Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

GPS-equipped Wireless Sensor Node for PermaSense

SEMESTER THESIS

Felix Sutton

Advisors: Bernhard Buchli
Dr. Jan Beutel

Professor: Prof. Dr. Lothar Thiele

Computer Engineering and Networks Laboratory
Department of Information Technology and Electrical Engineering
ETH Zürich

December 28, 2010

Acknowledgement

I would like to thank Prof. Dr. Lothar Thiele for giving me the opportunity to write this semester thesis in his research group.

I would also like to extend my gratitude to Bernhard Buchli and Dr. Jan Beutel for offering this exciting semester project and giving me the opportunity to explore the challenging world of wireless sensor networks. The technical guidance from Bernhard throughout this semester project is greatly appreciated.

Furthermore, I would like to take this opportunity to thank my family and girlfriend for their amazing support.

Abstract

Monitoring the long-term movement of alpine rock formations has significant research and commercial opportunities in areas such as geological modelling, infrastructure planning and public early warning systems.

The PermaSense project [1] has successfully deployed wireless sensor nodes for environmental monitoring in the alpine regions of Switzerland. Recent research projects [2] have successfully developed an over-provisioned test system for online GPS using commercial single-frequency Global Positioning System (GPS) receivers together with Differential-GPS post-processing techniques.

This semester thesis introduces a prototype GPS-equipped wireless sensor node for the PermaSense alpine environmental monitoring project, with the target application being the accurate measurement of alpine rock movement.

The GPS-equipped wireless sensor node consists of a low-cost commercial GPS receiver coupled with an ultra-low power TinyNode184 wireless sensor node. The captured satellite range and timing measurements are transmitted to a sink node using Dozer, an ultra-low power multi-hop communication protocol. Differential GPS post-processing techniques are used to achieve range accuracy within the order of a few centimeters.

The functional requirements are presented, followed by the overall system design of the wireless sensor node. A detailed discussion of hardware interfaces and software architecture of the prototype is given, followed by a brief description of the experimental setup. Finally, a detailed functional and operational anaylsis of the developed prototype is provided.

Contents

Tables

Figures

1 Introduction

1.1 Motivation

Monitoring the long-term movement of alpine rock formations promotes the development of geological prediction models which have significant research and commercial applications such as infrastructure planning and maintenance, natural disaster management and early warning systems. The remote monitoring of such vast, isolated and treacherous alpine regions poses difficult challenges, which may otherwise not be possible to overcome without the application of wireless sensor networks.

The Global Positioning System (GPS) provides accurate, continuous, global position and time information through the use of GPS receiver equipment. A constellation of satellites orbiting the Earth are synchronised using atomic clocks. A GPS receiver can determine a satellite's location and the propagation time of the received signal by decoding the received GPS message. The GPS receiver can then calculate the time of arrival of a given satellite, and extending this to four or more satellites, the receiver can determine its global position in three dimensions and the current time.

Commercially available low-cost single-frequency GPS receivers suffer from high localisation errors, typically in the order of meters [3]. Clearly, such GPS receivers alone are not sufficient for the monitoring target application, where an accuracy of centimeters is required. Recent research projects [2] have successfully demonstrated that the application of Differential-GPS (D-GPS) [4] post-processing techniques significantly reduces the ranging errors suffered by such low-cost commercial GPS receivers.

It is the aim of this semester thesis to design and implement a prototype GPS-equipped wireless sensor node using a low-cost commercial single-frequency GPS receiver, an ultra-low power wireless sensor platform, and an ultra-low power network protocol stack. By capturing satellite range and timing measurements for all available satellites together with GPS reference measurements and D-GPS algorithmic post-processing, the location of alpine rock formations can be accurately calculated.

The PermaSense project [1] has successfully deployed wireless sensor nodes for environmental monitoring in the alpine regions of Switzerland. It is the intention that this semester thesis work will motivate the future integration of GPS-equipped wireless sensor nodes into the existing PermaSense alpine wireless sensor network.

1.2 Contributions

The contributions of this semester thesis are as follows:

- A system design of a GPS-equipped wireless sensor node for monitoring the movement of alpine rock formations, with the intention to deploy the node within the PermaSense environmental wireless sensor network.
- A proof-of-concept implementation of the aforementioned GPS-equipped wireless sensor node system design.
- A functional and operational performance analysis of the developed prototype measurement sensor node.

1.3 PermaSense Project

PermaSense is a multi-disciplinary research project aimed at developing the next generation of wireless sensor monitoring equipment for deployment in extreme environmental conditions. The PermaSense project has successfully deployed two wireless sensor networks in the Swiss alpine region, namely at the Jungfraujoch and the Matterhorn. The success of these field deployments from a geological perspective has motivated the research into localisation monitoring of alpine rock formations.

1.4 Dozer Network Protocol

Dozer is an ultra-low power network protocol stack [5] tailored for data gathering applications in wireless sensor networks. The Dozer protocol forms the communication backbone in all the PermaSense wireless sensor network deployments.

The Dozer network protocol stack is implemented using TinyOS [6]. TinyOS is a light-weight event-centric operating system specifically designed for embedded systems with limited resources, and is very popular within the sensor network research community. TinyOS is written in nesC [7], which is an extension to the C language designed to implement the concepts and execution model of TinyOS.

The Dozer protocol stack is assumed in the system design of the GPS-equipped wireless sensor node, as it provides an advantageous integration path into the existing PermaSense network infrastructure while also maximising TinyOS software re-use.

1.5 Related Work

The concept of integrating a wireless sensor node with a GPS receiver for localisation applications has been previously introduced in projects such as [8], [9] and [10]. However, there are some significant differences between these previous studies and the prototype system developed in this work.

The Wildfire Monitoring [8] and Walking GPS [9] projects only rely on GPS positioning information where localisation errors of 1 to 2 meters were observed. The GLACSWEB glacial monitoring project [10] achieves higher localisation accuracy by using a Differential-GPS hardware module. The hardware D-GPS solution presumably has a higher component cost and overall power consumption than a single-frequency GPS receiver in conjunction with Differential-GPS algorithmic post-processing as used in this semester thesis.

Recent research projects at the ETH Computer Engineering and Networks Laboratory [2] have developed a corestation for prototyping Differential-GPS post-processing techniques. This system was designed as test-bed and therefore uses high-end microprocessors and communication interfaces which are not power-optimised for scalable wireless sensor network deployments. This semester thesis will extend the corestation system design into a scalable and power-efficient GPS-equipped wireless sensor node.

1.6 Outline

The remaining part of this semester thesis is structured into four chapters. Chapter 2 presents the measurement requirements of the target alpine environmental monitoring application, the system design for a GPS-equipped wireless sensor node, followed by a description of the hardware and software components used to implement the prototype. Chapter 3 contains a summary of the experimental setup used to develop the prototype. A functional performance, power and system deployment analysis is given in Chapter 4. Finally, Chapter 5 contains a conclusion and discussion of future research opportunities.

2

System Design & Implementation

2.1 Functional Requirements

In order to capture the GPS-equipped wireless sensor node's functional requirements, the system deployment of the sensor network into the PermaSense monitoring infrastructure must be considered. An example of such a deployment is illustrated in Figure 2-1.

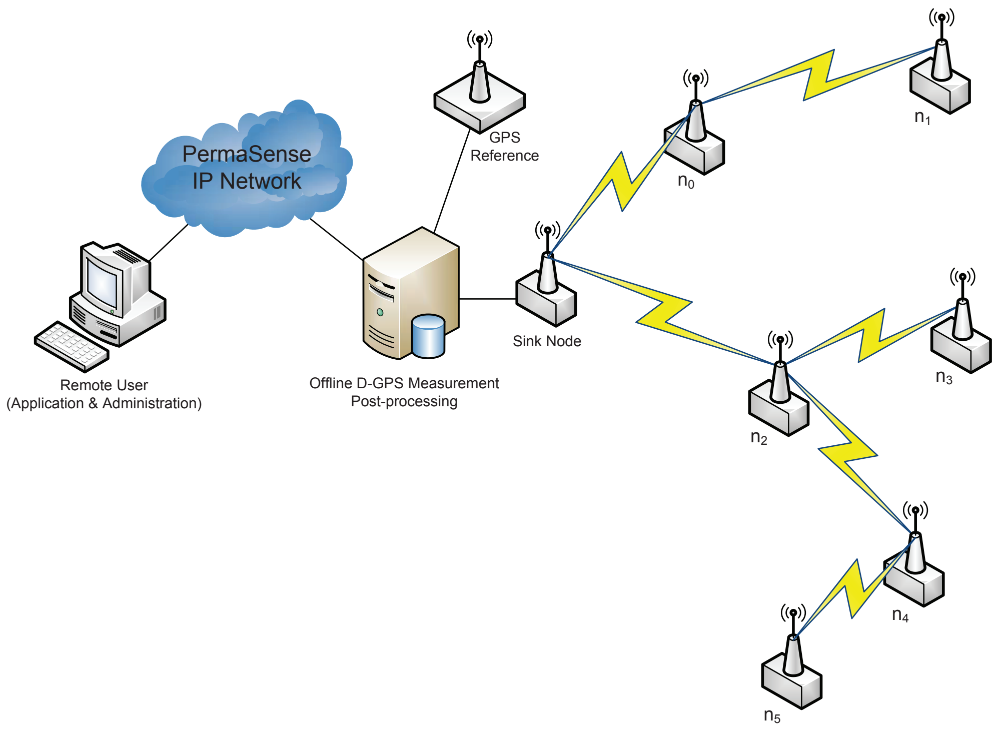


Figure 2-1
Example PermaSense GPS-equipped wireless sensor network deployment.

It is the intention to deploy a number of GPS-equipped wireless sensor nodes (n_0,n_1,...,n_5 in Figure 2-1) across an alpine region of particular geological interest. Each sensor node receives unmodified satellite measurement data from its attached GPS receiver and transmits the data to a sink node using Dozer, a low-bandwidth multi-hop network protocol. The sink node is strategically placed in the neighbourhood of the sensor nodes in order to receive all GPS measurement data over the wireless link before sending the data to an attached server for post-processing. The post-processing server is responsible for performing the Differential-GPS (D-GPS) algorithms on the collected satellite measurement data set. The D-GPS post-processing algorithms rely on GPS measurement information from reference base stations, which consist of GPS logger equipment positioned at a fixed and known location. The post-processing server may have a direct connection to such a GPS reference device, or remote access to publicly available GPS reference data. The post-processing server is connected to the PermaSense back-bone IP network for access to common IT services such as graphical data representation, data archiving, remote access, operation and maintenance functions.

A high localisation measurement resolution (in the order of centimeters) is achieved by harnessing D-GPS techniques at the post-processing server. The D-GPS algorithm are dependent on accurate, time-synchronised and periodic carrier-phase, pseudorange and doppler shift measurements from all available GPS satellites. It is vital that all deployed sensor nodes perform the GPS measurements in a synchronised manner, otherwise the calculated range measurements become uncorrelated in time and thus introduce significant spatial errors. Experimental results have indicated [11] that continuous GPS measurements for alpine rock monitoring are not necessary, but rather a set of 3 to 6 hour long measurement blocks at 30 second periodic intervals ensures the desired localisation accuracy. The measurement interval, block length and time-synchronised operation is exemplified in Figure 2-2.

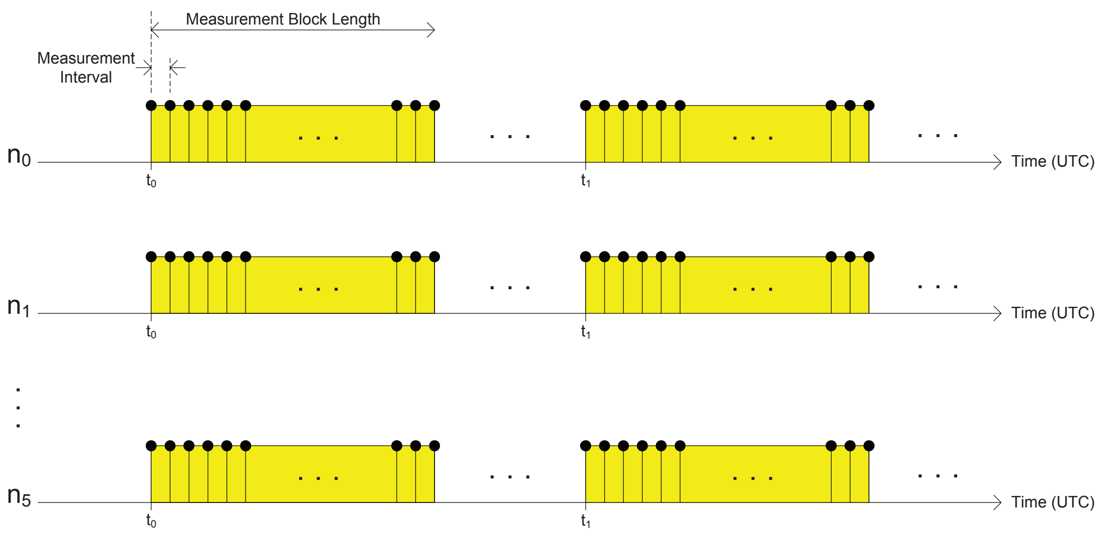


Figure 2-2
Example of synchronised GPS measurement scheduling.

The sensor nodes are to be deployed in high-altitude alpine regions where they will be exposed to extreme environmental conditions. Therefore the hardware components must be selected and the mechanical enclosure constructed to withstand such harsh conditions. The operational and mechanical requirements based on the already deployed PermaSense sensors are listed in [12].

The GPS-equipped wireless sensor node functional requirements can be summarised as follows:

- Receive GPS satellite range and timing data from all available satellites.
- Reliably communicate all measurement data to a sink node for D-GPS post-processing.
- UTC-time synchronised measurement sampling across all deployed sensor nodes.
- Support a continuous measurement block length of 3 to 6 hours, with a measurement interval of 30 seconds on each sensor node.
- Support a programmable measurement schedule allowing multiple measurement blocks per 24 hour period.
- Reliable secondary storage of captured measurement data.
- Support interfaces for additional environmental sensors.
- Power optimised for battery operated deployment with maximal operational lifetime.
- System designed for deployment in extreme environmental conditions.

2.2 System Architecture

The system architecture of the GPS-equipped wireless sensor node is illustrated in Figure 2-3. The system consists of an ultra-low power wireless sensor module (Shockfish SA TinyNode184), and a single-frequency GPS receiver module (u-box LEA-6T).

The TinyNode184 executes the application logic, interfaces to external sensors and communicates with the wireless sensor network. The TinyNode184 also interfaces to an external storage device (e.g. SD-card) for reliable storage of measurement data, and to miscellaneous application-specific sensors and/or actuators.

The GPS receiver provides satellite range and timing data to the TinyNode184 over a standard asynchronous serial interface (UART). The GPS module also provides a periodic monostable pulse synchronised with UTC time at configurable intervals, as described in Section 2.2.2. The power to the GPS module is gated by a latch controlled by the TinyNode184 module. The entire system is powered by a battery cell or external power source through a voltage regulator.

The TinyNode184 implements a programmable measurement schedule where the desired measurement interval may be statically defined. Internal timers are used to commence the measurement sequence at the specified UTC time, as described in Section 2.4.2. At such time, the TinyNode184 will power-on the GPS module using

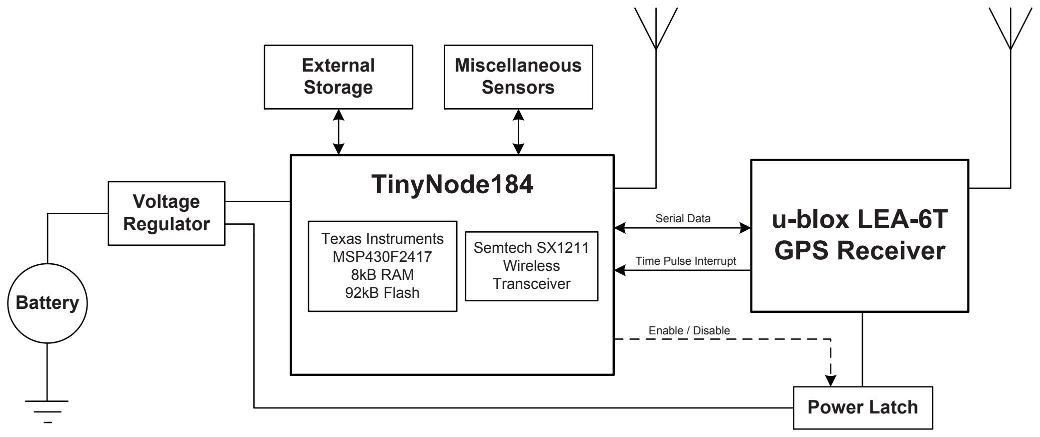


Figure 2-3
System architecture of GPS-equipped wireless sensor node.

the power latch (see Section 2.3.2), and await for a timepulse (i.e. a hardware interrupt) from the GPS module. When the interrupt is received, the TinyNode184 reads the GPS measurement data, partitions the data into a suitable packet format, and reliably transmits the packet to the sink node using the Dozer wireless sensor network protocol. The TinyNode184 will also store the measurement data in external storage (e.g. SD card) for redundancy purposes. Other environmental sensors (e.g. temperature, humidity, etc.) may also be read before the TinyNode184 enters a sleep state. The periodic measurement cycle repeats for the duration of the programmed measurement schedule.

The main features of the GPS-equipped wireless sensor node system design are as follows:

- High measurement accuracy through a precision GPS receiver module and the processing of measurement data at UTC synchronised time intervals.
- Reliable measurement retrieval through underlying sensor network protocol, Dozer, coupled with secondary storage mechanisms.
- Power efficiency through ultra-low power TinyNode184, GPS power optimised tracking modes and GPS duty cycling using the power latch.
- Customisable to wide-range of localisation sensing applications through availability of external interfaces for miscellaneous sensors and/or actuators.

2.2.1 TinyNode184

The TinyNode184 [13] is an ultra-low power module that provides easy integration of wireless communication together with sensors and/or actuators. The module consists of a Texas Instruments MSP430 low-power microcontroller [14] and a 868 / 915 MHz Semtech SX1211 ultra-low power wireless transceiver [15]. The MSP430 micro-controller has 92kB program flash, 8kB RAM, and a number of external interfaces including GPIO, UART, SPI and I2C.

A TinyNode184 extension board is used to assist in programming, debugging and interfacing to the TinyNode184 module. The extension board contains on-board power regulators, RS-232 line drivers, LEDs and wire-wrap/solder-pads for selected GPIO ports. The TinyNode184 together with the extension board is illustrated in Figure 2-4.

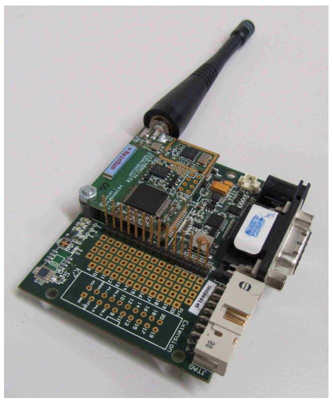


Figure 2-4
TinyNode184 with extension board.

2.2.2 LEA-6T GPS Receiver

The LEA-6T is a commercially available single-frequency GPS receiver specialised for precision timing applications. This particular GPS module provides simultaneous access to the unmodified satellite measurement data for a configurable number of satellites, which is a fundamental requirement for the application-specific D-GPS post-processing algorithms. The GPS module supports USB, UART and DDC (I2C compliant) interfaces over which the standard NMEA protocol [16] or the proprietary UBX binary protocol [17] can be used to transfer commands and data between a host and the GPS module.

The LEA-6T evaluation kit is used to configure, test and interface to the LEA-6T GPS module. The kit provides a USB interface (from which the unit is powered from), antenna connector and a LED indicating the timepulse trigger. The kit was modified by adding a new socket connecting directly to the timepulse pin on the LEA-6T integrated circuit. The modified LEA-6T evaluation kit is illustrated in Figure 2-5.

A precision GPS/UTC synchronised monostable timepulse is provided by the LEA-6T. The timepulse has a configurable pulse width and frequency (in integer multiples of Hz). The module can also be configured to send pre-configured commands shortly after the timepulse, thus providing UTC synchronised measurement delivery to a connected host.

Figure 2-5
u-blox LEA-6T GPS evaluation kit.

2.3 Hardware Interfaces

2.3.1 Serial Interface and Timepulse Operation

The standard asynchronous serial interface is used to interconnect the TinyNode184 and GPS module. The TinyNode184 extension board and the LEA-6T evaluation kit are both wired as RS-232 Data Circuit-terminating Equipment (DCE) devices, requiring a DCE-to-DCE (or cross-over) cable to interconnect the two devices. The timepulse trigger from the LEA-6T is connected to spare GPIO pin on the TinyNode184 extension board (i.e. Port1.3 of the MSP430 microcontroller). The ground pins of both devices are connected to ensure a common ground between the two devices. The wiring of the DCE-DCE cable, the timepulse and common ground is shown in Figure 2-6.

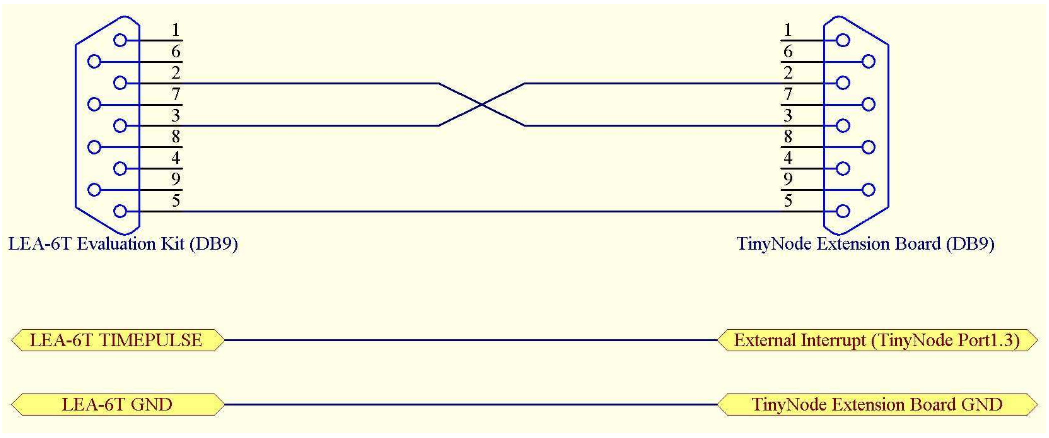

1
6
2
7
3
8
4
9
5
LEA-6T Evaluation Kit (DB9)
TinyNode Extension Board (DB9)
LEA-6T TIMEPULSE
External Interrupt (TinyNode Port1.3)
LEA-6T GND
TinyNode Extension Board GND

Figure 2-6
GPS serial and timepulse interfacing.

2.3.2 Power Supply and GPS Power Latch

A common +5V DC voltage regulator is used to power all components of the prototype, which allows flexibility for deployment with a battery cell or with higher-power external power supplies.

It is well-known that the GPS modules are power-hungry devices [8], and the LEA-6T is no exception. In the field of wireless sensor nodes, where power is an extremely scarce resource, methods have to be used to minimise the overall GPS power consumption. One such method is through power optimised tracking [18], and another more trivial method is to simply turn off the device when it is not being used.

A simple power latch circuit (see Figure 2-7) is used to gate the power supplied to the GPS module. The power latch is controlled by the TinyNode184. A dual-transistor circuit is used to ensure an active-high trigger from the TinyNode184 and to safely switch between the +3V and +5V DC power domains. A prototype circuit including debug LEDs is constructed on vero-board and conveniently mounted onto the TinyNode184 extension board as illustrated in Figure 2-8.

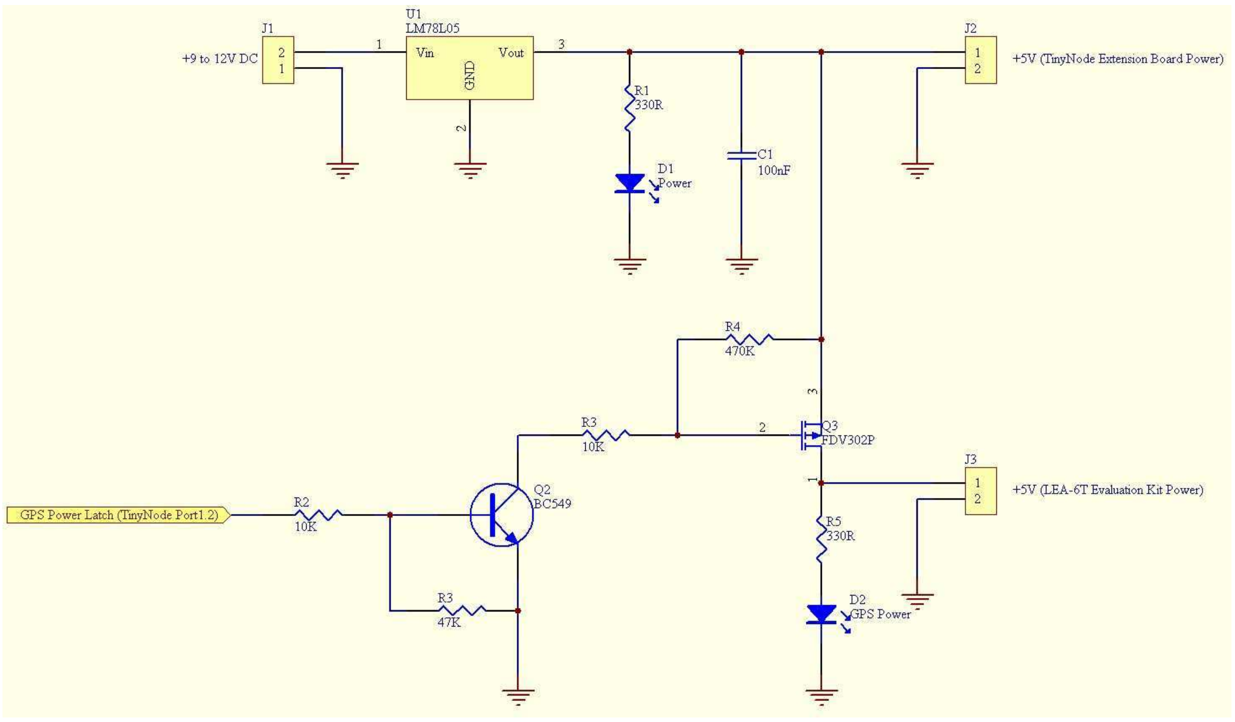


Figure 2-7
Power supply and GPS power latch circuit diagram.

The use of the TinyNode184 extension board and the GPS evaluation kit enables rapid prototyping and simplifies debugging at the cost of higher power consumption. A total of three power regulators (at two different power domains), several debug LEDs and unused sensor devices considerably increase the base-line power consumption. It is expected that a customised Printed Circuit Board (PCB) would provide significant power savings.

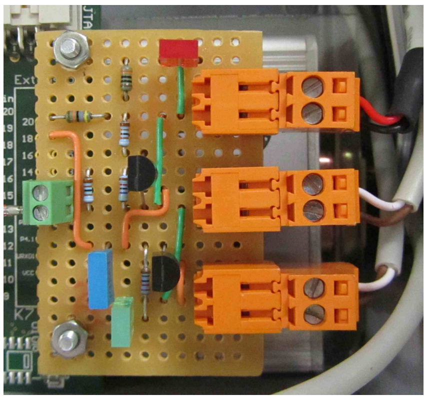


Figure 2-8
Prototype power supply and GPS power latch circuit.

2.4 Software Design & Implementation

The software architecture and interface description of the GPS-equipped wireless sensor node is illustrated in Figure 2-9. The following sections describe the main software components in detail.

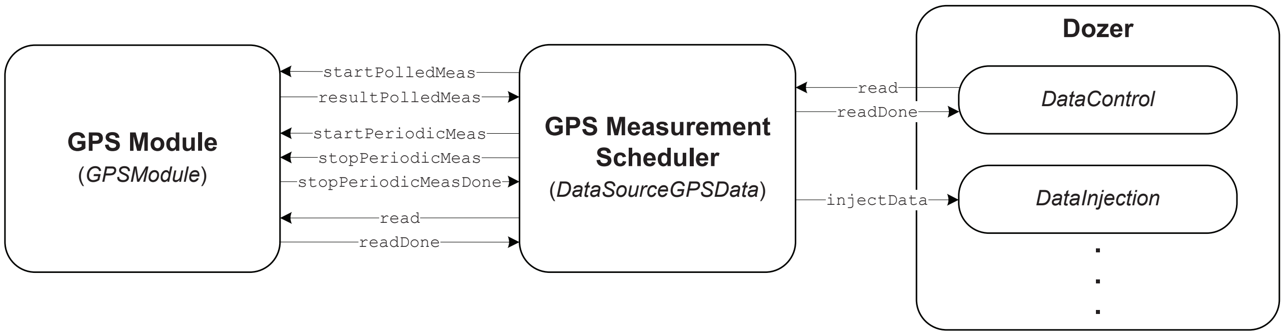


Figure 2-9
Software architecture.

2.4.1 GPS Module

The LEA-6T GPS module supports two modes of measurement retrieval, namely polled or periodic mode. The polled mode requires the host to explicitly request the measurement using a zero length payload request packet, whereas the periodic mode supplies the host with the configured message type at a defined interval.

The retrieval modes and low-level hardware interfaces to the GPS module are encapsulated into a TinyOS software component named GPSModule. The module wires to the serial interface components, the power latch GPIO component, and the GPIO external interrupt component to which the LEA-6T timepulse is connected to.

When the GPS module is powered-on it attempts to search for available satellites. Once synchronisation is achieved and the ephemeral data has been successfully downloaded and decoded, the module is able to perform polled or periodic measurements. In the case where there are no available satellites, a zero-length-payload packet is sent in polled mode, and no data is sent in periodic mode.

Synchronised GPS measurements between multiple nodes are achieved by utilising the LEA-6T timepulse to trigger the TinyNode184 when the latest measurement is available. The LEA-6T timepulse is always synchronised to a UTC time interval after satellite acquisition. In the case of a 30 second measurement interval, the timepulse is triggered at 15 and 45 seconds past the top of every UTC minute. Shortly after the timepulse arrives, the LEA-6T writes the measurement data to the serial interface. Figure 2-10 illustrates the GPS power-on and periodic measurement sequence.

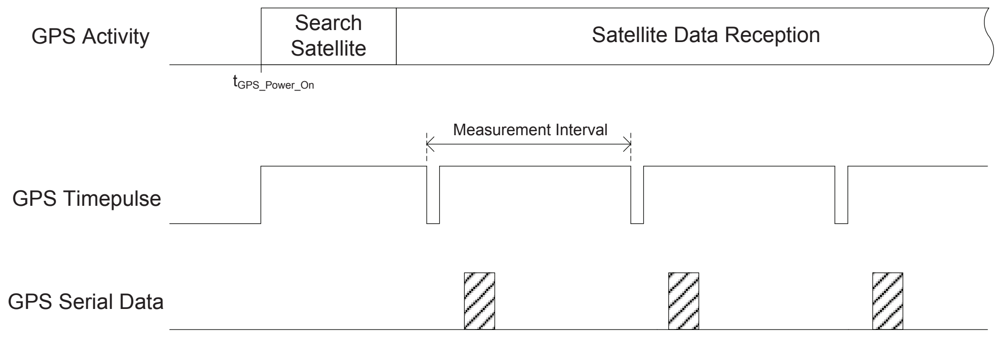


Figure 2-10
GPS timepulse sequence.

The GPS serial data is transferred to the TinyNode184 using the UBX protocol [17]. The protocol is a proprietary binary transport protocol from u-blox which defines a variable length packet structure as illustrated in Figure 2-11. The start of the packet is identified by a pair of synchronisation bytes, followed by the encoded message type by means of message class and message ID bytes. The length of the payload is then encoded followed by the payload and the packet checksum.

UBX protocol packets are decoded in the GPSModule component using a simple state-based parser. The StateChart of the parser is illustrated in Figure 2-12. Each received byte from the UART is placed into an input buffer before the parser state machine is invoked. The parser is implemented as a TinyOS task [19]. Since tasks execute atomically with respect to each other [20], potential race conditions in reading and writing from the input buffer are prevented. Once the UBX packet structure has been parsed and verified by comparing the encoded checksum with the calculated checksum, the message is placed into a message buffer for further processing.

The satellite measurement data is structured as a linear byte-wise array consisting of the GPS sample time, the number of satellites measured, followed by the individual satellite measurements (where each satellite has an identical data structure).

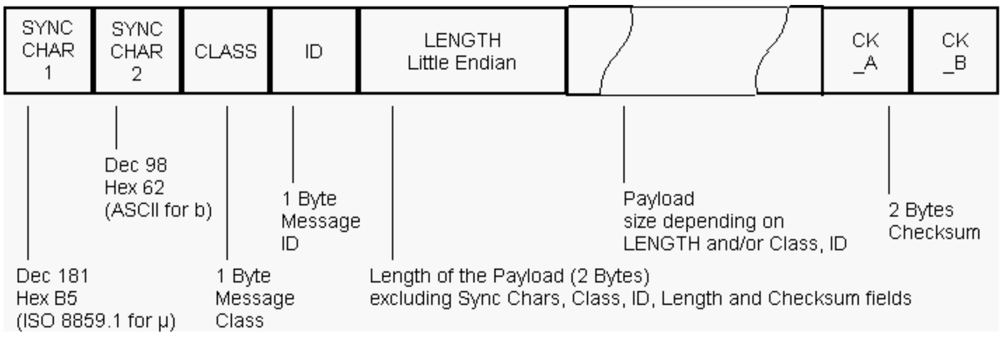


Figure 2-11
UBX protocol structure.

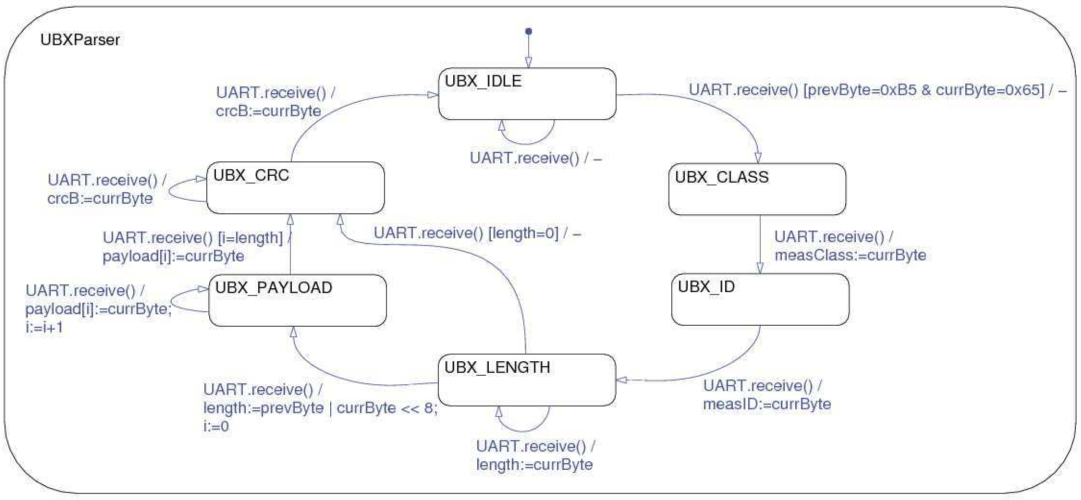

UBXParser
UBX_IDLE
UBX_CLASS
UBX_ID
UBX_LENGTH
UBX_PAYLOAD
UBX_CRC
UART.receive() [prevByte=0xB5 & currByte=0x65] / –
UART.receive() / –
UART.receive() / measClass:=currByte
UART.receive() / measID:=currByte
UART.receive() / length:=currByte
UART.receive() / length:=prevByte | currByte << 8; i:=0
UART.receive() / payload[i]:=currByte; i:=i+1
UART.receive() [i=length] / payload[i]:=currByte
UART.receive() [length=0] / –
UART.receive() / crcB:=currByte
UART.receive() / crcB:=currByte

Figure 2-12
UBX parser StateChart.

The length of the entire data structure is variable, based on the number of visible satellites at the time the measurement is taken. Although the LEA-6T can track a configurable number of satellites simultaneously, experiments have shown that a maximum of 12 satellites is a more realistic scenario in the field. A maximum of 12 simultaneous satellites requires a total of 296 bytes storage in the measurement buffer.

A TinyOS timer component [21] is instantiated as a UBX protocol timeout timer. The timeout timer is used to identify if a parsing error or unexpected signalling behaviour from the GPS module occurred (i.e. CRC errors). The timer also serves as a convenient synchronisation point for configuring the GPS measurement mode.

The GPS measurement mode is started using the respective polled / periodic split-phase commands [20] (*startPolledMeas* / *startPeriodicMeas*), as illustrated in Figure 2-9. When periodic measurements have been requested, the software execution sequence (see Figure 2-13) is as follows: (i) an asynchronous event (i.e. hardware interrupt) is triggered by the timepulse, (ii) the serial interface is enabled and each received byte is processed by the parser, when the last byte is processed, the packet is placed in measurement buffer for further processing, and finally (iii) the timeout timer expires and the serial interface is disabled to reduce power consumption.

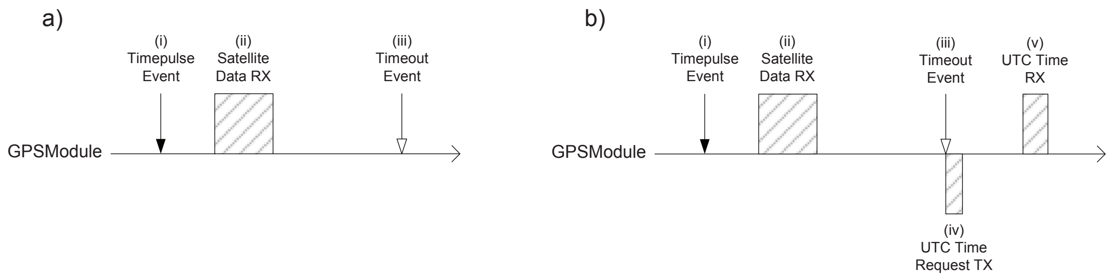


Figure 2-13
(a) Periodic and (b) polled GPS measurement sequence.

The polled measurement sequence is slightly more complicated, as by design, the TinyNode184 does not maintain UTC timing knowledge of when exactly the next timepulse will occur. So in order not to have the polled measurement request command collide with a periodic measurement result, the polled request is piggy-backed onto the timeout timer. The execution sequence for polled measurements (see Figure 2-13) is as follows: (i) the timepulse event occurs, (ii) periodic measurement data is received and discarded, (iii) the timeout timer expires, (iv) the zero length payload request command is sent to the GPS module, and finally (v) the polled measurement data is received and parsed.

The internal states of the software GPSModule mimic the operational states of the GPS receiver, namely GPS_MODULE_IDLE when the GPS module is powered off, GPS_MODULE_ACTIVE_POLLED and GPS_MODULE_ACTIVE_PERIODIC for polled and periodic modes respectively. The StateChart for the GPSModule internal state machine is illustrated in Figure 2-14.

The software architecture is such that the GPSModule can be easily extended to poll or periodically request any of the UBX commands supported by the LEA-6T GPS module. However, only two such commands (see Table 2-1) are required to implement the functional requirements of the prototype GPS-equipped wireless sensor node. The RXM-RAW command provides the complete unmodified range and timing information for all available satellites. The data returned by this command provides all the required inputs to the post-processing Differential-GPS carrier-phase algorithms. The NAV-TIMEUTC command provides the current UTC time and is used for internal measurement scheduling, as described in Section 2.4.2.

The writing of GPS measurement data is assumed constant (i.e. at 30 seconds) due to the UTC time precision of the timepulse. However, the consumer of the GPS measurements (the Dozer DataControl via the GPS Measurement Scheduler, as detailed in Section 2.4.2) reads slightly slower, while also experiencing jitter due to periods of higher execution load in the Dozer protocol stack. Due to this timing mismatch,

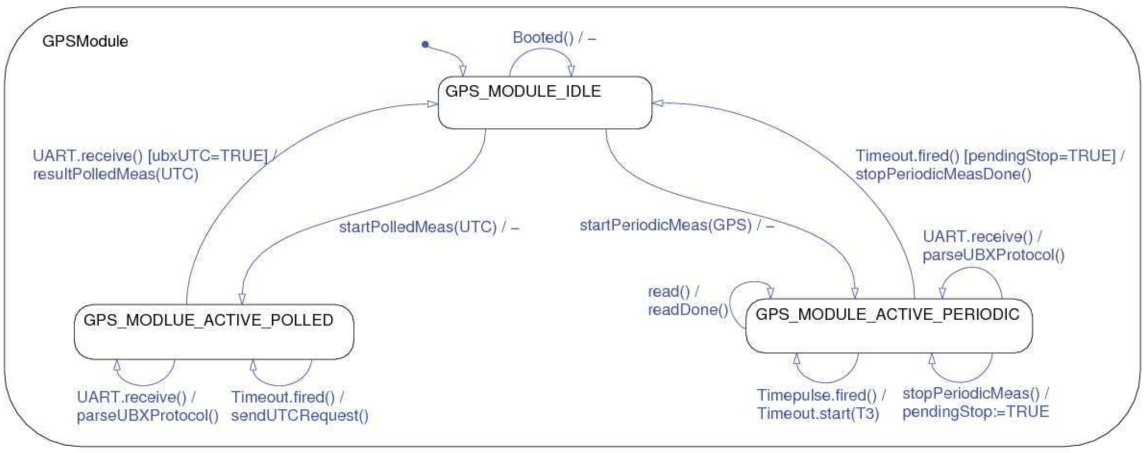

GPSModule
Booted() / –
GPS_MODULE_IDLE
UART.receive() [ubxUTC=TRUE] / resultPolledMeas(UTC)
Timeout.fired() [pendingStop=TRUE] / stopPeriodicMeasDone()
startPolledMeas(UTC) / –
startPeriodicMeas(GPS) / –
UART.receive() / parseUBXProtocol()
read() / readDone()
GPS_MODLUE_ACTIVE_POLLED
GPS_MODULE_ACTIVE_PERIODIC
UART.receive() / parseUBXProtocol()
Timeout.fired() / sendUTCRequest()
Timepulse.fired() / Timeout.start(T3)
stopPeriodicMeas() / pendingStop:=TRUE

Figure 2-14
GPSModule StateChart.

UBX Message Type	Mode	Interval	Response
RXM-RAW	Periodic	30 seconds	Satellite Measurement Data
NAV-TIMEUTC	Polled	-	Current UTC Time

Table 2-1: UBX messages implemented for target application.

an asynchronous FIFO buffer is implemented in the GPSModule to prevent race conditions and ensure measurement data is not overwritten.

The write and read operations on the measurement buffer are implemented in separate TinyOS tasks, therefore each task will run to completion and atomically with respect to each other. It is assumed that the consumer can only read data if there is exactly one or more measurements stored in the buffer. Under these assumptions, the worst-case buffer length for a measurement block length can be calculated using equation 2.1.

$$B = \left\lceil \frac{t_{\text{ReadJitter}}}{t_{\text{WriteInterval}}} * \frac{t_{\text{MeasBlockLength}}}{t_{\text{MeasInterval}}} \right\rceil \tag{2.1}$$

An average jitter of 600ms between the software event triggered by the consumer and the GPS timepulse was observed. Using a realistic deployment configuration as listed in Table 2-2 and substituting into Equation 2.1 gives a worst-case measurement buffer size of 10 measurements.

2.4.2 GPS Measurement Scheduler

The functional requirements of the GPS-equipped wireless sensor node require that a configurable periodic measurement schedule be supported. This scheduling behaviour, and indeed the interface toward the Dozer protocol stack, are encapsulated into a TinyOS software component named DataSourceGPSData. This behaviour is

Measurement Block Length	14400 seconds
Measurement Interval	30 seconds
Dozer Measurement Read Jitter	0.6 seconds
GPS Measurement Write Interval	30 seconds

Table 2-2: Parameters used for measurement buffer size calculation.

modelled in a simple state machine consisting of an idle and active state, as illustrated in Figure 2-15.

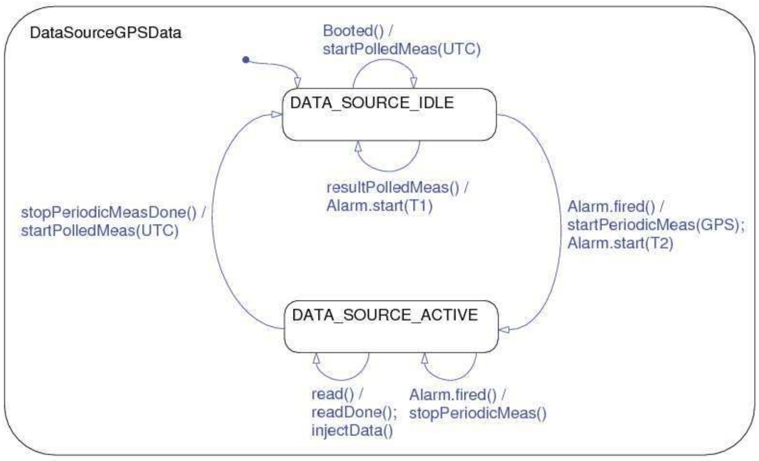

DataSourceGPSData
Booted() / startPolledMeas(UTC)
DATA_SOURCE_IDLE
resultPolledMeas() / Alarm.start(T1)
stopPeriodicMeasDone() / startPolledMeas(UTC)
Alarm.fired() / startPeriodicMeas(GPS); Alarm.start(T2)
DATA_SOURCE_ACTIVE
read() / readDone(); injectData()
Alarm.fired() / stopPeriodicMeas()

Figure 2-15
DataSourceGPSData StateChart.

In order to start and stop the measurement sequence at the correct time, UTC time knowledge is needed within the TinyNode184. The time accuracy only has to be in the order of seconds, since the LEA-6T will take some time to acquire synchronisation on a satellite. The TinyNode184 does not have an internal real-time clock (RTC), and the prospect of adding such a device would only cost in additional power consumption and software overhead. Instead, the UTC time knowledge can be determined from the GPSModule asynchronously together with a TinyNode184 internal one-shot alarm [21]. The measurement scheduling sequence is exemplified in Figure 2-16.

When the wireless sensor node is powered-on for the first time or during a reset cycle, the GPSModule is requested to read the current UTC time using a polled request. As described in Section 2.4.1, the GPSModule will execute the necessary sequence of events and return the UTC time in a format of <year, day, hour>. Since the measurement schedule is in the range of hours, the year and day values can be ignored. The DataSourceGPSData component will then calculate the time (in TinyNode184 ticks, where 1 tick = 1024 ms) until the start of the next periodic measurement, and configure a one-shot alarm. When the alarm event triggers at the beginning of the measurement block, the DataSourceGPSData component sends a periodic measurement request to the GPSModule before reconfiguring the alarm for

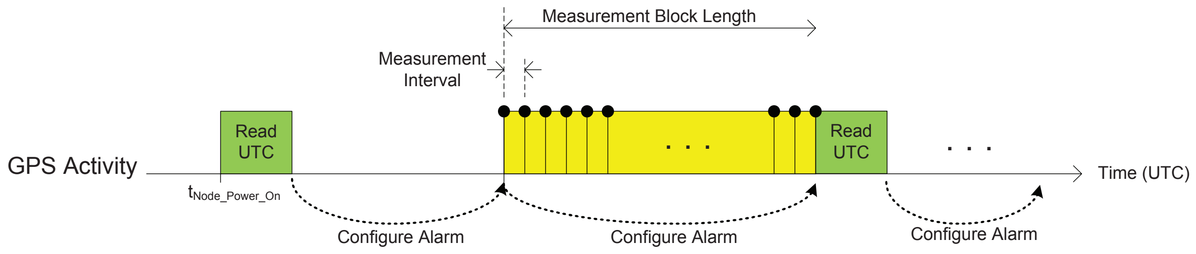


Figure 2-16
GPS Measurement scheduling sequence.

the end of the measurement block. During the measurement block the GPS is performing periodic measurements. At the end of the measurement block the current UTC time is read again, before configuring the one-shot alarm for the start of the next measurement schedule.

As the GPS-equipped wireless sensor node is intended to be deployed for a long period of time in the field, small schedule timing errors at each measurement interval will eventually cause the deployed sensors to measure asynchronously. In order to prevent this clock drift, the current UTC time is read at the end of each measurement schedule. As the LEA-6T is still powered-on at the end of the measurement sequence, the reading of the UTC time ensures synchronised measurements with minimal overhead.

The DataSourceGPSData component also acts as an interface between the Dozer protocol stack and the GPSModule. The Dozer protocol periodically requests measurement data from all data sources (i.e. miscellaneous sensors, as illustrated in Figure 2-3), regardless if there is any data to send. So to prevent stalling the Dozer periodic measurement cycle, the DataSourceGPSData immediately returns NULL data if there is nothing to send (i.e. in the DATA_SOURCE_IDLE state). If the GPS measurement cycle is active (i.e. in the DATA_SOURCE_ACTIVE state) the next set of satellite measurements are retrieved for injection into the Dozer uplink transmission queue.

The Dozer protocol stack implementation can not efficiently transmit variable-sized data packets, due to the TinyOS fixed-size message data structure [22]. Therefore, the measurement data structure received from the GPSModule is partitioned into a number of smaller, fixed-size data structures per satellite as illustrated in Figure 2-17. The satellite data structure contains all the relevant measurement data packed together with the common GPS time prefix. The replication of the GPS time prefix adds additional overhead to all packets, however this greatly simplifies data post-processing and verification at the sink node. Each satellite data structure has a fixed size of 30 bytes.

Each satellite data structure is encapsulated into a Dozer packet. The Dozer packet header includes the unique source node ID and a time stamp. The Dozer packet is further encapsulated into to radio frame (as defined in [15]) before being transmitted over the air. The complete packet structure is illustrated in Figure 2-18.

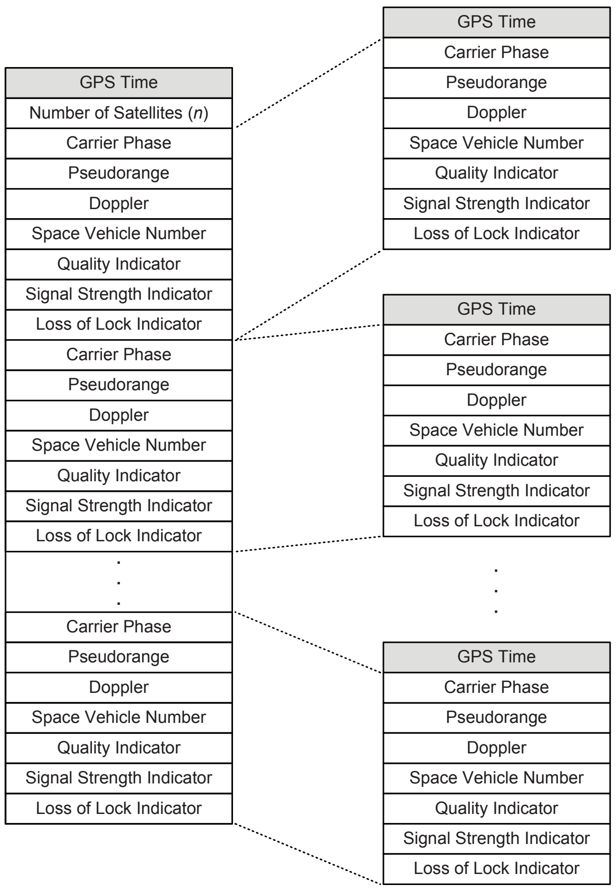


Figure 2-17
Satellite data structure partitioning.

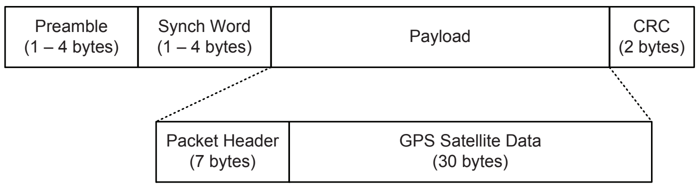


Figure 2-18
Radio frame structure.

2.4.3 GPSDozer Application

The GPSDozer application is an amalgamation of the GPSModule and DataSourceGPSData components together with the complete Dozer network protocol stack. The TinyOS component graph of the GPSDozer application is illustrated in Figure 2-19.

The TinyNode184 memory requirements of the GPSDozer application are summarised in Table 2-3. It can be seen that only 1% static RAM is available for further application development. The excessive static RAM consumption is attributed to the large measurement buffer as described in Section 2.4.1. The size of the measurement buffer is dependent on the timing jitter between the LEA-6T timepulse and the Dozer measurement request, and by the maximum number of satellites monitored for the target application. Significant memory savings can be achieved by fine-tuning the Dozer protocol to reduce the timing mismatch. Further memory savings can be achieved by reducing the maximum number of monitored satellites only if the D-GPS measurement accuracy is not adversely affected.

Memory	Size (bytes)	Used (bytes)	Available
Flash ROM	92208	34672	62.4%
Static RAM	8192	8110	1.0%

Table 2-3: TinyNode184 memory requirements of GPSDozer application.

2.5 Summary

This chapter described the functional requirements of the GPS-equipped wireless sensor node based on the PermaSense target application. The system architecture of the wireless sensor node and its features was presented.

The main hardware components in the architecture, namely the TinyNode184 and the LEA-6T GPS module, were introduced. The hardware interfacing between the two components and associated circuitry was described in detail.

The software architecture was also detailed, giving an insight into the internal states machines of the GPSModule and DataSourceGPSData software components. Finally, the TinyOS component graph of the combined prototype application, GPS-Dozer, was presented together with an analysis of its memory requirements.

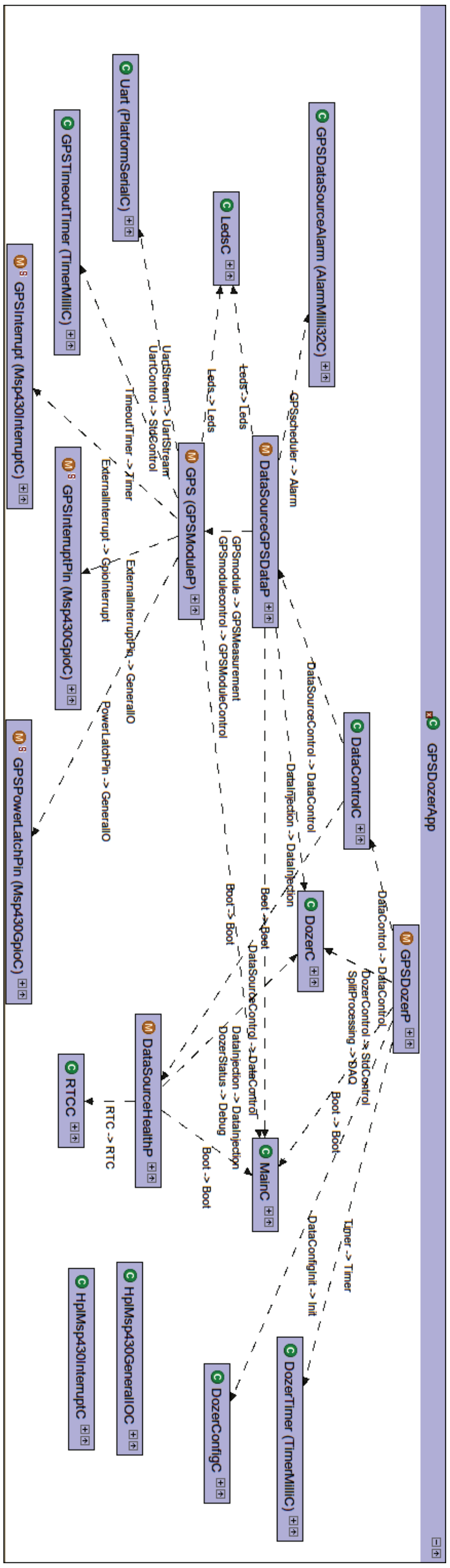

GPSDozerApp
GPSDataSourceAlarm (AlarmMilli32C)
LedsC
Uart (PlatformSerialC)
GPSTimeoutTimer (TimerMilliC)
GPSInterrupt (Msp430InterruptC)
GPS (GPSModuleP)
DataSourceGPSDataP
GPSInterruptPin (Msp430GpioC)
GPSPowerLatchPin (Msp430GpioC)
DataControlC
DozerC
GPSDozerP
DataSourceHealthP
RTCC
MainC
HplMsp430GeneralIOC
HplMsp430InterruptC
DozerConfigC
DozerTimer (TimerMilliC)
GPSScheduler -> Alarm
Leds -> Leds
UartStream -> UartStream
UartControl -> StdControl
TimeoutTimer -> Timer
ExternalInterrupt -> GpioInterrupt
ExternalInterruptPin -> GeneralIO
PowerLatchPin -> GeneralIO
GPSmodule -> GPSMeasurement
GPSmodulecontrol -> GPSModuleControl
DataSourceControl -> DataControl
DataInjection -> DataInjection
Boot -> Boot
DataControl -> DataControl
DozerControl -> StdControl
SplitProcessing -> DAQ
DozerStatus -> Debug
RTC -> RTC
Timer -> Timer
DataConfigInit -> Init

Figure 2-19
GPSDozer TinyOS component graph.

3

Experimental Setup

3.1 Software Build Environment

The rapid prototyping [23] software development process is used throughout the development of the TinyNode184 application software. This development approach enables fast verification of all TinyOS components in isolation, before combining multiple components together to form larger and more complex applications.

An open-source software build environment based on Ubuntu, Eclipse and Subversion is used to develop all software. The TinyNode184 application software is implemented using TinyOS [6].

The tools to flash the TinyNode184 microcontroller are included with the MSP430 compiler, which are contained within the default TinyOS distribution. A detailed description of TinyOS, nesC and MSP430 compiler installation, configuration and operation can be found in [24]. A description of building, flashing and customising the GPSDozer application is provided in Appendix A.

3.2 Test & Verification Tools

The prototype GPS-equipped wireless sensor node is tested incrementally in a single-hop topology as depicted in Figure 3-1. A TinyNode184 platform is configured as a sink node using the DozerBase application. The sink node receives all packets sent from the prototype, and transfers them to a host computer using a serial over USB interface.

The SerialForwarder application running on the lab computer creates a UDP socket, where multiple software clients can receive the Dozer packets as they arrive at the sink. The Listener program is used to display the contents of the received packets in hexadecimal format, which allows for fast debugging and verification.

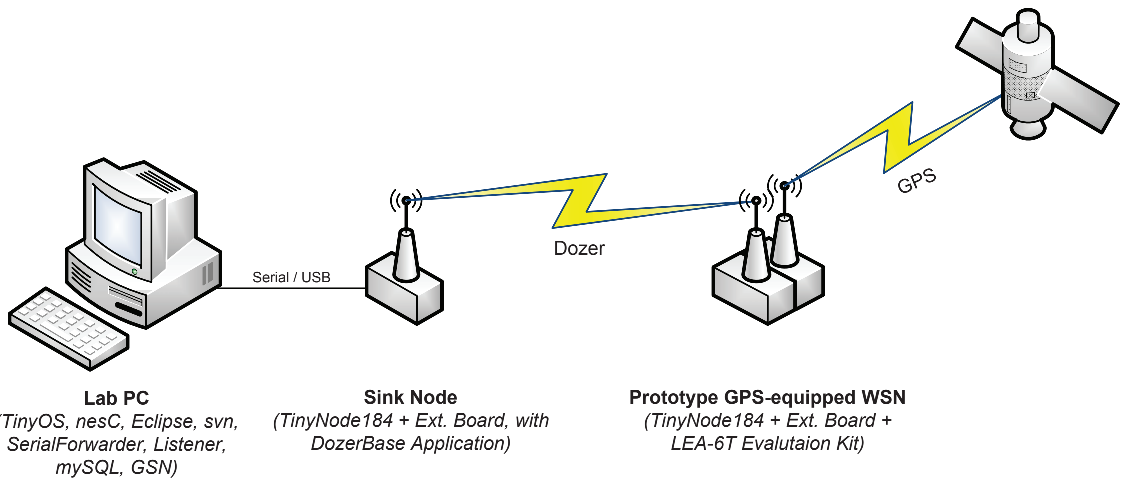

Serial / USB
Dozer
GPS
Lab PC
(TinyOS, nesC, Eclipse, svn, SerialForwarder, Listener, mySQL, GSN)
Sink Node
(TinyNode184 + Ext. Board, with DozerBase Application)
Prototype GPS-equipped WSN
(TinyNode184 + Ext. Board + LEA-6T Evalutaion Kit)

Figure 3-1
Experimental setup.

3.3 Data Analysis Tools

The Global Sensor Networks (GSN) application [25] is a popular method for the parsing and organisation of data from wireless sensor nodes. A custom GSN interface, called backlog, is used to transfer received packets into a mySQL database. The GSN front-end application is provided for graphically visualising the received data.

Direct SQL queries to the mySQL database allow for large measurement data sets to be exported to a comma separated file. A custom Java application is used to parse and sort the measurement data elements and generate Matlab scripts for plotting and analysis.

3.4 Development Challenges

A number of technical obstacles were encountered throughout the prototype development, which included the following:

- Experiencing intermittent behaviour from the GPS module due to weather conditions. The change of weather can reduce the number of visible satellites to zero, which can then cause unexpected behaviour during initial debugging phases.
- Intermittent parsing errors were encountered due to multiple parser tasks being en-queued onto the TinyOS task queue. TinyOS does not allow the same task to be placed on the task queue multiple times. A work-around for re-posting tasks was implemented.
- A new dozer message type for the GPS data needed to be integrated, which required a new Java wrapper class for the GSN tool. The data resolution between the TinyOS mig tool and the mySQL schema does not exactly match,

meaning single and double-precision floating-point (IEEE 754) values had to be stored in a byte-wise format in the mySQL database.

- Occasional data mismatch in the measurement buffer causes invalid satellite packets to be transmitted to the sink node. The suspected root-cause of this problem is attributed to the high jitter in the Dozer data request event. This jitter can be reduced by fine-tuning of the Dozer measurement interval.

3.5 Summary

This chapter briefly described the software build environment used to develop the prototype sensor node, highlighting the use of a rapid prototyping software development process coupled with an open-source software tool chain.

A single-hop network configuration was used to test and debug the prototype. Open-source tools and a custom Java parser application were used for system verification and data analysis.

4

Results

4.1 Functional Performance

The hardware components of the GPS-equipped wireless sensor node are mounted into a water-proof die-cast aluminium enclosure to allow functional testing of the measurement sensor node. The TinyNode184 is programmed with the GPSDozer application, which includes a static measurement schedule. The prototype is powered by an external power supply through a water-proof power socket. External antennas for the TinyNode184 and LEA-6T GPS are attached using water-proof SMA antenna connectors. Due to project time constraints, an external storage device (i.e. SD card) and miscellaneous sensors (i.e. temperature, humidity, etc.) are omitted from the final prototype, however, unused TinyNode184 hardware interfaces would make this integration feasible. The final prototype sensor node is shown in Figures 4-1 and 4-2.

The accuracy of GPS measurement is critical to the overall performance of the target localisation application. The pseudorange and carrier phase measurements are of most importance to the Differential GPS post-processing algorithms. In order to capture real-world measurement data and access the performance of the sensor node, the prototype GPS-equipped wireless sensor node was mounted on the roof of the ETH Electrical Engineering building. Figures 4-3 and 4-4 illustrate the pseudorange and carrier phase measurements for the 7 most available satellites taken over a period of 18 hours, where a measurement block length of 4 hours and an idle duration of 1 hour between measurement blocks was configured.

It can be seen that the pseudorange fluctuates in a sinusoidal pattern, which correlates with the arc movement of the satellite with respect to the stationary sensor node. As the satellite moves away from the horizon, the pseudorange decreases to a minimum where the satellite is closest to the node, before moving away from the node thus increasing the pseudorange. As the satellite moves, the carrier phase observed at the sensor node varies. This in turn creates discontinuities in the pseudorange due to the phase change of the received GPS signal. The periodic scheduling behaviour is also visible, where the GPS measurements are periodically interrupted

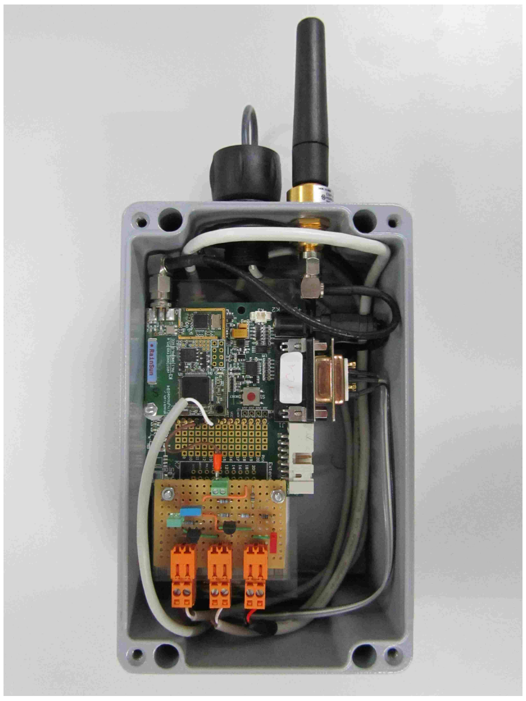


Figure 4-1
Prototype GPS-equipped wireless sensor node (top-view).

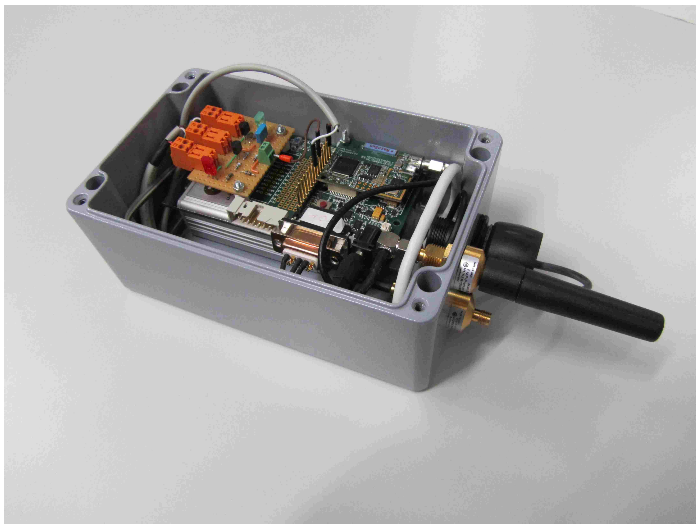


Figure 4-2
Prototype GPS-equipped wireless sensor node (side-view).

for a period of 1 hour, as per the configured measurement schedule. Based on the observed data set and previous experience in analysing the D-GPS input parameters, the prototype achieves the GPS-equipped wireless sensor node functional requirements.

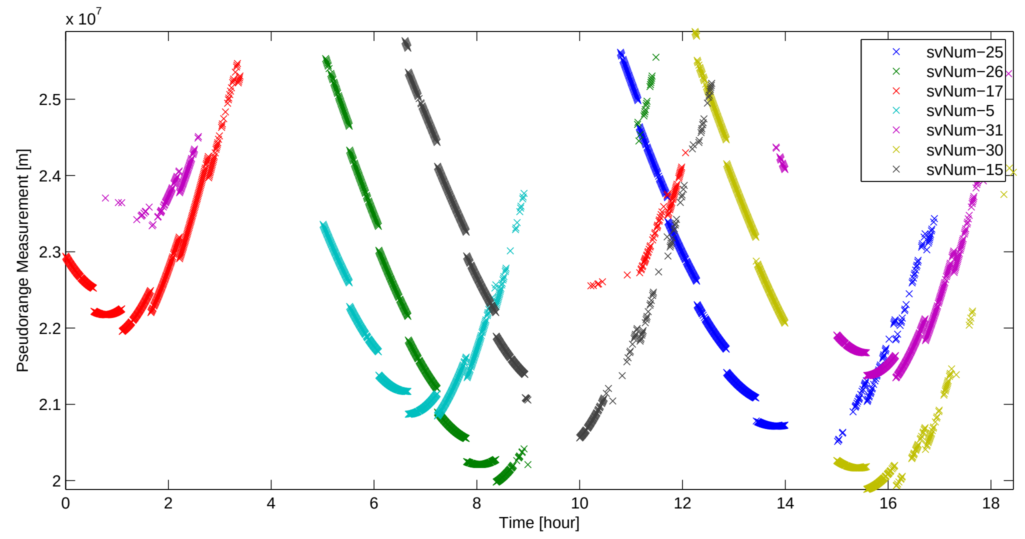


Figure 4-3
Pseudorange measurements from outdoor testing.

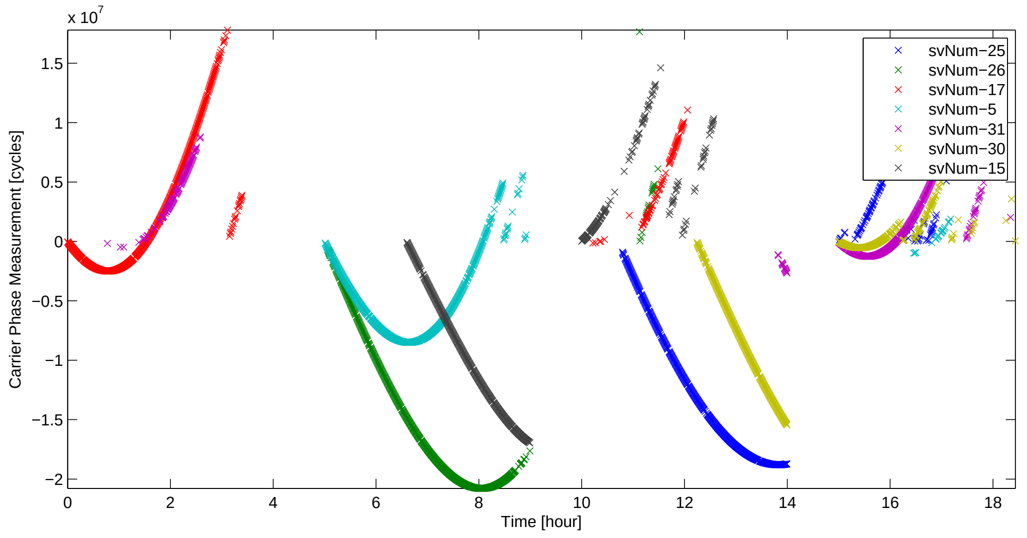


Figure 4-4
Carrier Phase measurements from outdoor testing.

4.2 Power consumption

The power consumption of a wireless sensor node is extremely important to the longevity of the device in the field. A power analysis of the prototype GPS wireless sensor node in a single-hop topology is used to assess the overall power consumption. A Dozer beacon interval of 10 seconds and a measurement interval of 30 seconds is chosen as a realistic deployment configuration. The LEA-6T is configured with Power Optimised Tracking (POT) in order to minimise GPS power consumption. In order to isolate the Dozer network activity from the GPS measurement processing, two scenarios are investigated and discussed in the following paragraphs.

The power profile in Figure 4-5 illustrates the periodic Dozer network activity without GPS measurements, and highlights the four main contributing components to the overall power consumption, namely the periodic (i) beacon reception from the parent (i.e. the sink node in this case), (ii) beacon transmission to child nodes, (iii) Dozer protocol and data processing overhead, and (iv) transmission of health node messages, which are required for Dozer network connectivity.

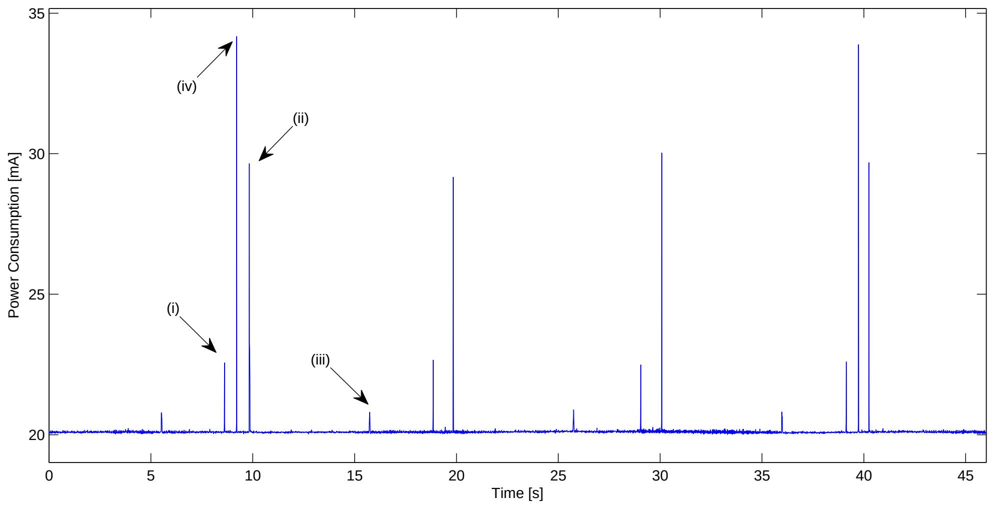


Figure 4-5
GPSDozer power profile without GPS measurements.

The prototype under test contains a number of voltage regulators and LEDs, due to the TinyNode184 extension and LEA-6T evaluation boards. This collectively contributes to an unusually high base-line current consumption of approximately 20mA. It is expected that a custom PCB would significantly reduce the base-line power consumption.

The power profile in Figure 4-6 illustrates the GPSDozer power profile with GPS measurements using a 5 minute measurement schedule. The plot illustrates the aforementioned 20mA base-line superimposed with Dozer network activity, followed by the powering of the GPS module for the approximate 5 minute measurement cycle. The GPS module adds approximately 55mA to the base-line when operating in

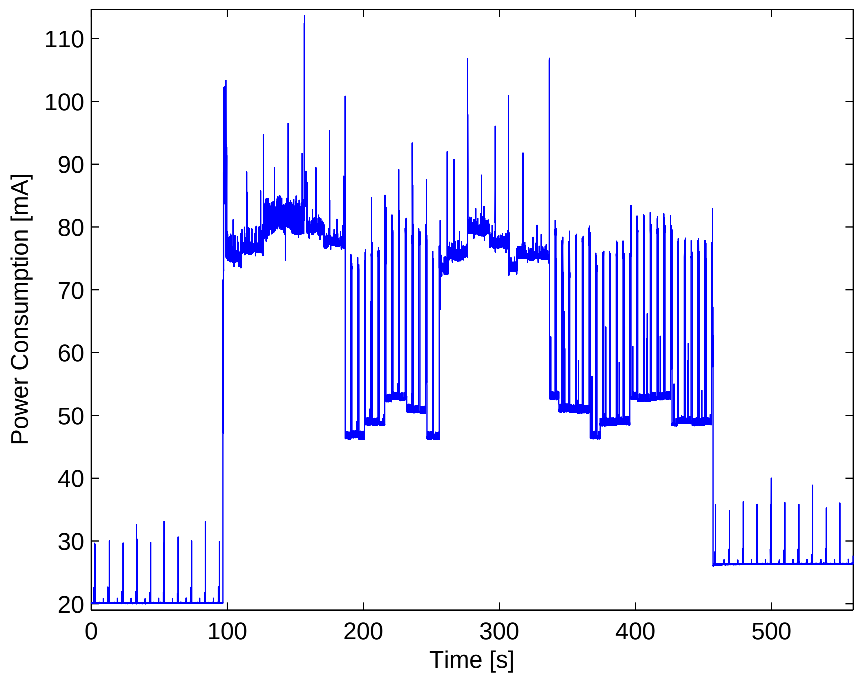


Figure 4-6
GPSDozer power profile with GPS measurements.

maximal-performance mode. At approximately 180 seconds into the test sequence, the satellite visibility increased such that the POT mode was enabled. This drastically reduced the GPS current consumption to 25mA. The current consumption of the POT mode iterates between the 55mA and 25mA in a ratio of 1:5, confirming previous power consumption measurements [2]. The stepped power levels of between 3 and 5mA throughout the GPS activity are due to the three integrated debug LEDs. The signal quality of visible satellites continued to fluctuate during the test, which accounts for the transition out and into POT mode at approximately 260 and 330 seconds respectively.

An estimate of the sensor node total average current consumption (mA) per hour I_{system} can be calculated using Equation 4.1, where I_i is the contributing current (mA) of component i, α_{i} is the average proportion of time (in seconds per minute) component i is active, and K is a constant equal to 60 sec/min. An approximate sensor node lifetime L can then be calculated using Equation 4.2, where I_{GPS_Active} and $I_{GPS\_Inactive}$ are the total average current consumption when the GPS is active and inactive during a 24 hour period respectively, N_{Blocks} is the number of measurement blocks per day of length t_{Active} hours and C is the battery cell capacity (mAh) with an estimated discharge efficiency β.

$$I_{\mathrm{system}} = \sum \left(\frac{\alpha_{\mathrm{i}}}{K}\right) I_{\mathrm{i}} \tag{4.1}$$

$$L = \frac{\beta C}{\left(24 - N_{\text{Blocks}} t_{\text{Active}}\right) I_{\text{GPS\_Inactive}} + N_{\text{Blocks}} t_{\text{Active}} I_{\text{GPS\_Active}}} \tag{4.2}$$

The parameters in Table 4-1 list the power components and their contributions assuming a custom PCB, where all prototype voltage regulators are consolidated, all debug LEDs and unused sensors are removed and with GPS POT always active (a realistic assumption in an alpine scenario). Using these parameters and assuming a twice daily 4 hour periodic measurement schedule, a 14000mAh battery cell with 75% efficiency, the node lifetime is approximately 40 days. The node lifetime could be extended by means of dual-battery backup and solar power charging solutions.

Power Component	**Avg. Contribution (sec/min)**	**Avg. Current (mA)**
Dozer Data TX (without GPS)	0.2	20
Dozer Data TX (with GPS)	0.4	20
GPS Module (POT mode inactive)	12	55
GPS Module (POT mode active)	48	25

Table 4-1: Summary of components and their contributions to total average current consumption measured at 5V DC.

4.3 System Deployment Considerations

In order to plan a network of GPS-equipped wireless sensor nodes, say into the existing PermaSense network, it is important to know how the network will perform in real-world deployment scenarios. The prototype GPS-equipped wireless sensor node has only been tested in an isolated single-hop topology. In a real-world deployment, it is desirable to deploy a larger number of nodes spanning a greater physical area, thus potentially requiring support for dynamic multi-hop topologies.

The Dozer sensor network protocol stack is by design a low-power and a low-bandwidth protocol stack. The success of Dozer within the PermaSense network is attributed to matching a low-bandwidth protocol stack with the low-bandwidth requirements of the sensors. With the introduction of higher-bandwidth measurement devices, such as GPS, the uplink bandwidth limitations of Dozer become evident.

The Dozer frame structure determines the uplink bandwidth for each child node. A simplified theoretical model based on this frame structure is used to analyse the link capacities for single and multi-hop network configurations using GPS measurement payload. Figure 4-7 presents the the frame structure used in the model. The nomenclature is as follows; the time between successive beacons is the beacon interval, where each beacon interval is partitioned into a finite number of slots. One of the slots is used for the parent beacon transmission, and all other slots may be allocated to child nodes for uplink transmission. Table 4-2 summarises the worst-case application-specific parameters used throughout the analysis.

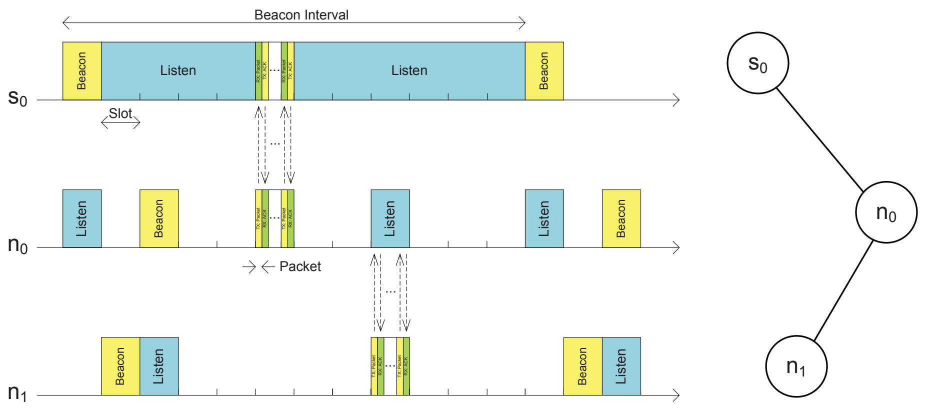


Figure 4-7
Dozer frame structure used in simplified model.

Maximum Number of Satellites	12
Radio Packet Size	50 bytes
Effective Transmission Rate	50 kbps
Beacon Interval	10 seconds
Number of Processing Slots	6

Table 4-2: Dozer network parameters used in analysis.

4.3.1 Single-hop Network Topology

In a single-hop sensor network, the transmission time between successive sink-node beacons is to be shared between a finite number of slots, where each child node uses one slot for uplink data transmission. The maximum number of child nodes (i.e. maximum number of nodes connected to the sink) is dependent on the uplink packet size and the transmission data rate. Using the parameters listed in Table 4-2, the maximum number of supported nodes can be determined by the intersection of the Dozer slot time and the required packet transmission time as illustrated in Figure 4-8.

It can be seen that 98 nodes can be supported under the worst-case GPS measurement scenario. If we were to implement some form of packet-level compression, an gain in network capacity is achieved. Assuming 10%, 20% and 30% packet-level compression, the maximum number of supported nodes is increased to 109, 124, and 142 respectively.

In a real-world deployment, the advantage with a single-hop topology is that nodes can be added (up to the maximum) and removed without risk of overloading the network. Capacity improvements can only be achieved by increasing Dozer network bandwidth and/or implementing packet-level compression. However, in certain scenarios multi-hop topology networks may provide greater flexibility, reliability and power efficiency.

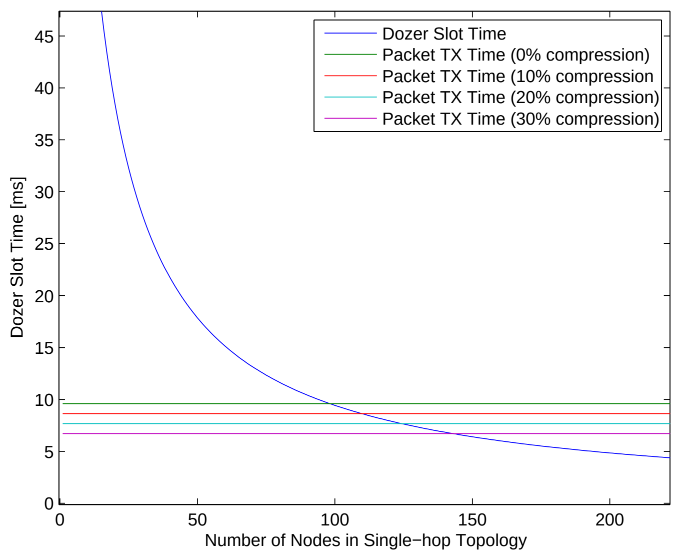


Figure 4-8
Maximum network capacity in a single-hop topology.

4.3.2 Multi-hop Network Topology

The complexity of the analysis increases somewhat in a multi-hop sensor network, as the Dozer frame structure allows for possible network topologies which may lead to uplink capacity overload. In order to calculate the maximum number of nodes supported in a multi-hop network, we must consider the worst-case uplink capacity to the sink node. The slot time allocated to a child node must be long enough to transmit the data generated by the given node and the data generated by all of the nodes' descendants. If the uplink capacity is not sufficient, the node must buffer the overflowed data (which in certain configurations may be unbounded) or drop the overflowed data (impacting the systems measurement accuracy and reliability).

The number of slots is dependent on the maximum number of children each node can support. Since the beacon interval is fixed, the number of slots determines the available uplink transmission time for each child node. Figure 4-9 plots the maximum number of children versus the maximum number of descendants each node can support within our simplified theoretical model. Then Figure 4-9 plots the maximum number of nodes that can be supported in the network, assuming a maximum hop-count of 2 (which defines the topology that gives the maximum number of nodes without uplink capacity overload). The two figures also plot the same curves assuming packet-level compression of 10%, 20% and 30%.

An interesting observation is that using a maximum number of 10 children, the maximum number of nodes reaches a local peak. This suggests that without using compression, there is negligible gain in configuring the Dozer network with any higher than 10 children per node. It can also be seen in Figure 4-9 that packet-level compression improves the network capacity in the multi-hop topology.

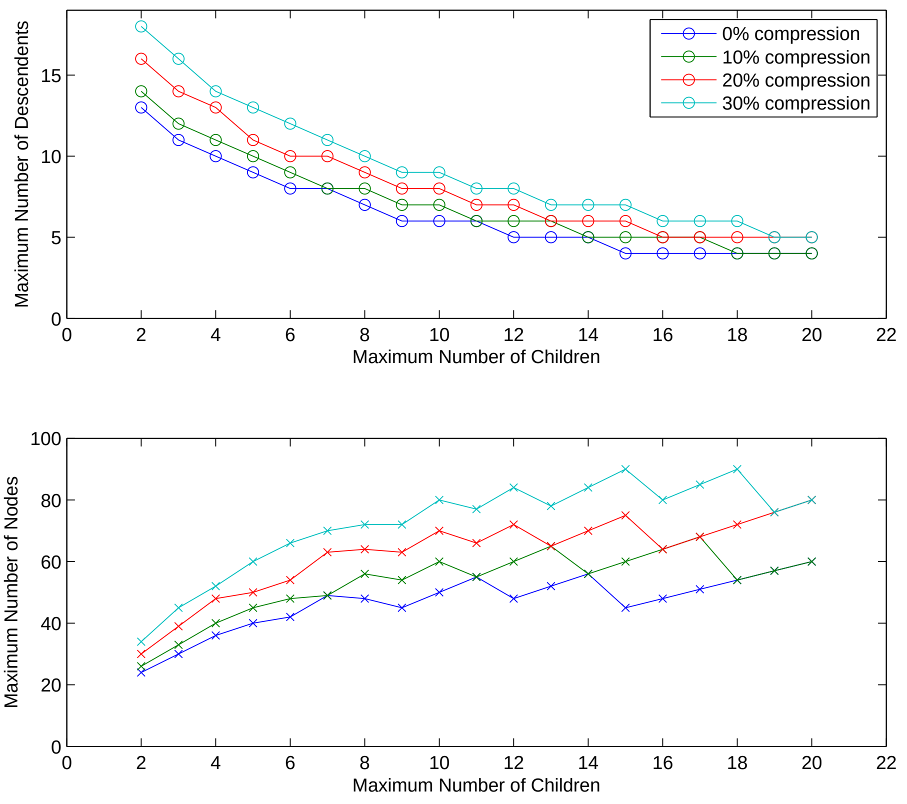


Figure 4-9
Maximum number of descendents (upper) and Maximum number of nodes (lower) in a multi-hop topology with a maximum hop-count of 2.

It is important to reiterate that the maximum number of nodes calculated in Figure 4-9 assumes an optimal topology, where the maximum hop-count is restricted to 2. If we were to relax the hop-count constraint, it is possible that a network topology may form (due to loss of link to a parent, node failures, etc.) where the uplink capacity is overloaded. Consider the example topologies in Figure 4-10, assuming a maximum number of 4 children and maximum number of 10 descendants. Figure 4-10(a) represents a valid topology where the number of descendants per link is bounded to 10, thus guaranteeing uplink data capacity to the sink node. However, if node j was to fail, node i would attempt to connect to nearby node k, as represented in Figure 4-10(b). In doing so, the number of descendants on the sink uplink now becomes 11, meaning the uplink slot time is not long enough to send all the generated data. The selection of the maximum number of hops may reduce the occurrence of the uplink capacity overload problem in some topologies, but not in general.

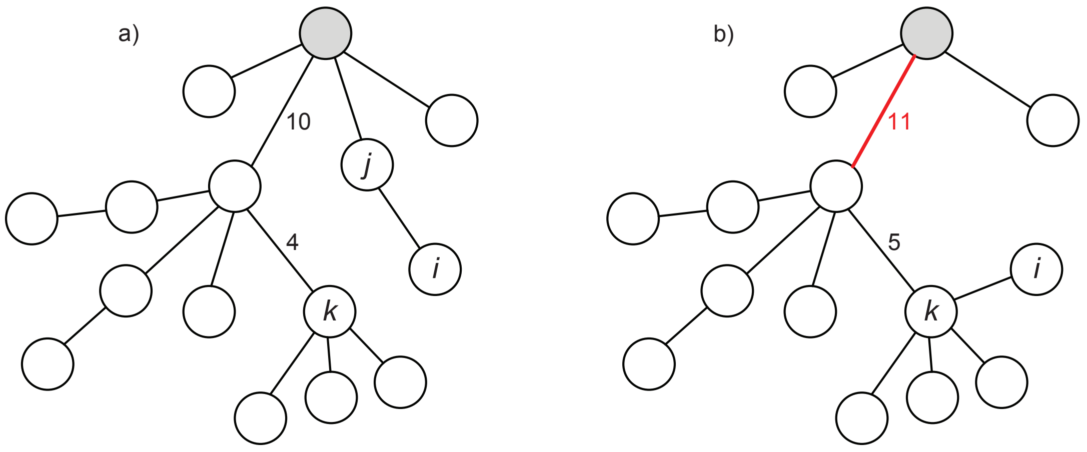


Figure 4-10
Example of uplink capacity overload in a multi-hop topology.

The topology of sensor networks is dependent on many factors including application-specific (e.g. placement of nodes), node-specific (e.g. availability of secondary storage), and network-specific (e.g. behaviour after loss of link) factors. It is not the intention here to extensively detail deployment strategies, but rather highlight some important implications for the deployment of the prototype GPS-equipped wireless sensor nodes using the low-bandwidth Dozer sensor network protocol stack. In summary, there are four deployment strategies to consider:

1. Use a single-hop topology. Calculate the worst-case data packet size, using the relation in Figure 4-8, this will determine the maximum number of nodes for the deployment. In the single-hop configuration, nodes can be added/removed without any risk of overloading the network. The limitation is that the deployment of the network is restricted to a star-topology.
2. Use a multi-hop topology. Select a maximum number of children for the network deployment, such that uplink link capacities can not be overloaded. That is, the maximum number of descendants must be less than or equal to one less the maximum number of children, and the maximum number of hops must be two. This configuration would guarantee no uplink capacity overload. This would also place a restriction on the deployment topology, as the breadth and depth of the node tree is restricted, which may affect the physical span of some deployments.
3. Use a multi-hop architecture. Select the maximum number of children for a desired network size. Accept that in certain network topologies, there will be link capacity overload. The loss of data incurred due to the link capacity overload can be mitigated by significantly increasing the nodes' uplink packet buffer or by using secondary storage to store overflow data messages from child nodes.
4. A final alternative would be to enhance the underlying network protocol, Dozer, to support higher bandwidth. The higher uplink bandwidth may not need to be constant, but rather more adaptive to selected overloaded links in

the routing tree. Such adaptive uplink bandwidth would allow for overloaded links to be flushed using a short burst of higher bandwidth, and therefore not restricting the deployment topology of the sensor network.

4.4 Payload Compression Opportunities

The single and multi-hop analysis presented in Section 4.3 have indicated promising increases in network capacity by compressing the GPS measurement payload prior to transmission using Dozer. The area of compression techniques is an extremely broad and well-researched field, and it is not the intention to recommend which algorithm or technique is best suited to in this application. However, there are a number of simple application-specific compression methods that could be implemented with minimal computational overhead on the prototype GPS-equipped wireless sensor node. These methods are loosely termed: payload bit-packing, differential encoding and measurement resolution reduction.

4.4.1 Payload Bit-packing

The byte-wise measurement data structure defined by the UBX protocol as shown in Figure 2-17 is not efficiently structured. The smallest unit of data representation is a byte, irrespective of whether a parameter requires the full byte resolution or not. For example, the space vehicle number, quality, signal strength and loss of lock indicator parameters all do not use the full 8-bit resolution. These parameters can be simply bit-packed into two bytes, as illustrated in Figure 4-11, effectively reducing the data packet size by 6.67%.

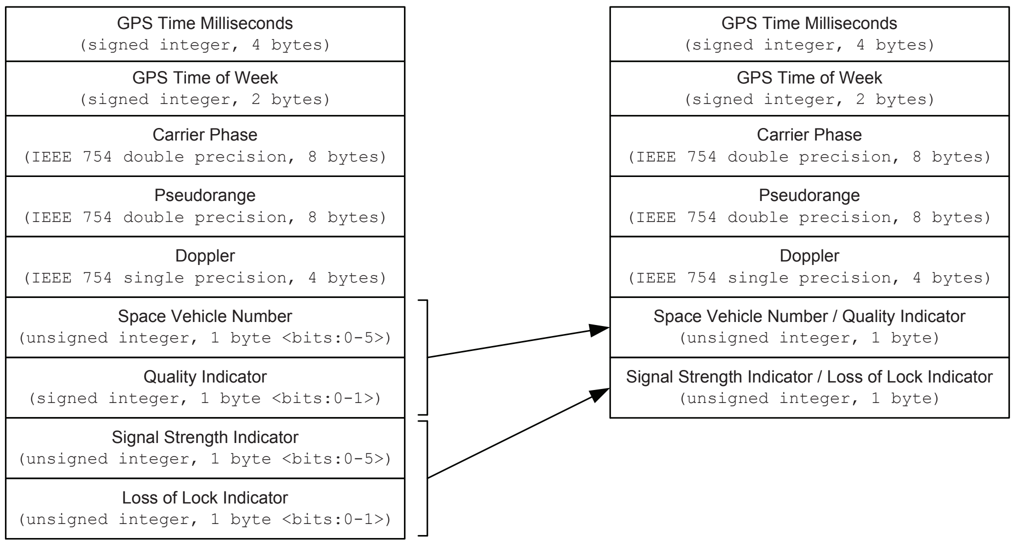

GPS Time Milliseconds
(signed integer, 4 bytes)
GPS Time of Week
(signed integer, 2 bytes)
Carrier Phase
(IEEE 754 double precision, 8 bytes)
Pseudorange
(IEEE 754 double precision, 8 bytes)
Doppler
(IEEE 754 single precision, 4 bytes)
Space Vehicle Number
(unsigned integer, 1 byte <bits:0-5>)
Quality Indicator
(signed integer, 1 byte <bits:0-1>)
Signal Strength Indicator
(unsigned integer, 1 byte <bits:0-5>)
Loss of Lock Indicator
(unsigned integer, 1 byte <bits:0-1>)
GPS Time Milliseconds
(signed integer, 4 bytes)
GPS Time of Week
(signed integer, 2 bytes)
Carrier Phase
(IEEE 754 double precision, 8 bytes)
Pseudorange
(IEEE 754 double precision, 8 bytes)
Doppler
(IEEE 754 single precision, 4 bytes)
Space Vehicle Number / Quality Indicator
(unsigned integer, 1 byte)
Signal Strength Indicator / Loss of Lock Indicator
(unsigned integer, 1 byte)

Figure 4-11
Packet-level compression using payload bit-packing.

4.4.2 Differential Encoding

Instead of sending the absolute value for all measurements per satellite, a differential packet could be used to transmit only the relative differences of each satellite measurement value. Two satellite data packets can be defined, one larger full-packet containing the absolute values of all measurements, and one smaller differential-packet containing the relative measurements, as depicted in Figure 4-12. One new status byte would need to be added to both the full and differential packets in order to indicate the packet type (full or differential) and the sequence number of sequential differential packets. The full-packet would need to be transmitted each time the GPS module adds a new satellite to its set of tracked satellites, otherwise the full-packet need only be sent after a fixed number of measurement intervals.

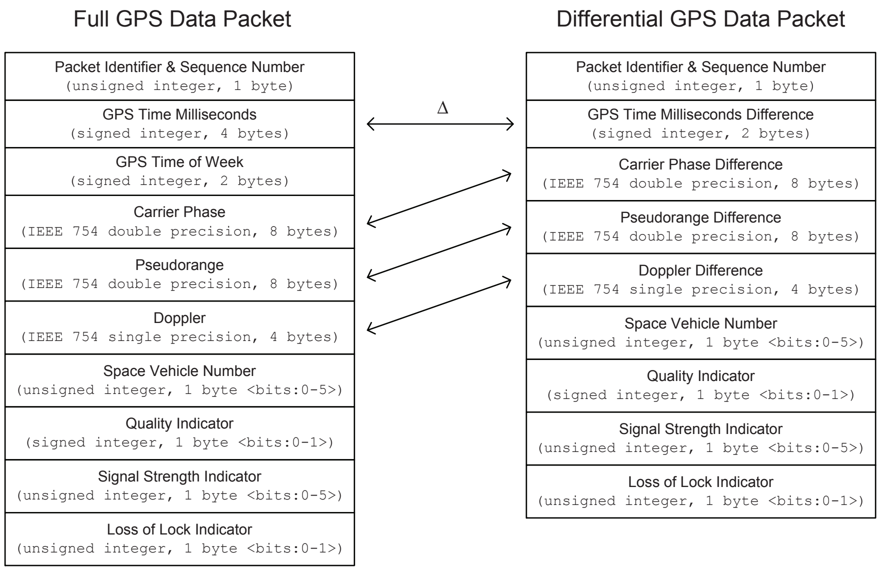

Full GPS Data Packet
Packet Identifier & Sequence Number (unsigned integer, 1 byte)
GPS Time Milliseconds (signed integer, 4 bytes)
GPS Time of Week (signed integer, 2 bytes)
Carrier Phase (IEEE 754 double precision, 8 bytes)
Pseudorange (IEEE 754 double precision, 8 bytes)
Doppler (IEEE 754 single precision, 4 bytes)
Space Vehicle Number (unsigned integer, 1 byte <bits:0-5>)
Quality Indicator (signed integer, 1 byte <bits:0-1>)
Signal Strength Indicator (unsigned integer, 1 byte <bits:0-5>)
Loss of Lock Indicator (unsigned integer, 1 byte <bits:0-1>)
Δ
Differential GPS Data Packet
Packet Identifier & Sequence Number (unsigned integer, 1 byte)
GPS Time Milliseconds Difference (signed integer, 2 bytes)
Carrier Phase Difference (IEEE 754 double precision, 8 bytes)
Pseudorange Difference (IEEE 754 double precision, 8 bytes)
Doppler Difference (IEEE 754 single precision, 4 bytes)
Space Vehicle Number (unsigned integer, 1 byte <bits:0-5>)
Quality Indicator (signed integer, 1 byte <bits:0-1>)
Signal Strength Indicator (unsigned integer, 1 byte <bits:0-5>)
Loss of Lock Indicator (unsigned integer, 1 byte <bits:0-1>)

Figure 4-12
Full and differential packets for packet-level compression.

4.4.3 Measurement Resolution Reduction

The pseudorange measurement from the GPS module is encoded in a double-precision floating point (IEEE 754) representation. The measurement value is fragmented into 8 bytes ($d_0, d_1, \ldots, d_7$) for byte-wise transmission over the UBX protocol. It was observed over number of measurement samples, as shown in Figure 4-13, the most significant byte of the pseudorange measurement d_7 does not change between successive measurement samples. The d_7 byte represents the sign bit and the highest 7 bits of the IEEE 754 exponent. The sign bit is unexpected to change from zero as a negative pseudorange is invalid, while the higher exponent bits are unlikely to change drastically due to the magnitude of the satellite trajectory. Therefore, the most significant byte of the pseudorange value could be removed, or at the very least, only transmitted when it changes. A similar technique could also be applied to the carrier phase values.

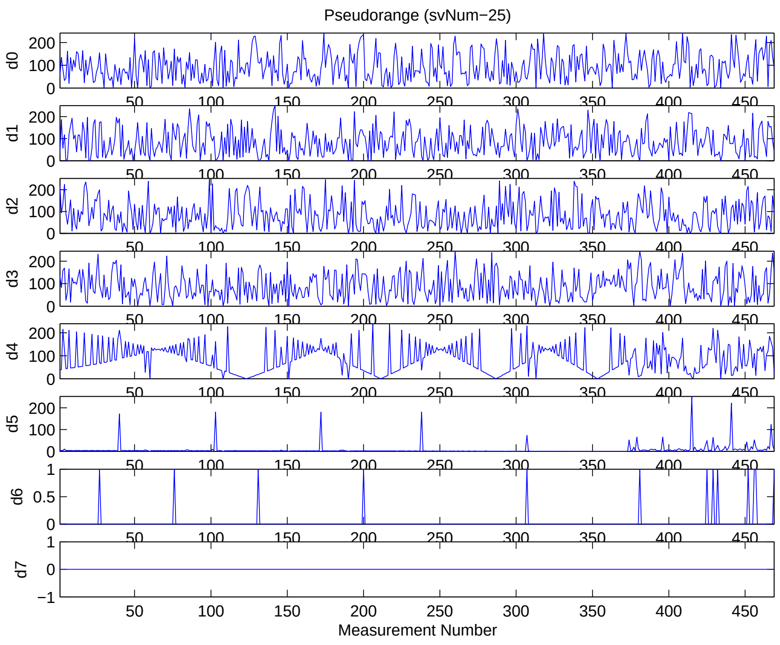


Figure 4-13
A byte-wise analysis of the difference between consecutive pseudorange measurements.

4.5 Summary

In this chapter the functional performance of the prototype GPS-equipped wireless sensor node was verified by analysis of captured measurements taken during outdoor experiments.

A detailed power profile of the prototype was presented, highlighting the Dozer network activity and GPS measurement power components. A node lifetime of approximately 40 days was estimated assuming a fully-integrated printed circuit board operating on a single battery cell.

A simplified model of the Dozer frame structure was used to investigate the scalability of deploying the GPS-equipped wireless sensor nodes in single and multi-hop topologies. The analysis found that a maximum of 98 nodes could be supported in a single-hop topology. In a multi-hop topology, it was found that certain topologies will lead to an uplink capacity overload within the network. The analysis also indicated promising capacity increases using packet-level compression. Three packet-level compression opportunities were presented, namely payload bit-packing, differential encoding and measurement resolution reduction.

5

Conclusion & Future Work

A system design of a GPS-equipped wireless sensor node was presented for monitoring the movement of alpine rock formations. A prototype was successfully implemented and tested in laboratory and outdoor conditions, where all functional requirements were successfully verified.

A power consumption analysis was performed on the prototype sensor node. Initial power estimates indicated a conservative node lifetime of approximately 40 days assuming a fully-integrated printed circuit board operating on a single battery cell. It is anticipated that the node lifetime can be significantly extended by harnessing advanced power management techniques such as dual-battery backup and solar power charging solutions.

A simplified model of the underlying wireless sensor network protocol, Dozer, was used to investigate the scalability of deploying the GPS-equipped wireless sensor nodes in single and multi-hop topologies. In a single-hop network a maximum of 98 sensor nodes could be supported without risk of overloading the network. However, in a multi-hop network it was found that certain topologies will lead to an uplink capacity overload which may result in measurement data loss.

A number of deployment strategies were suggested to mitigate the potential loss of data, however, at the cost of restricted deployment topology and increased uplink buffer requirements. One alternative approach is to enhance the Dozer network protocol to support higher bandwidth. The increase in bandwidth could be adaptive, which would allow for overloaded links to be flushed using a short burst of higher bandwidth, and therefore not restricting the deployment topology of the sensor network.

The network analysis also showed that packet-level compression is a promising method to improve node capacity in single and multi-hop network topologies. A number of compression opportunities were presented including payload bit-packing, differential encoding and measurement resolution reduction, which all could be implemented with minimal computational overhead. A deeper analysis of compression algorithms on the measurement payload may lead to even further improvements.

A

Developer's Guide to the GPS-equipped Wireless Sensor Node

The following sections describe the steps involved in building, configuring and flashing a TinyNode184 as a GPS-equipped wireless sensor node (using the GPSDozer application) and as a sink node (using the DozerBase application). A description of how to extend the Dozer message types is also provided.

A.1 GPS-equipped Wireless Sensor Node

1. Check out the PermaSense SVN repository.

   ```
   # mkdir ~/permasense_svn
   # cd ~/permasense_svn
   # svn checkout svn://svn.ee.ethz.ch/permasense/trunk/soft/tinyos-2.x
   ```

2. Check out the GPS-equipped Wireless Sensor Node SVN repository.

   ```
   # mkdir ~/gps_wsn_svn
   # cd ~/gps_wsn_svn
   # svn checkout svn://svn.ee.ethz.ch/buchli/SA2010/HS10
   ```

3. Copy the GPSDozer application files into the PermaSense TinyOS application directory.

   ```
   # cp ~/gps_wsn_svn/HS10/code/GPSDozer ~/permasense_svn/tinyos-2.x/apps/
   ```

4. Extract the following archive containing the modified Dozer library files:

   ```
   ~/gps_wsn_svn/HS10/code/TinyOS_Dozer_Directory/tos_directory.tar.gz
   ```

   Copy the following files into the PermaSense TinyOS directory:

   ```
   ./chips/msp430/statecounter/McuSleepC.nc
   ./chips/msp430/statecounter/StateCounterC.nc
   ./lib/DataControl/datacontrol.h
   ./lib/DozerConfig/DozerConfigP.nc
   ./lib/dozer/DozerConfiguration.h
   ./lib/dozer/DozerMessages.h
   ./lib/dozer/circqueue.h
   ./lib/serial/SerialDispatcherC.nc
   ./platforms/tinynode184/PlatformSerialC.nc
   ./platforms/tinynode184/TinyNodeSerialP.nc
   ```

5. Modify the GPS measurement schedule by updating the *MEAS_SCHEDULE_ACTIVE* and *MEAS_SCHEDULE_SLEEP* definitions located in file:

   ```
   ~/permasense_svn/tinyos-2.x/apps/GPSDozer/GPSModule.h
   ```

6. Connect the PC to the TinyNode184 extension board using a serial over USB cable. Take note of the Unix serial device name.

7. Build the GPSDozer application and flash the TinyNode184 using the custom script *flash_node.sh*, passing the serial device name as an input argument.

   ```
   # ~/permasense_svn/tinyos-2.x/apps/GPSDozer/flash_node.sh /dev/ttyUSB0
   ```

The *flash_node* script builds the tiny node application into an Intel hexadecimal binary file. The node ID of the sensor node is configured by modifying the binary file using the *tos-set-symbols* command. The modified binary file is then written to TinyNode184 program flash using the Texas Instruments MSP430 flash tool.

8. The u-blox LEA-6T GPS module is configured using the u-center [26] GPS evaluation and visualisation tool. The following configuration file must be loaded into the LEA-6T for the GPS-equipped wireless sensor node application:

   ```
   # ~/gps_wsn_svn/HS10/code/u-blox_configuration/lea_6t_gps_wsn_config.txt
   ```

9. Connect all serial and power cables between the TinyNode184 and the GPS module, and perform a hardware reset of the TinyNode184.

A.2 Wireless Sink Node

1. Build and flash a TinyNode184 platform with the default DozerBase application from the PermaSense SVN using the following command:

   ```
   # cd ~/permasense_svn/tinyos-2.x/apps/DozerBase
   # make clean
   # make tinynode184 install,bsl auto
   ```

   The default sensor node ID is zero, which defines a sink node in a Dozer network.

2. Start the *SerialForwarder* Java program using the command:

   ```
   # java net.tinyos.sf.SerialForwarder &
   ```

   This program receives ActiveMessages from the TinyNode184 over the specified serial interface, and pipes the data onto a specified UDP socket (i.e *sf@localhost:9002*) where multiple applications can simultaneously read and process the received packets.

3. Send the sink node a beacon request command using the following command:

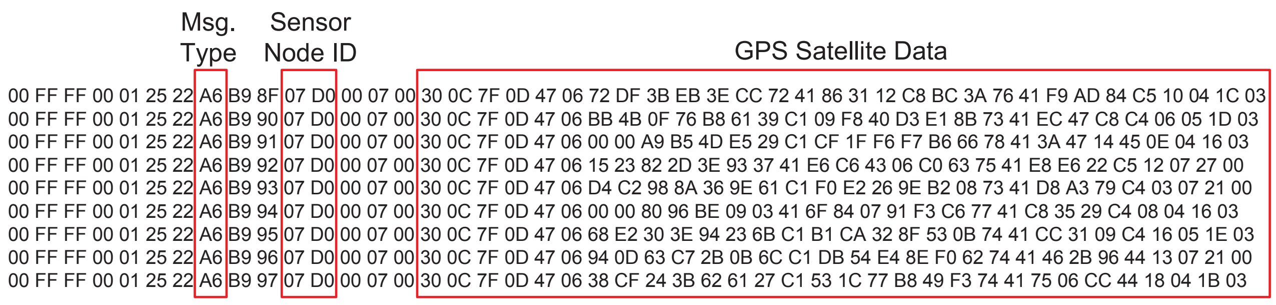
   ```
   # java net.tinyos.tools.Send 00 FF FF 00 00 05 22 50 ff ff 01 00 01
   ```

   This command will instruct the sink node to periodically send beacon messages. The sink node will eventually connect to the GPS-equipped wireless sensor node.

4. Start the *Listen* Java application using the following command:

   ```
   # java net.tinyos.tools.Listen
   ```

   The application receives the packets from the *sf@localhost:9002* UDP socket and prints the hexadecimal data to standard output. This is the most basic test to verify the reception of satellite measurement data. An example of one satellite measurement is listed in Figure A-1, where each line represents one satellite data packet, 0XA6 is the *AM_DATAGPS* message type, 0X07 0XD0 is the default GPS-equipped wireless sensor node ID (2000), and the last 30 bytes is the satellite data payload.



```
00 FF FF 00 01 25 22 A6 B9 8F 07 D0 00 07 00 30 0C 7F 0D 47 06 72 DF 3B EB 3E CC 72 41 86 31 12 C8 BC 3A 76 41 F9 AD 84 C5 10 04 1C 03
00 FF FF 00 01 25 22 A6 B9 90 07 D0 00 07 00 30 0C 7F 0D 47 06 BB 4B 0F 76 B8 61 39 C1 09 F8 40 D3 E1 8B 73 41 EC 47 C8 C4 06 05 1D 03
00 FF FF 00 01 25 22 A6 B9 91 07 D0 00 07 00 30 0C 7F 0D 47 06 00 00 A9 B5 4D E5 29 C1 CF 1F F6 F7 B6 66 78 41 3A 47 14 45 0E 04 16 03
00 FF FF 00 01 25 22 A6 B9 92 07 D0 00 07 00 30 0C 7F 0D 47 06 15 23 82 2D 3E 93 37 41 E6 C6 43 06 C0 63 75 41 E8 E6 22 C5 12 07 27 00
00 FF FF 00 01 25 22 A6 B9 93 07 D0 00 07 00 30 0C 7F 0D 47 06 D4 C2 98 8A 36 9E 61 C1 F0 E2 26 9E B2 08 73 41 D8 A3 79 C4 03 07 21 00
00 FF FF 00 01 25 22 A6 B9 94 07 D0 00 07 00 30 0C 7F 0D 47 06 00 00 80 96 BE 09 03 41 6F 84 07 91 F3 C6 77 41 C8 35 29 C4 08 04 16 03
00 FF FF 00 01 25 22 A6 B9 95 07 D0 00 07 00 30 0C 7F 0D 47 06 68 E2 30 3E 94 23 6B C1 B1 CA 32 8F 53 0B 74 41 CC 31 09 C4 16 05 1E 03
00 FF FF 00 01 25 22 A6 B9 96 07 D0 00 07 00 30 0C 7F 0D 47 06 94 0D 63 C7 2B 0B 6C C1 DB 54 E4 8E F0 62 74 41 46 2B 96 44 13 07 21 00
00 FF FF 00 01 25 22 A6 B9 97 07 D0 00 07 00 30 0C 7F 0D 47 06 38 CF 24 3B 62 61 27 C1 53 1C 77 B8 49 F3 74 41 75 06 CC 44 18 04 1B 03
```

Figure A-1
Example GPS satellite data output from Listen tool.

A.3 Extending Dozer Message Types

The following steps need to be performed when adding a new Dozer message type (i.e. a new message type for differential packet transmission).

1. Add the new ActiveMessage type to the enumeration in file: *./lib/dozer/DozerMessages.h*

```
enum {
    AM_DATANODEHEALTH    =  0x80,
    AM_DATANODEINFO      =  0x82,
    AM_DATAADCDIFF       =  0x84,
    AM_DATADIGITALDCX    =  0x86,
    AM_DATAADCMUX1       =  0x8A,
    AM_DATAADCMUX2       =  0x8C,
    AM_DATAWXT520WINDPTH =  0x92,
    AM_DATAWXT520PREC    =  0x94,
    AM_DATAWXT520SUP     =  0xA4,
    AM_DATADECAGONMUX    =  0xA0,
    AM_DATAGPS           =  0xA6,  // e.g. new message type
};
```

2. Add the new Dozer data source message type associated with the ActiveMessage to the enumeration in file: */tos/lib/DataControl/datacontrol.h*.

```
enum {
    DATA_SOURCE_INFO,
    DATA_SOURCE_HEALTH,
    DATA_SOURCE_ADCMUX,
    DATA_SOURCE_ADCCOMDIFF,
    DATA_SOURCE_DCXLOGGER,
    DATA_SOURCE_DRIFT,
    DATA_SOURCE_EVENTS,
    DATA_SOURCE_RSSI,
    DATA_SOURCE_STATECOUNTER,
    DATA_SOURCE_DECAGONMUX,
    DATA_SOURCE_WXT520,
    DATA_SOURCE_GPS, // e.g. new message type
    NUM_SOURCES // last entry in enumeration
};
```

3. Add the new Dozer data source message type to the DATA_DATACONTROL_CONFIG bit field in file: */tos/lib/DozerConfig/DozerConfigP.nc*.

```
#define DOZER_DATACONTROL_CONFIG \
    1<<DATA_SOURCE_INFO |\
    1<<DATA_SOURCE_HEALTH |\
    0<<DATA_SOURCE_ADCMUX |\
    0<<DATA_SOURCE_ADCCOMDIFF |\
    0<<DATA_SOURCE_DCXLOGGER |\
    0<<DATA_SOURCE_DRIFT |\
    1<<DATA_SOURCE_EVENTS |\
    1<<DATA_SOURCE_RSSI |\
    1<<DATA_SOURCE_STATECOUNTER |\
    0<<DATA_SOURCE_DECAGONMUX |\
    1<<DATA_SOURCE_WXT520 |\
    1<<DATA_SOURCE_GPS  // e.g. new message type
```

4. If the new message type increases the Dozer maximum message size of 30 bytes, change the MAX_DATA_LENGTH enumeration in *./lib/dozer/DozerMessages.h* file to the new maximum size. Note that the *Makefile* that is used to build the TinyOS application may override this definition:

```
CFLAGS+=DDOZER_CLEAN_QUEUE=1 -DTOSH_DATA_LENGTH=37
```

5. Ensure that the TinyOS top-component wires the Dozer DataControl component, indexed by the enumeration, to the new DataControl component that handles the new message type.

```
DataControlC.DataSourceControl[DATA_SOURCE_GPS]->DataSourceGPSData;
```

Bibliography

[1] *PermaSense Project*, `http://www.permasense.ch/`.

[2] P. Guillet, *Online GPS for PermaSense WSN*, Semester Thesis, Department of Information Technology and Electrical Engineering, ETH Zürich, October 2010.

[3] J.M Zogg, "GPS Compendium: Essentials of Satellite Navigation", `http://www.u-blox.com/images/downloads/Product_Docs/GPS_Compendium%28GPS-X-02007%29.pdf`, February 2009.

[4] E. Kaplan, *Understanding GPS: Principles and Applications*, Artech House, 1996.

[5] N. Burri, P. von Rickenbach, and R. Wattenhofer, "Dozer: Ultra-Low Power Data Gathering in Sensor Networks", *Proceedings of the 6th international conference on Information Processing in Sensor Networks*, April 2007.

[6] *TinyOS project*, `http://www.tinyos.net/`.

[7] D. Gay et al., "The nesC Language: A Holistic Approach to Networked Embedded Systems", *Proceedings of Programming Language Design and Implementation*, June 2003.

[8] D. Doolin, and N. Sitar, "Wireless sensors for wildfire monitoring", *Proceedings of Smart Structures and Materials*, May 2005.

[9] R. Stoleru, T. He, and J.A. Stankovic, "Walking GPS: A Practical Solution for Localization in Manually Deployed Wireless Sensor Networks", *29th Annual IEEE International Conference on Local Computer Networks (LCN'04)*, November 2004.

[10] K. Martinez, J. Hart, and R. Ong, "Environmental Sensor Networks", *IEEE Computer*, August 2004.

[11] P. Limpach, and D. Grimm, "Rock glacier monitoring with low-cost GPS receivers," *Abstract Volume 7th Swiss Geoscience Meeting*, Neuchatel, Switzerland, November 2009.

[12] A. Hasler et al., "Wireless Sensor Networks in Permafrost Research: Concept, Requirements, Implementation and Challenges", *Proceedings of the 9th International Conference on Permafrost*, Alaska, USA, 2008.

[13] *TinyNode184 Datasheet*, http://www.tinynode.com/?q=system/files/TinyNode%20Users%20Manual%20rev12.pdf, 2010.

[14] *Texas Instruments MSP430 Datasheet*, http://www.ti.com/lit/gpn/msp430f2417, 2009.

[15] *Semtech SX1211 Transceiver Dataasheet*, http://www.semtech.com/apps/filedown/down.php?file=sx1211.pdf, 2008.

[16] *National Marine Electronics Association 1083 Standard*, http://www.nmea.org/content/nmea_standards/nmea_083_v_400.asp, 2010.

[17] *u-blox 6 Receiver Description Including Protocol Specification*, http://www.u-blox.com/images/downloads/Product_Docs/u-blox6_ReceiverDescriptionProtocolSpec_%28GPS.G6-SW-10018%29.pdf, 2010.

[18] *u-blox LEA-6T GPS Module Data Sheet*, http://www.u-blox.com/images/downloads/Product_Docs/LEA-6_DataSheet_%28GPS.G6-HW-09004%29.pdf, 2010.

[19] *TinyOS TEP 106: Schedulers and Tasks*, http://www.tinyos.net/tinyos-2.x/doc/pdf/tep106.pdf.

[20] P. Levis, "TinyOS Programming", http://www.tinyos.net/tinyos-2.x/doc/pdf/tep111.pdf, October 2006.

[21] *TinyOS TEP 102: Timers*, http://www.tinyos.net/tinyos-2.x/doc/pdf/tep102.pdf.

[22] *TinyOS TEP 111: message_t*, http://www.tinyos.net/tinyos-2.x/doc/pdf/tep111.pdf.

[23] I. Sommerville, *Software Engineering - 6th edition*, Addison-Wesley, 2001.

[24] *ETH Computer Engineering and Networks Laboratory - PermaSense Wiki*, http://people.ee.ethz.ch/~nccr/permasense/.

[25] *Global Sensor Networks (GSN) Project*, http://sourceforge.net/apps/trac/gsn/.

[26] *u-center GPS Evaluation Software User Guide*, http://www.u-blox.com/images/downloads/Product_Docs/u-Center_User_Guide%28GPS-SW-08007%29.pdf, 2010.