Zeitsynchronisation in Sensornetzen

Tobias Krauer
Seminar Verteilte Systeme: Sensornetze
tkrauer@student.ethz.ch

1 Einleitung

Durch die zunehmende Miniaturisierung und den technischen Fortschritt ist es möglich geworden, immer kleinere und leistungsfähigere Sensoren zu konstruieren. Auch in der Kommunikationstechnologie setzt sich dieser Trend unaufhaltsam fort, was es ermöglicht, diese Sensoren zu einem drahtlosen Netzwerk zu verbinden, einem Sensornetz.
In solchen Sensornetzen ist Zeitsynchronisation eine wichtige Komponente. Die vorhandenen Protokolle und Algorithmen zur Zeitsynchronisation, die sich in herkömmlichen Netzen bewährt haben, befriedigen die Anforderungen der Sensornetze nicht, weshalb neue Methoden zur Synchronisation der Uhren entworfen werden müssen.

In Abschnitt 2 wird erläutert, wofür die Zeitsynchronisation in Sensornetzen gebraucht wird und weshalb sie eine kritische Komponente darstellt. Im nächsten Abschnitt folgt ein kleiner theoretischer Exkurs zur Zeitsynchronisation allgemein. In Abschnitt 4 wird der Anforderungskatalog zusammengestellt, dem Algorithmen zur Zeitsynchronisation in Sensornetzen genügen müssen. Abschnitt 5 erläutert dann anhand dieser Liste von Anforderungen, wieso die in konventionellen verteilten Systemen verwendeten Protokolle für Sensornetze nicht geeignet sind. In den beiden nächsten Abschnitten 6 und 7 werden zwei Verfahren vorgestellt, die sich zur Uhrensynchronisation in Sensornetzen eignen. Es sind dies die „Packet Stream Synchronization“ [1, 2] und die „Reference Broadcast Synchronization“ [3]. In Abschnitt 8 werden einige Schlussfolgerungen gezogen.

2 Motivation

Das Hauptanwendungsgebiet der Zeitsynchronisation in Sensornetzen ist die „Data Fusion“. Hinter diesem Begriff verbirgt sich das Einsammeln und Zusammenfügen der auf den einzelnen Sensorknoten gewonnenen Daten, um daraus die gewünschten Schlussfolgerungen ziehen zu können. Dieser Prozess ist allerdings meist nur möglich, wenn die einzelnen „Informationsstücke“ in eine exakte chronologische Reihenfolge gebracht werden können und zeitlichen Abstände zwischen zwei Ereignissen präzise gemessen werden können.
Als einfaches Beispiel soll eine Froschbeobachtung dienen: Um die Position eines Frosches anhand seines Quakens zu bestimmen, können die Laufzeitunterschiede der Signale, die von verschiedenen Sensorknoten (Mikrophone) empfangen wurden, ausgewertet werden. Um diese Laufzeitunterschiede aber bestimmen zu können, müssen die Uhren der Sensorknoten synchronisiert sein.

Ein weiteres Anwendungsgebiet der Zeitsynchronisation stellt die energiesparende Kommunikation zwischen den Knoten des Sensornetzes dar. Um Energie zu sparen, sind die Teilnehmer des Netzes die meiste Zeit im „Schlafmodus“. Je präziser sie aus diesem aufwachen können, um an einem vorher vereinbarten Zeitpunkt wieder empfangsbereit zu sein, desto weniger Energie wird verbraucht.

Aber natürlich wird die Zeitsynchronisation auch für die schon aus den traditionellen verteilten Systemen bekannten Anwendungen benötigt: Zeitstempel für die Kryptographie, Koordination von zukünftigen Aktionen (Scheduling), usw.

3 Grundlagen

Mit Hilfe der internen Zeitsynchronisation möchte man einen Zustand erreichen, in dem alle zu synchronisierenden Uhren die gleiche Zeit anzeigen und im gleichen Takt laufen. Zeigen alle Uhren die Realzeit (UTC: Universal Coordinated Time) an, so spricht man von einer externen Zeitsynchronisation. In einem Sensornetz genügt jedoch meist die interne Synchronisation, da ein solcher Verbund von Sensoren im allgemeinen als geschlossenes System betrachtet werden kann.
Unsynchronisiert zeigt jede Uhr eine andere Zeit an und die Uhren laufen auch mit unterschiedlicher Frequenz. Diese Frequenz ist ausserdem nicht stabil. Dieser sogenannte Uhrendrift beträgt je nach verwendetem Oszillator ca. 10^{-6}, potentiell driftet die Uhr in einer Sekunde bereits eine Mikrosekunde ab. Dieser Effekt ist von verschiedenen Faktoren abhängig: Temperatur, Spannung, Alter des Oszillators, usw.

4 Anforderungen an Zeitsynchronisations-Algorithmen in Sensornetzen

Ein Sensornetz weist einige Besonderheiten auf, die ein Algorithmus berücksichtigen muss. Folgende Bedingungen müssen deshalb erfüllt sein, damit ein Protokoll für ein Sensornetz geeignet ist [4]:

- Energieeffizienz: Energie ist in einem Sensornetz die limitierende Grösse. Je effizienter der Algorithmus mit diesem kostbaren Gut umgeht, desto geeigneter ist er. So ist darauf zu achten, die CPU so wenig wie möglich zu nutzen und die Kommunikation auf ein Minimum zu beschränken. Dieser Grundsatz steht im krassen Gegensatz zu den klassischen verteilten Systemen, in denen für das Design eines Algorithmus angenommen werden kann, dass die CPU immer gebraucht werden kann und dass das Netzwerks ohne Einschränkungen dauernd benutzt werden kann. Es ist vor allem dieser Punkt, der den Einsatz der klassischen Zeitsynchronisations-algorithmen in Sensornetzen nicht praktikabel macht.
- Dynamik: Ein Sensornetz ist dynamisch, Knoten schliessen sich ad hoc zusammen, können aber auch zeitweise nicht oder nur über andere Knoten erreichbar sein. Ein Algorithmus sollte mit all diesen Eigenschaften umgehen können.
- Präzision: Wegen der engen Kopplung der Sensornetze an die Realität sollte die Synchronisation sehr genau sein. Nur so kann die Beobachtung hinreichend wirklichkeitsgetreu gemacht werden. Um ein Objekt anhand von Laufzeitunterschiede akustischer Signale zentimetergenau zu orten, ist bereits eine Präzision im µs-Bereich nötig.
- Kosten und Grösse: Der Entwurf eines Algorithmus hat auch Auswirkungen auf die Kosten und Abmessung der Hardware, die verwendet werden muss. Dieser Punkt verbietet also den trivialen Ansatz, in jeden Sensorknoten eine Atomuhr einzubauen.
- Skalierbarkeit: Die Vision hinter den Sensornetzen wird durch das Schlagwort „Smart Dust“ verdeutlicht: In Zukunft sollen hunderttausende Sensorknoten sich zu riesigen Netzen verbinden. Deshalb sollten Algorithmen zur Uhrensynchronisation skalierbar sein, um auch bei einer so grossen Population von Sensorknoten zuverlässig arbeiten zu können.
- Robustheit: Genauso wie die Skalierbarkeit ist auch die Robustheit (zum Beispiel gegen Nachrichtenverlust oder hohe Netzlast) ein wichtiger Aspekt, der insbesondere bei einer grossen Knotenzahl an Bedeutung gewinnt.

Doch auch bezüglich der Zeitsynchronisation selbst werden Anforderungen an den Algorithmus gestellt:

- Die Uhr darf nie zurückgestellt werden, da sonst der gleiche Zeitpunkt zweimal existieren würde. Dies würde es verunmöglichen, die Zeitpunkte von Ereignissen in eine exakte chronologische Reihenfolge zu bringen.
- Der Uhrendrift muss korrigiert werden. Dieser beträgt etwa 10^{-6}, der Zeitgeber driftet also potentiell in einer Sekunde ein Mikrosekunde von der Realzeit ab. Würde diesem Effekt nicht entgegengewirkt werden, so könnte kaum eine Präzision im µs-Bereich erreicht werden.
  Formal ist der Drift r einer Uhr *u(t)* definiert als die um eins verminderte Ableitung nach der Zeit der Funktion *u(t)* [1]. Da die ideale Uhr eine Steigung von eins hat, ist somit der Uhrendrift gerade null, was auch der Definition eines idealen Taktgebers entspricht.

$$\text{Uhrendrift: } r^{u}(t) = \frac{\partial u(t)}{\partial t} - 1$$

5 Protokolle zur Zeitsynchronisation in traditionellen verteilten Systemen

Das bekannteste und meist verbreitete System zur Zeitsynchronisationen in herkömmlichen Netzwerken ist das sogenannte „Network Time Protocol“ (NTP) [5]. Dieses von David L. Mills entworfene Protokoll hat sich durch seinen langjährigen Einsatz im Internet als sehr robust, skalierbar und zuverlässig erwiesen. Diese Skalierbarkeit und Robustheit stand auch im Vordergrund beim Design dieses Protokolls und wird mit einer Baumstruktur von Netzknoten erreicht. Ein Server mit einer exakten Uhr (z.B. Atomuhr) schickt periodisch die aktuelle Zeit an den Client. Der Client kann nun selbst auch wieder als Server fungieren und Zeitstempel versenden. Bei der Ankunft dieser Zeitstempel ist allerdings wegen der Latenz des Netzwerkes die enthaltene Information schon veraltet. Diese Verzögerungszeit wird mit der Hälfte der sogenannten „round-trip-time“, also der Zeit die eine Nachricht vom Sender zum Empfänger und wieder zurück benötigt, geschätzt, dies setzt allerdings relativ symmetrische Latenzen voraus.
Obwohl sich NTP im Internet durchgesetzt hat, ist es für Sensornetze ungeeignet: Mit diesem Verfahren wird eine Genauigkeit im ms-Bereich erreicht, abhängig von der Position in der Baumstruktur. Die Ungenauigkeit wird durch die Tatsache verstärkt, dass in einem Sensornetz mit dynamischem Routing in den seltensten Fällen eine symmetrische Latenz vorausgesetzt werden kann. Die Präzision ist also etwa um den Faktor 1000 ungenügend.
Des Weiteren horcht der Client bei diesem Protokoll dauernd am Netzwerk und wartet auf den Empfang eines Zeitstempels. Dieses fortwährende Lauschen am Netz ist jedoch sehr energieaufwändig und ist somit in Sensornetzen nicht sinnvoll.
Ausserdem wird mit diesem Protokoll der dynamischen Topologie eines Sensornetzes nicht Rechnung getragen. NTP setzt eine statische baumartige Struktur voraus, ein Widerspruch zu den dynamischen ad hoc Sensornetzen. Zudem können auch Knoten, die zur Zeit nicht erreichbar sind, Ereignisse registrieren, die man im nachhinein synchronisieren möchte. Dies ist mit NTP nicht möglich.

Einen anderen Weg zu einer möglichst genauen Approximation der Realzeit besteht in der Ausnutzung von global vorhandener Zeitinformation. So lässt sich mit einem GPS-Empfänger die Realzeit mit einer Genauigkeit von etwa 200ns bestimmen [6], die Präzision wäre also auf jeden Fall ausreichend. Doch die Nachteile überwiegen: Der Empfänger braucht mehrere Minuten zur Initialisierung und Sichtkontakt zu den Satelliten, dies würde also zum Beispiel einen Einsatz in Gebäuden nicht zulassen. Das Gewicht, die Kosten und der hohe Stromverbrauch der heutigen GPS-Hardware machen diese Art der Zeitsynchronisation für Sensornetze gänzlich unbrauchbar.

Erwähnenswert sind auch noch die logischen Uhren (Lamport [7], Vektorzeit [8]), mit denen sich eine kausaltreue Beobachtung realisieren lässt. Auch diese Verfahren haben sich in den traditionellen verteilten Systemen als günstiger Ersatz für die fehlende Realzeit bewährt. Doch die logischen Uhren verfolgen ein ganz anderes Konzept: Mit ihnen ist es möglich interne Ereignisse (z.B. Nachrichtenempfang, -versand) in eine kausale Ordnung zu bringen. In einem Sensornetz besteht allerdings der Wunsch, externe Ereignisse, die nicht kausal zusammenhängen müssen, in eine exakte chronologische Abfolge zu bringen, um beispielsweise präzise zeitliche Abstände zu messen.

6 Packet Stream Synchronization

6.1 Grundidee

Das Systemmodell dieses Verfahrens besteht im wesentlichen aus zwei Komponenten: Einem Produzenten und einem Konsumenten. Der Produzent verfügt über eine ideale Uhr (kein Uhrendrift) und sendet in regelmässigen Abständen Zeitstempel dieser Referenzzeit *t* aus. Der Konsument empfängt diese und startet daraufhin den Algorithmus zur Zeitsynchronisation. Der Empfänger verfügt über zwei Uhren: Eine Hardware-Uhr *h(t)* mit einem unbekannten Uhrendrift und eine vom Algorithmus berechnete Software-Uhr *c(t)*. Beide Uhren können als Funktionen der Referenzzeit *t* interpretiert werden, deren Funktionswert abhängig von der aktuellen Zeit *t* ist.

Diese Methode verfolgt einen ähnlichen Ansatz wie NTP. Das Grundproblem eines solchen Lösungsansatzes ist ein klassisches Henne-Ei-Problem: Um die Uhren synchronisieren zu können, müsste die Verzögerungszeit des Netzwerkes bekannt sein. Um aber diese Latenz zu messen, müssten die Uhren schon synchronisiert sein. Um diesen Zyklus zu durchbrechen, wird bei dieser Methode die Annahme getroffen, dass die Verzögerung des Netzwerkes gleich Null sei.

Die Kommunikation erfolgt nur unidirektional vom Produzenten zum Konsumenten. Es werden keine besonderen Anforderungen an diese Verbindung gestellt, die Latenz kann beliebig sein und muss keine normierte Verteilung aufweisen. Allerdings wurde in realen Netzwerken die Beobachtung gemacht, dass die minimale Verzögerungszeit relativ konstant ist, unabhängig von der Netzlast oder dem Empfänger. Diese Beobachtung trifft nur für das Minimum zu, für den Durchschnitt oder das Maximum gelten diese Erkenntnisse nicht.

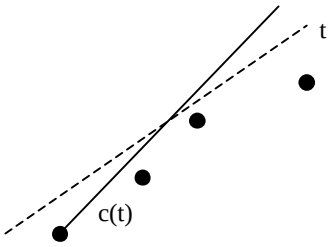

Abbildung 6.1: Ist die Steigung der Software-Uhr grösser als eins, besteht die Gefahr, dass die Präzision unendlich schlecht wird.

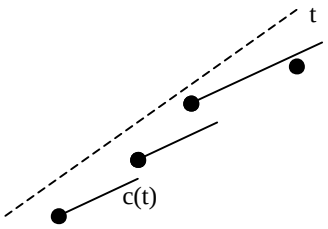

Abbildung 6.2: Nur wenn die Software-Uhr einen negativen Drift hat und aktuellere Zeitstempel berücksichtigt werden, wird ein Abdriften der Präzision verhindert.

Basierend auf dieser Tatsache wird bei diesem Verfahren versucht, eine untere Schranke für die Realzeit zu approximieren. Würde der Empfänger ebenfalls über eine Uhr ohne Drift verfügen, so könnten ankommende Zeitstempel mit einem aktuelleren (sprich höherem) Zeitwert als neuer Startwert der Uhr dienen. Die Annäherung würde immer exakter und hätte schliesslich eine Genauigkeit in der Grösse der minimalen Verzögerungszeit. Da nun alle Knoten aufgrund der Konstanz der minimalen Latenz die gleiche Genauigkeit erreichen würden, wäre die Präzision der internen Synchronisation der Knoten untereinander extrem genau.

Doch leider verfügen die Empfänger nicht über einen solche driftlosen Zeitgeber, es muss also versucht werden, eine solche Uhr zu simulieren. Das Resultat dieser Simulation ist eine sogenannte Software-Uhr. Diese kann allerdings niemals ohne Drift sein, sie sollte aber zwei Eigenschaften haben:

1) Weisen die ankommenden Zeitstempel einen höheren Zeitwert als der momentane Stand der Software-Uhr auf, so stellen sie einen aktuelleren Wert dar. Dieser Wert muss dann als neuer Initialwert der Software-Uhr genommen werden.
2) Der Drift sollte negativ sein, d.h. die Steigung der Software-Uhr darf maximal eins sein. Ein positiver Drift würde bedeuten, dass die Uhr schneller läuft als die Referenzzeit. Nach einiger Zeit hätten ankommende Zeitstempel einen Wert, der immer kleiner als die aktuelle Zeit der Software-Uhr ist, diese würde dadurch nie mehr korrigiert werden und potentiell unendlich abdriften (Abbildung 6.1). Durch den negativen Drift wird sichergestellt, dass die Software-Uhr immer langsamer läuft als die Referenz-Uhr des Produzenten. Dadurch wird die Uhr nicht mehr „nach oben“ abdriften und wegen der ersten Bedingung wird sie nicht unendlich schlecht „nach unten“ abdriften (Abbildung 6.2).

6.2 Der Algorithmus

Da der Knoten nur über eine driftbehaftete Hardware-Uhr verfügt, lässt sich ohne weitere Einschränkung keine Software-Uhr mit den beiden oben genannten Eigenschaften programmieren. Es wird daher die Annahme getroffen, dass der Drift der Hardware-Uhr beschränkt ist:

$$\left| r^{h}(t) \right| < r^{h}_{\max}$$

Durch diese Einschränkung lässt sich nun eine Software-Uhr realisieren, welche die beiden oben genannten Bedingungen erfüllt. Mit Hilfe dieser lässt sich der in der Grundidee beschriebene Algorithmus implementieren, welcher bei jedem Erhalt eines Zeitstempels ausgeführt wird und versucht, die Software-Uhr näher an die Referenzzeit heranzubringen [2]:

- Der erste Initialwert c_1 der Softwareuhr wird gleich dem ersten ankommenden Zeitstempel s_1 gesetzt:

$$c_1 = s_1$$

- Als Initialwerte c_i der nächsten Runden wird das Maximum aus Ankunftszeitpunkt r_i bezüglich der aktuellen Software-Uhr und des Zeitstempels s_i genommen:

$$c_i = \max(s_i, c_{i-1}(r_i))$$

- Die Funktion der Software-Uhr in Runde i sieht dann wie folgt aus:

$$c_i(t) = c_i + \frac{1}{1 + r^{h}_{\max}} \cdot (h(t) - h(r_i)) \qquad (\forall\, t \geq r_i)$$

Durch Ableiten dieser Funktion nach *t* lässt sich leicht überprüfen, dass der Drift dieser Software-Uhr immer negativ ist. Wegen der Maximum-Wahl bei den Initialwerten jeder Runde wird auch der Forderung (1) Rechnung getragen.

Eine solcher simulierter Zeitgeber hat eine Abweichung von der Referenzzeit in der Grössenordnung der minimalen Latenz. Da diese minimale Verzögerung eine vom Empfänger unabhängige Konstanz aufweist, erhalten alle Knoten in etwa die selbe untere Schranke, es wird also eine interne Synchronisation der Empfänger erreicht.

6.3 Vergleich mit Anforderungskatalog

- Energieeffizienz: Diese Anforderung wird nicht genügend beachtet, da der Sender regelmässig Zeitstempel aussenden muss. Bei der Simulation in [2] beträgt der Abstand zwischen zwei Nachrichten 20ms. Der Produzent verbraucht also alle 20ms Energie zum Senden, aber auch die Konsumenten der Zeitstempel müssen in diesem regelmässigen Abstand am Netz lauschen und die Zeitstempel empfangen.
- Dynamik: Auch dieser Punkt ist nicht erfüllt. Nicht erreichbare Knoten können nicht im Nachhinein synchronisiert werden. Des Weiteren benötigt dieses Verfahren eine statische Infrastruktur, da der spezielle Produzenten-Knoten über eine ideale Uhr verfügen muss. Würde der Senderknoten nicht über eine solche Uhr verfügen, könnte bei einigen Knoten das Szenario in Abbildung 6.1 eintreten, andere würden sich gemäss Abbildung 6.2 verhalten, die interne Synchronisation könnte potentiell unendlich schlecht werden.
- Präzision: In der Simulation wurde mit diesem Verfahren eine Präzision von 10µs erreicht, die Präzision ist also ausreichend.
- Kosten und Grösse: Dieses Verfahren benötigt keine zusätzlichen Instrumente, die Abmessung und Kosten der Knoten bleiben deshalb unverändert.
- Skalierbarkeit: Ist innerhalb dem Empfangsgebiet eines Produzenten sicher gegeben, da nur eine unidirektionale Kommunikation verwendet wird. Somit könnte für die Übertragung der Zeitstempel auch eine Broadcast gebraucht werden.
- Robustheit: In der Simulation [2] hat der Algorithmus eine hohe Robustheit gegen Netzlast aufgewiesen. Dies lässt sich damit erklären, dass die minimale Latenz nicht von der Netzlast abhängt. Auch ein gewisser Nachrichtenverlust wird von dieser Methode verkraftet.
- Kein Zurückstellen der Uhr: Die Uhr wird nur vorgestellt und niemals zurückgestellt, diese Bedingung ist somit erfüllt.
- Korrektur des Uhrendrift: Bei dieser Variante des Verfahrens wird der Drift nicht korrigiert, es existiert allerdings eine Version dieses Algorithmus, die den Effekt des Uhrendrift mittels einer Heuristik zu korrigieren versucht [2].

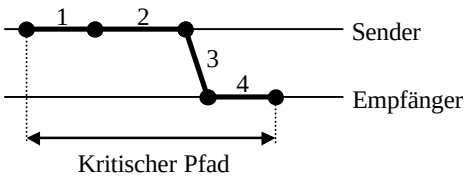

Abbildung 7.1: Die vier Komponenten des kritischen Pfades bei einer Punkt-zu-Punkt Kommunikation.

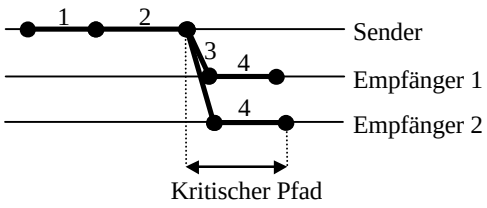

Abbildung 7.2: Durch einen Broadcast auf physischer Ebene wird der kritische Pfad stark verkürzt.

7 Reference Broadcast Synchronization

7.1 Grundidee

Das Hauptproblem bei der Zeitsynchronisation stellt der Weg vom Sender eines Zeitstempels zum Empfänger dieser Nachricht dar. Auf diesem kritischen Pfad treten nicht-deterministische Verzögerungen auf, die ein Abschätzen der Latenz schwierig machen. Bei dem Verfahren der „Reference Broadcast Synchronization“ wird dieser kritische Pfad durch Verwendung eines Broadcasts auf physischer Ebene verkürzt. Die Verzögerung einer Nachricht vom Sender zum Empfänger setzt sich aus vier Komponenten zusammen (Abbildung 7.1):

1) Sendedauer: Konstruieren der Nachricht, Einfügen von Headers, Berechnung von Prüfsummen, usw.
2) Zugriffszeit auf das Netz: Warten bis das gemeinsame Kommunikationsmedium frei ist und die Nachricht gesendet werden kann.
3) Flugzeit vom Sender zum Empfänger.
4) Empfangsdauer: Entgegennahme der Nachricht.

Durch einen Broadcast wird der kritische Pfad stark beschnitten (Abbildung 7.2): Die Komponenten (1) und (2) sind für alle Empfänger gleich und haben somit keinen Einfluss mehr. Die Flugzeit kann vernachlässigt werden, da sich elektromagnetischen Wellen mit Lichtgeschwindigkeit ausbreiten. Diese Dauer liegt demzufolge im unteren ns-Bereich und hat damit wenig Einfluss auf die geforderte µs-Präzision. Somit bleibt einzig die Empfangsdauer weiterhin nicht-deterministisch. Doch lässt sich auch diese Zeitspanne und somit die Varianz hardwaremässig stark verkürzen, indem bei der Ankunft des ersten Bits bereits ein Interrupt ausgelöst wird, bei dem die lokale Systemzeit ausgelesen wird.

Die Länge des verkürzten kritischen Pfades ist zwar nicht bei allen Empfängern gleich gross, doch sind die Unterschiede zwischen der Laufzeit einer Nachricht an zwei verschiedene Empfänger normalverteilt mit einem Mittelwert gleich Null. Wird nun also angenommen, dass eine Broadcast-Nachricht bei allen Empfängern gleichzeitig ankommt, ist der dadurch entstehende Fehler gleich der Standardabweichung dieser Normalverteilung. Allerdings fällt dieser Fehler nicht mehr allzu sehr ins Gewicht, wenn der verbleibende kritische Pfad nur noch eine Länge von wenigen Mikrosekunden hat.

7.2 Der Algorithmus

Durch das Treffen der Annahme, dass die Broadcast-Nachricht bei allen Adressaten gleichzeitig eintrifft, verfügt man über ein Ereignis mit einem in der Broadcast-Domäne einheitlichen Zeitpunkt. Tauschen die Empfänger den Empfangszeitpunkt der Nachricht bezüglich ihrer lokalen Zeit aus, kann jeder Knoten den Offset zwischen seiner Zeitskala und

der seiner Nachbarn berechnen. Die Präzision kann zudem noch verbessert werden, indem mehrere Broadcasts durchgeführt werden und der durchschnittliche Offset verwendet wird.

Durch die berechtigte Annahme, dass der Uhrendrift über einen gewissen Zeitraum konstant bleibt und sich die Abweichung zwischen zwei Uhren linear verhält, kann der Drift mit einer linearen Regression korrigiert werden. Jede Broadcast-Nachricht liefert einen zum Empfangszeitpunkt gültigen Offset. Aus mehreren Broadcasts wird mit Hilfe der linearen Regression ein Zusammenhang zwischen der Zeit und Offset hergestellt.

Mit der Funktion dieser Regressionsgeraden kann zu jedem beliebigen Zeitpunkt der zugehörige Offset zur Zeitskala eines anderen Knoten errechnet werden. Durch die Annahme, dass der Uhrendrift über einen gewissen Zeitraum konstant bleibt, lässt sich mit dieser Funktion auch der Unterschied zweier auf verschiedenen Knoten gemessener Zeitpunkte aus der Vergangenheit rekonstruieren. Es ist also eine sogenannte Post-Facto Synchronisation möglich, die Knoten müssen also den Synchronisationsalgorithmus erst starten, wenn das Abgleichen der Uhren tatsächlich nötig ist.

Die mit diesem Verfahren erreichte Präzision der internen Synchronisation liegt im unteren Mikrosekunden-Bereich. Bei einer Messung [3] mit drei IPAQs, die ad hoc über ein 802.11 WLAN verbunden waren, konnte eine Präzision von 1.85µs erreicht werden. In dieser Messung wurde auch eine gute Robustheit gegen hohe Netzlast gezeigt, die Präzision der Synchronisation wurde dadurch nur um etwa einen Drittel reduziert. Allerdings ist dieses Resultat mit Vorsicht zu geniessen, da die Netzlast nur durch einen Dateitransfer „im Hintergrund“ erzeugt wurde, die an der Messung beteiligten Knoten waren weder Sender noch Empfänger dieser Daten.

Dieses Verfahren bietet auch die Möglichkeit, sehr simpel eine Synchronisation über mehrer Hops durchzuführen, also eine Multi-Hop Synchronisation. Die Grundidee besteht darin, dass die auszutauschenden Zeitwerte bei jedem Hop konvertiert werden. Allerdings erhöht sich der Fehler bei jeder Umwandlung, doch wächst dieser nur mit der Wurzel der Anzahl Hops [3].

7.3 Vergleich mit Anforderungskatalog:

- Energieeffizienz: Wegen der Möglichkeit der Post-Facto Synchronisation müssen die Sensorknoten erst bei Bedarf synchronisiert werden, was sich auch im Energiehaushalt der Knoten niederschlägt, da sie nicht dauernd zum Empfangen von Synchronisations-nachrichten aufwachen müssen.
- Dynamik: Dieser Punkt wird zum Einen wegen der Post-Facto Synchronisation erfüllt, da Knoten, die im Augenblick nicht erreichbar sind, auch nachträglich noch synchronisiert werden können. Zum Andern ist mit diesem Verfahren eine Multi-Hop-Synchronisation möglich. Allerdings wird vorausgesetzt, dass es sich um ein broadcast-fähiges Netz handelt.
- Präzision: Die erreichte Präzision ist mit 1.85µs sehr exakt.
- Kosten und Grösse: Dieses Verfahren benötigt keine zusätzlichen Instrumente, die Abmessung und Kosten des Knoten bleiben unverändert.
- Skalierbarkeit: Die Nachrichtenkomplexität ist in der Grössenordnung $m \cdot n^2$, wobei m die Anzahl der Broadcasts pro Regression ist und n die Anzahl der zu synchronisierenden Knoten darstellt. In der Messung [3] wurde die hohe Präzision durch die Verwendung von 30 Broadcasts erreicht. Eventuell müssen allerdings nicht alle Knoten des Sensornetzes am Synchronisationsprozess partizipieren, sondern nur die Teilmenge, welche synchronisierte Uhren benötigt.
- Robustheit: In der Simulation wies der Algorithmus eine hohe Robustheit gegen Netzlast auf. Wie schon erwähnt, ist dieses Resultat mit Vorsicht zu geniessen. Zudem

verkraftet diese Methode einen gewissen Nachrichtenverlust, da die Regression auch mit weniger Nachrichten noch eine ausreichende Genauigkeit aufweist.

- Kein Zurückstellen der Uhr: Da nur Offsets zwischen den Uhren berechnet werden, werden die Uhren nicht verstellt.
- Korrektur des Uhrendrift: Der Uhrendrift wird unter der Annahme einer gewissen zeitlichen Konstanz des Drifts korrigiert.

8 Schlussbetrachtung

Zum Thema Zeitsynchronisation in Sensornetzen bzw. allgemeiner in ad hoc Netzen existieren viele Arbeiten mit sehr unterschiedlichen Ansätzen und ebenso unterschiedlichen Voraussetzungen und Eigenheiten. Dies macht auch Sinn, da es in dem Bereich der Sensornetze auch unterschiedliche Anwendungen mit sehr unterschiedlichen Anforderungen gibt. Aufgrund dieser Tatsache ist es überflüssig, nach dem optimalen Algorithmus zur Zeitsynchronisation zu suchen, da es diesen wegen dieser Vielfalt der Problemstellung nicht geben kann.

Anhand der beiden vorgestellten Verfahren sollte auch klar geworden sein, dass immer ein Trade-Off zwischen Präzision und Energieverbrauch gefunden werden muss. Ein Algorithmus der nie oder nur sehr selten Nachrichten zur Synchronisation verschickt, spart zwar Energie, doch ist die damit erreichte Präzision schlecht. Ein dauerndes Senden von Zeitstempeln würde zwar bei der „Packet Stream Synchronization“ die Präzision erhöhen, doch dies geschieht dann zum Nachteil des Energiehaushaltes. Um diesem Trade-Off gerecht zu werden, sollten sich die Algorithmen parametrisieren lassen, um sich an die jeweilige Anwendung anzupassen. Bei der ersten Variante entspricht der Abstand zwischen zwei Zeitstempeln diesem Parameter, bei der Zweiten wird er durch die Anzahl der Broadcast-Nachrichten verkörpert.

Die „Reference Broadcast Synchronisation“ erfüllt den Anforderungskatalog fast, jedoch wird ein broadcastfähiges Netz vorausgesetzt. Um eine hohe Präzision zu erreichen, muss hardwaremässig ein Interrupt unterstützt werden, der bei der Ankunft des ersten Bits einer Nachricht ausgelöst wird. Bei der „Packet Stream Synchronization“ wäre es von Vorteil, wenn die ankommenden Zeitstempel ebenfalls gleich als Nachricht mit hoher Priorität erkannt werden und nicht erst noch lange im Netzwerk-Stack verweilen würden. Es lässt sich also sagen, dass der Aspekt der Zeitsynchronisation bereits zu Beginn beim Hardware-Design des Sensornetzes berücksichtig werden sollte. Es macht wenig Sinn, softwareseitig am Algorithmus zu feilen um die Präzision einige Mikrosekunden zu steigern, wenn mit einer simplen Hardwareunterstützung eine grössere Präzisionssteigerung möglich ist.

9 Referenzen

[1] Philipp Blum and Lothar Thiele. Clock Synchronisation using Packet Streams. *16th International Symposium on Distributed Computing (DISC)*, Toulouse, France, October 2002.

[2] Philipp Blum. Precise and Low-Jitter Wireless Time Synchronizazion, unpublished.

[3] Jeremy Elson, Lewis Girod and Deborah Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *Proceeding of the Fith Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, December 2002.

[4] 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)*, Princeton, New Jersey, USA, 28-29 October 2002.

[5] David L. Mills. *Conference on Communication Architectures (ACM SIGCOMM'94)*, London, UK, August 1994.

[6] Leslie Lamport. Time, clocks, and the ordering of events in an distributed system. *Communications of the ACM*, 21(7):558-65, 1978.

[7] F. Mattern. Virtual Time and Global States in Distributed Systems. In *Workshop on Parallel and Distributed Algorithms*, Chateau de Bonas, October 1988.