	0xA241BE
60 s	230	0xA241BE
120 s	456	0x144837C
180 s	685	0x144837C
240 s	913	0x144837C

Tabelle 6.3: HEARTBEATS_CYCLE und INTERVAL

7 Einschränkung

7.1 Ungenauigkeit durch Fließkommazahlen mit einfacher Genauigkeit

Der verwendete Compiler unterstützt in der Version 3.0.0 Fließkommazahlen nur mit einfacher Genauigkeit. Der Darstellungsbereich der Zahlen ist für diese Implementierung zwar ausreichend groß, allerdings sind hierbei für die Mantisse nur 23 Bits reserviert. Die Timeline arbeitet mit 32-Bit-Integer-Werten. Typumwandlungen von `long` auf `float`, wie z.B. in der `for`-Schleife aus Programmauszug 4.6, führen zum Abschneiden und Runden von Stellen. Dadurch entsteht eine Ungenauigkeit, die vor allem bei größeren Werten zunimmt.

7.2 Einschränkung der Ganzzahl-Umsetzung

Die im Abschnitt 4.7 vorgestellte Implementierung als reine Integer-Umsetzung ist nur beschränkt einsatzfähig. Die Einführung der Skalierungsfaktoren verringert zwar die Gefahr von Bereichsüberläufen, verhindert diese jedoch nicht gänzlich. Die im Abschnitt 6.3 erwähnten Werte wurden experimentell in einer Phase bestimmt, in der in gleichmäßigen periodischen Zeitabständen Synchronisationsnachrichten eingegangen sind. Setzen diese für einige Zyklen aus, so können sich größere Werte für die Differenzen `a` und `b` aus Programmauszug 4.8 ergeben, sodass die Größe `SCALEFACTOR1` nicht ausreicht. Problematisch ist vor allem die Multiplikation aus Programmauszug 4.9. Erfragt die überliegende Applikation einen Zeitpunkt, der sehr weit von der aktuellen Zeit abweicht, dann kann die Differenz `(*time - lAverage)` so groß sein, dass die Multiplikation mit der Variable `zaehler` wieder zu einem Bereichsüberlauf führt.

7.3 Testverfahren

Das im Abschnitt 6.1 beschriebene Testverfahren geht idealisiert davon aus, dass ein QUERY-Paket alle Sensorknoten gleichzeitig erreicht. Tatsächlich wird eine solche Nachricht abhängig von dem Abstand unterschiedlich schnell erreichen. Dieser Fehler dürfte jedoch sehr klein ausfallen, da sich elektromagnetische Wellen mit Lichtgeschwindigkeit ausbreiten.

Ein weiteres Problem stellen Hardware-Ausfälle dar. Davon betroffen waren vor allem die CC2531 USB-Dongle. Die genaue Ursache der Hardware-Ausfälle konnte nicht ermittelt werden. Auffällig war jedoch, dass bei hoher Netzwerkauslastung diese Ausfälle häufiger auftraten. Dies lässt vermuten, dass die Probleme im Protokollstack oder der Interrupt-Behandlung zu suchen sind.

8 Zusammenfassung und Ausblick

Ziel dieser Arbeit war die Analyse und Implementierung eines Verfahrens zur Zeitsynchronisation in drahtlosen Sensornetzen. Dazu wurden verschiedene etablierte Synchronisationsprotokolle analysiert, wobei sich das Flooding Time Synchronization Protocol für die potentiellen Anwendungen aus dem Bereich der Strukturüberwachung als am geeignetsten herausgestellt hat. Dieses Protokoll zeichnet sich durch Robustheit, einer hohen Synchronisationsgenauigkeit und einem geringen Verbrauch an Energieressourcen aus. Unter Einsatz des ereignisgesteuerten Betriebssystems Contiki wurde dieses Synchronisationsprotokoll als Netzwerkschicht auf den Plattformen CC2531 und CC2530 von Texas Instruments realisiert.

Um die Güte der Implementierung beurteilen zu können, wurde diese unter verschiedenen Konfiguration getestet. Zur Repräsentation der Timeline wurde ein Hardware-Zeitgeber eingesetzt. Die Timeline kann dabei Auflösung von 8 μs oder 31,25 ns pro Tick annehmen. Eine höhere Auflösung geht jedoch mit einer Verkürzung des darstellbaren Zeitraums einher. Damit eine vergleichbare Aussage bezüglich der Synchronisationsgenauigkeit getroffen werden kann, wurde das in [23] beschriebene Testverfahren übernommen und paarweise die maximale und durchschnittliche Abweichung zwischen je zwei Sensorknoten gebildet. Es wurde gezeigt, dass theoretisch eine Genauigkeit von 20 ns pro Hop möglich ist. Dies stellt im Vergleich zu [23] eine Verbesserung dar, die im Wesentlichen auf die verwendete Hardware zurückzuführen ist. Die Sensorknoten unterstützen den IEEE 802.15.4 Standard und ermöglichen hardwareseitig die Erhebung von präzisen Zeitstempel zum Sende- bzw. Empfangszeitpunkt des SFD-Bytes. Bei diesem Messergebnis ist jedoch zu beachten, dass mit 4 s kürzere Synchronisationsintervalle verwendet wurde als in [23]. Dies war nötig, da die Timeline mit einer Auflösung von 31,25 ns nur einen Zeitraum von etwa 2,34 Minuten darstellen kann. Die Implementierung einer Timeline, die eine hohe Auflösung und einen größeren darstellbaren Zeitraum unterstützt, bleibt als weitere Aufgabe bestehen. Erst dann können auch größere Synchronisationsintervalle getestet werden. In [22] wird dem Austausch von Sychronisationspaketen in einer hohen Frequenz in Synchronisationsprotkollen wie dem FTSP ein negativer Einfluss auf die Synchronisationsgenauigkeit unterstellt. Dieser Sachverhalt wurde für den Multi-Hop-Fall in der vorliegenden Arbeit nicht überprüft und bildet somit einen Ansatz für weitergehende Untersuchungen. Die Messergebnisse aus Abschnitt 6.5 lassen jedoch darauf schließen, dass die Größe des Synchronisationsintervalls keinen signifikanten Einfluss auf die Synchronisationsgenauigkeit hat. Kurze Synchronisationsintervalle haben den Vorteil, dass die Netzwerkknoten durch die Füllung der Regressionstabelle sich schneller im synchronisierten Zustand befinden. Größere Synchronisationszyklen gehen hingegen mit einem niedrigerem Energieverbrauch ein. Ein Kompromiss zwischen schneller Synchronisationsgeschwindigkeit und geringem Energieverbrauch wäre die Etablierung von dynamischen Intervallen, d.h. zu Beginn der Synchronisation könnten kürzere und, nachdem alle Sensorknoten synchronisiert sind, größere Intervalle gewählt werden.

Die erreichte Synchronisationsgenauigkeit beruht auch auf der Fließkomma-Implementierung der linearen Regression. Die Einbindung der Fließkomma-Biblio-theken führt jedoch zur Belegung von mehr Speicher, welcher eine limitierende Ressource in drahtlosen Sensornetzen darstellt. Daher wurde ebenfalls eine Ganzzahl-Implementierung erprobt, die ohne die Einbindung dieser Bibliotheken auskommt. Mit dieser Implementierung wurde jedoch im Vergleich zur Fließkomma-Implementierung eine niedrigere Genauigkeit erreicht. Aufgrund dieser und der im Abschnitt 7.2 erwähnten Einschränkungen stellt die Ganzzahl-Implementierung keine zufriedenstellende Lösung dar. Eine alternative Möglichkeit wäre der Einsatz von Fixpunkt-Arithmetik oder die Auslagerung der Berechnung der linearen Regression auf eine rekonfigurierbare Recheneinheit. Letzteres setzt die Verwendung einer heterogenen WSN-Architektur vorraus, wie sie beispielsweise in [10] vorgeschlagen wurde. Weiterhin kann eine höhere Genauigkeit erwartet werden, wenn statt auf Fließkommazahlen mit einfacher Genauigkeit auf doppelte Genauigkeit gesetzt wird. Ein Compiler, der dies unterstützt, ist z.B. der Crossware 8051 C Compiler.

Auch die Optimierung der hier vorgestellten Implementierung bietet Raum für zukünftige Arbeiten. Als Netzwerkschicht bietet das FTSP die Möglichkeit, Nutzdaten zusammen mit den Synchronisationsinformationen zu verschicken. Derzeit werden im unsychronisierten Zustand Nutzdaten mit ungültigen Synchronisationsinformationen verschickt. Zur Schonung der Energieressourcen könnte dabei auf diese ungültigen Daten verzichtet werden. Ein Sensorknoten befindet sich jedoch nur für einen relativ kleinen Zeitraum im nicht synchronisierten Zustand, sodass diese Maßnahme nur eine geringe Verbesserung des Energieverbrauchs erwarten lässt. Weiterhin kann die Header-Information eines FTSP-Pakets nach Tabelle 4.1 um das Längenfeld gekürzt werden, da ein Paket nach dem IEEE 802.15.4 Standard bereits die Längeninformation beinhaltet.

Aktuell werden mit jedem versendeten Paket automatisch Zeitstempel generiert und an die Nachricht angehängt. Dadurch könnten Pakete, die nicht über die FTSP-Schicht verschickt werden, unbrauchbar werden. Durch die Einführung von unterschiedlichen Pakettypen könnte sichergestellt werden, dass solche Stempel nur an FTSP-Synchronisationspakete angehängt werden.

Momentan müssen die IDs der Sensorknoten beim Programmieren der Sensorknoten festgelegt werden. Um sich dies zu ersparen, könnten die IDs aus der Netzwerkadresse abgeleitet werden.

Sofern die bisherige Timeline weiter verwendet werden soll, könnte die Dauer einer Synchronisationsperiode über den Timer 2, statt wie bisher über den Timer 1, festgelegt werden.

9 Anhang

9.1 Lineare Regression

In diesem Abschnitt werden die Formeln zur Berechnung der Parameter m und b aus den Gleichungen 2.12 und 2.13 hergeleitet.

In Abbildung 9.1 sind beispielhaft Werte für y in Abhängigkeit von der Größe x eingetragen. Man erkennt einen linearen Zusammenhang, wobei die Datenpunkte nicht ideal auf einer Geraden liegen. Es gibt mehrere Möglichkeiten eine Gerade durch diese Datenpunkte zu legen. Eine davon ist in diesem Diagramm eingezeichnet. Dabei weisen die Datenpunkte sowohl negative als auch positive Abweichungen von dieser Geraden auf. Allgemein möchte man den linearen Zusammenhang durch eine Gerade $y(x) = m \cdot x + b$ beschreiben, die möglichst geringe Abweichungen von den erhobenen Messwerten y_i aufweist. Ein Kriterium kann mithilfe der Methode der kleinsten Fehlerquadrate gefunden werden. Durch diese wird versucht, den folgenden Ausdruck zu minimieren:

$$S(m,b) = \sum_{i=1}^{n} (y_i - m \cdot x_i - b)^2 \tag{9.1}$$

Da die Wertepaare (x_i, y_i) bereits bekannt sind, hängt die Funktion $S(m,b)$ nur von den Paramtern m und b ab. Um ein Minimum zu finden, werden die partiellen Ableitungen gebildet und zu null gesetzt.

$$\begin{aligned}
0 &\overset{\text{minimiere 9.1}}{=} \frac{\partial S(m,b)}{\partial b} \\
&= -\sum_{i=1}^{n} 2 \cdot (y_i - m \cdot x_i - b) \\
&= 2 \cdot n \cdot b + 2 \cdot m \sum_{i=1}^{n} (x_i) - 2 \sum_{i=1}^{n} (y_i) \\
\Rightarrow \quad b &= \frac{1}{n} \sum_{i=1}^{n} (y_i) - m \cdot \frac{1}{n} \sum_{i=1}^{n} (x_i) \\
&= \overline{y} - m \cdot \overline{x}
\end{aligned}$$

Der Achsenabschnitt b ist weiterhin von der Steigung m abhängig. Setzt man 2.13 in 9.1 ein, so erhält man eine Gleichung, die nur von einem Parameter m abhängt:

$$S(m) = \sum_{i=1}^{n} ((y_i - \overline{y}) - m \cdot (x_i - \overline{x}))^2 \tag{9.2}$$

Das Nullsetzen der Ableitung dieser Gleichung und das anschließende Aufteilen der Summe ergibt die gewünschte Formel aus 2.12:

$$
\begin{aligned}
0 &\overset{\text{minimiere 9.2}}{=} \frac{\partial S(m)}{\partial m} \\
&= 2 \cdot \sum_{i=1}^{n} (((y_i - \overline{y}) - m \cdot (x_i - \overline{x})) \cdot (-(x_i - \overline{x}))) \\
&= \sum_{i=1}^{n} ((y_i - \overline{y}) \cdot (x_i - \overline{x}) - \sum_{i=1}^{n} (m \cdot (x_i - \overline{x}) \cdot (x_i - \overline{x})) \\
\Rightarrow \quad m &= \frac{\sum_{i=1}^{n} ((y_i - \overline{y}) \cdot (x_i - \overline{x}))}{\sum_{i=1}^{n} ((x_i - \overline{x}) \cdot (x_i - \overline{x}))} \\
&= \frac{\sum_{i=1}^{n} ((y_i - \overline{y}) \cdot (x_i - \overline{x}))}{\sum_{i=1}^{n} (x_i - \overline{x})^2}
\end{aligned}
$$

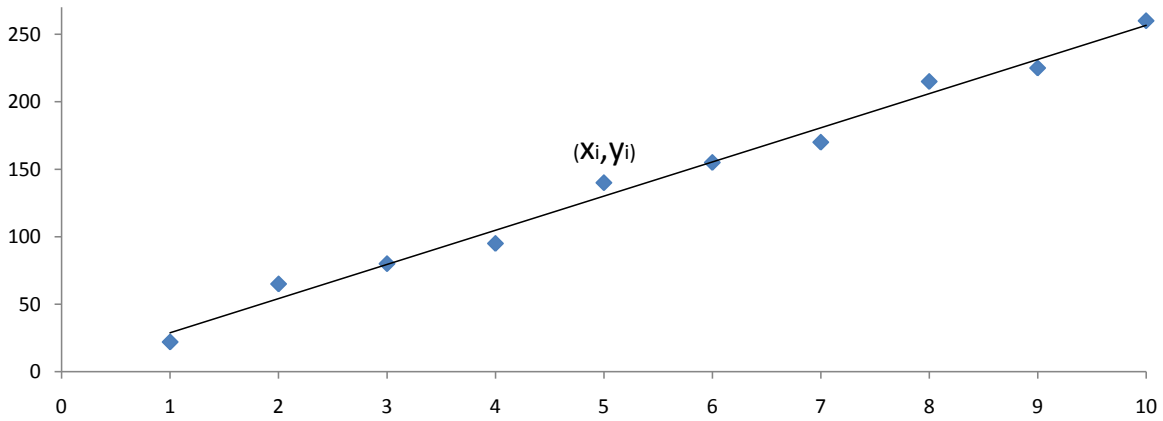

Abbildung 9.1: Linearer Zusammhang der Größen x und y

9.2 Fehlerabschätzung zur Berechnung der lokalen Zeit

Die lokale Zeit kann exakt nach Gleichung 2.20 berechnet werden. Zur Verkürzung der Berechnungszeit wird diese mithilfe der Schätzung $L_i(t) \approx G_i(t) - \overline{O_i}$ und Gleichung 2.19 zu

$$\widetilde{L_i}(t) = G_i(t) - \overline{O_i} - m_i \cdot (G_i(t) - \overline{O_i} - \overline{L_i}) \tag{9.3}$$

vereinfacht.

Der Fehler $\epsilon_i := L_i(t) - \widetilde{L_i}(t)$ zwischen exaktem und vereinfachtem Verfahren zur Bestimmung der lokalen Zeit soll hier ermittelt werden:

$$\begin{aligned}
\epsilon_i &= L_i(t) - \widetilde{L_i}(t) \\
&= \frac{G_i(t) - \overline{O_i} + m_i \cdot \overline{L_i}}{m_i + 1} - (G_i(t) - \overline{O_i} - m_i \cdot (G_i(t) - \overline{O_i} - \overline{L_i})) \\
&= \frac{G_i(t) - \overline{O_i} + m_i \cdot \overline{L_i}}{m_i + 1} - \frac{(m_i + 1) \cdot (G_i(t) - \overline{O_i} - m_i \cdot (G_i(t) - \overline{O_i} - \overline{L_i}))}{m_i + 1} \\
&= \frac{m_i^2}{m_i + 1}(G_i(t) - \overline{O_i} - \overline{L_i}) \\
&= \frac{1}{\frac{1}{m_i} + \frac{1}{m_i^2}}(G_i(t) - \overline{O_i} - \overline{L_i})
\end{aligned}$$

Laut Gleichung 2.7 beschreibt m_i das Verhältnis der Oszillatorfrequenzen $\frac{f_i - f_r}{f_i f_r}$ zweier Knoten i und r. Für die verwendeten SoCs ist eine maximale Frequenzungenauigkeit von $\pm$ 40 ppm angegeben. Dabei kann mithilfe von f_r eine obere und untere Schranke für f_i angegeben werden:

$$f_r \cdot (1 - \frac{40}{10^6}) \leq f_i \leq f_r \cdot (1 + \frac{40}{10^6}) \tag{9.4}$$

Für m_i folgt daher:

$$\begin{aligned}
m_i &= \frac{f_i - f_r}{f_i f_r} \\
&\overset{9.4}{\approx} \frac{f_r \cdot (1 \pm \frac{40}{10^6}) - f_r}{f_r^2 \cdot (1 \pm \frac{40}{10^6})} \\
&= \frac{\pm\frac{40}{10^6}}{f_r \cdot (1 \pm \frac{40}{10^6})} = \pm\frac{40}{f_r \cdot (10^6 \pm 40)}
\end{aligned}$$

Für $f_r \to 32\ MHz$ strebt $m_i \to \pm 0$, daher folgt für ϵ_i:

$$\lim_{m_i \to 0} \epsilon_i = \lim_{m_i \to 0} \frac{1}{\frac{1}{m_i} + \frac{1}{m_i^2}}(G_i(t) - \overline{O_i} - \overline{L_i}) = 0. \tag{9.5}$$

9.3 Überblick der Module

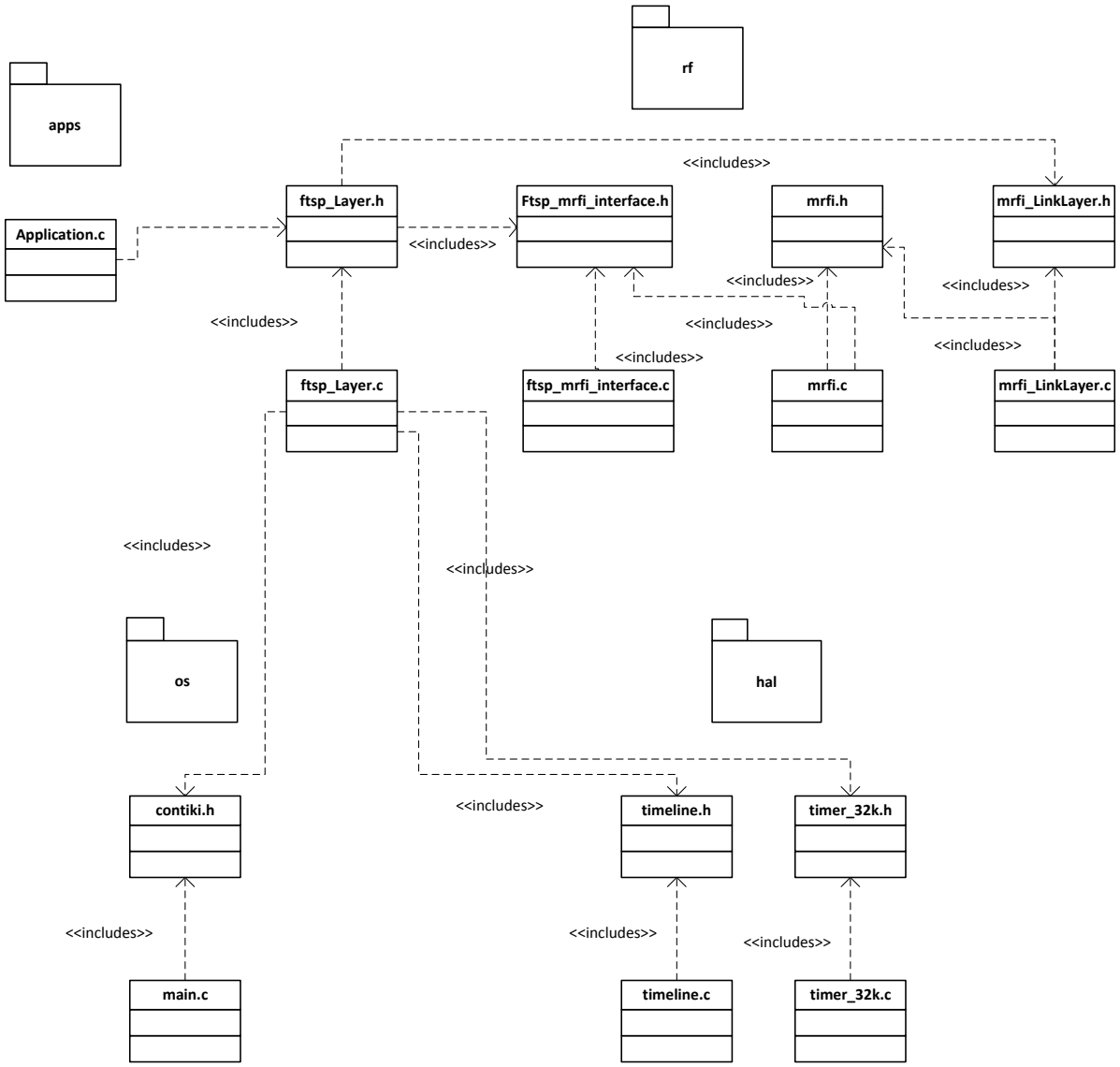

Abbildung 9.2: Module

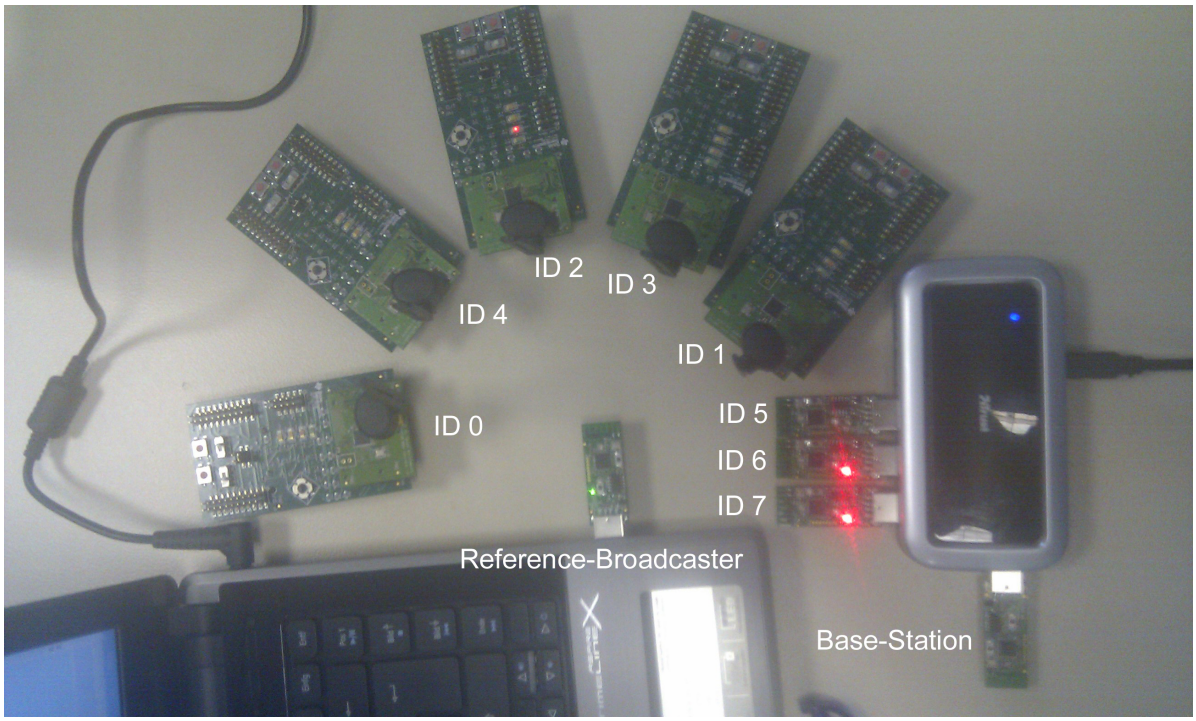

Abbildung 9.3: Testaufbau für eine 7-Hops-Kommunikation

9.5 Messergebnisse

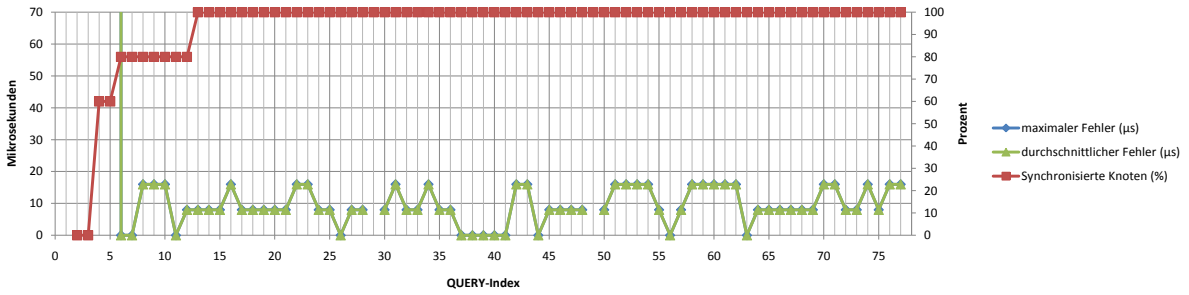

Abbildung 9.4: Messergebnis für 1 Hop. Fließkomma-Implementierung mit 8 μs Timeline.

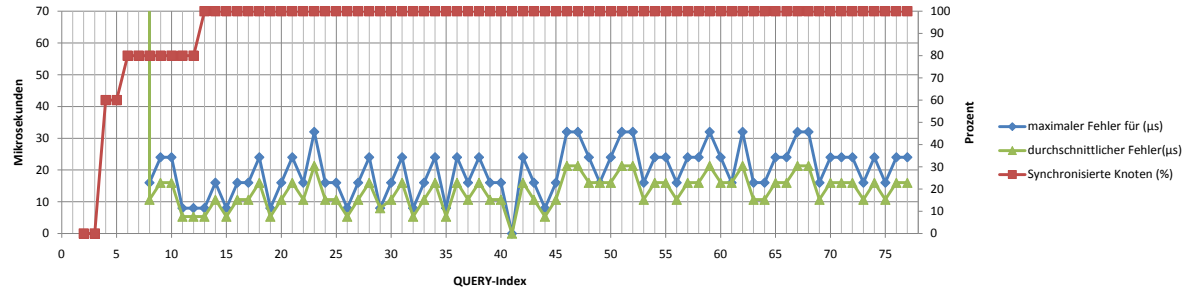

Abbildung 9.5: Messergebnis für 2 Hops. Fließkomma-Implementierung mit 8 μs Timeline.

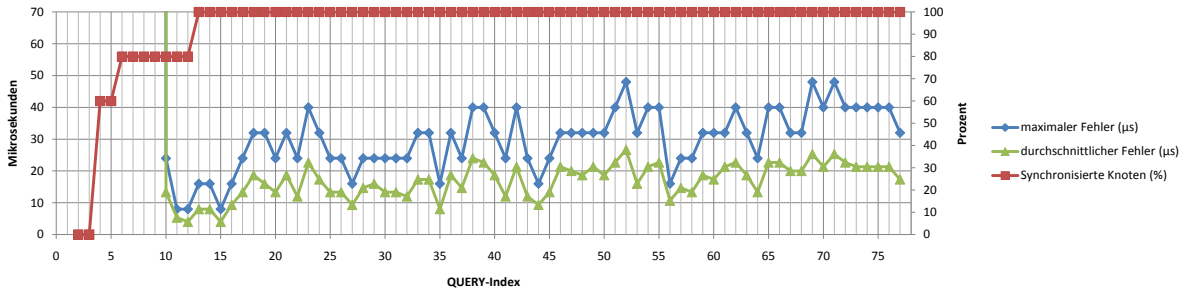

Abbildung 9.6: Messergebnis für 3 Hops. Fließkomma-Implementierung mit 8 μs Timeline.

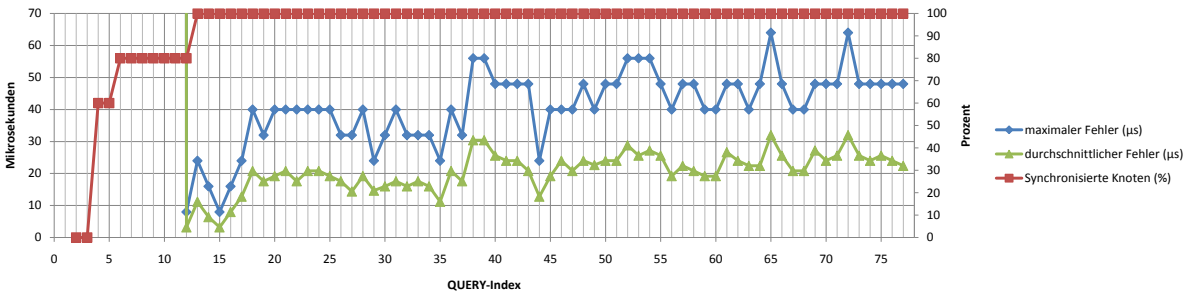

Abbildung 9.7: Messergebnis für 4 Hops. Fließkomma-Implementierung mit 8 μs Timeline.

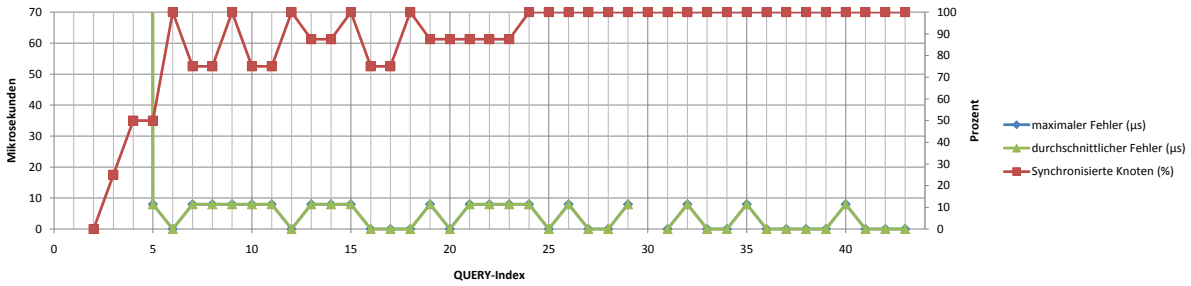

Abbildung 9.8: Messergebnis für 1 Hop. Fließkomma-Implementierung mit 8 μs Timeline. OFFSET_CORRECTION = 1

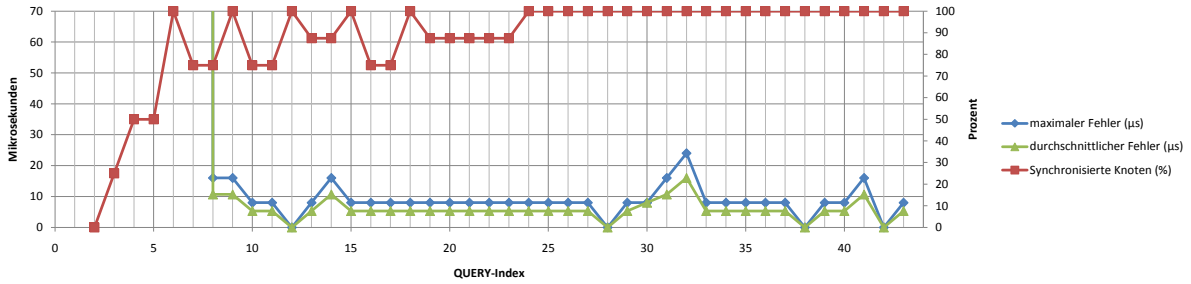

Abbildung 9.9: Messergebnis für 2 Hops. Fließkomma-Implementierung mit 8 μs Timeline. OFFSET_CORRECTION = 1

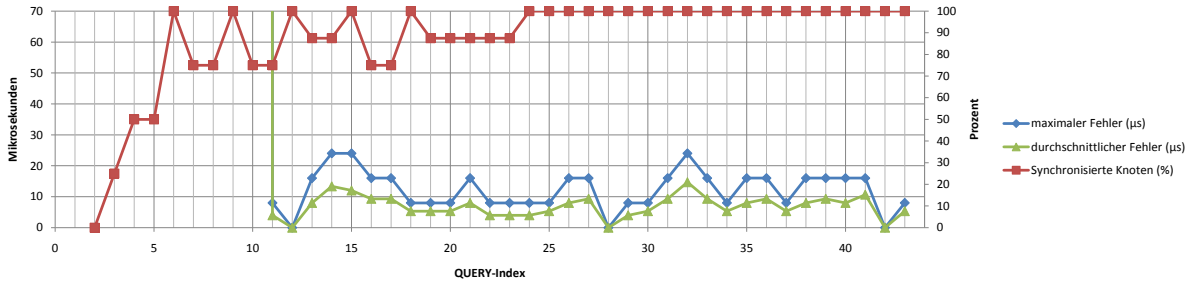

Abbildung 9.10: Messergebnis für 3 Hops. Fließkomma-Implementierung mit 8 μs Timeline. OFFSET_CORRECTION = 1

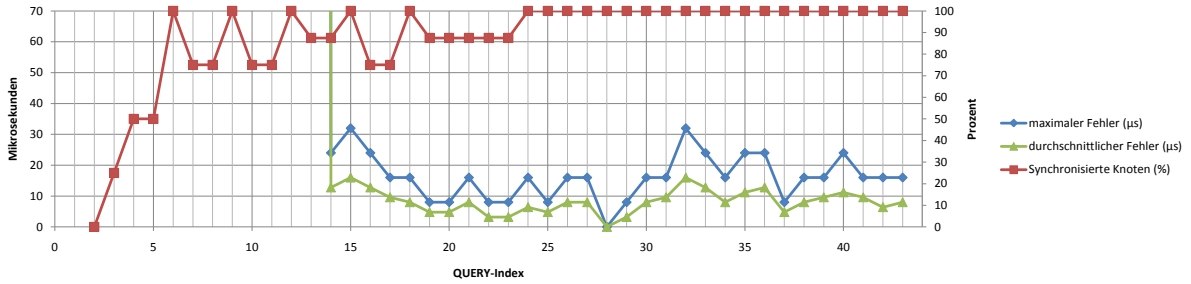

Abbildung 9.11: Messergebnis für 4 Hops. Fließkomma-Implementierung mit 8 μs Timeline. OFFSET_CORRECTION = 1

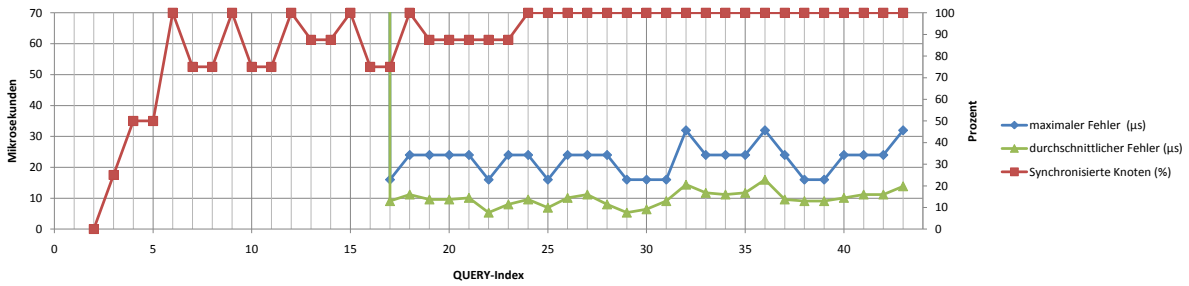

Abbildung 9.12: Messergebnis für 5 Hops. Fließkomma-Implementierung mit 8 μs Timeline. OFFSET_CORRECTION = 1

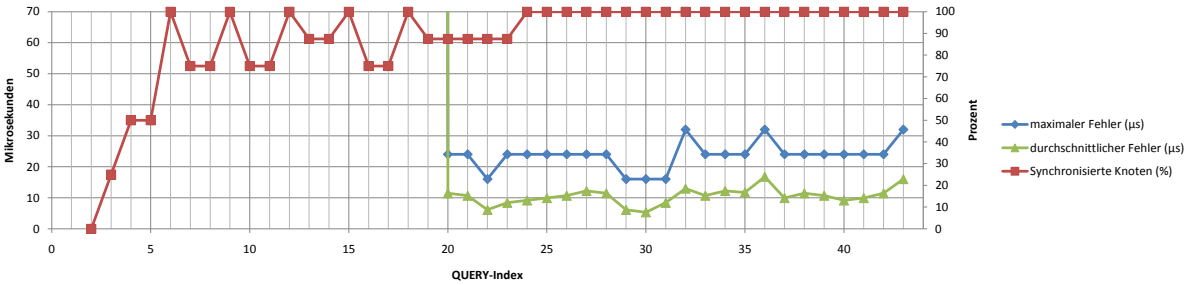

Abbildung 9.13: Messergebnis für 6 Hops. Fließkomma-Implementierung mit 8 μs Timeline. OFFSET_CORRECTION = 1

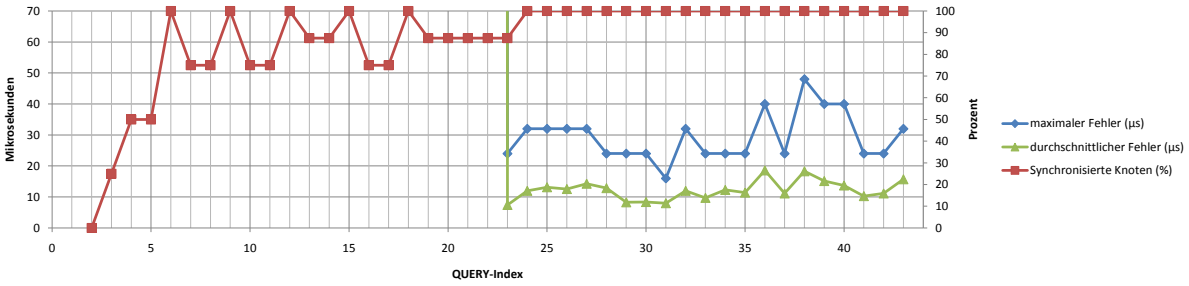

Abbildung 9.14: Messergebnis für 7 Hops. Fließkomma-Implementierung mit 8 μs Timeline. OFFSET_CORRECTION = 1

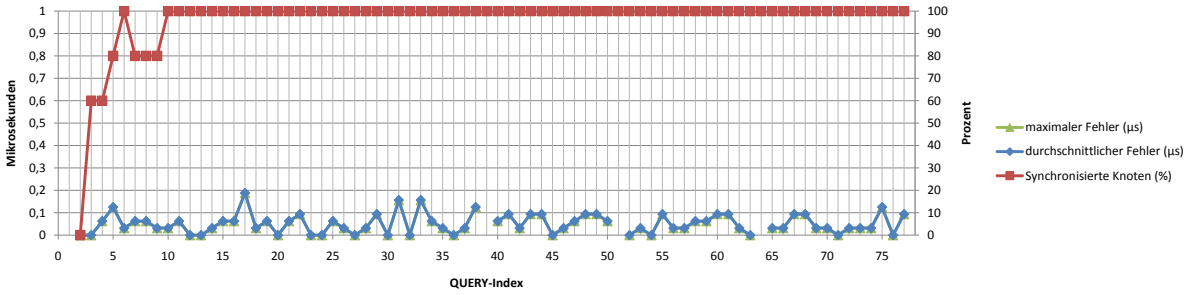

Abbildung 9.15: Messergebnis für 1 Hop. Fließkomma-Implementierung mit 31,25 ns Timeline. OFFSET_CORRECTION = 110

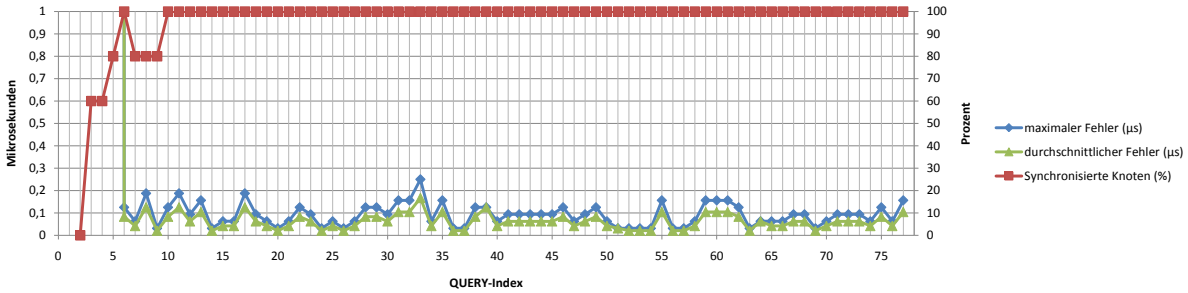

Abbildung 9.16: Messergebnis für 2 Hops. Fließkomma-Implementierung mit 31,25 ns Timeline. OFFSET_CORRECTION = 110

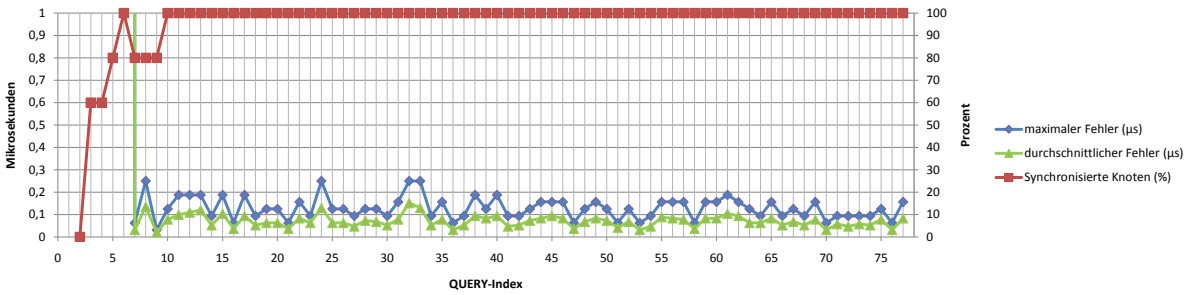

Abbildung 9.17: Messergebnis für 3 Hops. Fließkomma-Implementierung mit 31,25 ns Timeline. OFFSET_CORRECTION = 110

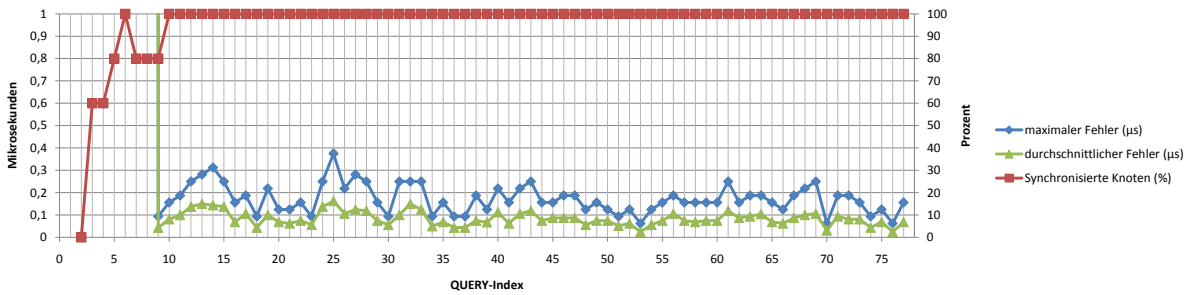

Abbildung 9.18: Messergebnis für 4 Hops. Fließkomma-Implementierung mit 31,25 ns Timeline. OFFSET_CORRECTION = 110

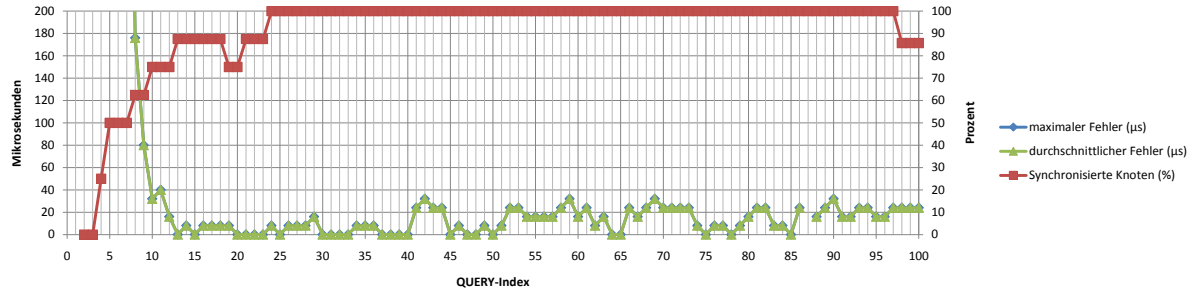

Abbildung 9.19: Messergebnis für 1 Hop. Integer-Implementierung mit 8 μs Timeline.

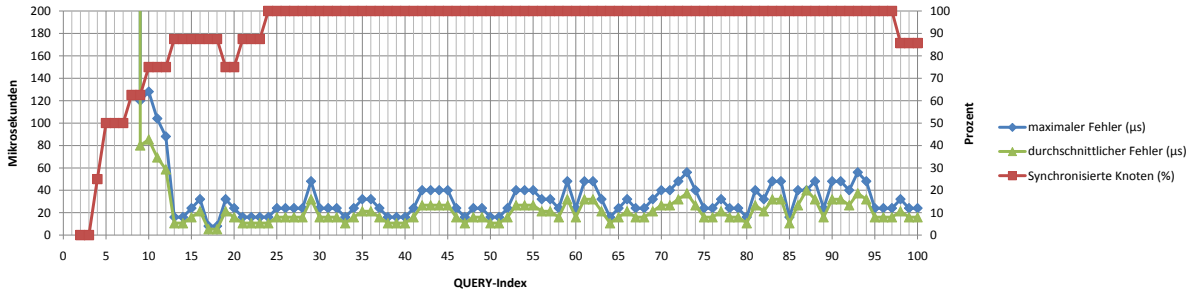

Abbildung 9.20: Messergebnis für 2 Hops. Integer-Implementierung mit 8 μs Timeline.

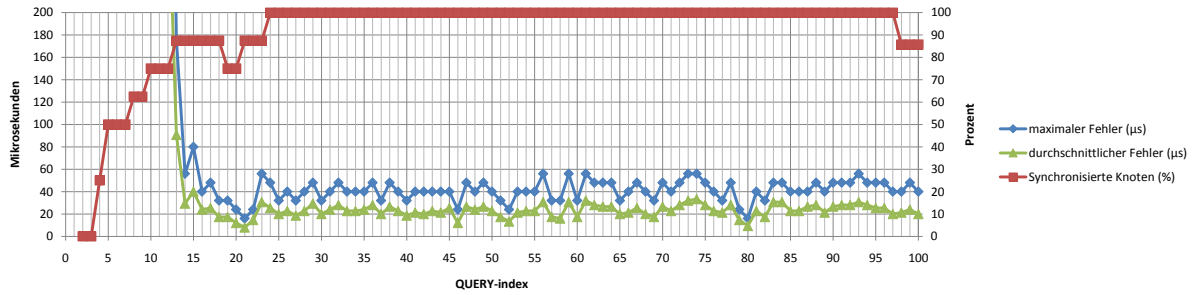

Abbildung 9.21: Messergebnis für 3 Hops. Integer-Implementierung mit 8 μs Timeline.

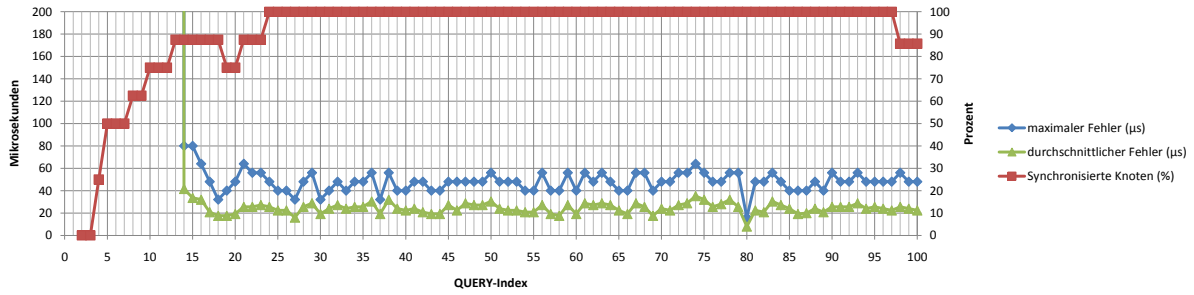

Abbildung 9.22: Messergebnis für 4 Hops. Integer-Implementierung mit 8 μs Timeline.

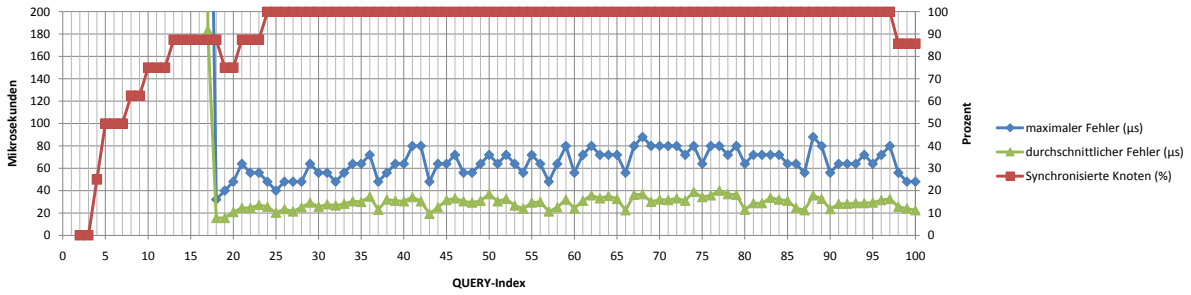

Abbildung 9.23: Messergebnis für 5 Hops. Integer-Implementierung mit 8 μs Timeline.

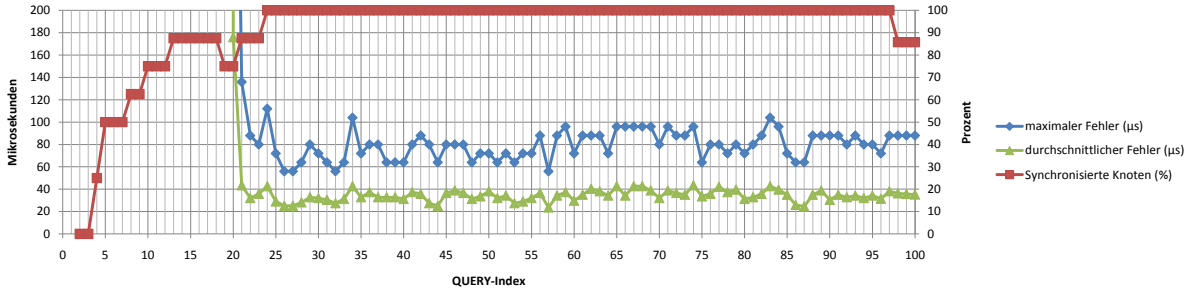

Abbildung 9.24: Messergebnis für 6 Hops. Integer-Implementierung mit 8 μs Timeline.

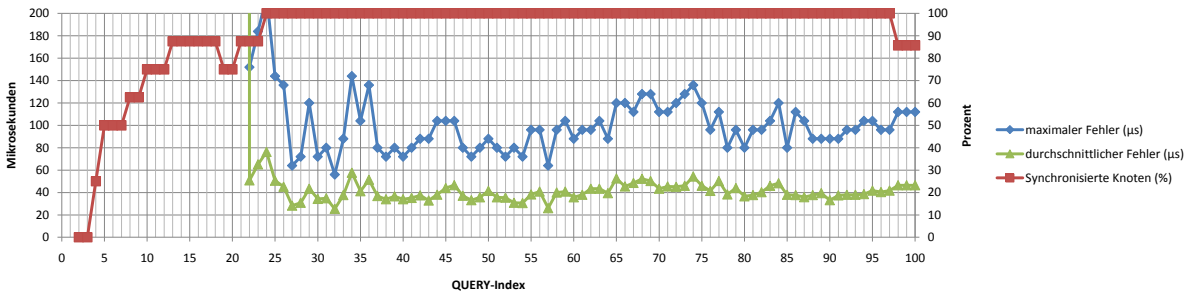

Abbildung 9.25: Messergebnis für 7 Hops. Integer-Implementierung mit 8 μs Timeline.

Aufbau der CD-ROM

Die CD-ROM ist folgendermaßen aufgebaut:

Hauptverzeichnis	
- `Bachelorarbeit.pdf`	Ausarbeitung zur Bachelor-Thesis.
- `Praesentation.ppt`	Präsentation zur Bachelor-Thesis.
halomote	Entwicklungsumgebung. Enthält die Quelltext-Dateien bzw. die Module.
Latex-Files	Latex-Dateien und Abbildungen zu `Bachelorarbeit.pdf`.
Messergebnisse	
- `analyze.rb`	Ruby-Auswertungsskript zur Berechnung der paarweisen und durchnittlichen Fehler pro Hop (bereitgestellt von Andreas Engel).
- `Diagramme.xls`	Ausgewertete Daten zu den verschiedenen Messungen inkl. der Diagramme.
- `Messungen.xls`	Terminal-Ausgabe zu den einzelnen Messungen in Form einer Excel-Tabelle. Enthält weiterhin die gemessenen Ausführungzeiten und ein Excel-Blatt zur Bestimmung von `SCALEFACTOR1` bei der Ganzzahl-Implementierung.
- `PairwiseError.xls`	Mittelwerte der paarweisen und durchschnittlichen Fehler zu den einzelnen Messungen.
- `Speicher-Float.xml`	Darstellung der Speicherbelegung der Fließkomma-Implementierung.
- `Speicher-Integer.xml`	Darstellung der Speicherbelegung der Ganzzahl-Implementierung.

Abkürzungsverzeichnis

BMC	Best Master Clock
CCA	Clear Channel Assessment
CRC	zyklische Redundanzprüfung (engl. Cyclic Redundancy Check)
FCF	Frame Control Field
FCS	Blockprüfzeichenfolge (engl. Frame Check Sequence)
FTSP	Flooding Time Synchronization Protocol
IEEE	Institute of Electrical and Electronics Engineers
IRQ	Unterbrechungsanforderung (engl. Interrupt Request)
ISM-Band	Industrial, Scientific and Medical Band
ISR	Unterbrechungsroutine (engl. Interrupt Service Routine)
MAC	Medienzugriffssteuerung (engl. Media Access Control)
MCU	Mikrocontroller
MRFI	Minimal-Radio-Frequency-Interface
NTP	Network Time Protocol
ppm	parts-per-million
PTP	Precision Time Protocol
RBS	Reference Broadcast Synchronization
RF	Radio Frequency
RTT	Paketumlaufzeit (engl. Round Trip Time)
SDCC	Small-Device-C-Compiler
SFD	start-of-frame-delimiter
SoC	System on Chip
TPSN	Timing-sync Protocol for Sensor Networks
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
WSN	Wireless Sensor Network

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] IEEE 1588-2002. Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. IEEE Std 1588-2002, pages i –144, 2002.

[2] IEEE 802.15.4-2006. Ieee standard for information technology- telecommunications and information exchange between systems- local and metropolitan area networks-specific requirements part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans). IEEE Std 802.15.4-2006 (Revision of IEEE Std 802.15.4-2003), 2006.

[3] Jocelyn Adams, Tanya Roosta, J. Mikael Eklund, Ruzena Bajcsy, and Shankar Sastry. Consideration of security in telehealth wireless sensor network monitoring systems. In The Third IASTED International Conference on Telehealth, pages 57–60, Anaheim, CA, USA, 2007. ACTA Press.

[4] Hyuntae Cho, Jeonsu Jung, Bongrae Cho, Youngwoo Jin, Seung-Woo Lee, and Yunju Baek. Precision time synchronization using ieee 1588 for wireless sensor networks. In Computational Science and Engineering, 2009. CSE '09. International Conference on, volume 2, pages 579 –586, 2009.

[5] Adam Dunkels. Contiki, 2010. `http://www.sics.se/contiki/`.

[6] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, LCN '04, pages 455–462, Washington, DC, USA, 2004. IEEE Computer Society.

[7] J. Eidson and Kang Lee. Ieee 1588 standard for a precision clock synchronization protocol for networked measurement and control systems. In Sensors for Industry Conference, 2002. 2nd ISA/IEEE, pages 98 – 105, 2002.

[8] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. SIGOPS Oper. Syst. Rev., 36:147–163, December 2002.

[9] Jeremy Elson and Kay Römer. Wireless sensor networks: A new regime for time synchronization. In Proceedings of the First Workshop on Hot Topics In Networks (HotNets-I), 2002.

[10] Andreas Engel, Björn Liebig, and Andreas Koch. Feasibility analysis of reconfigurable computing in low-power wireless sensor applications. In Andreas Koch, Ram Krishnamurthy, John McAllister, Roger Woods, and Tarek El-Ghazawi, editors, Reconfigurable Computing: Architectures, Tools and Applications, volume 6578 of Lecture Notes in Computer Science, pages 261–268. Springer Berlin / Heidelberg, 2011.

[11] Sandeep Dutta et al. Small device c compiler, 2010. http://sdcc.sourceforge.net/.

[12] Federico Ferrari, Marco Zimmerling, and Lothar Thiele. Accuracy and duty-cycle of ftsp on a lpl-mac. Technical Report 319, ETH Zürich, Schweiz, 2010.

[13] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In Proceedings of the 1st international conference on Embedded networked sensor systems, SenSys '03, pages 138–149, New York, NY, USA, 2003. ACM.

[14] Tobias Hammer. Hterm, 2010. http://der-hammer.info/index.htm.

[15] Texas Instruments. Simpliciti, 2007. http://www.ti.com/corp/docs/landing/simpliciTI/index.htm?DCMP=hpa_rf_general&HQS=NotApplicable+OT+simpliciti.

[16] Texas Instruments. CC253x/4x User's Guide (Rev. B), 2010.

[17] Texas Instruments. Smartrf flash programmer, 2010. http://focus.ti.com/docs/toolsw/folders/print/flash-programmer.html.

[18] Texas Instruments. A USB-Enabled System-On-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee Applications, 2010.

[19] Texas Instruments. Smartrf protocol packet sniffer, 2011. http://focus.ti.com/docs/toolsw/folders/print/packet-sniffer.html.

[20] Texas Instruments. A True System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee Applications, 2011.

[21] Michael Koch. Entwicklung und Umsetzung von verteilten Systemen zur Vibrationsanalyse und Strukturidentifikation. Diplomarbeit, Fachhochschule Koblenz, 2009.

[22] Liangping Ma, Hua Zhu, G. Nallamothu, Bo Ryu, and Zhensheng Zhang. Impact of linear regression on time synchronization accuracy and energy consumption for wireless sensor networks. In Military Communications Conference, 2008. MILCOM 2008. IEEE, pages 1–7, 2008.

[23] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In Proceedings of the 2nd international conference on Embedded networked sensor systems, SenSys '04, pages 39–49, New York, NY, USA, 2004. ACM.

[24] Miklós Maróti, Brano Kusy, and Janos Sallai. FTSP für TinyOS, 2008. http://www.tinyos.net/tinyos-2.x/tos/lib/ftsp/.

[25] D.L. Mills. Internet time synchronization: the network time protocol. Communications, IEEE Transactions on, 39(10):1482 –1493, October 1991.

[26] D. Wobschall and Yuan Ma. Synchronization of wireless sensor networks using a modified ieee 1588 protocol. In Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2010 International IEEE Symposium on, 27 2010.

[27] Na Xu, Xiaotong Zhang, Qin Wang, Jing Liang, Guangrong Pan, and Meng Zhang. An improved flooding time synchronization protocol for industrial wireless networks. In Embedded Software and Systems, 2009. ICESS '09. International Conference on, pages 524 –529, May 2009.