

**"ZEITSYNCHRONISATION IN TMOTESKY  
SENSORNETZEN "**

Serge Shumilov  
Boris Iven, Oliver Mali, Roman Saul

Dezember 2006

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
<b>2</b>	<b>Das Synchronisationsverfahren</b>	<b>3</b>
2.1	Das Verfahren . . . . .	3
2.1.1	Die Streuung der Offsets . . . . .	4
2.1.2	Sync Points und Average Offsets . . . . .	4
2.1.3	Uhrendrift verschiedener Motes . . . . .	6
2.1.4	Berechnung des Skews . . . . .	8
2.1.5	Berechnung der globalen Zeit . . . . .	9
2.2	Parametrisierung . . . . .	9
2.3	Genauigkeit . . . . .	10
<b>3</b>	<b>Anwendung</b>	<b>11</b>
3.1	Interfaces . . . . .	11

# 1 Einleitung

Unter dem Begriff „Sensornetz“ versteht man einen Verbund von kleinsten, drahtlos miteinander vernetzten Computern, die durch die Ausstattung mit Sensoren zur feinmaschigen und umfassenden Beobachtung von Phänomenen der realen Welt eingesetzt werden. Aufgrund dieser Einbettung von Sensornetzen in die reale Welt haben die Kategorien Raum und Zeit eine fundamentale Bedeutung für viele Anwendungen. Für die Koordination der Sensorknoten untereinander und die Interpretation von Beobachtungen spielt der Zeitpunkt eines Ereignisses eine wichtige Rolle. Viele Anwendungen erfordern eine gemeinsame Zeitbasis der beteiligten Sensoren. Hierzu ist das Synchronisieren der Uhren aller Geräte erforderlich. Wenn die Hardware diesen Dienst nicht anbietet, so muss die Synchronisation über spezielle Protokolle erfolgen. Die vorhandenen Protokolle (Network Time Protokoll (NTP)) und Algorithmen zur Zeitsynchronisation, die sich in herkömmlichen Netzen bewährt haben, erfüllen leider die Anforderungen der Sensornetze nur unzureichend, weshalb neue Methoden zur Synchronisation der Uhren entworfen werden müssen. Eine Analyse und eine Reihe von verschiedenen Synchronisationsverfahren findet sich z.B. in [EGE02, ER02, MKSL04, Röm05].

Der Kern dieses Projektes ist die Untersuchung der Anforderungen an die Synchronisation und die Implementierung eines Synchronisationsverfahrens für die neue Tmote Sky Plattform der Firma Moteiv. Leider unterstützt TinyOS diese Hardware zur Zeit noch nicht vollständig, daher gibt es im System noch keinen Dienst, den man für die Synchronisation dieser Motes nutzen könnte. Existierende Implementierungen [Van] sind an ältere Mica2-ähnliche Plattformen orientiert. Im Unterschied zu diesen Plattformen besitzen die Tmote Sky Geräte einen neuen Transceiver (CC2420), der den Standards IEEE 802.15.4 und ZigBee entspricht. Durch die Nutzung von Start of Frame Delimiter Signalen (SFD) [CJM05] kann das hier vorgestellte Verfahren eine bessere Genauigkeit erzielen, als es bisher auf der MAC-Ebene dieser Plattform möglich war.

## 2 Das Synchronisationsverfahren

Bei diesem Protokoll wird die Zeit durch Verschicken von speziellen Paketen mit Zeitstempeln synchronisiert. Dabei werden bewusst die unterschiedlichen Zeitbasen der verschiedenen Geräte beibehalten. Prinzipiell wird dabei ausgehend von einem bestimmten Sensor (Master) dessen Zeit (die globale Zeit) an alle anderen Sensoren (Slaves) per Broadcast gesendet. Diese berechnen dann die Differenz (Offset) und den Drift (Skew) zwischen ihrer Uhr und der des Masters. Dadurch sind sie in der Lage, aus ihrer lokalen Zeit eine globale Zeit zu berechnen und diese möglichen Anwendungen bereitzustellen.

### 2.1 Das Verfahren

Der Master sendet in definierbaren Intervallen (Sync Interval) per Broadcast Blöcke von  $n$  Nachrichten mit einem 32-Bit Zeitstempel (globale Zeit) in das Netzwerk. Empfängt ein Slave eine solche Nachricht, wird die lokale Empfangszeit gemessen und die Differenz zur globalen Zeit (Offset) berechnet. Um möglichst genaue Zeitstempel zu erhalten, verwenden wir den sog. START OF FRAME DELIMITER INTERRUPT (SFD), den TinyOS zur Verfügung stellt. Tupel aus lokaler Empfangszeit und Offset werden in einem Array der Größe  $n$  auf den Slaves gespeichert. Dieses Verfahren lehnt sich an das in [MKSL04] vorgestellte FTSP an, bei dem jedoch die Berechnung des Skews durch eine Lineare Regression

erfolgt. Diese Vorgehensweise zeigte bei unserer Implementierung eine unzureichende Genauigkeit des berechneten Skews, so dass es schien, als sei dieser nicht konstant. Zur Ermittlung der Ursache diente uns der folgende Versuch.

### 2.1.1 Die Streuung der Offsets

Wir haben zunächst einzelne Zeitstempel ( $n = 1$ ) in schneller Folge (2 pro Sekunde) verschickt und die Differenz zur Uhr des Empfängers (Offset) protokolliert. Abbildung 1 zeigt

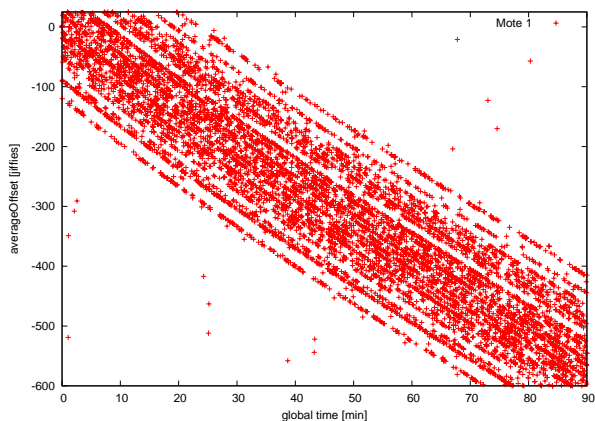


Abbildung 1: Offset( $t$ ), 1 Msg/Block

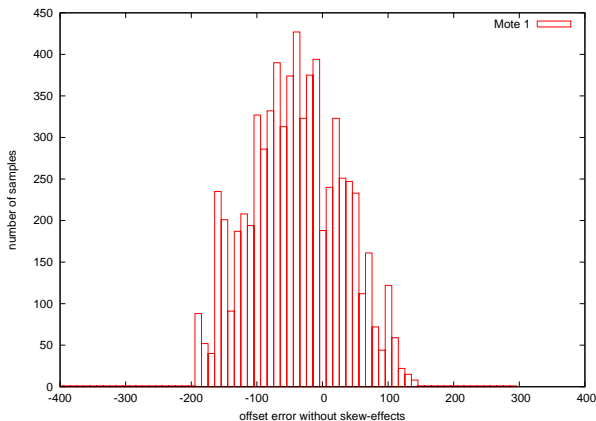


Abbildung 2: Verteilung

den Verlauf der Offsetwerte. Auffallend ist, dass die Werte in einem begrenzten Bereich etwa gleichverteilt sind und darüberhinaus nur vereinzelte Ausreißer auftreten. Weiterhin ist bereits zu erkennen, dass der Uhrendrift (Skew) konstant ist. Weitere Messungen über einen längeren Zeitraum werden diese Vermutung später bestätigen. Wir haben den Drift mit Hilfe einer Ausgleichsgerade korrigiert und so die Abweichung der Werte von dieser Gerade in einem Histogramm (siehe Abb. 2) aufbereitet. Das „Band“ in dem die Werte etwa gleichverteilt sind, ist etwa 350 jiffies<sup>1</sup> breit. Bei diesem Ausmaß der Streuung ist es natürlich nicht sinnvoll, den Skew mittels Linearer Regression aus  $n$  zeitlich dicht aufeinanderfolgenden Zeitstempeln zu berechnen.

Wir modifizierten das Verfahren dahingehend, dass die Slaves nach dem Empfang der  $n$ -ten Nachricht nur die Mittelwerte der lokalen Empfangszeiten (sync point) und die Mittelwerte der Offsets (Average Offset) berechnen. Der Skew wird nun anschließend durch einen Algorithmus (siehe Abschnitt 2.1.4) bestimmt, der mit den gerade ermittelten Average Offsets und Sync Points arbeitet. Dafür müssen jedoch diese Mittelwerte in einer akzeptablen Genauigkeit bereitgestellt werden. Der Parameter, der hier justiert werden kann ist die Anzahl der Samples. Hierzu wurde folgender Versuch zur Ermittlung eines optimalen Anzahl als Kompromiss zwischen Genauigkeit und Energieeffizienz durchgeführt.

### 2.1.2 Sync Points und Average Offsets

In dieser Messreihe speichern die Slaves Blöcke zu je 5, 10, 25 und 50 Wertepaaren ( $t_s, t_r$ ), wobei  $t_s$  ein Zeitstempel vom Master ist, als dieser das Paket gesendet hat. Die lokale Empfangszeit des Slaves ist in  $t_r$  gespeichert. Aus jeweils einem dieser Blöcke erhält man durch Mittelung der  $t_r$  einen sogenannten Sync Point. Zu diesem wird wiederum durch

<sup>1</sup>1 jiffy = 1/32768 s

Mittelung der Differenzen  $t_s - t_r$  ein Offset berechnet, diesen nennen wir Average Offset. Der Abstand zwischen zwei Nachrichten in einem Block betrug hierbei 250 ms, der Abstand zwischen zwei aufeinander folgenden Blöcken 500 ms. Die Messergebnisse sind in den Abbildungen 3 bis 10 graphisch dargestellt. Sie legen den Schluss nahe, dass durch Mittelung über etwa 20-30 Nachrichten eine akzeptable Genauigkeit der Average Offsets erzielt werden kann und eine weitere Erhöhung nur noch die Netzlast erhöht. Wir werden darauf in Abschnitt 2.3 näher eingehen.

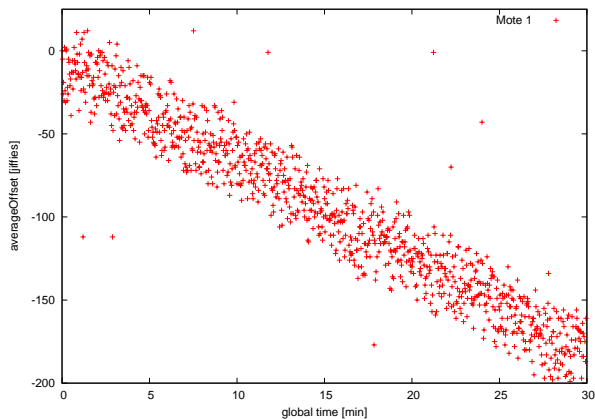
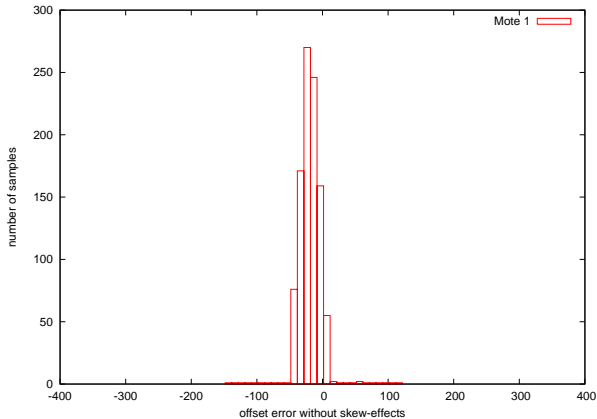
Abbildung 3: Offset( $t$ ), 5 Msgs/Block

Abbildung 4: Verteilung

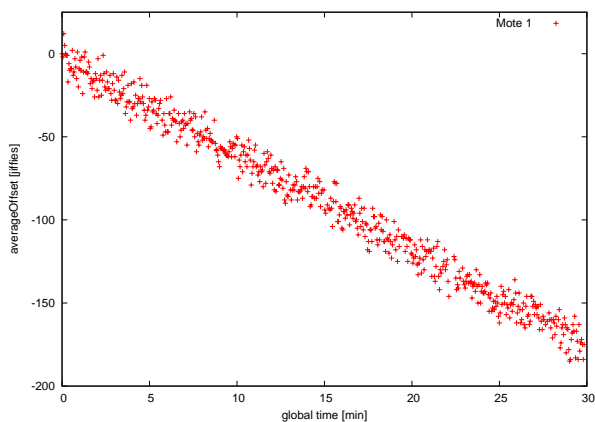
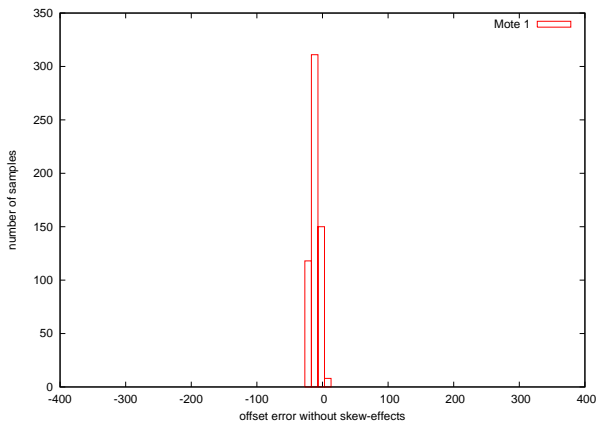
Abbildung 5: Offset( $t$ ), 10 Msgs/Block

Abbildung 6: Verteilung

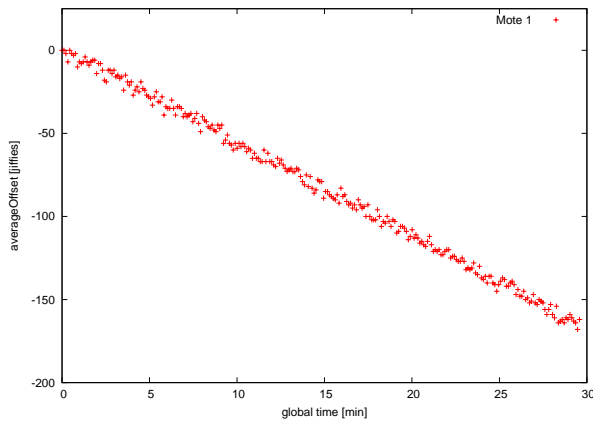
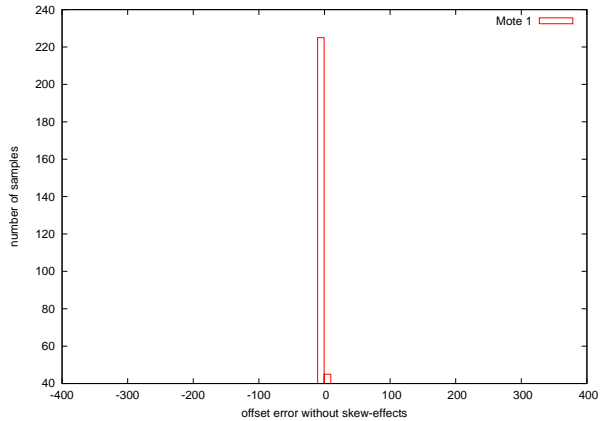
Abbildung 7: Offset( $t$ ), 25 Msgs/Block

Abbildung 8: Verteilung

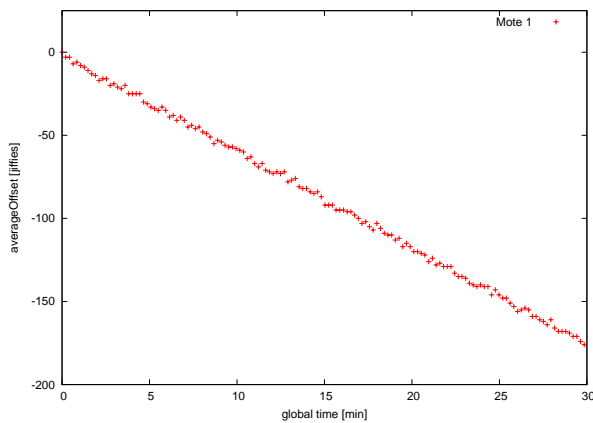
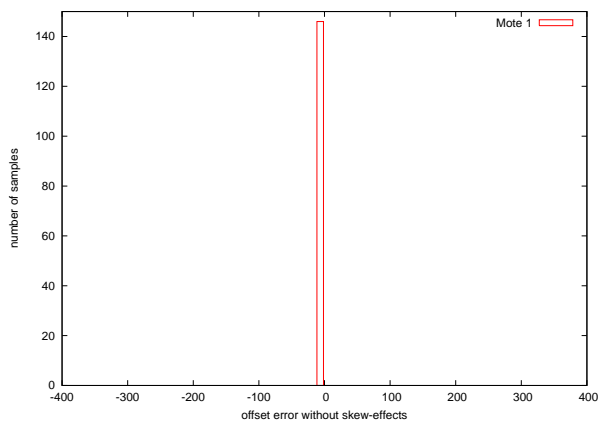
Abbildung 9: Offset( $t$ ), 50 Msgs/Block

Abbildung 10: Verteilung

### 2.1.3 Uhrendrift verschiedener Motes

Wie verhält sich nun der Uhrendrift über einen längeren Zeitraum und bei verschiedenen Motes? Die folgende Messung (Abbildung 11) zeigt den Verlauf der Average Offsets bei 6 Motes über einen Zeitraum von etwa einer Stunde. Bei diesen Messungen wurde über Blöcke zu je 25 Synchronisationsnachrichten gemittelt. Der Abstand zwischen zwei Blöcken betrug 500 ms. Es ist zu erkennen, dass der Uhrendrift, also die Steigung der Geraden, eine konstante Größe ist, die von Mote zu Mote jedoch stark variiert. Eine weitere Messreihe bestätigt die Gültigkeit dieses Ergebnisses für einen Mote über einen Zeitraum von etwa 24 Stunden (Abbildung 12).

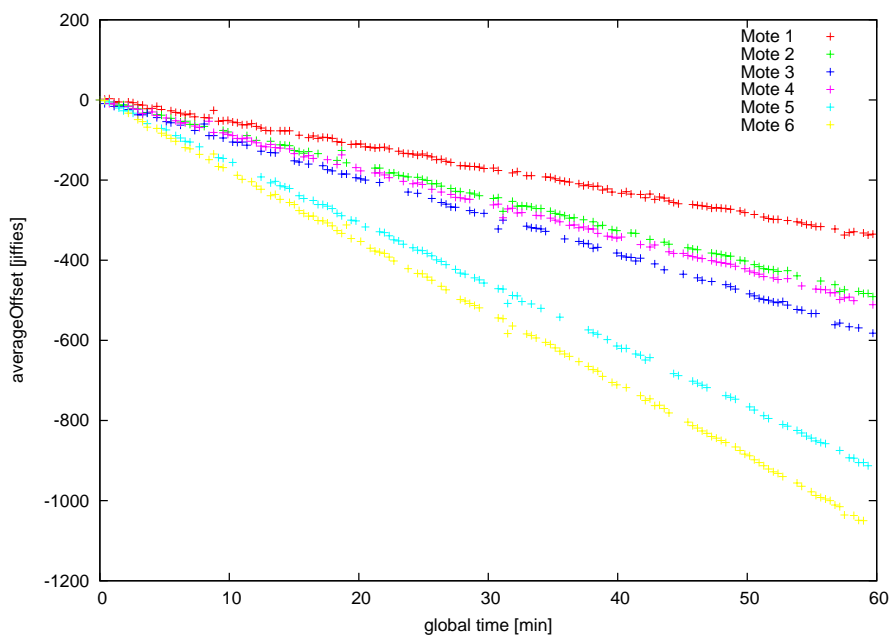


Abbildung 11: Offsetverlauf bei 6 motes

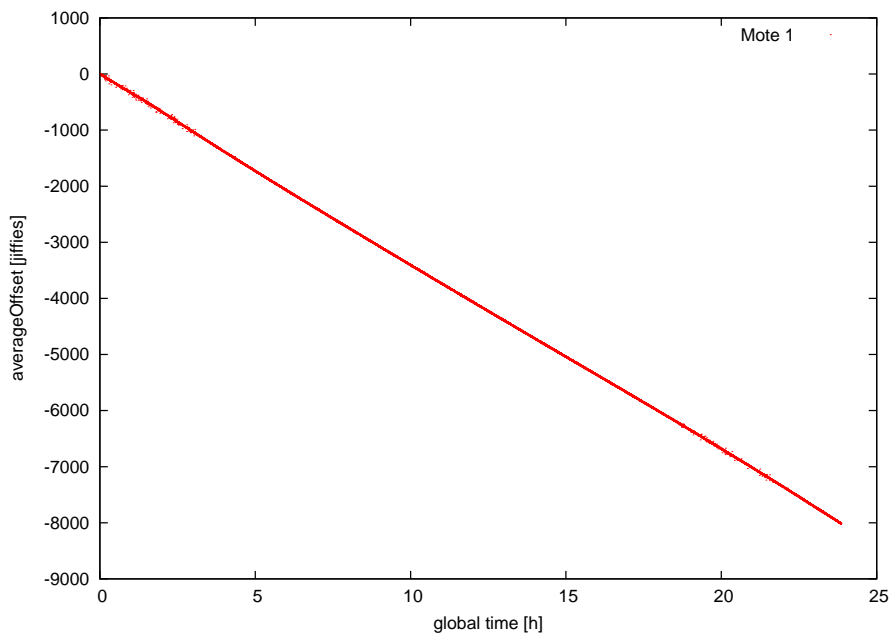


Abbildung 12: Offsetverlauf über 24 h

### 2.1.4 Berechnung des Skews

Bestimmt man neben dem Mittel der Offsetwerte ( $y$ -Werte) noch das Mittel der Sendezeiten ( $x$ -Werte), so erhält man eine Folge von Synchronisationspunkten (Sync Points). Wir ermitteln nun den Uhrendrift (Skew) als Steigung der Geraden durch zwei aufeinander folgenden Wertepaaren (siehe Abb. 17). Zusätzliche Genauigkeit erhalten wir durch eine Alterungsfunktion, die die letzten beiden Skew-Werte gewichtet verknüpft:

$$\begin{aligned}\bar{s}_0 &:= 0 \\ \bar{s}_n &:= \frac{m-1}{m} \bar{s}_{n-1} + \frac{1}{m} s_n \\ \bar{s} &:= \bar{s}_n\end{aligned}$$

Wobei  $m$  eine geeignet gewählte Konstante und  $s_n$  die neueste berechnete Steigung ist. Initial setzen wir  $\bar{s}_0 = 0$ . Dies hat zur Folge, dass  $\bar{s}_n$  erst nach und nach gegen den gesuchten Wert konvergiert.

Um möglichst rasch nach Start der Messung auf gute Schätzwerte für den Drift zurück greifen zu können haben wir die Konvergenz zu Beginn der Messung beschleunigt:

$$\begin{aligned}\bar{s}_n (1 \leq n \leq m) &= \frac{n-1}{n} \bar{s}_{n-1} + \frac{1}{n} s_n \\ \bar{s}_n (n > m) &= \frac{m-1}{m} \bar{s}_{n-1} + \frac{1}{m} s_n\end{aligned}$$

Dies bewirkt, dass die ersten Messwerte stärker gewichtet werden. Die Gewichtung (alter Wert/neuer Wert) verläuft jetzt wie folgt:

$$\begin{aligned}\bar{s}_1 &\quad \left(0/\frac{1}{1}\right) \\ \bar{s}_2 &\quad \left(\frac{1}{2}/\frac{1}{2}\right) \\ \bar{s}_3 &\quad \left(\frac{2}{3}/\frac{1}{3}\right) \\ \bar{s}_4 &\quad \left(\frac{3}{4}/\frac{1}{4}\right) \\ &\quad \vdots \\ \bar{s}_m &\quad \left(\frac{m-1}{m}/\frac{1}{m}\right)\end{aligned}$$

### 2.1.5 Berechnung der globalen Zeit

Haben die Slaves erst einmal Skew und Offset berechnet, lässt sich daraus globale Zeiten  $t_g$  zu einer lokalen Zeit  $t_l$  auf der Slave-Uhr bestimmen. Sei weiterhin  $\bar{o}$  der letzte Average Offset,  $\bar{t}_r$  das letzte Sync Point (Mittel der lokalen Empfangszeiten) und  $\bar{s}$  der letzte berechnete Skew, dann ist

$$t_g = \bar{o} + \bar{s} \cdot (t_l - \bar{t}_r) . \quad (1)$$

## 2.2 Parametrisierung

Für die beschriebenen Parameter müssen jetzt geeignete Werte gefunden werden. Hierbei gilt es zwischen verschiedenen Gesichtspunkten abzuwägen:

- **Distanz zwischen zwei Sync Points (Sync Interval)**

Große Synchronisationsintervalle erhöhen die Genauigkeit der Messung, kleine liefern dafür schnellere Ergebnisse, sind aber ungenauer und erhöhen die Netzlast, was wiederum zu kürzeren Batterielaufzeiten führt.

- **Anzahl der Sync Messages pro Block**

Eine hohe Anzahl an Synchronisationsnachrichten erhöht die Genauigkeit, aber auch die Netzlast. Eine niedrige Anzahl führt zu ungenaueren Messergebnissen.

- **Die Alterungskonstante  $m$**

Ein hoher Wert für  $m$  erhöht die Genauigkeit, allerdings reagiert das System dann träger auf Veränderungen des Drifts.

Die vorigen Messungen haben gezeigt, dass 25 Sync Messages pro Block einen guten Kompromiss zwischen Genauigkeit und niedriger Netzlast darstellen. Wir haben nun mit verschiedenen Synchronisationsintervallen experimentiert. Die Ergebnisse sind in den Abbildungen 13 bis 16 zu sehen. Die Alterungskonstante  $m$  haben wir auf 7 festgelegt. Zum Vergleich ist die Steigung der berechneten Ausgleichsgerade durch die gemittelten Offsetwerte eingezeichnet:

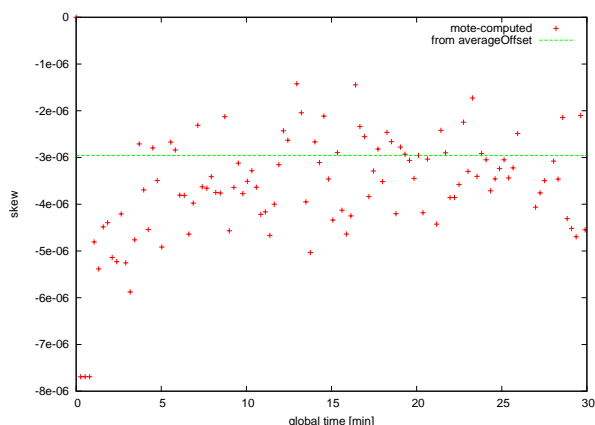


Abbildung 13: Sync Interval 10s

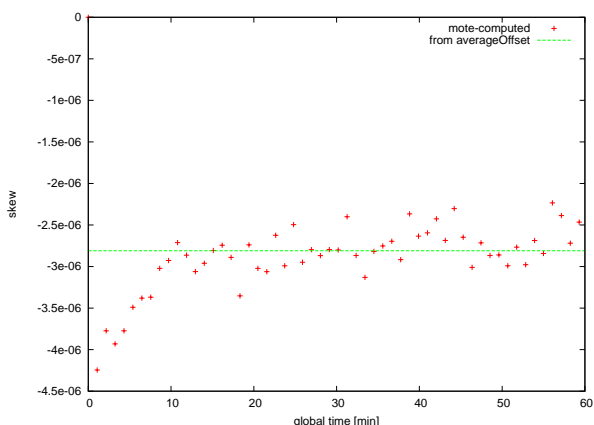


Abbildung 14: Sync interval 60s

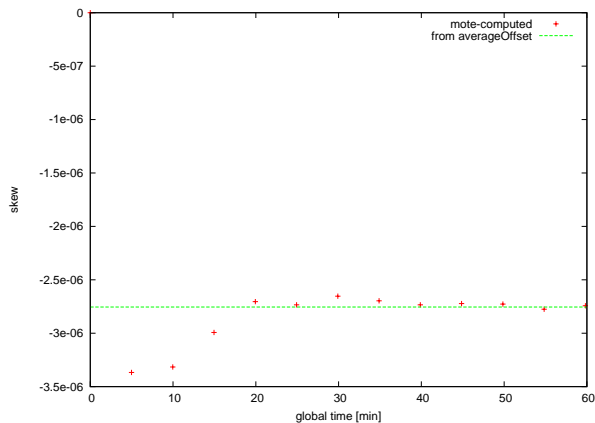


Abbildung 15: Sync Interval 5 min

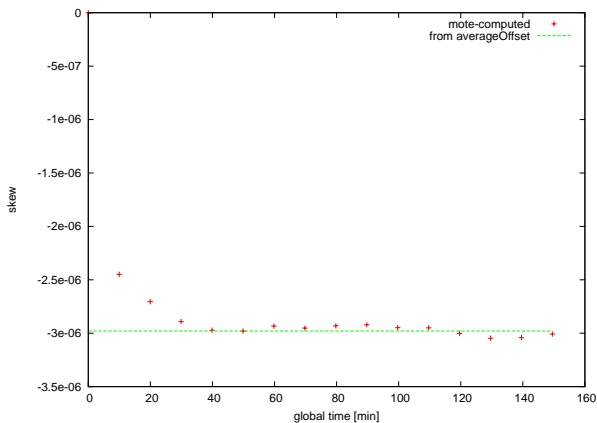


Abbildung 16: Sync Interval 10 min

**Ergebnis:** Ein Sync Interval von 5 min liefert schon nach etwa 20 min gute Ergebnisse (vgl. auch das Beispiel im folgenden Abschnitt 2.3). Das dürfte für die meisten Anwendungen eine gute Einstellung sein. Kurze Synchronisationsintervalle sind auf jeden Fall zu vermeiden!

## 2.3 Genauigkeit

Wir wollen nun die Genauigkeit des berechneten Skews zu bestimmen. Den Fehler der Offset- und Skewmessung können wir durch eine Fehlerrechnung eingrenzen. Die Steigung  $s_i$  zwischen zwei aufeinander folgender Sync Points wird durch eine Ausgleichsgerade bestimmt:

$$s_i = f(\bar{o}_i, \bar{o}_{i-1}, t_i, t_{i-1}) = \frac{\bar{o}_i - \bar{o}_{i-1}}{t_i - t_{i-1}}$$

Als fehlerbehaftet betrachten wir nur jeweils die beiden Offsetwerte  $\bar{o}_i$  und  $\bar{o}_{i-1}$ . Weiterhin

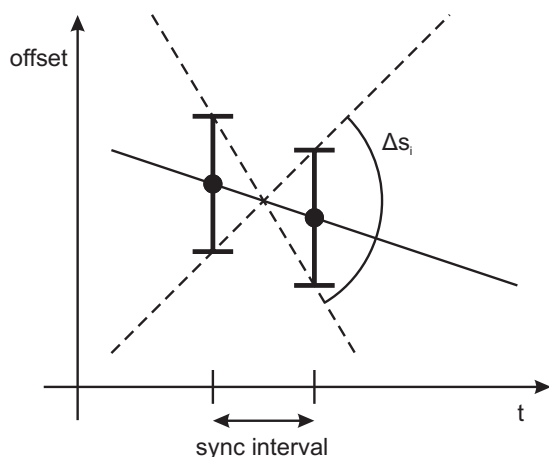


Abbildung 17: Großer Offset-Fehler, kleines Sync Interval

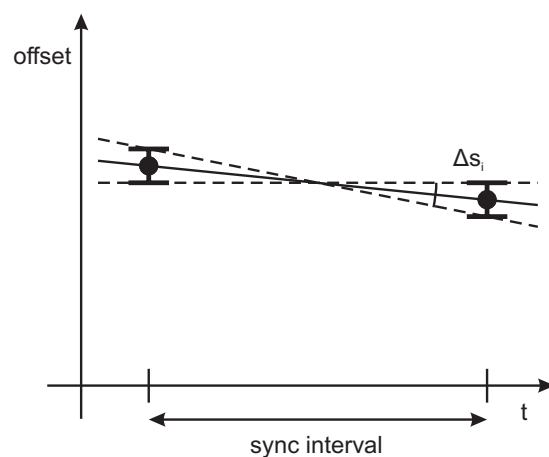


Abbildung 18: Kleiner Offset-Fehler, großes Sync Interval

ist  $t_i - t_{i-1}$  der zeitliche Abstand der Nachrichtenblöcke (Sync Interval). Der Fehler des Skews  $\Delta s_i$  lässt sich mittels Gausscher Fehlerfortpflanzung daraus ableiten:

$$\Delta s_i = \sqrt{\left(\frac{\partial f(\bar{o}_i, \bar{o}_{i-1}, t_i, t_{i-1})}{\partial \bar{o}_i} \Delta \bar{o}_i\right)^2 + \left(\frac{\partial f(\bar{o}_i, \bar{o}_{i-1}, t_i, t_{i-1})}{\partial \bar{o}_{i-1}} \Delta \bar{o}_{i-1}\right)^2} \quad (2)$$

Bei Annahme von gleichen Fehlern der Average Offsets  $\Delta\bar{o}_i = \Delta\bar{o}_{i-1} = \Delta\bar{o}$  ergibt sich aus (2):

$$\Delta s_i = \sqrt{2 \left( \frac{\Delta\bar{o}}{t_i - t_{i-1}} \right)^2} \quad (3)$$

Wir können für die Fehlerrechnung die Alterungsfunktion in guter Näherung als gleitenden Mittelwert betrachten. Der Fehler des berechneten Skews ist daher:

$$\Delta\bar{s} \approx \frac{\Delta s_i}{\sqrt{m}} \quad (4)$$

Nachdem nun die Fehler für den Average Offset und den Skew bekannt sind, können wir eine Abschätzung für die Genauigkeit der zu einer lokalen Zeit  $t_l$  berechneten globalen Zeit  $t_g$  machen. Die Fehlerfortpflanzung nach Gauss für die Formel (1) sieht dann wie folgt aus.

$$\Delta t_g = \sqrt{\Delta\bar{o}^2 + ((t_l - \bar{t}_r)\Delta\bar{s})^2} \quad (5)$$

**Beispiel:** Die Konfiguration unseres Verfahrens mit den Parametern 25 Sync Messages je Block (vgl. Abb. 7) zeigt eine Standardabweichung der Average Offsets  $\bar{o}_i$  von etwa 5 jiffies (siehe Abb. 8). Wird die Alterungskonstante  $m$  etwa auf 7 gesetzt und wählt man als Synchronisationsintervall 5 min ergibt sich ein Fehler  $\Delta\bar{s}$  von etwa  $2,7 \cdot 10^{-7}$ . Das entspricht einem relativen Fehler von etwa 10 %. Vergleiche hierzu auch Abb. 15.

Seien nun Offset und Skew mit dieser Genauigkeit zum Zeitpunkt  $\bar{t}_r$  bekannt, kann man das Versenden von Synchronisationsnachrichten einstellen. Die Abweichung der globalen Zeit nach 24 Stunden wäre nach Gleichung (5):  $\Delta t_g = 760 \text{ jiffies} \approx 23 \text{ ms}$ .

**Fazit:** Besonders kleine Synchronisationsintervalle können den Skew-Fehler so erhöhen, dass das Messverfahren unbrauchbar wird. Größere Offset-Fehler, die entstehen, wenn nur wenige Synchronisationsnachrichten verwendet werden, lassen sich durch ein langes Intervall teilweise kompensieren. Vergleiche hierzu auch Abb. 17 und 18. Unsere weiteren Messungen werden dieses Ergebnis bestätigen.

## 3 Anwendung

Wie lässt sich unser Synchronisationsverfahren in Applikationen anwenden, welche Dateien und Interfaces werden benötigt und wie werden diese konfiguriert?

### 3.1 Interfaces

Wir bieten folgende Interfaces:

- `TimeSyncControl.nc`
- `GlobalTime.nc`
- `TimeSyncInfo.nc`,

sowie zur Konfiguration die Header-Datei `TimeSyncConf.h`.

**Das Interface TimeSyncControl.nc**

```
interface TimeSyncControl {
    command result_t start(uint32_t syncInterval,
                          uint32_t innerSyncInterval);
    command result_t stop();
    command uint32_t getSyncInterval();
    command result_t setSyncInterval(uint32_t syncInterval);
    event result_t isSynced();
}
```

Dieses Interface stellt die grundlegenden Werkzeuge für die Zeitsynchronisation zur Verfügung. Neben dem Starten mit festem `innerSyncInterval`, das den Abstand zwischen zwei Synchronisationsnachrichten angibt und `syncInterval`, der Abstand zwischen den Synchronisationsblöcken, kann mit `getSyncInterval()` auf den Slaves die Startzeit des nächsten Blocks abgefragt werden. Auf dem Master kann mit `setSyncInterval(uint32_t)` der Abstand dieser Blöcke dynamisch, während der Laufzeit, noch verändert werden (alle Zeitparameter in ms).

**Das Interface GlobalTime.nc**

```
interface GlobalTime {
    command result_t getGlobalTime(uint32_t *timeRef);
    command result_t local2Global(uint32_t *timeRef);
    command result_t localTilGlobal(uint32_t globalTime, int32_t *timeRef);
}
```

Mit `getGlobalTime(uint32_t)` liefert dieses Interface zu einer beliebigen Zeit, die dazu gehörige globale Zeit. `local2Global(uint32_t)` rechnet die lokale Zeit in die globale um und `localTilGlobal(uint32_t, int32_t)` gibt den zeitlichen Abstand von einer lokalen Zeit zu einer gegebenen globalen an.

**Das Interface TimeSyncInfo.nc**

```
interface TimeSyncInfo {
    async command int32_t getOffset();
    async command float getSkew();
}
```

Ein einfaches Interface, um den grade aktuellen Offset bzw. Skew auszugeben.

**Das Header-File TimeSyncConf.h**

```
#define TSYNC_ROOT_ID
#define NUM_SAMPLES
#define CONV_RATIO
```

Neben dem Format der Zeitsynchronisationsnachrichten enthält die Header-Datei noch die Parameter, `TSYNC_ROOT_ID`, die ID des Masters, `NUM_SAMPLES`, die Anzahl der Zeitsynchronisationsnachrichten pro Synchronisationsblock und `CONV_RATIO` die Alterungskonstante  $m$ .

Das Modul `TmoteTimeSyncM.nc` implementiert diese Interfaces oder leitet sie an das darunter liegende `TmoteFTSP.nc` Modul weiter, das dann wiederum den größten Teil der Implementierung beinhaltet, unter anderem die Möglichkeit via Compiler-Schalter `TMOTE_MAC_TSYNC` zwischen den Modi PC und TMote zu variieren. PC verwendet hierbei das Modul `SimpleTime`, um die von der Mote-CPU bereitgestellte Zeit zu simulieren. Hier ein Beispiel(s.a. `TestTmoteTimeSync.nc`), wie das Wiring für unsere Synchronisation aussehen könnte:

### Beispiel Wiring `TestTmoteTimeSync.nc`

```
configuration TestTmoteTimeSync {}
implementation {
    components TestTmoteTimeSyncM, TmoteTimeSyncC, GenericComm,
               SimpleTime, Main, TmoteFTSPC, LedsC;

    Main.StdControl -> SimpleTime;
    Main.StdControl -> GenericComm;
    Main.StdControl -> TestTmoteTimeSyncM;
    TestTmoteTimeSyncM.Leds -> LedsC;
    TestTmoteTimeSyncM.SendMsg -> GenericComm.SendMsg[0x17];
    TestTmoteTimeSyncM.TimeSyncInfo -> TmoteFTSPC;
    TestTmoteTimeSyncM.TimeSyncControl -> TmoteTimeSyncC.TimeSyncControl;
    TestTmoteTimeSyncM.GlobalTime -> TmoteTimeSyncC.GlobalTime;
}
```

## Literatur

- [IEEE03] 802.15.4-2003 IEEE Standard for Information Technology-Part 15.4: *Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Low Rate Wireless Personal Area Networks (LR-WPANS)*, 2003
- [CJM05] Cox, D., Jovanov, E., Milenkovic, A.: *Time Synchronization for ZigBee Networks*, Proc. of the 37th IEEE Southeastern Symposium on System Theory (SS-ST'05), Tuskegee, 2005
- [EGE02] Elson, J., Girod, L., Estrin, D.: *Fine-Grained Network Time Synchronization using Reference Broadcasts*, Proc. of the 5th Symposium on Operating Systems Design and Implementation, Boston, 2002
- [ER02] Elson, J., Römer, K.: *Wireless Sensor Networks: A New Regime for Time Synchronization*, Proc. of the First Workshop on Hot Topics in Networks (HotNets-I), Princeton, New Jersey, USA, 28-29 October 2002
- [MKSL04] Maroti, M., Kusy, B., Simon, G. and Ledeczi, A.: *The Flooding Time Synchronization Protocol*, Proc. of the 2nd International Conference on Embedded Network Sensor Systems (SenSys '04), 2004
- [Röm05] Römer, K.: *Time Synchronisation and Localisation in Sensor Networks*, PhD Thesis, ETH Zurich, 2005
- [tmote] tmoteSky platform: <http://www.moteiv.com>
- [Van] Vanderbilt's implementation of the FTSP in TinyOS:  
<http://cvs.sourceforge.net/viewcvs.py/tinyos/minitasks/02/vu/tos/lib/TimeSync/>