Precise Time Synchronization for Wireless Sensor Networks using the Global Positioning System

Semester Thesis

Michael Meier
meiermic@ee.ethz.ch

Advisor:
Philipp Sommer

Supervisor:
Prof. Dr. Roger Wattenhofer

Distributed Computing Group
Computer Engineering and Networks Laboratory (TIK)
Department of Information Technology and Electrical Engineering

January 2010

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing Group

Abstract

For sensor networks a wide variety of time synchronization protocols exist. These protocols establish a common logical clock between participating nodes. This logical clock has no relation with wall clock time.
While a synchronized logical clock enables many interesting applications, such as more energy efficient MAC protocols, it does not cater much to the needs of data gathering, the main application of sensor networks. Sensor data is often useless if the time of the measurement is unknown. Time synchronization protocols in conjuction with a method of converting logical clock values to wall clock time, such as UTC, would enable one to order events in applications where sensor data is sampled from multiple, independent sensor networks.
In this thesis a method is proposed to establish a relationship between the logical clock provided by time synchronization protocols and wall clock time with the aid of the Global Positioning System. Furthermore the proposed approach is, implemented on a small network of sensor nodes with the Zigbit900 chip using TinyOS.

Acknowledgements

I would like to thank Prof. Dr. Roger Wattenhofer for offering me the opportunity to write this semester thesis at the Distributed Computing Group at ETH Zürich.
I would also like to thank my advisor, Philipp Sommer. He was always quick to offer his invaluable support. Without his help this semester thesis would not have been possible.

Contents

1 Introduction **9**
- 1.1 Wireless Sensor Networks . . . 9
- 1.2 Time Synchronization Protocols . . . 9
- 1.3 The Global Positioning System . . . 10
- 1.4 nesC . . . 10
- 1.5 TinyOS . . . 10
- 1.6 Goal . . . 11

2 Design and Implementation **13**
- 2.1 System Overview . . . 13
- 2.2 Hardware . . . 14
  - 2.2.1 Pixie Node . . . 14
  - 2.2.2 Zigbit 900 . . . 14
  - 2.2.3 ATMega1281 . . . 15
  - 2.2.4 AT86RF212 . . . 15
  - 2.2.5 Pixie Base Station . . . 15
  - 2.2.6 LEA-5H . . . 15
- 2.3 Time Synchronization Protocols . . . 15
  - 2.3.1 The Radio Clock . . . 16
  - 2.3.2 Modifications in TinyOS . . . 16
- 2.4 GPS Module . . . 17
- 2.5 Synchronization Pairs . . . 17
- 2.6 Regression . . . 18

3 Evaluation **21**
- 3.1 Time Synchronization Protocols . . . 21
- 3.2 Demo Application . . . 22
  - 3.2.1 Accelerometer . . . 22
  - 3.2.2 Experiment . . . 24

4 Conclusion and Further Work **25**
- 4.1 Conclusion . . . 25
- 4.2 Further Work . . . 25

Bibliography **27**

1 Introduction

1.1 Wireless Sensor Networks

A wireless sensor network, or WSN for short, consists of a number of sensor nodes which can communicate wirelessly. Sensor nodes usually comprise a microcontroller, a low-power radio transceiver, sensors, and a source of energy. Sensor nodes are programmed to monitor a given set of environmental conditions, such as light intensity, temperature or vibrations. They may be programmed to form an ad-hoc multiple hop network.
Wireless sensor networks are different from conventional sensor installations in many aspects. To be cost efficient, WSNs have to survive on a very limited power supply for a long time, e.g. on a pair of batteries for three years. WSNs are often installed in very harsh environmental conditions, such as in high alpine environments[1] or on volcanoes[2], forcing them to run unattended for most of the time. Due to the unreliability of equipment in general and the uneconomicality of carrying out maintenance operations when the sensors are already deployed, WSNs have to cope with node failure dynamically.

1.2 Time Synchronization Protocols

For many applications of wireless sensor networks, such as data gathering, it is of utmost importance to know when data was sampled or when a given event happened. Therefore, sensor nodes need to have a common understanding of time. Synchronized clocks can also help in other applications such as the development of more energy efficient MAC protocols. For wireless sensor networks there exists a range of time synchronization protocols such as RBS[3], TPSN[4], FTSP[5] or GTSP[6]. They strive to establish a two way relationship between the local time of a node and a “global” sensor network time. When used in conjunction with a 1Mhz clock they are able to deliver pairwise synchronization errors in the range of $5\mu s$.

1.3 The Global Positioning System

The Global Positioning System, short GPS, is a satellite navigation system operated by the United States of America[1].
To provide positioning information to receivers on or near the earth's surface, about 30 satellites are placed in earth orbit. GPS satellites are equipped with an atomic clock which is regularly synchronized with the clock of all other GPS satellites to within a few nanoseconds. GPS satellites continuously broadcast their time and position. Using the signals from four satellites, a GPS receiver can determine its position and its time. If GPS receivers were equipped with clocks precisely synchronized to the GPS satellites' clocks, only three signals would be needed. Using four signals to determine the position also yields information about time. GPS receivers are able to calculcate UTC with an error of some hundred nanoseconds or even less.

1.4 nesC

The nesC programming language[7] developed at University of Californica, Berkeley, is an extension of the well known C programming language.
nesC forces the programmer to implement functionality in components. Components provide the implemented functionality to other components via interfaces. To offer their services components may also use, or require, interfaces offered by other components. Interfaces are bidirectional. They specify a set of functions, so called commands, to be implemented by the interface provider and another set of functions, so called events, to be implemented by the interface user. This allows for a concise representation of event driven programs. As an illustration an interface user may call a "send packet" method offered by the interface provider, a radio driver. This call is non-blocking. As soon as the send is done the interface provides signals, or calls, the corresponding "send done" event. Typically commands call downwards, i.e. from high level application towards hardware drivers while events call in the other direction. Programs are built by wiring together interface providers and interface users.
In nesC concurrency is a first class citizen. The nesC execution model uses only a single stack, therefore concurrency is managed in an event driven fashion. The nesC language offers language constructs to call commands, signal events and to defer some task to be run at a later point in time.
Components are statically linked to each other. This allows a compiler to generate very efficient code.

1.5 TinyOS

TinyOS[8] is an open source operating system specifically designed for the requirements of wireless sensor networks. It is programmed in the nesC programming language. TinyOS offers a very modular driver architecture and has been ported to over a dozen platforms.
In the wireless sensor network community, TinyOS is a widely used standard for the reference implementation of proposed new algorithms.

[1] `http://www.gps.gov/`

1.6 Goal

The goal and contribution of this thesis is to present a simple and cheap method of synchronizing wall clock time and a logical sensor network time provided by time synchronization protocols such as FTSP or GTSP.
Said method is presented in Section 2.1, while Sections 2.2 - 2.6 deal with implementation issues on the Pixie base station and on Pixie sensor nodes. Chapter 3 evaluates the chosen approach. A conclusion is reached in chapter 4.

2
Design and Implementation

2.1 System Overview

The approach presented in this thesis is intended to be simple and low in cost. Furthermore the system is designed in a minimally intrusive way, such that precise time synchronization between sensor network clock and wall clock time can be added to existing networks.

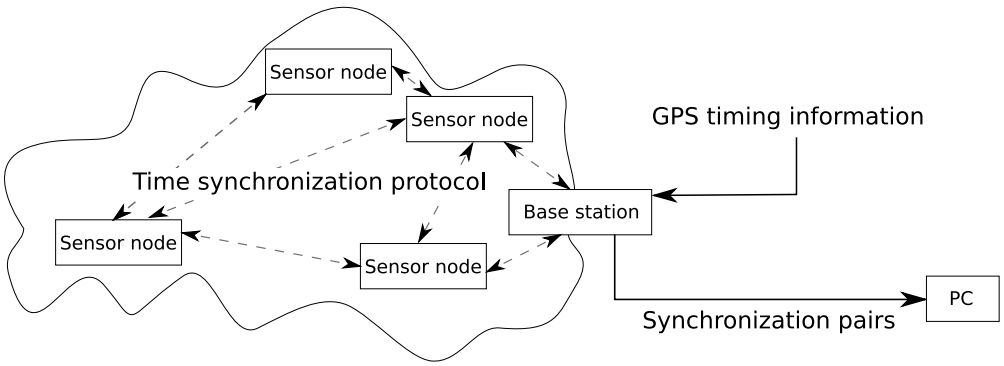

Figure 2.1: High level system view

A high level overview of the proposed system can be found in Figure 2.1. The system consists of a wireless sensor network, formed by wireless sensor nodes, a base station, and a PC attached to the base station. The sensor nodes and the base station run among themselves a time synchronization protocol such as FTSP[5] or GTSP[6]. The base station is equipped with a GPS receiver from which it acquires precise wall clock time information. Although every node could be equipped with a GPS receiver, a decision has been made to design an application with only one GPS receiver. This is due to three reasons:

1. GPS receivers are prohibitively expensive to be put in every sensor node.

2. GPS receivers consume a lot of energy. A typical GPS receiver can be expected to require about $130mW$[9] whereas a typical radio transceiver consumes about $60mW$ when active. To mitigate this problem radio transceivers are usually turned off for most of the time. This is either not or only hardly possible with GPS receivers as they have to observe GPS signals for about 3 to 30 seconds in order to get a good estimate of position and time. Because one wants to synchronize sensor network and wall clock time pretty often, say every 10 seconds, this means that the GPS receiver has to be turned on all the time.

3. Not all sensor nodes may be able to obtain a sufficient GPS signal, e.g. they may be placed indoors or in other places with poor or no GPS reception.

The base station periodically acquires wall clock time t_{WT} from the GPS receiver. Whenever t_{WT} is measured the base station also takes a measurement of the corresponding synchronized sensor network time, called t_{GT}. Together they form the tuple (t_{WT}, t_{GT}), which is called a synchronization pair.

Whenever such a synchronization pair is generated, it is emitted to the PC connected to the base station. A program on the PC records the synchronization pairs. With the knowledge of two or more synchronization pairs, software will be able to convert any sensor network time to its corresponding wall clock time. This process may, but does not have to happen in real time. Note that the PC might be replaced by any kind of computer. The processing could be handled by much less powerful systems, even on the base station itself. However, due to the ease of programming on a PC with very high level languages and the availability of robust and comprehensive date and time handling libraries, conversion from sensor network to wall clock time is not handled on the base station. This is a reasonable decision, given that many sensor network deployments feature a quite powerful computer, e.g. a Laptop or a Gumstix module[10], connected to their base station.
Given this system architecture, sensor network timestamps associated with measured data may be converted to wall clock timestamps.

2.2 Hardware

2.2.1 Pixie Node

The Pixie sensor node can be considered a prototyping platform for wireless sensor networks. It consists of a Meshnetics Zigbit 900 module[11], an onboard antenna, three LEDs and connectors wired to every pin of the Zigbit 900. The Pixie sensor node does not contain any sensor. For further information on the Zigbit900 module, see Section 2.2.2.

2.2.2 Zigbit 900

The Zigbit 900 module[11] sold by MeshNetics consists of an Atmel ATMega1281 microcontroller, an Atmel AT86RF212 Transceiver and all the necessary passive components. For further information on the microcontroller and the transceiver see Sections 2.2.3 and 2.2.4 respecively. The internal wiring of the Zigbit 900is a bit unfortunate. The AT86RF212 features an interrupt request line indicating the arrival of a new packet. This line is connected to an interrupt pin of the ATMega1281. This means that timestamping of incoming packets must be done in software, e.g. the interrupt handler invoked due to reception of packet has to sample the value of a timer register as fast as possible. Any atomic sections in different code running on the node this result in

jitter. This forms a contrast to platforms such as the MICA2 mote[12], where this timestamping is done in hardware and thus very robust to long atomic sections.

2.2.3 ATMega1281

The Atmel ATMega1281 microcontroller[13] is an 8 bit low power microcontroller part of Atmel's well known AVR series. It features 128 KByte of program memory, 8 KByte of RAM, 6 hardware timers, multiple peripheral interfaces such as an UART, SPI and I^2C bus. Some hardware timers also include capture capability which makes the ATMega1281 well suited for applications with very precise timing requirements.

2.2.4 AT86RF212

The Atmel AT86RF212[14] is a low power transceiver implementing the IEEE 802.15.4 standard. The AT86RF212 can be connected to a microcontroller via an SPI interface and a low number of digital I/O lines. The AT86RF212 is very similar to the AT86RF230[15] which is extensively used on other wireless sensor platforms such as the IRIS mote[16].

2.2.5 Pixie Base Station

The Pixie base station[17] can be considered superset of the Pixie node presented in Section 2.2.1. The Pixie base station is equipped with additional hardware such as a digital temperature sensor, a Flash memory chip for extended long term storage and 4 LEDs. The UART of the Zigbit 900 module is connected to a Digi Connect ME [18] housed in an enlarged Ethernet jack. The Digi Connect ME is configured as a bride between the Zigbit's UART and a TCP/IP network. It is possible to observe the whole serial output of the Zigbit module via a TCP connection established with the DigiConnectME module.
The Pixie base station also features a u-blox LEA-5H GPS receiver chip[9]. The GPS receiver is connected to the microcontroller via an I^2C bus interface. The reset pin of the LEA-5H chip can be controlled by the microcontroller. The time pulse line of the GPS receiver is connected to a capture input pin of the ATMega1281 so as to enable precise timestamping. For further information on the LEA-5H module, see section 2.2.6.
An overview of the hardware configuration of the Pixie Base station can be obtained from Figure 2.2.5.

2.2.6 LEA-5H

The LEA-5H GPS module [9] is a GPS receiver manufactured by u-blox. One can interface to this module via SPI, I^2C, RS-232 or USB. The LEA-5H GPS module can be configured to generate a pulse on a digital I/O line, e.g. to generate a $100ms$ long pulse at the start of each UTC [1] second.

2.3 Time Synchronization Protocols

A considerable amount of effort went into enabling the two time synchronization protocols FTSP and GTSP on the Pixie and the Pixie base station platforms. Those protocols worked out of the box with millisecond resolution, however, for high precision it has been decided to run time synchronization with microsecond resolution.

[1] Universal Coordinated time. See `http://en.wikipedia.org/wiki/UTC`

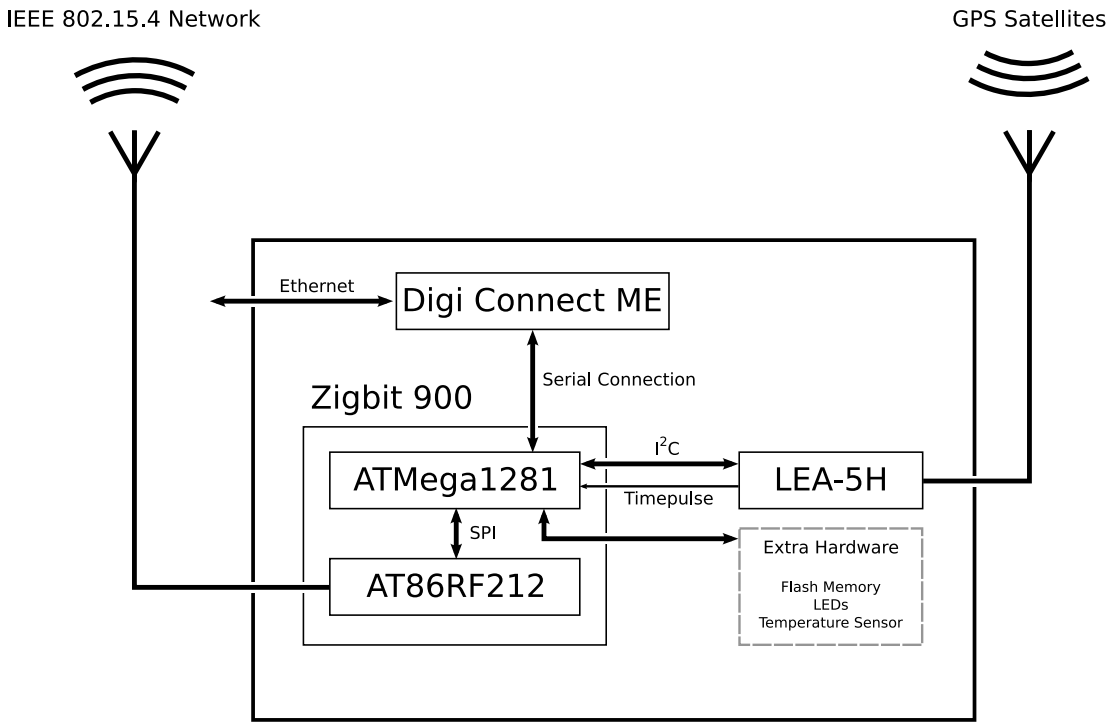

Figure 2.2: A block diagram of the Pixie base station.

2.3.1 The Radio Clock

The AT86RF212 radio module, which is part of the Zigbit module, features a 1Mhz clock output. Timing experiments conducted with the time pulse feature of the GPS receiver proved the radio clock to be very stable. The ATMega1281 offers an internal 8MHz RC oscillator, which may be divided by eight to produce a 1Mhz clock. Due to its stability the radio clock has been preferred to the internal oscillator. As the Pixie node and the Pixie base station feature no internal connection between the radio clock output and an input of one of the four timers of the ATMega1281, an external one had to be soldered. See Figure 2.3 for a picture of such a connection on the Pixie node. The radio clock is connected to the input of Timer 3 of the ATMega1281.

2.3.2 Modifications in TinyOS

Timers To enable microsecond timing in the time synchronization protocols, the `LocalTimeMicroC` component of the `meshbean` platform was rewired from Timer 1 to Timer 3. Additionally a new file *tos/platforms/pixie/InitThreeP.nc* has been created, which overrides *tos/platforms/mica/InitThreeP.nc*. If the macro `PIXIE_RADIO_CLOCK` is defined, the new `InitThreeP` component configures Timer 3 to use an external clock source, thus making Timer 3 a 1MHz clock driven by the radio module.

AT86RF212 driver In *tos/chips/rf2xx/rf212/RF212DriverLayerP.nc* the constant `RX_SFD_DELAY` was changed from $9\mu s$ to $29\mu s$. This constant indicates how much time lapses between the arrival of the SFD (the *Start Frame Delimiter*, a method of framing used by IEEE 802.15.4 transceivers) at the antenna and the generation of the corresponding interrupt

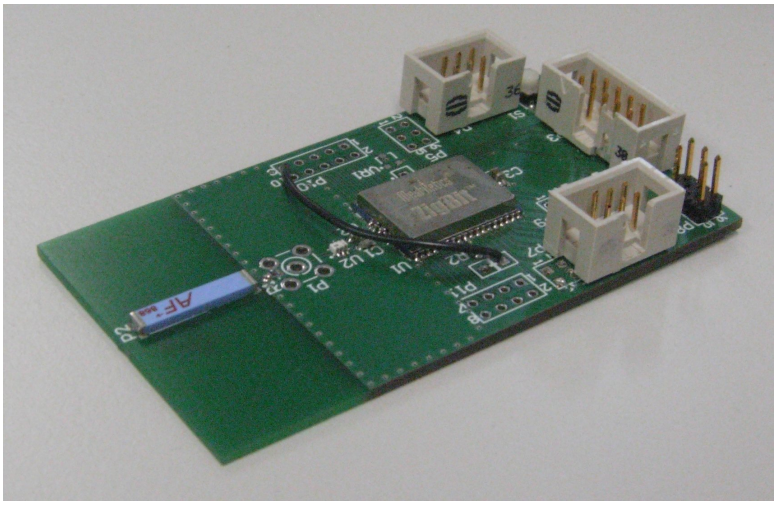

Figure 2.3: A pixie sensor node. The black wire soldered to the node is the connection between the radio clock and the input of Timer 3.

by the AT86RF212 module. The procedure used to find the new value for `RX_SFD_DELAY` is detailed in Section 3.1.

2.4 GPS Module

A new interface `SetTimePulse` was added to the driver for the LEA-5H GPS module. The interface features only one command, `set`, which enables one to configure how often the LEA-5H should generate a time pulse. For this purpose a new file *tos/platforms/pixiebase/chips/lea5x/LEA5x.P* has been created. It replaces the hitherto existing *tos/platforms/pixiebase/chips/lea5x/LEA5xReaderP.nc* and incorporates both the already existing and the new time pulse configuration functionality.
Prior work has shown that the combination of the LEA-5H GPS receiver interfaced via the I^2C bus and TinyOS works only flakily. This is in accordance with observations made during the work leading up to this thesis. Afer some time, usually in the range of about two minutes, the LEA-5H module would stop working and both the ATMega1281 and the LEA-5H had to be reset. According to extensive analysis of the problem, this is due to two reasons: the TinyOS I^2C driver for the ATMega1281 is broken and the protocol specification issued by u-blox[19] is unspecific and incomplete. u-blox was not willing to share any additional insight on the working of their protocol. To mitigate this unstable behaviour, two changes have been introduced to the driver. First, whenever a command is written to the GPS receiver, the driver waits for a fixed amount of time, in the range of 100 milliseconds, before reading the answer to said command. Second, the driver now offers a reset interface allowing to reset the GPS receiver.
These changes can not be considered as a solution, they are merely a work-around. They, however, increased the usable running time of the system from mere minutes to multiple hours.

2.5 Synchronization Pairs

To generate synchronization pairs according to Section 2.1 a new component `UTCCorrelateC` has been introduced. It offers the interface `UTCCorrelate` which allows an application to

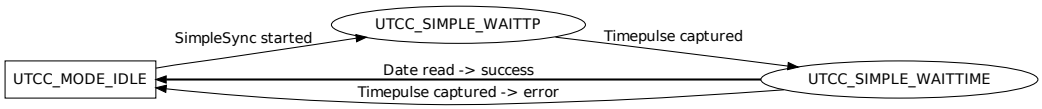

Figure 2.4: State machine for SimpleSync

trigger the generation of synchronization pairs by invoking the command `startSimpleSync`. Generation of a synchronization pair is signalled by the event `syncDone`. `UTCCorrelateC` is implemented according to the state machine shown in Figure 2.4. To enable precise timing, the time pulse line from the LEA-5H module is connected to the capture input of Timer 3, the Timer used by the time synchronization protocol. This means that whenever a time pulse occurs, the current value of Timer 3 is saved in a register and an interrupt is generated. This is all done in hardware. As long as the capture interrupt gets serviced in less than 2^{16} microseconds, or about 65 milliseconds, even long atomic sections have no influence on the precision of the generated synchronization pairs.
After generation of a synchronization pair is initiated, `UTCCorrelateP` waits for the next time pulse to happen. As soon as said time pulse arrives, `UTCCorrelateP` initiates a date and time read out from the LEA-5H module. Additionally the capture value is saved aside. The capture value is a timestamp of the timepulse expressed in local time. In case the next time pulse happens before the date read is done, or the read returned an error, the process is considered to be failed. If the read is successful the saved local timestamp is converted to a global timestamp with the `TimeSync.convertToGlobal` command. This global timestamp together with the date and time read from the LEA-5H is printed via the UART. An example of such a line follows:

```
syncpair 2009/12/2/13/58/13 692d5911
```

This line means that wall clock time 2:58:13 PM UTC on December 2, 2009 corresponds to sensor network time $692d5911_{16}$. Sensor network time is expressed as a 32 bit unsigned integer, it is printed in hexadecimal notation.

2.6 Regression

All analysis of synchronization pairs is done on a PC in programs written for the *Python*[20] programming language.
Conversion between sensor network and wall clock time is handled by class `UTCSync` in *globaltoutc/readsyncpairs.py*. An instance of this class may be fed with synchronization pairs by calling its `addsyncline` method. `addsyncline` parses the line and converts the wall clock time given in the syncpair to a UNIX timestamp. The sensor network time given in the syncpair needs a little more processing. As sensor network time is expressed using a 32 bit unsigned integer, time will wrap around after 2^{32} clock ticks. Assuming microsecond precision this means a wrap around will take place after about one hour. Clearly, this is too early. Therefore `UTCSync` detects wrap around conditions and maintains a counter to keep track of how many wrap arounds have already taken place. Given a 32 bit sensor network time t_{GT} and a wrap around counter $globalmsb$ one may calculate the extended sensor network time t_{eGT} as follows:

$$t_{eGT} = 2^{32} * globalmsb + t_{GT}$$

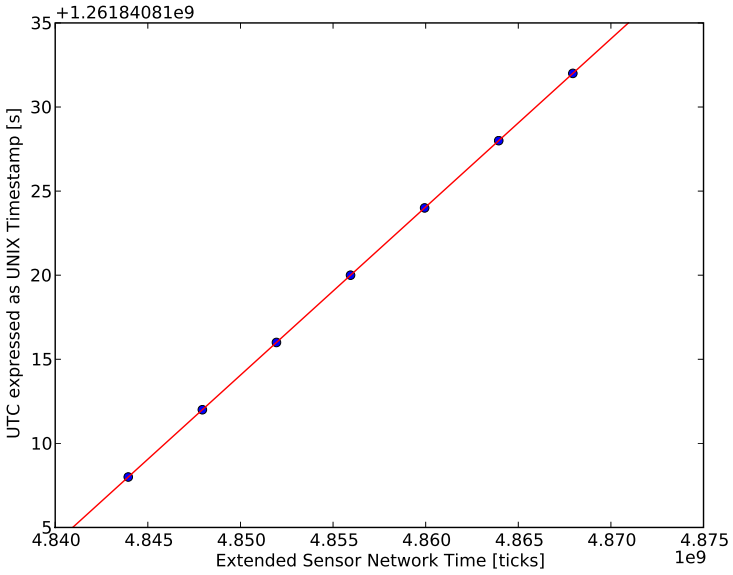

Figure 2.5: Example of a linear regression between extended sensor network time and UTC. The points represent synchronization pairs.

Should this feature work properly, not more than $2^{32} - 1$ clock ticks must pass between two synchronization pairs. This is a reasonable assumption given that one prefers to have fairly short intervals, such as some seconds, between to synchronization pairs due to clock drift. The tuple consisting of the extended sensor network time and UNIX time is appended to a regression table. As soon as there is more than one entry in the linear regression table a linear regression is calculated. In this linear regression sensor network time is placed on the x-Axis and UTC, represented with UNIX timestamps, is placed on the y-Axis, see Figure 2.6. Linear regression yields two values, namely the $slope$ and $intercept$ of the fitted line. Given a sensor network time t_{GT} the corresponding UTC t_{WT}, expressed as a UNIX timestamp can be calculated as follows:

$$t_{WT} = t_{GT} * slope + intercept$$

Class `UTCSync` offers a method `convertrecent` which does exactly the abovementioned conversion. It has to be noted that this conversion is only valid for events that happened recently with respect to the last synchronization pair.

3 Evaluation

3.1 Time Synchronization Protocols

To evaluate the changes presented in Chapter 2 a test setup with three Pixie nodes has been created. A graphical illustration of the setup may be found in Figure 3.1.

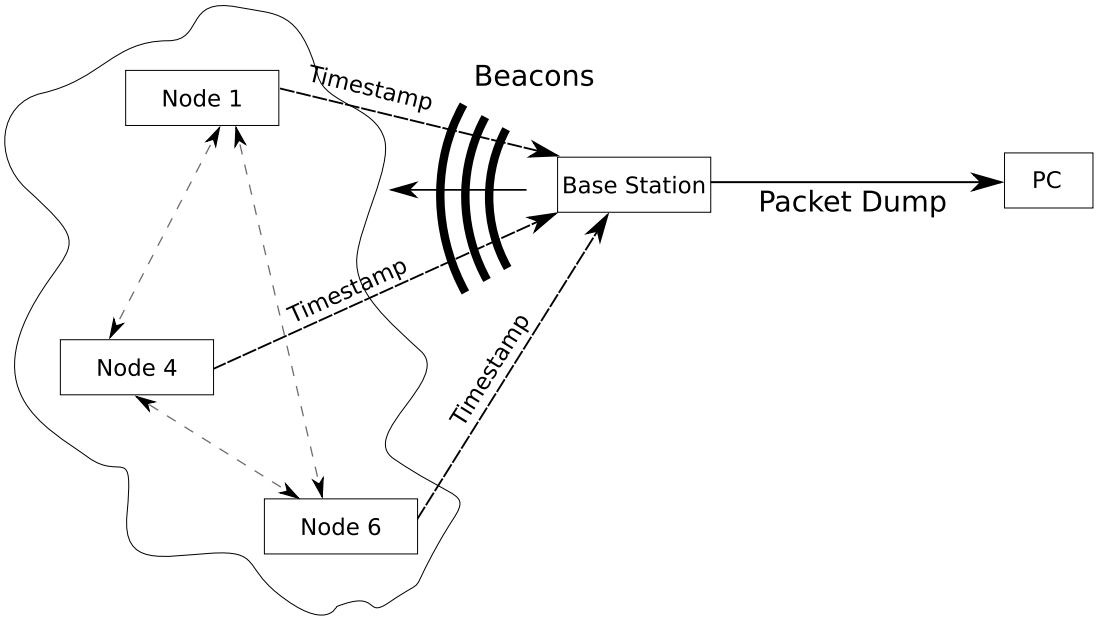

Figure 3.1: Setup to determine the quality of time synchronization. Dashed grey lines indicate activity of time synchronization protocols, dashed black line indicates transmission of event timestamp back to the base station.

The three Pixie nodes were assigned the node IDs 1, 4 or 6 respectively. The experiment setup also consists of a fourth node, the base station, which was a Meshbean900 evaluation kit

node[21]. The base station was flashed with TinyOS' `BaseStation` application and its UART was connected to a PC. Using the program *globaltoutc/beacon.py* beacon packets were injected into the network. All packets observed in the network were dumped to the PC and written to a file. The three nodes were placed few centimetres away from each other. Therefore a radio signal sent by the base station arrives at all the nodes at approximately the same time. The difference between reception times of any two nodes is well under $1ns$ while timestamping works at 1Mhz. Whenever a participating node receives a packet it takes a timestamp of the arrival time of said packet and then waits for a backoff time depending on its node ID. This is a measure taken to avoid congestion which would arise if all the nodes would try to send their reports at the same time. After waiting for their respective backoff time, the nodes report the measured timestamp to the base station. Beacons were injected every four seconds.
At first, only operation of FTSP was tried to be ovserved. FTSP did not behave as expected but instead showed random offsets and fluctuations. This problem could be solved by installing version 1.6.7 of the AVR C library[22] in the place of version 1.4.7, which is part of the TinyOS toolchain. Apparently the floating point arithmetic implementation of version 1.4.7 is buggy.
After the abovementioned floating point issue was resolved, FTSP showed a constant offset of $20\mu s$ between a node and its parent node in the FTSP tree. Therefore, the constant `RX_SFD_DELAY` was adjusted from $9\mu s$ to $29\mu s$. After this change, FTSP was able to achieve very precise time synchronization, e.g. with time differences between nodes in the order of 3 to $5\mu s$.
GTSP worked "out of the box". Its precision is very similar to the one offered by FTSP, see Figure 3.1.
The three nodes employed in the experiment were under very light load, they were just programmed to respond to beacon packets. Under these conditions software timestamping as employed by nodes with a Zigbit 900 proved to be sufficient. No experiments were conducted to determine the behaviour of software timestamping under heavy load. It is however expected that performance will degrade rapidly.

3.2 Demo Application

To demonstrate the working of the approach proposed in this thesis, a small demonstration application has been developed. An application setup consists of one or more Pixie nodes equipped with accelerometers, a base station capturing all packets sent in the network and a PC. Nodes are able to detect acceleration events, e.g. knocking on a table, with the accelerometer. When nodes detect an event they take a timestamp and report this timestamp via the radio. Software on a PC reads these timestamps and converts them to UTC, thus indicating at what wall clock time a given event has taken place. The code used to program the sensor nodes can be found in *src/Respond/src/*, the code used on the PC can be found in the *globaltoutc* directory.

3.2.1 Accelerometer

The Pixie nodes have been equipped with a LIS3LV02DQ[23] accelerometer manufactured by STMicroelectronics, see Figure 3.2.1. The LIS3LV02DQ is a digital 3-axis accelerometer and can be interfaced to a microcontroller via SPI or I^2C bus and features a configurable interrupt mechanism. The accelerometer may be configured to generate an interrupt when a certain set of conditions, e.g. acceleration above or below a given threshold, is met.
A driver using the I^2C interface was programmed but showed to be unreliable. When writing packets longer than 2 bytes the I^2C implementation provided by TinyOS for the ATMega1281

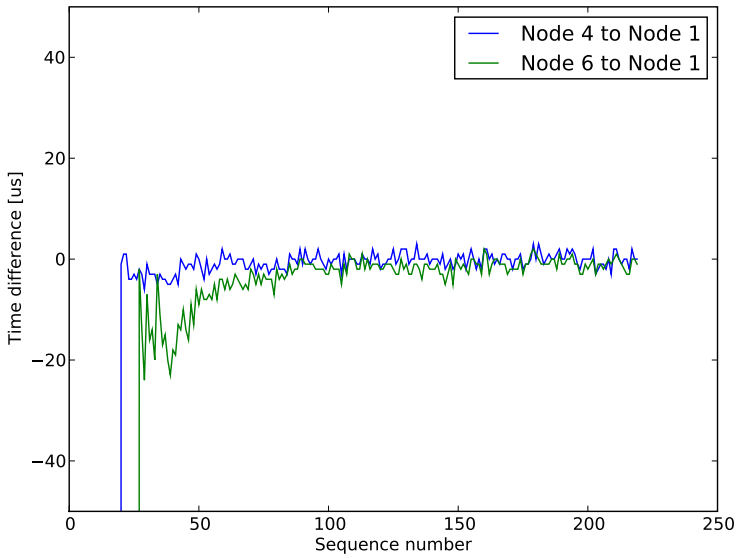

Figure 3.2: Example of GTSP time differences. At the start of the experiment it can be observed how a node running GTSP adapts to the clock of its neighbors.

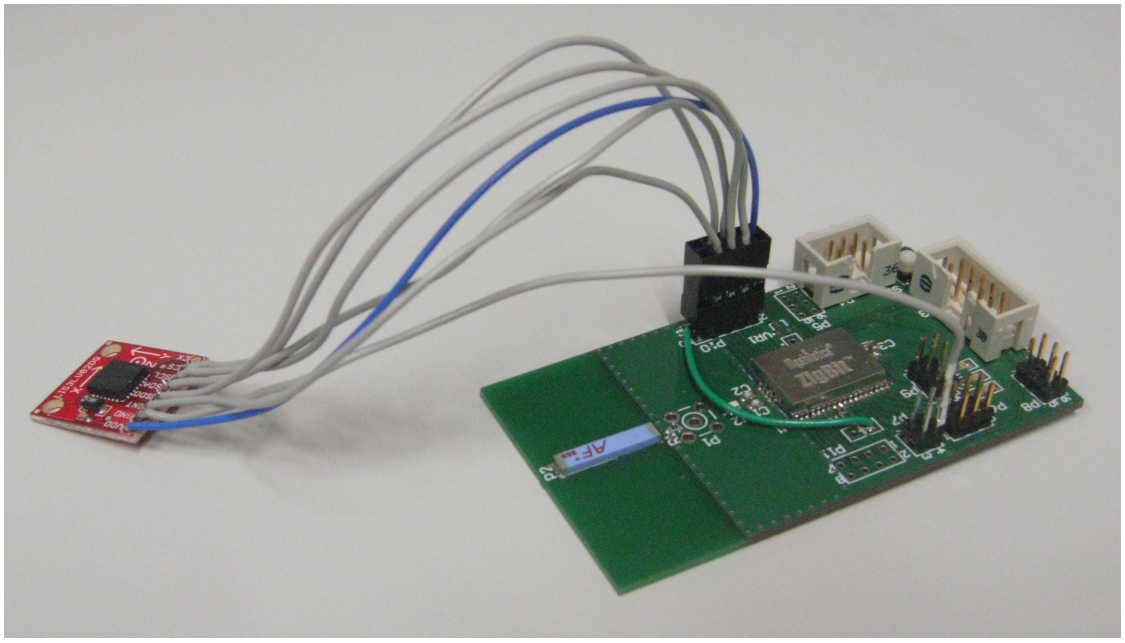

Figure 3.3: A Pixie sensor node equipped with a LIS3LV02DQ accelerometer connected via the SPI bus. The extra wire in the lower right corner is attached to Ground.

sometimes introduced bogus data into the stream. Therefore, it was decided to re-use the already existing driver for the LIS3L02DQ[24] accelerometer, an ancestor of the LIS3LV02DQ. This driver uses the SPI bus to communicate with the accelerometer. To adapt the driver to the new accelerometer, some addresses of registers had to be changed. Also, the driver was augmented with new functionality to allow limited configuration of interrupt sources.

3.2.2 Experiment

To verify the demo application an experiment with two Pixie nodes and a Pixie base station has been conducted. The Pixie base station was used to generate synchronization pairs. The two Pixie nodes, equipped with accelerometers, were placed on the opposite ends of a table, the distance between the two nodes being 2.5 metres. Accelerometer events were triggered by knocking on the table. The network traffic was monitored with a Meshbean900 node. Before any knocking events were generated it was made sure that the participating nodes were correctly synchronized.
The synchronization pairs and the network dump from the experiment were used as input for the program *globaltoutc/knocktimes.py*. This program reads events from the network dump and converts their timestamp from sensor network time to UTC. Some sample output is shown below:

```
event 0x0002 on node 1 happened at 2009/12/26 15:22:00.740895
event 0x0002 on node 4 happened at 2009/12/26 15:22:00.740670
event 0x0003 on node 1 happened at 2009/12/26 15:22:02.089241
event 0x0003 on node 4 happened at 2009/12/26 15:22:02.091243
```

In the above sample the two nodes, numbered 1 and 4, recorded 2 pairs of events. The time difference between the registration of the same knock is in the order of milliseconds or below.

4 Conclusion and Further Work

4.1 Conclusion

The goal of this semester thesis is reached. An approach to achieve synchronization between sensor network time provided by time synchronization protocols and wall clock time was proposed. Said approach was implemented on sensor nodes equipped with a Zigbit900 module and a base station. The implementation is simple, low in cost and easy to integrate into already existing sensor network deployments. Sensor nodes do not need to be fitted with any additional hardware, it suffices for them to run a standard time synchronization protocol. Only one sensor node, the base station, is equipped with a costly and energy hungry GPS receiver. The conversion between sensor network time and UTC is handled on a PC, thus enabling easy programming and guaranteeing access to robust date and time handling libraries.
The implementation was verified using two Pixie sensor nodes equipped with accelerometers. The sensor nodes registered acceleration events and reported them to a base station. A PC connected to the base station then correlated the sensor network timestamps sent by the Pixie nodes with timestamps in UTC.
Future applications can now correlate timestamps registered in sensor networks with wall clock time with high precision. Knowing when data was sampled or when a given event took place greatly enhaces the value of data recorded in wireless sensor networks. Furthermore, even measurements made in different, independent sensor networks can now be correlated.

4.2 Further Work

Although a basic and working implementation was presented in this thesis, many issues remain to be solved:

- The TinyOS I^2C driver for the ATMega1281 is very unreliable. Especially long writes lead to spurious behaviour. In order for the LEA-5H driver to operate correctly, the issues with the I^2C driver have to be resolved.
- In order for the issues with the LEA-5H GPS receiver to be solved, the LEA-5H protocol needs to be better understood. The current implementation is based on an incomplete

understaning of the LEA-5H protocol.

- The routines converting sensor network time to UTC are very simple. They could be made robust by taking into account more information, such as the arrival times of synchronization pairs received by the PC.
- No long term experiment has been conducted to verify the approach proposed in this thesis under real world conditions.

Bibliography

[1] Matthias Keller, Jan Beutel, Andreas Meier, Roman Lim, and Lothar Thiele. Learning from sensor network data. In *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 383–384, New York, NY, USA, 2009. ACM.

[2] Geoffrey Werner-Allen, Konrad Lorincz, Matt Welsh, Omar Marcillo, Jeff Johnson, Mario Ruiz, and Jonathan Lees. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 10(2):18–25, 2006.

[3] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 147–163, New York, NY, USA, 2002. ACM.

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

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

[6] Philipp Sommer and Roger Wattenhofer. Gradient Clock Synchronization in Wireless Sensor Networks. In *8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), San Francisco, USA*, April 2009.

[7] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, volume 38, pages 1–11, New York, NY, USA, May 2003. ACM.

[8] An open-source OS for the networked sensor regime. `http://www.tinyos.net/`.

[9] LEA-5H GPS receiver module with Flash memory. `http://www.u-blox.com/en/gps-modules/pvt-modules/lea-5h.html`.

[10] The gumstix homepage. `http://www.gumstix.com`.

[11] Zigbit 900 Module. `http://www.meshnetics.com/zigbee-modules/zigbit900/`.

[12] MICA2 Data Sheet. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf`.

[13] Atmel ATMega1281 Data Sheet. `http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf`.

[14] Atmel AT86RF212 Data Sheet. http://www.atmel.com/dyn/resources/prod_documents/doc8168.pdf.

[15] Atmel AT86RF230 Data Sheet. http://www.atmel.com/dyn/resources/prod_documents/doc5131.pdf.

[16] IRIS 2.4 GHz. http://www.xbow.com/Products/productdetails.aspx?sid=264.

[17] Georg Oberholzer. Gateway for IEEE 802.15.4 based Wireless Sensor Network. 2009.

[18] Digi Connect ME - Secure network device server module. http://www.digi.com/products/embeddedsolutions/digiconnectme.jsp.

[19] u-blox 5 Protocol Specification. http://www.u-blox.com/images/downloads/Product_Docs/u-blox5_Protocol_Specifications(GPS.G5-X-07036).pdf.

[20] The Python Programming Language. http://python.org/.

[21] Meshbean900. http://www.meshnetics.com/dev-tools/meshbean/.

[22] The AVR C Library. http://www.nongnu.org/avr-libc/.

[23] LIS3LV02DQ Data Sheet. http://www.st.com/stonline/products/literature/ds/11115.pdf.

[24] LIS3L02DQ Data Sheet. http://www.st.com/stonline/products/literature/od/10175.pdf.