Secure and scalable dataprocessing and communications for an Animal Tracking System

by

Yngve F. Johansen
yfjohans@siving.hia.no
Øystein Vinningland
ovinning@siving.hia.no

Master Thesis in
Information and Communication Technology

Agder University College

Grimstad, 29th May 2003

ABSTRACT

Modern technology is constantly being employed in new fields, by new industries and people. The cost of advanced technological equipment decreases continously, and equipment that was reserved for people and corporations with a lot of money has now become household products.

This enables many interesting ideas and projects, Kitron Development's radio-collar project being one of them. Kitron Development is involved in a project to track the position of animals by substituting the traditional bell collar with a collar that contains a GPS receiver. The collar transmits its position to a base station using VHF radio, and the base station forwards the received data to a central using cellular technology.

In this thesis we have surveyed, designed and implemented the central, called Animal Tracking System, that receives and processes data from the base stations. The Animal Tracking System consists of four different components that work together. A server that communicates with the base station, and stores data in a database, an alarm monitor that survey the data in the database and a web application that allow the farmers to view data about their animals, as well as configure the system.

Keywords: Security, Scalability, Python, Java, Servlets, GPRS, GSM, DBMS, SQL

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

0. PREFACE

This thesis is the final project by Yngve Farbu Johansen and Øystein Vinningland at Agder University College, faculty of engineering and science during Spring of 2003.

First of all we would like to thank our supervisor, Rune Fensli, for valuable guidance during the project. Thanks to Kitron Development, especially Erik Hardeng and Ole Jørgen Lium, for giving us the opportunity to participate in such an interesting project and for valuable help and advice. We would also like to thank our fellow students and friends for the discussions and support these months, and Bent Andre Solheim for providing us with help and guidance regarding LaTeX. Finally we would like to thank the director of studies, Stein Bergsmark, for all tips and information given this Spring.

Grimstad, May 2003

Yngve Farbu Johansen and Øystein Vinningland

1. INTRODUCTION

1.1 Thesis Definition

The following is our thesis definition, on which our work is based.

Kitron Development has been working on a system for tracking domesticated animals, with the main focus on sheep, for about two years. They have developed a "radio-bjelle" that contains a GPS receiver and a VHF transmitter that transmits the received GPS data along with possible alarms, and other parameters to a base station. The base station then sends its gathered data to a central server for processing, and as Kitron Development is preparing for a pilot test of the animal tracking system during the summer 2003, this master thesis will be an important part of developing a central server solution for this new Animal Tracking System.

The thesis will focus on evaluation of possible technologies and solutions between the base station and the central server, with respect to cost, scalability, fail-safety and security in the design of the application and the communication principles. A study shall be conducted regarding construction of a central database for storing the logged data from the animals together with their tracking data, where it also is possible to process analysis of the tracking data and trigger predetermined alarms based on different events. A demo of this database is to be tested as an implemented part of Kitron´s central server solution.

1.2 Background

Each year more than 2.2 million sheep grazes in Norway. By various causes, 100.000 of these die during the three and a half months long grazing season. This means 1000 sheep per day, or 40 per hour [1]. These numbers are the background for Kitron Development's radio collar project. The project was originally started by Grønvold Matheson Technology. Kitron Development became involved at a later stage and are now in charge of developing the system. The project is supported by the Ministry of Agriculture and has recently been mentioned in national media.

1.3 Description of Animal Tracking System

Each animal is equipped with a radio collar instead of the traditional bell. This collar transmits its posistion by VHF communication to a nearby base station. The main purpose of the base station is to receive the messages from the radio collar and forward them to our server by wireless modem (GSM/GPRS). The server process the incoming data and stores it in the database. An alarm monitor checks the database continuously to detect any irregularites which should trigger an alarm, and then alerts the farmer. The farmer can log in to the web application which gives acess to the farmer's requested information from the database. Figure 1.1 is an illustration of the complete system.

1.4 Introduction to GPS

A GPS (Global Positioning System) terminal enables you to establish your position at any time, anywhere on the surface of the earth. GPS was developed by the US Department of Defense to provide all-weather round-the-clock navigation capabilities for military ground, sea, and air forces. Since its implementation, GPS has also become an integral asset in numerous civilian applications and industries around the globe, including recreational uses.

GPS is a navigation system that employs 24 satellites orbiting 20 200 km above earth. The system functions everywhere, 24 hours a day and as opposed to astronomical navigation, the GPS technology is indepent of meteorological conditions. The GPS system concists of 24 satellites designated NAVSTAR (Navigation System with Timing And Ranging). 21 of these is active at all times, while the rest functions as backup. The NAVSTAR satellites continuous transmitts signals containing the satellite's position, the date and time. With a GPS terminal you are able to recieve these signals. If the reciever has contact with at least three of the satellites it can calculate your position with degree of longitude and latitude, together with speed and direction. If the reciever has contact with four or more satellites it can also calculate your position 3-dimensionally. Today the accuracy of good GPS recievers is typically 5-8 meters [2].

1.5 Limitations

In section 1.1 we wrote that a part of the technological solution is to evaluate possible technologies and solutions between the base station and the central server, with respect to, among others, fail-safety. At an early stage of the project we were informed by Kitron Development that fail-safety was an issue we did not have to take into consideration, as they felt it was not as important as other issues. We have

therefore excluded it as a part of this assignment, both in the theoritical study and the prototype implementation.

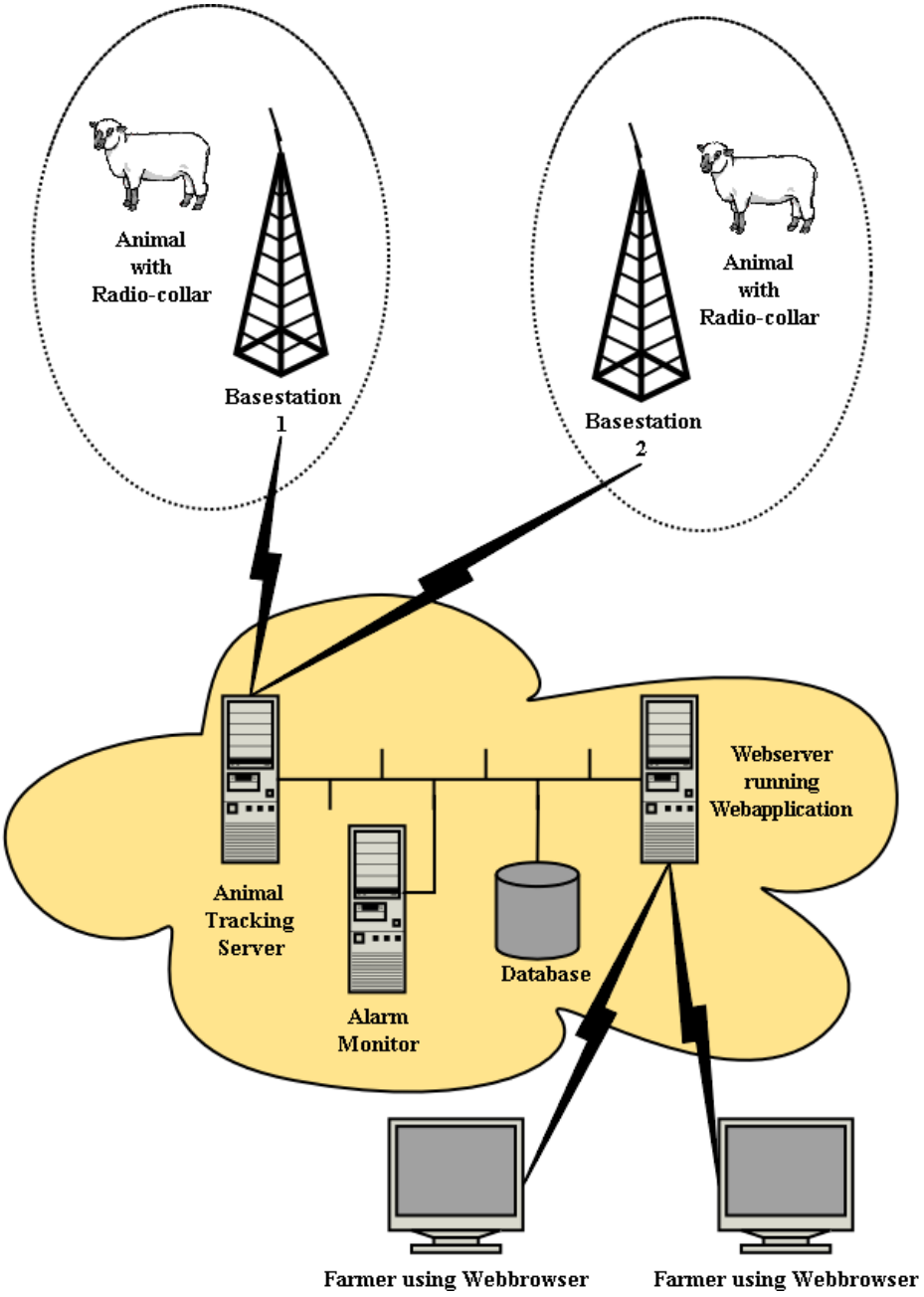


Figure 1.1: *An overview of the Animal Tracking System*

2. PRESTUDY

2.1 Alternative Communication Technologies

2.1.1 GSM

Today's second-generation GSM networks deliver high quality and secure mobile voice and data services (such as SMS Messaging) with full roaming capabilities across the world and the GSM platform is a very successful wireless technology. GSM was first introduced in 1991 and is the worlds leading and fastest growing mobile standard. More than one tenth of the worlds population in 174 countries use GSM. The number of subscribers is expected to surpass one billion at the end of 2003 [3].

In 1982 the Conférence Européenne des Postes et des Télécommunications (CEPT) formed the Groupe Spécial Mobile (GSM) to make a pan-European digital cellular system. In 1989, the European Telecommunications Standards Institute (ETSI) made the GSM specifications.

The GSM commercial service was started in 1991, but handset were not readily available before 1992. In 1993, there were 36 GSM networks in 22 countries, including non-European countries such as Australia and South Africa. Today, there are more than 470 GSM operators in 174 countries, and by January 2002 there were 646 million users.

GSM allows up to eight users to share a single 200 kHz radio channel by allocating a unique time slot to each user. This technique is called TDMA. GSM is used in the 900 and 1800 MHz bands all over the world except North America (which uses the 1900 MHz band). Eventually, new frequencies will be used in the 450 and 850 MHz bands. Since the beginning, GSM has offered SMS, a connectionless packet service limited to transmitting text messages containing less than 160 characters. Data transfer is also made possible using a service called circuit-switched data (CSD), which offers throughput up to 14.4 kbps. These limitations led to the standardization of the High Speed Circuit Switched Data (HSCSD) and General Packet Radio Service (GPRS).

2.1.2 SMS

The Short Message Service (SMS) is the ability to send and receive text messages to and from mobile telephones. The text can comprise of words or numbers or an alphanumeric combination. SMS was created as part of the GSM Phase 1 standard. The first short message is believed to have been sent in December 1992 from a personal computer to a mobile phone on the Vodafone GSM network in the UK. Each short message is up to 160 characters in length – when Latin alphabets are used.

2.1.3 HSCSD

High Speed Circuit Switched Data (HSCSD) is an enhancement of data services (Circuit Switched Data - CSD) of all current GSM networks. Non voice services are three times faster than normal, which means subscribers are able to send and receive data from their portable computers at a speed of up to 28.8 kbps; this is currently being upgraded in many networks to rates of and up to 43.2 kbps. The theoretical maximal data transfer speed is 57.6 kbps, but in use this transfer rate does not occur.

The HSCSD solution enables higher rates by using up to four communication channels, allowing subscribers to enjoy faster rates for their Internet, e-mail, calendar and file transfer services. HSCSD is currently (2002) available to 90 millions subscribers across 25 countries around the world.

HSCSD is offered to subscribers using either voice terminals that support the feature, or a PCMCIA portable computer card, with a built in GSM phone. This means that connection of a HSCSD mobile phone or a PCMCIA card with a computer can turn notebook computers and other portable devices into a mobile networking device with the ability to transfer data at HSCSD speeds, as well as make voice calls on the computer/portable device.

As mentioned, HSCSD enables higher rates, but like CSD it is circuit-based. Therefore, it is inherently not efficient for bursty traffic. This weakness of HSCSD has contributed to only around 30 operators having introduced it so far. Most operators use or will introduce GPRS instead. A positive effect of the circuit-based technique though, is that once a connection is established, the data capacity will be constant through the whole duration of the connection, as channels that are allocated are not lost until disconnection.

2.1.4 GPRS

GPRS, General Packet Radio Service, is often called generation 2.5 in mobile communication. The technology lies between GSM (2nd generation) and UMTS (3rd

generation). One of the characteristics of GPRS is persistent connection to the Internet, another is high speed data transfer. But the GPRS technology is really nothing more than a development of GSM, from lineswitching to packetswitching. GPRS is a technology that transfers data through packetswitching technology with speed up to 171.2 kbps, which is 17 times faster than GSM. This speed enables the first video solutions on the cellular phone and transferal of datafiles with sound. At the same time the terminal has a persistent connection which means that a user do not have to wait for the connection to be set up, as you have to with GSM. Usually users will be invoiced for the quantity of data sent or received.

Limitations: The maximum speed, 171.2 kbps, assumes that all eight timeslots (four in and four out) are used and the terminal is capable of this. If many users are on the GPRS-net at the same time, the maximum speed will be reduced. GPRS is not very secure for users travelling at high speed, for instance on a train or a car.

2.1.5 Cost Study

Introduction

During the project we have performed a cost study for three different wireless network transmission technologies, GSM, GPRS and SMS. In Section 2.1.3 we look at the HSCSD technology, but we have not taken it into account in our cost study. HSCSD enables higher throughput by bundling several GSM channels, but at the same time increasing the cost. The amount of data transmitted by the base stations is relatively small, and there is no need for high throughput. For instance if data for 500 animals are transmitted it will take approximately 40 seconds to transfer using standard GSM (9600 bps).

Some of the foundation we have based our cost study upon are shown in Table 2.2 and Table 2.1. All the prices given are NOK each per month. Even though VAT is not included for Netcom's prices in these tables, we have included it in all the graphs in this section. In addition to the price information we assumes that the base station transmits the animals' position four times each day, and for all the animals at the same time. The size of each message is 92 bytes including receipt.

Table 2.1: *GSM Price Information*

Price Element	**DataCall (Netcom)***	**BareData (Telenor)**
Monthly costs	NOK 125	NOK 50
Price per minute	NOK 0.80	NOK 0.89
Start-up cost	NOK 0.29	NOK 0.59
SMS	NOK 0.75	NOK 0.89

*VAT not included

Table 2.2: *GPRS Price Information*

Price Element	**GPRS (Netcom)***	**GPRS (Telenor)**
Price per KB 0 - 0.5 MB	N/A	NOK 0.10
Price per KB 0 - 1 MB	NOK 0.08	N/A
Price per KB 0.5 - 20 MB	N/A	NOK 0.015
Price per KB more than 1 MB	NOK 0.016	N/A
Price per KB more than 20 MB	N/A	NOK 0.01

*VAT not included

In addition to the information mentioned above, we have also ignored the registration fee concerning the various services in the cost study. This is done because you must pay the same fee for all three services and the difference between Telenor and Netcom is very small. If the system is operational over a period of time the influence of the registration fee is insignificant.

In our cost study, only messages concerning positions have been taking into consideration. Alarms and other messages from the base station to the server will influence the cost, but since the system is not operational yet it is virtually impossible to predict how many of these will be sent each month.

GSM

With regard to the GSM part of our cost study we found that the connection fee plays a significant role. If the number of animals are limited, the connection fee is actually more expensive than the transfer itself.

The maximum number of animals a base station is able to handle at the current stage is 7000, that is the reason our graph extends to that number. From the graph (Figure 2.1) we see that Netcom is cheaper than Telenor unless the number of animals is very high.

GRPS

As opposed to GSM, GPRS has a persistent connection and therefore no connection fee.

GPRS also differs from GSM with regards to which operator that is cheapest. The two graphs (Figure 2.2) crosses at around 75 animals, after that Telenor is cheapest. The reson for this is given in Table 2.2 and Table 2.1. There we saw that Telenor is cheaper than Netcom when the data amount surpasses 0.5 MB per month. Even

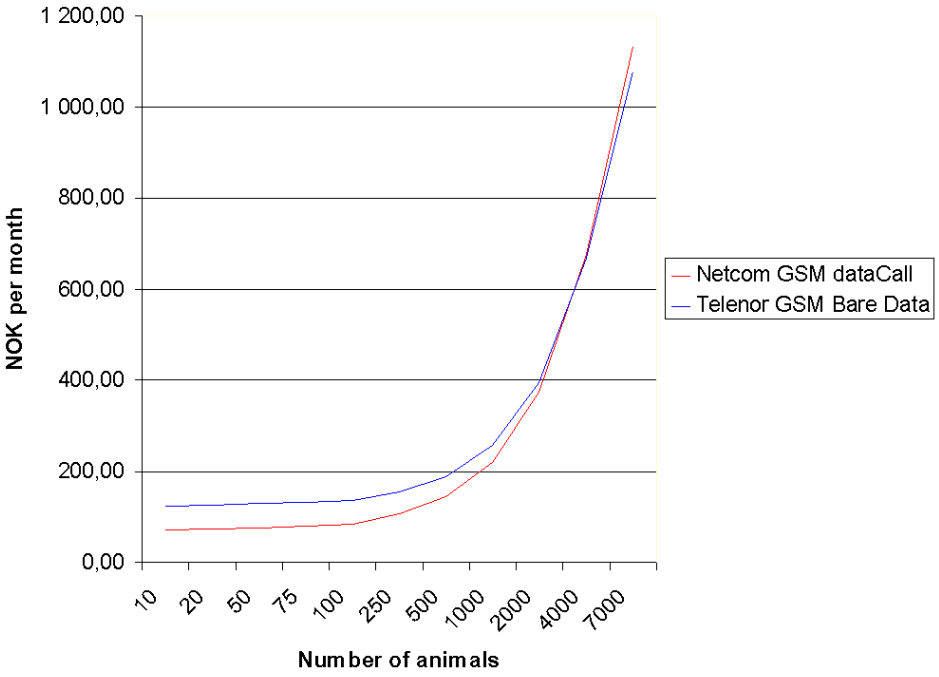


Figure 2.1: *GSM graph*

though the data amount surpasses 0.5 MB per month when then number of animals are just under 50, Telenor still need some additional data to "catch up with" Netcom.

SMS

With SMS you can send messages up to 160 characters, but the default character set is limited to 128 different characters [4]. The current message format is not very suitable for transmission using SMS. If the system were to use SMS as the transmission technology, one would probably change the message format to suit the SMS alphabet. Since the message format was specified by Kitron Development from the start (Appendix A.1) the calculation are based on a primitive message coding format using two characters to represent one byte. By using this method we have 128*128 different characters, but it is a highly innefficent method. To make the method somewhat more efficient, we have stripped the messages by removing the error correction codes (ECC). Further more we do not send receipt with the SMS message, and thereby bringing the messagesize down to 73 bytes.

The only real information the SMS graph (Figure 2.3) give us is that SMS is highly unsuitable as transmission technology in this case. In addition to the above men-

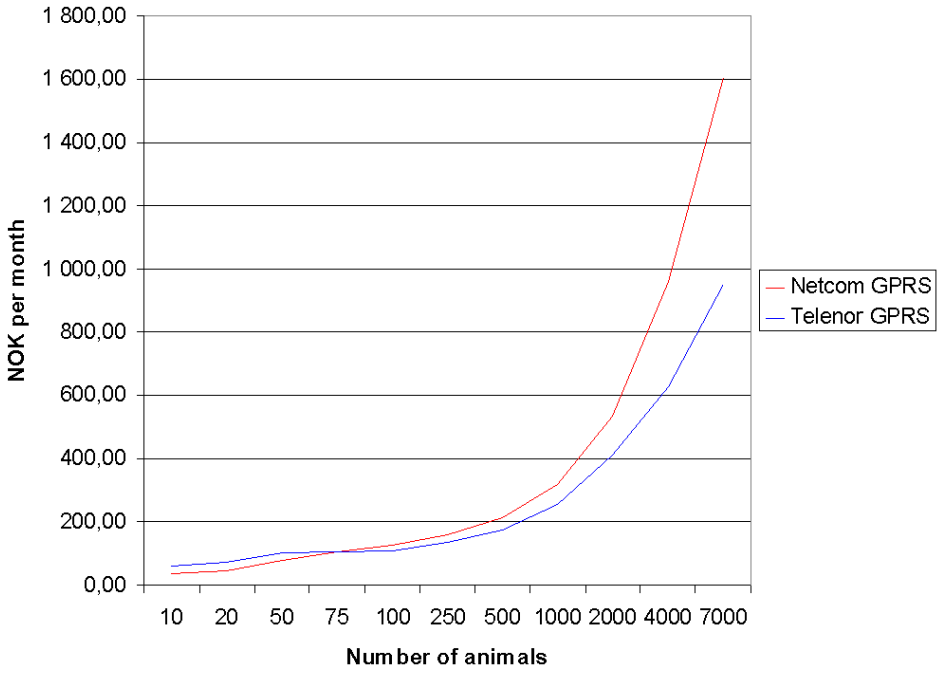


Figure 2.2: *GPRS graph*

tioned points the price is totally unacceptable, even when the number of animals are very few. For instance if we have 10 animals, the cost with our inefficient method is well above NOK 500 per month. If the amount of data were to increase the cost would increase dramatically. An early conclusion must be that SMS is not an option in this project.

Summary

First of all we can discard the SMS option, which leaves us with the GSM and the GPRS options. In Figure 2.4 we see the cheapest GSM alternative (Netcom) compared to the cheapest GPRS alternative (Telenor).

As we can see from the graph, the price difference is relative small. GSM is cheapest when the number of animals is low, but as the amount of data transmitted increases, GPRS becomes cheaper. The reason for this is that with GSM each bit costs the same, but with GPRS the cost decreases the more data you send. If the messageformat, on a later stage, is altered tp transfer more data, the impact on the GSM cost would be bigger than on the GPRS cost. As mentioned in the introduction to this section we assumed that only four transmissions took place in one day, but if this is incorrect the GSM cost also increase more than the GPRS cost due

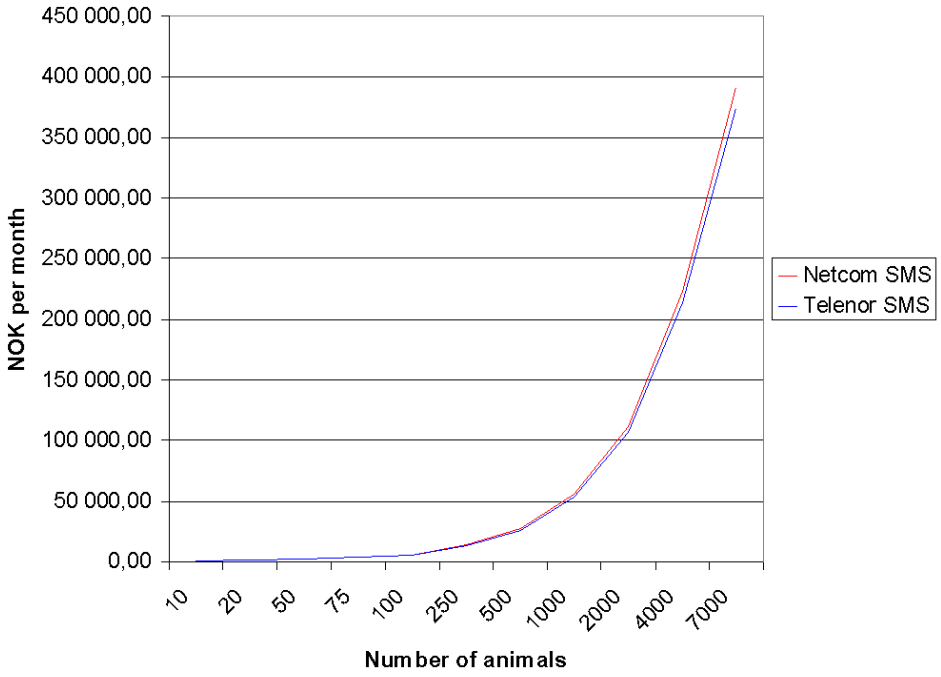


Figure 2.3: *SMS graph*

to the connection fee. The connection fee is also the reason for the GSM graph's gentle rising in the beginning. The same connection fee will also play an important role in the collection season, when position data is sent at higher frequency. Once again, the GPRS cost handle this adjustment better than the GSM cost.

One of Kitron Development's objectives is that the cost per animal per grazing season does not exceed NOK 50. The graph in Figure 2.5 shows that the price for transmitting the data is way below this objective. The reason Telenor is quite expensive with few animals is that the monthly cost is higher than with Netcom. Not unexpected the price per animal drops as the number of animals increases, this is of course because more animals share the monthly cost and in the case of GSM, the connection fee.

2.1.6 Security

Introduction

In this section we will describe the security models used in GSM and GPRS. Figures 2.6 and 2.7 along with most of the information is taken from a report by Lauri Pesonen [5].

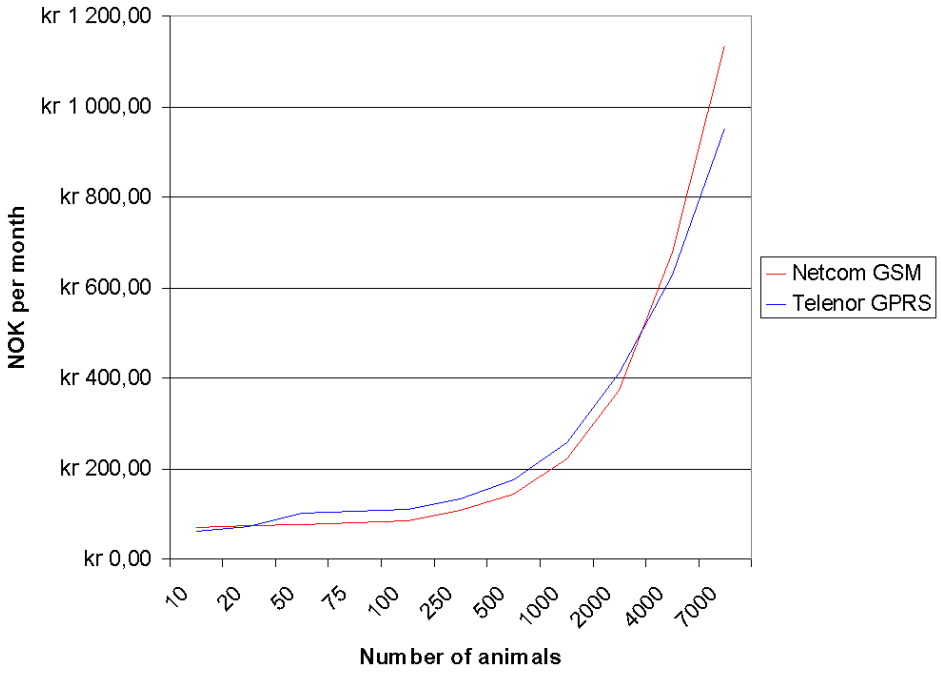


Figure 2.4: *Comparison between GSM and GPRS*

GSM security

GSM uses three security algorithms, A3, A5 and A8. A3 is the authentication algorithm used in the GSM system. A5 is the the encryption algorithm while A8 is the key generation algorithm used in the GSM system

The GSM Security Model is based on a shared secret, called Ki, between the subscriber's home network's HLR and the subscriber's SIM. Ki is 128-bit key used to generate a 32-bit signed response, called SRES, to a Random Challenge, called RAND, made by the MSC, and a 64-bit session key, called Kc, used for the encryption of the over-the-air channel. When a MS first signs on to a network, the HLR provides the MSC with five triples containing a RAND, a SRES to that particular RAND based on the Ki and a Kc based again on the same Ki. Each of the triples are used for one authentication of the specific MS. When all triples have been used the HLR provides a new set of five triples for the MSC [6].

When the MS first comes to the the area of a particular MSC, the MSC sends the Challenge of the first triple to the MS. The MS calculates a SRES with the A3 algorithm using the given Challenge and the Ki residing in the SIM. The MS then sends the SRES to the MSC, which can confirm that the SRES really corresponds to the Challenge sent by comparing the SRES from the MS and the SRES in the

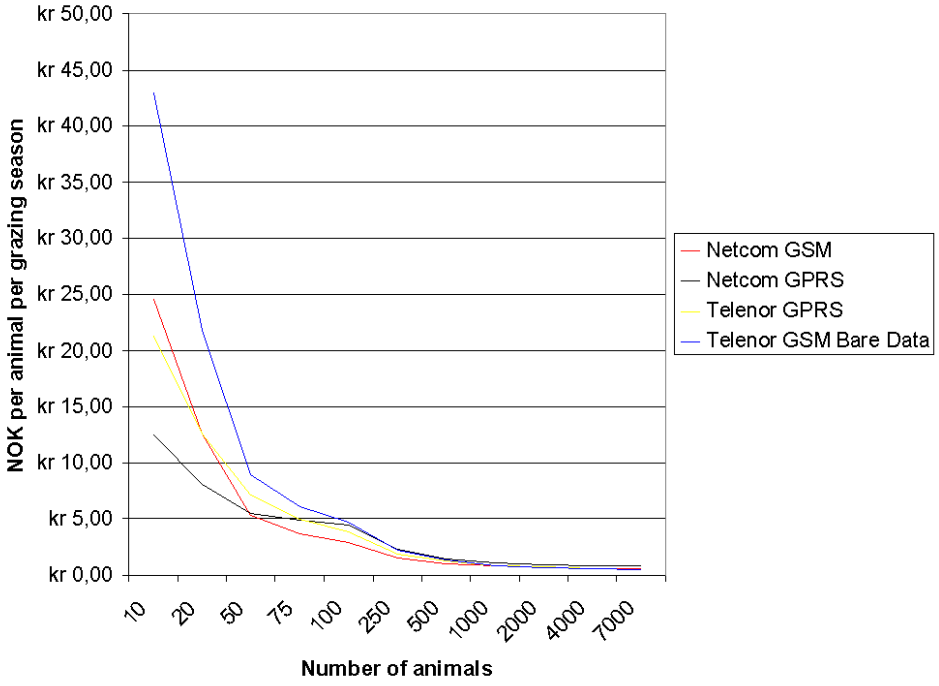


Figure 2.5: *Cost per animal per grazing season*

triple from the HLR. Thus, the MS has authenticated itself to the MSC (Figure 2.6).

The MS then generates a Session Key, Kc, with the A8 algorithm using, again, the Challenge from the MSC and the Ki from the SIM. The BTS, which is used to communicate with the MS, receives the same Kc from the MSC, which has received it in the triple from the HLR. Now the over-the-air communication channel between the BTS and MS can be encrypted.

Each frame in the over-the-air traffic is encrypted with a different keystream. This keystream is generated with the A5 algorithm. The A5 algorithm is initialized with the Kc and the number of the frame to be encrypted, thus generating a different keystream for every frame. This means that one call can be decrypted when the attacker knows the Kc and the frame numbers. The frame numbers are generated implicitly, which means that anybody can find out the frame number at hand. The same Kc is used as long as the MSC does not authenticate the MS again, in which case a new Kc is generated. In practice, the same Kc may be in use for days. The MS authentication is an optional procedure in the beginning of a call, but it is usually not performed. Thus, the Kc is not changed during calls.

Only the over-the-air traffic is encrypted in a GSM network. Once the frames have been received by the BTS, it decrypts them and send them in plaintext to the

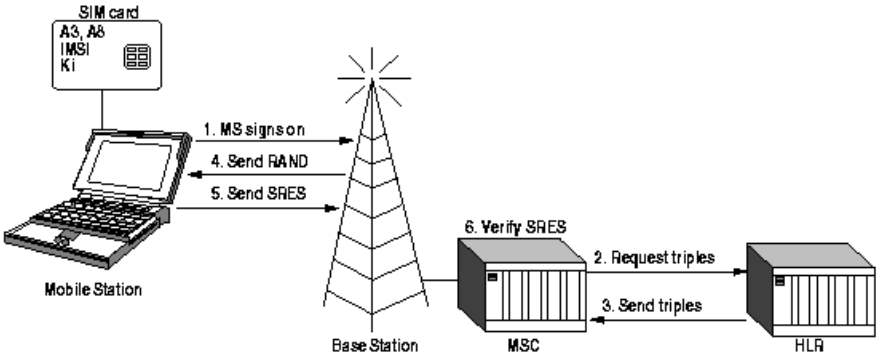


Figure 2.6: *Mobile station authentication*

operator's backbone network [6].

GPRS security

In the GPRS system, the frames are transmitted as cipher text from the MS to the SGSN, Serving GPRS Support Node. This is done because the GPRS system uses multiple timeslots in parallel in order to achieve a greater transmission rate. One GPRS phone can be allocated multiple timeslots by the network, thus increasing the transmission rate of that MS. The frames can be sent in parallel timeslots to the same BTS or to two different BTSs if the MS is handed over from one BTS to another.

To a BTS the use of one timeslot is seen as a separate call. Thus, the BTS is unable to put the frames from different timeslots together. This means that there has to be a network component that is able to receive the frames from one MS, defragment them and send them onwards to the actual destination. The BTSs are also unable to decrypt the frames, because consecutive frames on one channel have not got consecutive frame numbers. See Figure 2.6. To simplify the implementation, the frames are decrypted at the SGSN where all of the frames end up and it is thus easy to keep track of frame numbers. The solution is based on the ease of implementation and has not been implemented in order to increase system security. As a side effect, the GPRS system effectively prevents eavesdropping on the backbone between the BTS and SGSN, because the frames are still encrypted at this point. In GPRS, the triples from the HLR are transmitted to the SGSN and not to the MSC. Thus, security of GPRS depends largely on the placement and security of the SGSNs.

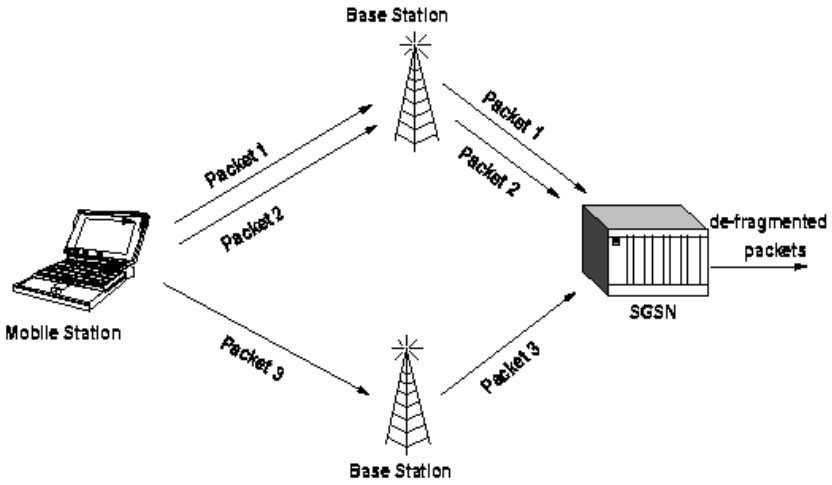


Figure 2.7: *GPRS architecture*

2.2 Network Programming

2.2.1 Introduction

Sockets are a term you often encounter when you study network programming. The term originate from the Unix world, where every I/O related operation is done by reading and writing to a file descriptor [7].

There exists several different socket types, but when you want to communicate using the IP protocol there are mainly two types of sockets.

Datagram Socket/UDP

Datagram Socket, referred to as SOCK_DGRAM, is a connectionless socket, without any form of handshake performed by the protocol – UDP [8]. You specify a destination address, and write data to the socket, and it will hopefully arrive at the destination. You will not get any acknowledgment if the recipient gets the packet, neither will you get a negative acknowledgment if it does not. The packets you send may not be received in the order they was sent.

Stream Socket/TCP

Stream Socket, referred to as SOCK_STREAM, is a connection oriented socket, using the handshake (ACK/NACK) of TCP [9]. The data sent using a Stream

Socket will be received in the same sequence as they was sent. If you operate over a bad connection, and lose packets, TCP will retransmit until all data is correctly transferred.

2.2.2 Multiplexed Socket Programming

Multiplexed Socket programming, often referred to as Asynchronous Socket Programming, is a method of programming sockets to handle multiple connections, by "multiplexing" the requests. There is no multithreading or process forking, used. Everything is handled by one process, which uses the `select()` call to block execution until data is received on any of the file descriptors, and then handling that event. When finished, the process loops back and performs a new `select()` call.

The benefit of using asynchronous sockets in a socket application, is that it scales well. By using only one process for handling the requests there is no CPU and memory overhead used to spawn threads or fork new processes. There is also no need for locking shared resources, using semaphores or mutexes [10] [11]. Using the select call usually lets the server scale to more connections, and achieve higher throughput than other alternative socket programming methods [12].

Unfortunately this method is rather complex to use. It is normal to design state machines for the different processes the server should execute. If you need to do more than trivial processing of the data you receive, like error checking and extensive database or disk I/O, it becomes even more complex. This is because you have to limit the time used by each state transition, because the select call needs to be issued continuously.

> [10] «With async, or 'event-driven' programming, you cooperatively schedule the CPU or other resources you wish to apply to each connection. How you do this really depends on the application - and it's not always possible or reasonable to try. But if you can capture the state of any one connection, and divide the work it will do into relatively small pieces, then this solution might work for you. If your connections do not require much (or any) state, then this is an ideal approach.»

2.2.3 Threaded Synchronous Socket Programming

Pure synchronous socket programming would mean that you could only service on request at a time, not exactly a scalable solution. A solution to this is to use threads, or processes if the target platform have little or no support for threads. Threads are more lightweight than forking processes.

When using multithreading to handle several connections, you usually have one thread that listens for new connections to the socket, and spawns a new thread or forks a new process to handle that connection. The benefit of this is that it is relatively easy to implement. Since each thread handles only one connection you do not have to worry about other incoming connections, or other sockets. You program it like it was a synchronous process, and let the OS handle the "multiplexing" indirectly, through the scheduler.

This way of handling socket programming scales to a certain extent, dependent mainly on the platform and hardware available. The spawning of new threads upon incoming connection consumes quite a bit of CPU time, and the system needs to store each thread's state information, when performing context switching. This means that the more threads you use, and therefor how many concurrent connection you can handle, the more memory will be used by the system. And if each thread uses some memory extensive operations, like storing large amounts of data in a database, the total memory usage will could become rather high.

There exists methods to minimize some of the drawbacks of multithread socket programming. To minimize the CPU overhead of spawning new processes upon incoming connections, you may use what is called a *thread pool*, which is a pool of pre-spawned threads that can be used to service a request, and when finished will return to the pool, ready for the next connection. This results in a bit longer startup time for the application, because the threads are spawned at startup, but no extra CPU overhead upon receiving a request.

Multithreaded programming often use semaphores or mutexes to serialize access to shared resources which do not support multiple access, causing other threads to wait until the shared resource is available before using it. Some resources, like some database libraries, are what is referred to as *thread safe*. This term has lots of different meanings, but in one form or another the software component is constructed to be used simultaneously by multiple threads.

2.2.4 Security

SSL

SSL (Secure Socket Layer) is a protocol, positioned below the application layer in the OSI model (Figure 2.8), that uses cryptographic algorithms to provide authentication and data confidentiality [13]. Both sides of the communication may be authenticated, but very often only the server is authenticated. Most web sites using HTTP over SSL, often referred to as HTTPS, uses server certificates to authenticate the server, but do not use client certificates.

SSL is optimized for use with HTTP [15], but can be used to authenticate and encrypt any application protocol.

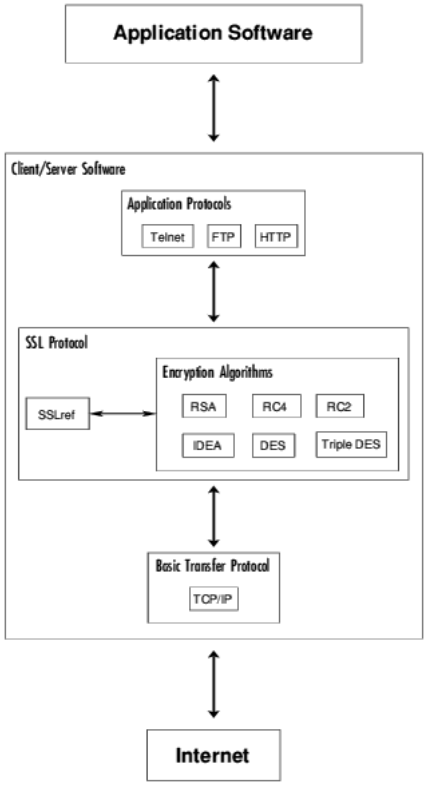


Figure 2.8: *SSL runs above TCP/IP and below high-level application protocols (Taken from «The SSLP Reference Implementation Project», J. Bradley, 1995 [14])*

To use SSL in an application you can use third party components, like OpenSSL [16] which is an free implementation of the SSL protocol. SSL libraries are available for many programming languages, like C/C++, Java and Python. A second alternative is to program using normal sockets, and tunnel the data through a SSL Wrapper, like Stunnel [17]. Stunnel enables non-SSL applications to utilize SSL, without having to modify the application.

IPSec

The IP Security Protocol Workgroup [18] have developed several Internet Drafts and RFCs related to security and the Internet Protocol (IP).

IP Authentication Header (AH) [19] provides connectionless integrity and data origin authentication for IP datagrams [19], and optionally protection against replay attacks.

IP Encapsulation Security Payload (ESP) [20] is designed to provide a mix of security services in IPv4 and IPv6 [20]. There are several ways to use ESP, it can be used alone, along with IP Authentication Header, or in tunnel mode with an IP AH packet encapsulated in an IP ESP packet.

IP ESP provides confidentiality, data origin authentication, connectionless integrity, protection against replay attacs, and limited traffic flow confidentiality. What services are provided depends on options specified during the establishment of a Security Association.

Internet Security Association and Key Management Protocol [21] provides a framework that provides mechanisms for performing *authentication*, establishing *security associations*, and *key management*.

The term Security Association (SA) is central in the IP Security Protocol. A SA is a negotiated agreement between sender and reciever that includes all of the information required to prove authentication, and encrypt and decrypt messages in transit. Security associations are unidirectional, meaning that for bi-directional communication two SAs have to be negotiated.

The problem with IPsec is that it consists of several Internet Drafts and RFCs, and is very complex. It contains numerous options and modes of operation, giving the user a high degree of flexibility.

Bruce Schneier, a highly acclaimed security expert, has reviewed IPsec [22]. His main issue with IPsec is its complexity.

> [22] Security's worst enemy is complexity.

Among the problems he addresses are:

Several Modes of Operation	IPsec has two modes of operation, transport mode and tunnel mode. Both AH and ESP can use these two modes. AH provides authentication, while ESP provides authentication, encryption, or both. Since ESP overlaps AHs functionality in most areas, AH should be removed and ESP should be modified to incorporate all of AHs functionality.
ESP allows encryption without authenication	Encryption without authentication is not useful. ESP should be modified to always provide authentication.
Security Associations are unidirectional	Since there are few situations where IP packets are sent without any reply, or where on direction of communication need to be secure while the other is not, SAs should be bi-directional instead of unidirectional.

While IPsec is complex in nature, and there are some problems with it, it is regarded, even by one of its most aggressive opponents Bruce Schneier, as the best IP security protocol available.

It is implemented by most VPN manufacturers, and exists on most modern Operating Systems.

IPsec is a work in progress, and several new proposed standards are under development, e.g. how to make IPsec and NATs to work together.

Using IPsec to secure an network application requires no changes to the source code, the security mechanisms are transparent. Each IP packet is encapsulated within an IPsec packet, and the receiving end removes the IPsec encapsulation before delivering the standard IP packet to the socket. But this also means that an IPsec implementation have to be set up on both ends.

2.3 Databases

2.3.1 General Database Theory

ACID properties

ACID properties are an important concept for databases. The acronym stands for Atomicity, Consistency, Isolation and Durability.

- Atomicity - The entire sequence of actions must be either completed or aborted. The transaction cannot be partially successful.
- Consistency - The transaction takes the resources from one consistent state to another.
- Isolation - A transaction's effect is not visible to other transactions until the transaction is committed.
- Durability - Changes made by the committed transaction are permanent and must survive system failure.

The ACID properties of a DBMS allow safe sharing of data. Without these ACID properties, everyday occurrences such using computer systems to buy products would be difficult and the potential for inaccuracy would be huge. Imagine more than one person trying to buy the same size and color of a sweater at the same time, a regular occurrence. The ACID properties make it possible for the merchant to keep these sweater purchasing transactions from overlapping each other, saving the merchant from erroneous inventory and account balances.

Transactions

A transaction is a group of statements whose changes can be made permanent or undone only as a unit. A transaction ends with a COMMIT or ROLLBACK statement. If it ends with a COMMIT statement, all the changes made to the database by the statements are made permanent. If the transaction fails or ends with ROLLBACK, none of the statements takes effect.

Integrity

Data Integrity is an umbrella term that refers to the consistency, accuracy, and correctness of data stored in a database. Data integrity is not about physical security,

fault tolerance, or data preservation (backups). It means, in part, that you can correctly and consistently navigate and manipulate the tables in the database. There are two basic rules to ensure data integrity; *entity integrity* and *referential integrity*.

The *entity integrity* rule states that the value of the primary key can never be a null value. A null value is one that has no value and is not the same as a blank. Because a primary key is used to identify a unique row in a relational table, its value must always be specified and should never be unknown. The integrity rule requires that insert, update, and delete operations maintain the uniqueness and existence of all primary keys.

The *referential integrity* rule states that if a relational table has a foreign key, then every value of the foreign key must either be null or match the values in the relational table in which that foreign key is a primary key.

Database Replication

Database replication is the process of sharing data between databases in different locations. You use database replication to make special copies, called replicas, of a database so that users at different locations can all work on their own copy and share, or synchronize, their changes. If you design your applications for multiple users, database replication can improve the way your users share data.

Database replication can be used for:

- Accesing data - If your work takes you out on the road, you can maintain a replica of a corporate database on your laptop computer. Upon connecting to the corporate network, you can synchronize the changes you made to the replica on your laptop with the changes someone else made in the corporate office replica.
- Distrubuting software - You can add new tables, queries, forms, reports, macros, and modules to the Design Master, or you can modify the existing objects. The next time the Design Master is synchronized with its replicas, the changes will be dispersed to the replicas.
- Backing up databases - You can automatically back up a database by keeping a replica on a different computer. Unlike traditional backup methods that prevent users from getting access to a database during backup, replication allows you to continue making changes online.
- Load Balancing - You can replicate a database on additional network servers and reassign users to balance the loads across those servers. You can also give users who need constant access to a database their own replica, thereby reducing the total network traffic.

- Internet or intranet replication - You can configure an Internet or intranet server to be used as a hub for propagating changes to participating replicas.

2.3.2 Database Connection Pooling

Database connection pooling eliminates the overhead of making a new connection to the database every time the database is needed, much in the same way as thread pooling (Section 2.2.3). An configurable amount of connections to the database is made at startup, and when a connection is needed by the application it gets one from the connection pool, and returns it when finished [23].

2.3.3 PostgreSQL

PostgreSQL is an object relational database management system (ORDBMS) based on POSTGRES, Version 4.2, developed at the University of California at Berkeley Computer Science Department. PostgreSQL is an open source descendant of this original Berkeley code.

POSTGRES pioneered many of the object-relational consepts now becoming available in some commercial databases. Traditional relational database management systems (RDBMS) support a data model consisting of a collection of named relations, containing attributes of a specific type. In current commercial systems, possible types include floating point numbers, integers, character strings, money and dates. It is commonly recognized that this model is inadequate for future data-processing applications. The relational model sucessfully replaced previous models in part because of its "Spartan simplicity". However, this simplicity makes the implementation of certain applications very difficult. PostgreSQL offers substantial additional power by incorporating the following additional consepts in such a way that users can easily extend the system:

- Inheritance
- Data types
- Functions

Other features provide additional power and flexibility:

- Constraints
- Triggers
- Rules

- Transactional integrity

These features put PostgreSQL into the category of databases referred to as *object-relational*. Note that this is distinct from those referred to as *object-oriented*, which in general are not as well suited to supporting traditional relational database languages. So, although PostgreSQL has some object-oriented features, it is firmly in the relational database world.

2.3.4 MySQL

MySQL, one of the most popular open source SQL database, is developed, distributed, and supported by MySQL AB. MySQL AB is a commercial company, founded by the MySQL developers, that builds its business providing services around the MySQL database. MySQL is a relational database management system that is designed from the start to work with medium size databases – 10-100 million rows, or about 100 MB per table – on small computer systems.

2.3.5 MS-SQL Server

SQL Server is a DBMS provided by Microsoft. MS SQL Server is a client/server relational database system. Client/server, also referred to as distributed computing, means that all of the data processing of the program does not occur on a single computer. With SQL server, the database engine part of the program resides on a designated server computer, whereas the other part of the program, the client interface resides on a user's desktop computer. The components of the client/server system communicate over a network as if they were one and the same program. MS SQL Server is not a physical server itself, but a computer program that runs on a physical server. Only one copy can run on a single physical server at a time.

2.3.6 Summary

Two summarize the two open source alternatives briefly described above, we have used a comparison made by Wolfgang Andreas Klimke [24].

Generally, it can be stated that PostgreSQL currently offers a richer set of features in processing the stored data. MySQL does not support some critical features, such as safe transactions or foreign keys that are usually part of relational database management systems. However, MySQL offers more standard data types and functions according to ANSI SQL. Furthermore, the additional processing required for handling the transaction-safe tables in PostgreSQL gives MySQL a significant speed advantage for insert and update statements.

Table 2.3: *Comparison between MySQL and PostgreSQL*

Feature	**MySQL**	**PostgreSQL**
License	MySQL Free Public License (very restrictive for Open Source standards; e.g. a license needs to be purchased for applications requiring MySQL to function)	Unproblematic licensing
Kernel architecture	Multi-threaded allowing for utilization of multiple CPU's	Single-threaded
Transactions with rollbacks	Not supported	Supported
Views	Not supported	Supported
Sub-selects	Not supported	Supported
Foreign Keys	Not supported	Supported
Extendable type system	Not supported	Supported
Stored procedures	Not supported	Supported
Size limitations:		
Table size	2GB to 8TB (operating system dependent)	64TB (all O/S)
Row size	65534 (without BLOB's	Unlimited
Columns/table	3398	1600

2.4 Programming Web Applications

2.4.1 Introduction

A web interface is a convenient and cross-platform way of presenting information. HyperText Markup Language (HTML) is the basis of the web, but static HTML pages is mostly a thing of the past. Today most HTML documents are generated from data stored in a database by one of the many different Web programming technologies.

2.4.2 PHP

PHP is a very popular open source programming language, mainly used to create dynamic web pages with some kind of database backend. It runs on several different platforms and supports an array of different web servers [25].

The core functionality of PHP is supported "out of the box", but more advanced features, like database access, cryptography and SSL must either be compiled into the PHP runtime at build time, or compiled as modules which can be loaded.

This makes deployment of web-applications difficult, because some libraries are only available on some of the platforms PHP support. It also means that you have to know exactly what non-standard functionality is used in the web application, if you are to move the web application from on server to another, without breaking the application.

2.4.3 Active Server Pages and ASP.NET

Active Server Pages (ASP) is Microsoft's web-scripting environment. It has been around since the end of 1996 [26], and having been continuously developed and upgraded it is a mature solution.

ASP is generally programmed using Visual Basic, but it is possible to use other programming languages as well. In the beginning ASP only ran under Windows, using Internet Information Services, but has since been made available on other platforms and web servers.

The newest incarnation of ASP is really a whole new environment, called ASP.NET. It is more of a generic framework, with Web Services and CLR being the most talked about.

CLR is an acronym for Common Language Runtime, and is a multi-language execution environment. Theoretically every language can be used to program ASP.NET, all that is need is a CLR compiler. Microsoft recommends the use of their new language, C#, which was made specifically for the .NET framework, and has features similar to C/C++ and Java.

ASP.NET has been around for a couple of years, and the amount of third-party libraries and components has grown steadily. Lately there has also emerged solutions for running ASP.NET applications on other platforms than Windows, and other servers than Internet Information Services.

2.4.4 Java Servlets and JavaServer Pages

Java is Suns object oriented programming language from 1995, primarly pitched at developing Java Applets for web pages. Since then, it has grown into a major programming language, used in products as different as mobile phones and mainframes. Java has also grown to handle task well beyond Java Applets, and is currently used for desktop applications, mobile phone applications, databases, web applications and much more [27].

Servlets are Java classes that adhere to the Java Servlet API, and are run by a Java Servlet Container. There exists several Java Servlet Engines, from many vendors, as well as open source alternatives. In our study we used Tomcat, the open-source official reference Servlet Container by the Apache Group [28]. It is implemented in Java, and thus is available on all platforms that the Java Runtime Environment exists for.

Servlets are easy to deploy, as they are self-contained WAR files, an archive file similar to Javas JAR achives, which contains all needed libraries and configuration files. Since a Java Servlet Container has to conform to a specific standard, a servlet that is developed and tested using for instance Apache Jakarta Tomcat, will run unmodified on IBM's WebSphere.

> [29] JavaServer Pages technology is an extension of the Java Servlet technology. Servlets are platform-independent, 100% pure Java server-side modules that fit seamlessly into a Web server framework and can be used to extend the capabilities of a Web server with minimal overhead, maintenance, and support.

JavaServer Pages allows for the direct insertion of servlet code into an otherwise static HTML file. These code segments are called scriptlets. But although JavaServer Pages and Servlets are two different programming methods, JavaServer Pages is automatically compiled into a Servlet by the server when it is run.

2.4.5 Web Application Security

Confidentiality

To ensure the confidentiality of the data transferred between the client and the server, SSL is the obvious choice. Its confidentiality is strong enough that virtually all online banking systems use it, and is the standard way of encrypting communication using the HTTP protocol.

> [30] ... this is completely transparent to servlets and servlet developers. You just need to obtain an appropriate server certificate, install it, and configure your server appropriately. Information transferred between servlets and clients is now encrypted.

This quote is regarding Java Servlets, but most other web technologies share the feature of transparant implementation of server authenticated SSL communication. It is the server that is configured to handle the SSL algorithm, allowing the web application to run unmodified over ordinary HTTP and SSL secured HTTP (HTTPS).

If the system needs to authenticate the clients some modification is needed. The web application must be able to verify the client certificates it receives, and grant access accordingly. SSL enabled web technologies that support client authentication have APIs to retrieve information about the certificate of the requester. Storing the client certificates in the database instead of usernames and passwords is more secure, because SSL certificates offer better authentication of the client and if an attacker gains access to the database a SSL certificate will not gain him anything.

Common Vulnerabilities

Web applications contribute a considerable amount of security vulnerabilities to Security Focus' Bugtraq [31] mailing list. Two of the most frequently reported bugs related to web applications are:

- SQL Insertion Vulnerabilities
- Cross-Site Scripting Vulnerabilities

SQL Insertion is the act of passing SQL code into an application that was not intended by the developer. This is most often caused by nonexistent or poor validation of data input from the user. An attacker usually abuses the fact that most SQL statements are constructed from concatenated strings.

```
sql = "SELECT userid FROM users WHERE loginname='" +
    username_input + "' AND password='" + userpass_input + "'";
```

Listing 2.1: *Typical string concatenation for generating a login SQL statement*

Listing 2.2 show what the SQL statement generated from concatenating the string in Listing 2.1 with form data from the user, if *frode* logs in without trying to exploit the system.

```
SELECT userid FROM users WHERE loginname='frode' AND password='
    hemmelig';
```

Listing 2.2: *Concatenated SQL statement without malicious code*

Listing 2.3 show what the result could look like if an attacker tries an SQL insertion attack by inputting

```
'; DROP TABLE users --
```

in the username field.

```
SELECT userid FROM users WHERE loginname='' ; DROP TABLE users
    -- ' AND password='';
```

Listing 2.3: *SQL Insertion Successful*

The SQL statement in Listing 2.3 have great implications if the web application lets it through unaltered. The *users* table will be deleted, meaning all userdata would be lost, and the service would be unavailable until the database is reconstructed from backup. This particular attack depends on the attacker to either know or guess the name of the table he wants to delete.

If the attacker is able to know or guess a username, for instance the user *admin*, which have administrator privileges, he is able to login without using a password. Inputting

```
admin'; --
```

in the username field results in the SQL statement shown in Listing 2.4, which would give the attacker full access.

```
SELECT userid FROM users WHERE loginname='admin' ; -- ' AND
    password='';
```

Listing 2.4: *SQL Insertion Successful*

Cross-Site Scripting (XSS) is a problem for web applications that allow users to post data that will be viewed by other users, like for instance a message forum. If the application is not secured against XSS attacks, a cracker may insert malicous Javascript, HTML or some other web technology. This will not give an attacker direct access to the server, nor is it possible to bring the server using only a XSS attack, but it may harm the users of the service. Depending on the scripting language used for the attack the attacker may be able to read data from a form, possibly giving the attacker confidential data like passwords [32].

Preventing SQL Insertion and XSS is usually done by the web application when input forms are processed. SQL Insertions can be prevented by encoding all characters that could escape a SQL statement, into something that does not pose a threat. Different DBMS' specify different ways of escaping these characters, thus the developer has to consult the documentation for the particular RDBMS used. Listing 2.5 shows Listing 2.4 if it had been properly processed, based on the character encodings used by PostgreSQL [33].

```
SELECT userid FROM users WHERE loginname='admin\'\; \-\-' AND
    password='';
```

Listing 2.5: *SQL Insertion Prevented*

XSS is also prevented through filtering and encoding strings input by the user. Special HTML entities and characters must be escaped, as well as input that may insert script elements [32].

3. REQUIREMENTS

3.1 Introduction

The project has been a development project in cooperation with Kitron Development A/S. Our work has been done in parallel with the team at Kitron, and we have continuously received specification and requirements proposals. We had no requirement analysis or system specification we should adhere to, but solved this together with the development team at Kitron as we moved along.

We had two meetings with Kitron, where we charted some of their requirements. When we encountered situations where we had several solutions, we communicated with Kitron to resolve what would fit best with the other parts of the project (Figure 1.1).

Because Kitron Development was working simultaneously with much of the surrounding elements with which our project should fit in, the specifications and requirements kept changing. We had to reimplement some parts, and some of the parts in our project could unfortunately not be brought up to par with the latest specifications we received.

This simultaneous development also meant that we had no radio-collar or base stations to test our prototype components against. We therefore chose to develop an radio-collar and base station emulator to help us test the components during development.

3.2 Overview of the Complete System

Kitron Development wanted us to study and implement a system to receive and store all the generated data from the radio-collars. In Figure 1.1 our project is illustrated, as the colored sky, together with the other parts of the system which Kitron is developing.

The computers, and the database storage, illustrates different components in the Animal Tracking Central. The different components do not necessarily have to run on different computers, or be separate processes.

3.3 Server

The server receives position data from all the radio-collars through one or more base stations. It has to handle several base stations, each of which is able to handle up to 7000 radio-collars. Each radio-collar transmits position data to a base station several times per day with the delay between message transmission being configurable.

If the server receives the same position data for an animal from multiple base stations, only one copy should be stored in the database. Each message from the radio-collar contain the latest position, as well as the previous two positions. All positions should be analyzed and entered into the database if they do not already exist.

3.3.1 Communication Interface

Section 2.1.5 concludes with GSM being the cheapest alternative if only position data is to be transmitted. After meeting with Kitron Development, we were informed that they had chosen to implement a GPRS module in their base stations, and that we should develop a server with a TCP/IP socket interface. If they were unable to implement a TCP/IP interface in their GPRS module, they had a GSM server module they could use between the base stations and our server.

3.3.2 Security

Ideally the communication between the server and the base stations should be secured through the use of IPsec (Section 2.2.4). The server could be developed without having to implement any security mechanism in the code. In addition the following security measures could be used:

- Authentication of base stations, preventing malicious attackers to pose as base stations.
- Authentication of the server, preventing malicious attackers to intercept communication and pose as the server.
- Encryption of the communication, ensuring the confidentiality of the data.
- Protection against replay attacks.

SSL (Section 2.2.4) would provide us with much of the same security mechanisms as IPsec. If we limited the authentication to only let the base stations authenticate the server, the SSL-specific code in the server would be minimal. Should the server

authenticate the base stations, certificates for each base station would have to be accessable to the server, either by storing the public keys locally, or using a Certificate Authority. An alternative to implementing SSL specific code in the server would be to use Stunnel [17] (Section 2.2.4).

Unfortunately neither IPsec nor SSL are to be implemented at this stage, because the base stations are controlled by an integrated chip. Kitron found it difficult to implement any of the proposed security mechanisms on the client-side. Initially the base stations was controlled by a laptop computer, which would have made the above suggested security mechanisms easier to implement.

3.4 Alarms

3.4.1 Introduction

The system may produce three different alarmtypes:

1. Animal Alarm, generated by radio-collar or base station because of external factors (low battery, tampering etc).
2. Animal Alarm, generated by the Animal Tracking Central after analysing incoming data.
3. Base station Alarm, generated by the base stations because of external factors.

Alarm type 1 and 3 are specified by Kitron in the specification A.3.

Alarm type 2, on the other hand, is a set of user configurable and chooseable alarms. This should enable the farmer to specify situations he want to be alerted of, and how he wants to be alerted. The system should be able to adapt to new suggestions, and it should be possible to add new alarms to the system, with out having to change the source code of the main system.

3.4.2 How to alarm the Farmer

When an alarm is generated – either by the radio-collar, the base station or by analysing the data received – the farmer should be notified in some way. Possible notification methods are:

- Email
- SMS

- By Phone, via a call-center

Ideally the design should be extensible, making it easy to add new notification methods when the need arise.

3.5 Database

3.5.1 Database Management System (DBMS)

The database must be designed to store relevant information gathered from the radio-collars and the base stations. This is best done by using a Relational Database Management System (RDBMS), which is the most popular type of DBMS these days.

The following criterias should be considered when choosing a RDBMS to implement in the system:

- Data integrity
  - Support for transactions
  - Support for constraints
- Scalability

3.5.2 Database Model

The database model should as closely as possible reflect the specifications provided by Kitron Development (Appendix A). Design decissions should promote flexibility and scalability.

The raw messages received by the server (Section 3.3) is to be stored in the database, to enable research and debugging.

3.6 Web Application

The web interface is in essence an adapted frontend to the database, designed with ease of use in mind. It should give the farmer access to relevant data, and allow him to configure the system to his liking, as much as possible.

It should be possible to get a list of all the animals that each farmer has registered in the system. A listing of all positions and other registered data for each animal should also be available.

Different farmers have different needs when it comes to the Alarm Monitor. There should therefore be an interface for choosing and configuring the alarm modules.

An administrative interface should be developed. This should enable a system administrator to administer user accounts, alarm modules and possibly other aspects of the system.

It would be nice to be able to redesign the web application without having to make too many modifications to the business logic of the web application, but this modularity is not required.

3.6.1 Security

Security is important in any web application. The web application should be secured against known vulnerability types.

The Web Application should support server authenticated SSL communication (HTTPS), but client authentication is not necessary.

3.7 Requirements overview

A short summary of the requirements follow:

Server

- IP based communication
- Multiple concurrent connections
- Scaleable
- No security

Alarms

- Modular design
- Farmer should be notified

Database

- Choose proper DBMS
- Database Model based on specifications

Web Application

- Interfaces for the farmer
- Interfaces for an administrator
- Secured against common vulnerabilities
- Encrypted communication using SSL
- Separation of business logic and presentation

4. PROTOTYPE IMPLEMENTATION

4.1 Introduction

Our assignment was to design and implement a system according to the requirements in Chapter 3. Kitron's focus had been on the radio-collar, the radio transmission system and the base station design. Because of this, little work had been done on the design and implementation of the server system. We therefore had little restrictions laid upon our work.

Figure 4.1 shows the different components in our prototype, and how they interact. On the far left the Base station Emulator (Chapter 4.2) is shown. The emulator only communicates with the Animal Tracking Server and is only for testing and debugging purposes. It is not a part of the system. The three orange circles depict three separate components running independently, all of which communicate with the fourth component – the RDBMS.

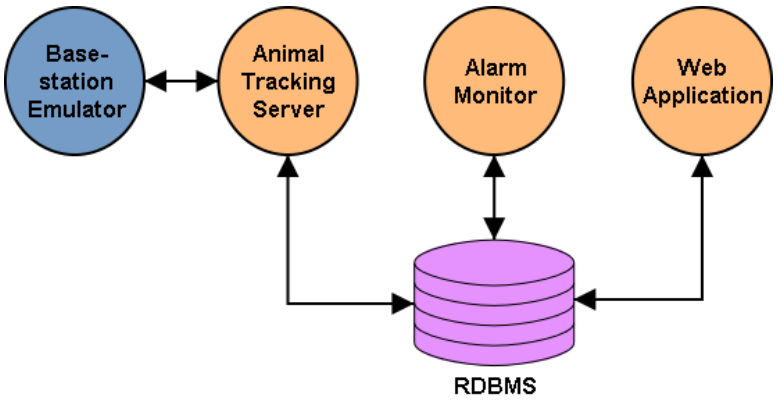


Figure 4.1: *Communication between prototype components*

4.2 Radio-collar and Base station Emulator

The Radio-collar and Base station Emulator – dubbed SimSheep – was the first component we developed. As mentioned earlier Kitron had no base stations or

radio-collars we could use for testing the other parts of the system, thus we needed an emulator.

We implemented the emulator using Java, and created classes for parsing and generating data coded in the format specified by Kitron in their two first specifications (Appendix A.1 & A.2). In addition we created utility classes to control GPS location data. To be able to test the socket interface of the Animal Tracking Server the emulator was programmed with socket support, as well as possibilty to save to files for debugging purposes.

4.3 Animal Tracking Server

4.3.1 Introduction

The server should be able to handle multiple connections simultaneously, and store the data received on these connections in a database. Kitron Development gave no restrictions on what programming language we were to use to develop the server.

4.3.2 Python

The server was implemented using the Python [34] programming language. This is a fairly new, interpreted, programming language, supporting dynamic typing and several high-level dynamic data types. The Python interpreter, needed for executing programs written in Python, is ported to several operating systems, including Windows, MacOS and most Unix variants.

4.3.3 Socket Interface

As stated in Chapter 3.3.1, Kitron wanted us to develop an IP-based system. The communication between the base stations and the server is in most cases two-sided and it is important that all data sent from one side arrives in good condition on the other side.

The Transfer Control Protocol (TCP) offer flow-control and retransmission of faulty data. TCP is also the most widely used protocol used on the Internet, carrying protocols like HTTP and POP3, which means that there exists a great amount of tools for it.

To support multiple concurrent connections using sockets, there are two alternatives – nonblocking, multiplexed IO or threaded, synchronous IO. Each alternative has its advantages and drawbacks (Section 2.2.2 & 2.2.3), and we chose to use the multithreaded model, also called the Dispatcher/Worker Model. The reason we did

not choose nonblocking, multiplexed sockets was because none of us had any previous experience with this way of handling socket programming. Our assessment of the throughput generated by the communication with the base stations indicated that putting the effort into implementing nonblocking, multiplexed sockets would be premature optimization.

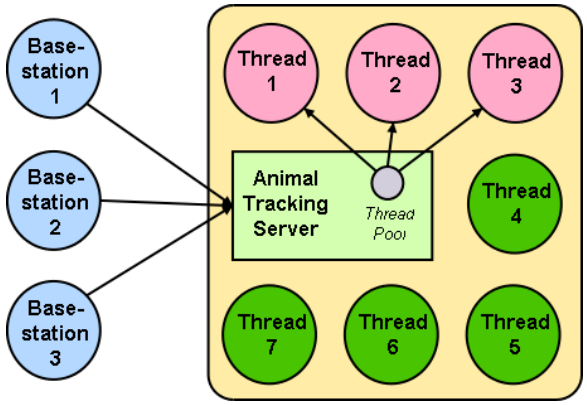


Figure 4.2: *Dispatcher/Worker Model as used in the Server*

The basic Dispatcher/Worker Model spawns new threads for each incoming socket connection, and the threads "die" when the work is finished. This gives a considerable amount of overhead, because threads are generated over and over again. To reduce the overhead of continously spawning new threads we implemented a threadpool (Chapter 2.2.3).

Figure 4.2 illustrates the concept of the Dispatcher/Worker Model, as implemented in the Animal Tracking Server, with a thread pool. Three base stations are simultanously communicating with the server, which has three active threads that do the work of receiving data from the base stations – one thread per base station. The four green threads illustrate Worker threads that are idling, waiting for connections.

4.3.4 Database Connection Pool

A third party Python module, *dbpool* from the *jonpy* library [35], was used to implement a database connection pool for the server.

> [35] The dbpool module is a wrapper for Python DB-API 2.0-compliant database modules to (a) keep a pool of physical connections available and (b) upgrade the modules to threadsafety level 2, which means that threads can share logical database connections.

The database connection pool limits the problem of scaling, to some extent, because there is no need to give each Worker Thread its own cursor object related

to a specific physical connection. Instead we provide the Workers with a logical connection from the database pool, which reduces the total amount of physical connections.

4.4 Alarm Monitor

The Alarm Monitor was, like the Server Module, implemented using Python [34]. One of the requirements for the Alarm Monitor was that it should be designed to have a pluggable interface for different modules.

Listing 4.1 show the main loop of the Alarm Monitor. In the first two lines a connection to the database is created, and then the Alarm Monitor enters a while loop which it will stay in until the program is shut down.

```
1  dbcon = PgSQL.connect(host=HOST, database=DB, user=USER,
       password=PASSWORD)
2  dbcur = dbcon.cursor()
3
4  while not stop:
5   try:
6    definedalarms = getDefinedAlarms(dbcon)
7    log.debug(definedalarms)
8
9    dbcur.execute(chosenalarmSQL)
10   chosenalarmRes = dbcur.fetchall()
11   log.debug("Chosenalarms from DB: " + str(chosenalarmRes))
12
13   farmerID = None
14
15   # For each ChosenAlarm, sorted by farmer
16   for chosenalarm in chosenalarmRes:
17    if farmerID != chosenalarm[1]:
18     farmerID = chosenalarm[1]
19     # Fetch animals for this farmer
20     dbcur.execute(animalSQL % (farmerID))
21     animalRes = dbcur.fetchall()
22     log.debug(animalRes)
23
24    # Execute alarmcode
25    log.info("Executing code for ChosenAlarmID comment: " +
          definedalarms[chosenalarm[0]]['comment'])
26    try:
27     exec(definedalarms[chosenalarm[0]]['pythoncode'])
28    except SyntaxError:
29     log.error("Syntax Error in Alarm Module ID: %d" %
           chosenalarm[0])
30
31    # For each Animal
32    for animalID in animalRes:
33     if chosenalarm[2] == None:
```

```
34          arguments = definedalarms[chosenalarm[0]]['defaultargs'].
                split(",")
35        else:
36          arguments = chosenalarm[2].split(",")
37        # Execution of the alarmmodule loaded from the DB
38        try:
39          alarmres = alarm_check(dbcon, animalID[0],arguments)
40          if alarmres >= 0:
41            log.warn("Alarm Module ID: %d produced an alarm for
                    AnimalID: %d" % (chosenalarm[0], animalID[0]))
42            # Register alarm in DB
43            registerAlarm(dbcon, animalID[0], chosenalarm[0],
                    alarmres)
44          elif alarmres == None:
45            log.info("Alarm Module ID: %d produced no alarm for
                    AnimalID: %d" % (chosenalarm[0], animalID[0]))
46          else:
47            log.error("Alarm module (ID: %d) should return None or a
                    positive integer" % chosenalarm[0])
48        except:
49          log.error("Runtime error with Alarm Module (ID=%d)" %
                chosenalarm[0])
50
51    except AlarmModuleCodingError:
52      log.error("An error occured during execution of an Alarm
            Module")
53    # Sleep for specified amount of time
54    time.sleep(10)
55
56  # Closing DB connections
57  dbcur.close()
58  dbcon.close()
```

Listing 4.1: *Main Loop of Alarm Monitor*

The modularity of the Alarm Monitor is achieved with the use of Pythons builtin `exec` function, which takes a string argument of Python source code, and compiles and executes it. In the database we have a table called *DefinedAlarms* (Section 4.5.3) which holds all the Alarm modules that is available to the user of the system. The table *ChosenAlarms* (Section 4.5.3) represents information about which alarm(s) each user of the system has chosen, and what arguments they have defined for each alarm module.

In line 9 and 10 (Listing 4.1) all the chosen alarms are retrieved from the database, and in lines 24-29 (Listing 4.1) the runtime execution of the Python code for the current alarm is done. For this to work all alarm modules have to follow the specification as shown in Table 4.1 and Listing 4.2.

This design allows the Alarm Monitor modules full access to the database, which could be a security risk if you choose to run a module which you have not checked the sourcecode of, to see if it tries to write to the database. A solution to this would be to maintain two separate database connections in the Alarm Monitor, one which

Table 4.1: *Alarm Module Specification*

Function name	alarm_check
Arguments	`dbcon` - Which is a pointer to the database connection object. `animalid` - An integer value identifying an animal. `args` - A list of arbitrary length, with arguments specific for this module.
Return value	ID of AnimalPosition record that triggered the alarm. 0 if no alarm generated.

```
1 def alarm_check(dbcon, animalid, args):
2  xpos = int(args[0])
3  ypos = int(args[1])
4
5  if xpos > 10 and ypos < 10:
6   return 1
7  else:
8   return None
```

Listing 4.2: *Alarm Monitor Module Example*

the monitor uses – which needs write access to register in the database that an alarm has occured – and one that is read only which the modules should use. This was not done because of time constraints.

4.5 Database

4.5.1 Introduction

Chosing a Database Management System (DBMS), is very much dependent on what you want to use it for, as well as the resources you have to spend on it. A fullfledged Oracle system may cost as much as NOK 292.204 , while MySQL, PostgreSQL and several other DBMS' are available for free. The different DBMS' have different things they excel at, but virtually all modern DBMS' use the SQL language, which makes it easy to port a database design from one system to another. Kitron Development had used MS-SQL in previous projects, but gave us no restrictions on what DBMS to use.

The database design was initially done using Modelator, but later transferred to Microsoft Visio because it was what Kitron Development used internally. We had two meetings with Kitron, where different design decisions were discussed. These talks are reflected in Chapter 3 and in our database design.

4.5.2 Database Management System (DBMS)

We chose to use PostgreSQL [36] during development, because we had prior experience with it. PostgreSQL is an open source ORDBMS that supports important features like transactions and constraints.

Changing the DBMS should not be difficult since all components use APIs which abstracts the underlaying database system, exposing identical interfaces regardless of DBMS used.

Kitron had previous experience with MS-SQL Server and we were supposed to test the prototype components against one of their servers, using Remote Desktop Sharing. Unfortunately the server was stolen shortly before testing was to begin, and there was no time to perform the tests.

4.5.3 Database Model

Implemented Database Model

The implemented database model, shown in Figure 4.3, was initially based on the first revision of the specifications (Appendix A.1). It was updated to fit with the second revision (Appendix A.2) prior to developing the other components. All other components in the prototype rely on it, thus it had to be developed early in the project phase.

The database contains tables used by the server, the Alarm Monitor and the web application. The web application uses practically all tables, as it is a frontend for the database. Below some of the tables will be described briefly.

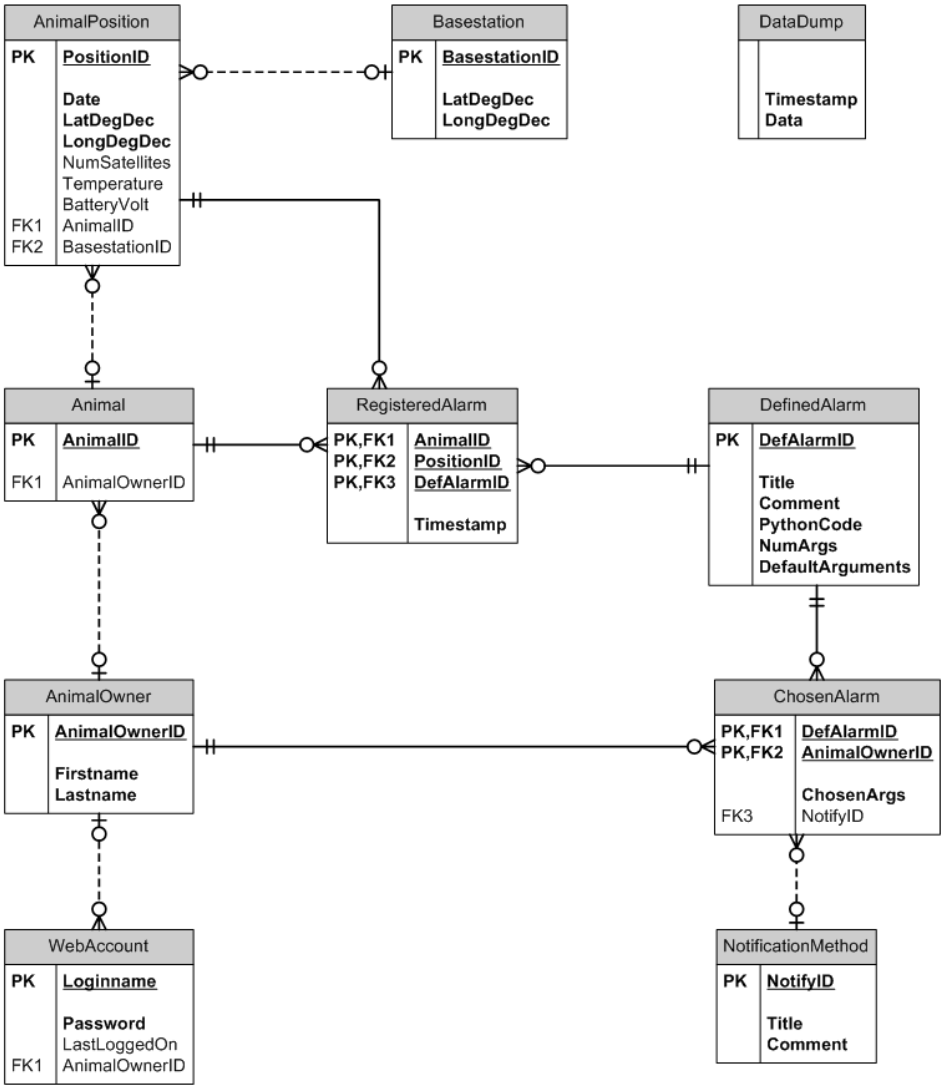


Figure 4.3: *ER diagram of the implemented database model*

DataDump This is, as the name implies, a table to dump data. Kitron wanted the server to store all received messages in their raw format, to use for research and debugging purposes.

AnimalPosition The table stores all the data related to an animal and its position. It uses a counter, called serial in PostgreSQL, which leads to an absolute maximum amount of positions possible to store in the database. Although the database has a roof on the number of positions it can store, this number is relatively large, and should not present a problem. The `bigserial` datatype we used for the *PositionID* field has a range of 1 - 9223372036854775807. Assuming 2000000 animals which sends 10 positions every day for a year, the system is able to run for somewhat over 1.2 billion years.

DefinedAlarm This is where the Alarm Monitor modules are stored. Information about the alarm, like number of arguments and the Python source code that constitutes the module is used by the Alarm Monitor for execution of the modules, and by the web application when users choose alarms.

RegisteredAlarm Stores data about alarms that are triggered. When an Alarm Monitor module triggers an alarm, the *AnimalID*, *PositionID* and *DefAlarmID* is stored in this table, which in retrospect is somewhat redundant. The foreign key *AnimalID* is unnecessary, because it is accessible through the *AnimalPosition* table.

Revised Database Model based on Latest Specification

Late in the project Kitron was about to test their base stations, and wanted to look at our database design. We received the last specification document (Appendix A.3) by mail 2003-04-25 (Appendix B.2 and Kitron requested our database design 2003-04-23 (Appendix B.1). The database design was updated to adhere to the received specification, but because of time constraints we could not implement it together with the other components. Figure 4.4 shows the revised database model.

AnimalPosDump and ControlMsgDump These two tables are the *DataDump* table from Figure 4.3 divided into two separate tables. Animal Position messages are stored in *AnimalPosDump*, while all other messages are stored in *ControlMsgDump*. The Server stores the data it receives in these two tables as binary strings.

Basestation The latest revision we received gave quite a lot of information about what the base stations would send, and possible state-information that could be received. This table is rather large, and should probably be split into several tables, to separate the data that changes often and the data that is rather static.

AnimalOwner In the revised database model, the table *WebAccount* has been eliminated, and all information have been moved to this table. The implemented database model makes it possible to have several login accounts in the Web Application, for each farmer. Implementing Alarm Notification made this design choice unfortunate, because each WebAccount entry had its own adressing information.

4.6 Web Interface

4.6.1 Introduction

Presenting the data gathered from the radio-collars and base stations using web technology gave us a plethora of alternative solutions. A survey of three widely used alternatives for generating HTML was performed (Section 2.4).

ASP.NET by Microsoft

Serving ASP.NET applications was originally only possible by using Microsofts Internet Information Services, but lately alternatives like Mono [37] have emerged. We had no previous experience with the .NET framework, nor any of the programming languages Microsoft recommend for use with it. In addition, the .NET framework relies on software that is commercial and not available to us, or free alternatives that are in the early stages of development.

These factors discouraged us from choosing ASP.NET as the technology for creating the web application. With more time on our hands, ASP.NET may very well have been a viable choice.

PHP: Hypertext Preprocessor by the PHP Group

PHP is a free, open source product which runs on several operating systems and web servers. As mentioned in Section 2.4.2 one of PHPs largest shortcomings is the deployment aspect. There is no way to bundle the non-standard libraries together with the web application, unless it is written in PHP.

PHP is very similar to Java Server Pages and basic ASP (not ASP.NET) in nature, all technologies allow you to embed executable code in plain HTML. This may lead to a mix of business logic and presentation specific code.

The problematic issues with deployment of PHP code, and no prior knowledge with any of the template or Model-View-Controller frameworks available for PHP, made it less than fortunate for usage in our project.

Java Servlets and JavaServer Pages by Sun

Javas fundamental cross-platform design, which makes Java available on practically all used systems of today and – very likely – of the future, is a strong incentive to choose technology based on it. In addition the Java Servlet and Server Pages technologies have a very broad range of available libraries, both commercial and free. Servers implementing the Servlet Container specification, needed to

serve Servlets and JavaServer Pages, is available from a wide range of vendors who compete in delivering the fastest and most scalable implementation.

The simplicity of deploying Java Servlets, the broad industry backing it has, and the fact that we had previous experience with Java made us choose Servlets as the technology to use for our Web application component.

We chose not to use JavaServer pages, and instead use a template engine called Velocity to separate the business logic and presentation layer.

4.6.2 Servlets

We chose to implement the web interface using the Java Servlet technology. Servlets is able to use the same libraries and modules as ordinary Java applications, which gave us access to a whole array of tools. This was especially useful for us, with regard to support for SSL and other Security mechanisms.

> [28] Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. The Java Servlet and JavaServer Pages specifications are developed by Sun under the Java Community Process.

To execute Java Servlets a Servlet Container is needed. There exists several alternatives, both commercial and free. We chose to use the free and open source Servlet Container Jakarta Tomcat, by the Apache Group.

4.6.3 Velocity

Java Servlets can be rather comprehensive to use for programming anything more than the most primitive HTML document. All data that is to be part of the HTML document must be printed by the Servlet using the method `out.println()` from the *HttpServlet* class. In addition to being tedious, this mixes layout information together with business logic. Separating business logic and presentation is an advantage because it makes it easier to change one part, without affecting the other.

Sun offers JavaServer Pages (JSP) as a solution to this problem, by enabling the programmer to mix HTML and Java code in a JSP document. The Java code can be business logic, and thus present the same problem, but it is also possible to make Java Servlets that take care of the business logic, and call them from a JSP document. Other alternatives exist to solve the problem of separating business logic and presentation, and many of the alternatives use different methods, for instance Jakarta Struts [38] which implements the Model-View-Controller (MVC) design paradigm.

Velocity is a Java-based template engine [39]. It allows data to be rendered based on templates. The engine can be used as a stand-alone utility or as an integrated component of other systems. As a general template engine, Velocity suits many purposes, such as code generation [40], XML generation and transformation, as well as dynamic webpage design. The way Velocity works, is that data is generated using on a context and a template. Separating page logic and design allow programmers and designers to focus their efforts at what they are good at.

Listing 4.3 shows one of the Velocity Templates we made for the Web Application. This template generates the HTML interface used to add and configure new alarms, shown in Figure 5.12. All lines starting with the character '#' is Velocity code, while everything else is pure HTML.

```
1  #parse("header.vm")
2
3  <H1>Configure Alarm</H1>
4
5  <H2>$alarm.getTitle()</H2>
6
7  <FORM ACTION="../farmer/saveAlarm" METHOD="POST">
8    <INPUT TYPE="HIDDEN" NAME="aid" VALUE="$!{alarm.getID()}">
9    <INPUT TYPE="HIDDEN" NAME="method" VALUE="$!{method}">
10 <TABLE WIDTH="100%" CELLPADDING="5">
11   <TR BGCOLOR="#6699FF">
12     <TD COLSPAN=2><B>Description</B></TD>
13   </TR>
14   <TR BGCOLOR="#CCCCCC">
15     <TD COLSPAN=2>$alarm.getDescription()</TD>
16   </TR>
17   <TR>
18     <TD COLSPAN=2> </TD>
19   </TR>
20   <TR BGCOLOR="#6699FF">
21     <TD COLSPAN="2"><B>Alarm Arguments</B></TD>
22   </TR>
23 #foreach($i in [1..$alarm.getNumArgs()])
24   <TR BGCOLOR="#CCCCCC">
25     <TD WIDTH="25%">Argument $i</TD>
26     <TD><INPUT TYPE="text" SIZE="10" VALUE="$!{alarm.getArg($i)}"
           NAME="args"></TD>
27   </TR>
28 #end
29   <TR>
30     <TD COLSPAN=2> </TD>
31   </TR>
32   <TR BGCOLOR="#6699FF">
33     <TD COLSPAN="2"><B>Notification</B></TD>
34   </TR>
35   <TR BGCOLOR="#CCCCCC">
36     <TD>Notification Method</TD>
37     <TD>
```

```
38     <SELECT name="notifymethod">
39 #foreach($not in $notifys)
40       <OPTION value="$!{not.getID()}">$!{not.getTitle()}</OPTION>
41 #end
42     </SELECT>
43    </TD>
44   </TR>
45   <TR>
46    <TD COLSPAN=2> </TD>
47   </TR>
48   <TR>
49    <TD COLSPAN=2><INPUT TYPE="SUBMIT" VALUE="Save Alarm Settings
          "> <INPUT TYPE="RESET" VALUE="Reset"></TD>
50   </TR>
51 </TABLE>
52
53 </FORM>
54 #parse("footer.vm")
```

Listing 4.3: *Velocity Template for generating the Configure Alarm HTML interface*

4.6.4 Security in the Web Application

Confidentiality

We used SSL with server certificates for our web application. Kitron felt that this would be secure enough for a prototype, and that client authentication could be added later if necessary. A selfsigned SSL certificate was generated for the Servlet Container server, which provides the same degree of cryptographic security as a trusted SSL certificate, but is less trustworthy from a client perspective.

Commercial vendors like Verisign [41] and Comodo [42] provide trusted SSL certificates, which may be purchased at a later stage if needed.

Securing against known vulnerabilites

We implemented three different functions to secure the application against SQL Insertion and XSS vulnerabilities (Section 2.4.5. One function filters and processes a string for input into an SQL statement, `parseSQLString`. The other, `parseXSSString`, processes data input that is to be presented to other users. The last method, `parseAllString`, is a method that executes both `parseXSSString` and `parseSQLString` and returns a string that is filtered for both SQL Injection and XSS possibilities. All methods are part of the class *DefaultServlet*.

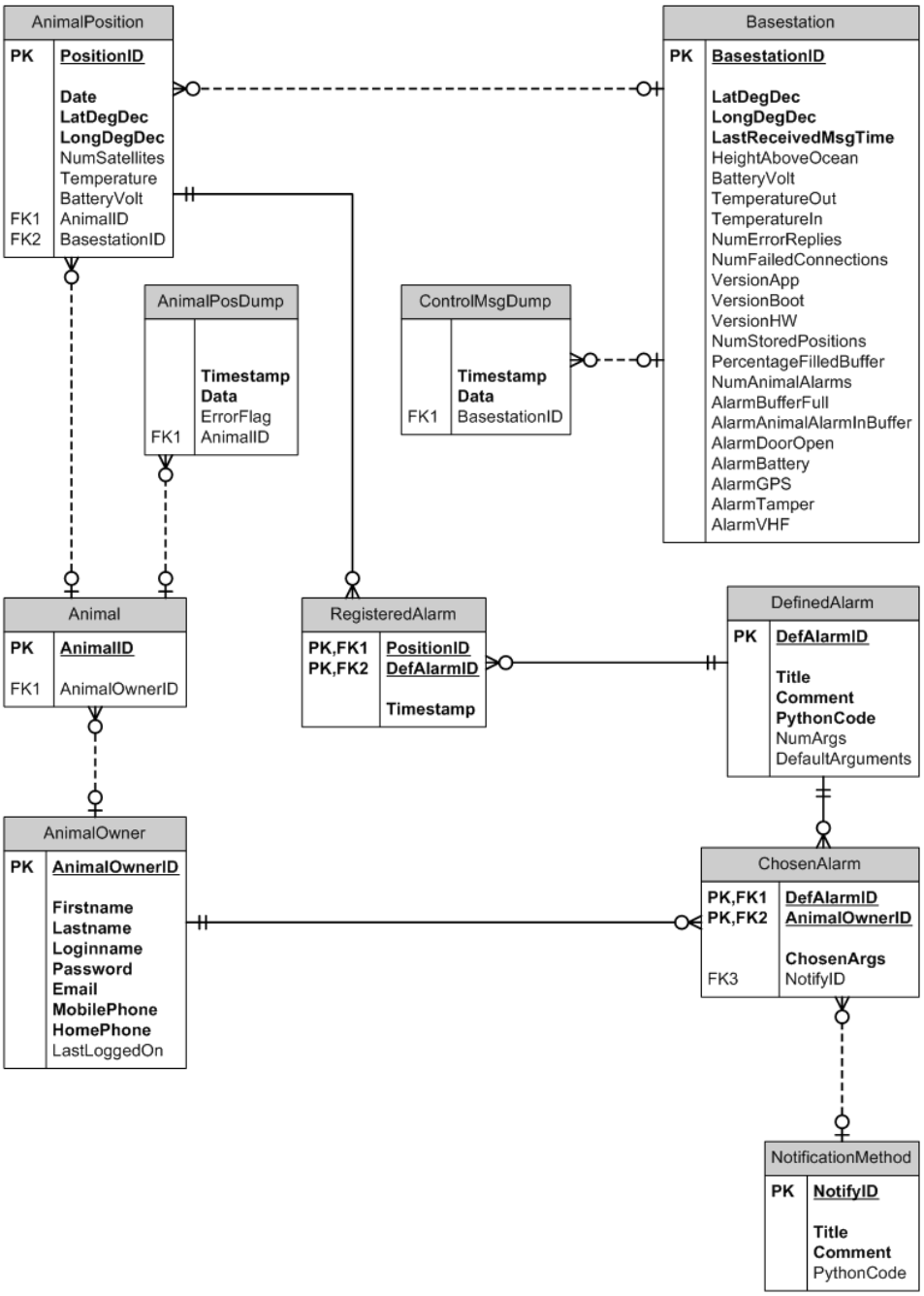


Figure 4.4: *ER diagram of the revised database model*

5.1 Radio-collar Emulator

Figure 5.1 shows the Radio-collar Emulator generating data for the animal with ID 1. The data seen in the figure is transmitted to the server via TCP/IP. In addition to the most recent position, each message also contains the two previous positions as specified in Appendix A.2.

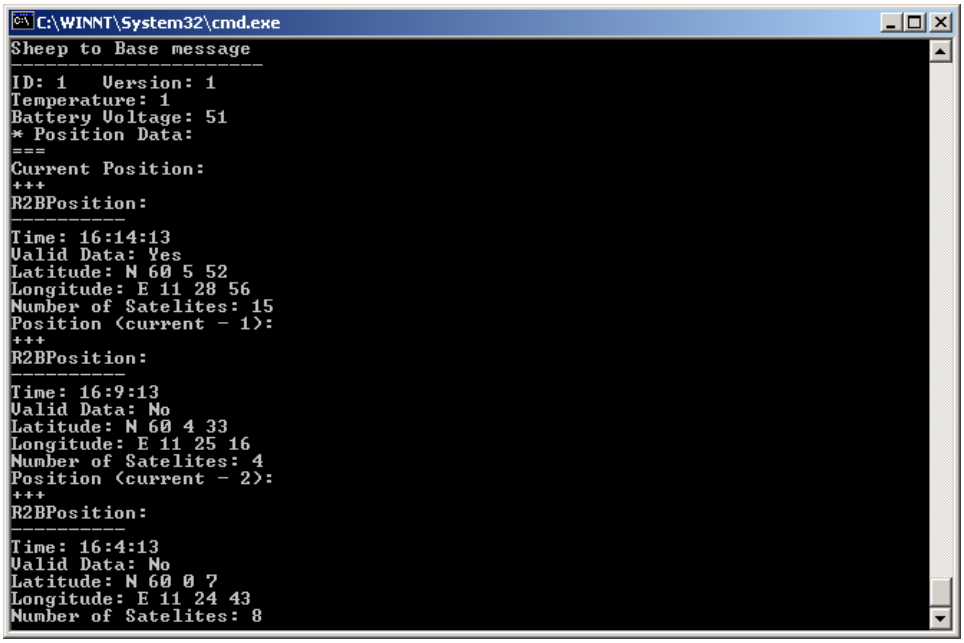


Figure 5.1: *Radio-collar Emulator sending data to Server*

5.2 Animal Tracking Server

The server receives the data from the Radio-collar Emulator, and stores it in the database. Figure 5.2 shows that the Radio-collar Emulator is connecting three times, using IP-address `127.0.0.1`. Each connection spawns a new worker thread from the threadpool, in this example designated Worker 0, 1 and 2.

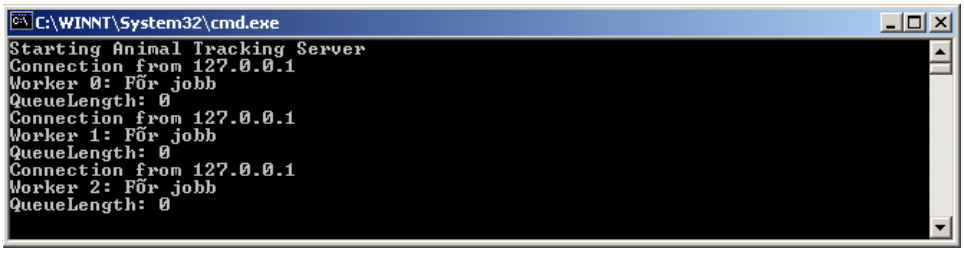


Figure 5.2: *Server receiving data from Radio-collar Emulator*

5.3 Alarm Monitor

Figure 5.3 show what the output when running the Alarm Monitor is. We see the Alarm Monitor executing the Alarm Monitor module with ID 1, for all animals. After it has run the first module through all the animals, it loads the next module and execute it. The arguments that the farmer has specified for the different modules is loaded from the table *ChosenAlarm*, as seen by the output:

```
Arg: ['10']
```

and

```
Arg: ['63.1234', '12.6572', '2500']
```

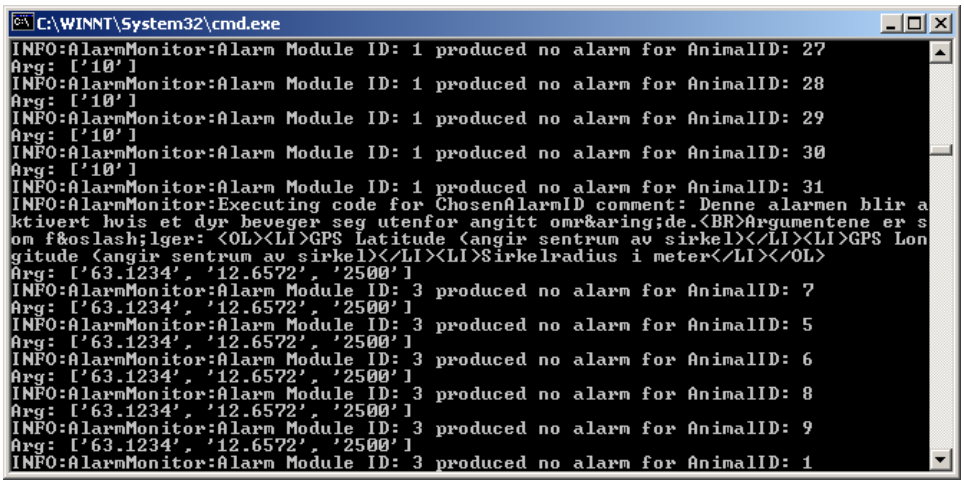


Figure 5.3: *Animal Monitor running two different modules*

5.4 Database

5.4.1 Animal Position Data

Data sent by the Radio-collar Emulator to the server, is stored in the database as shown in Figure 5.4. The latitude and longitude values are converted by the server from the original form shown in Figure 5.1 to the form shown in Figure 5.4, in the fields *latdegdec* and *longdegdec*. This is done to ease the processing and extraction of the position values, by storing it as a single numeric instead of in three separate numerical fields or as a string.

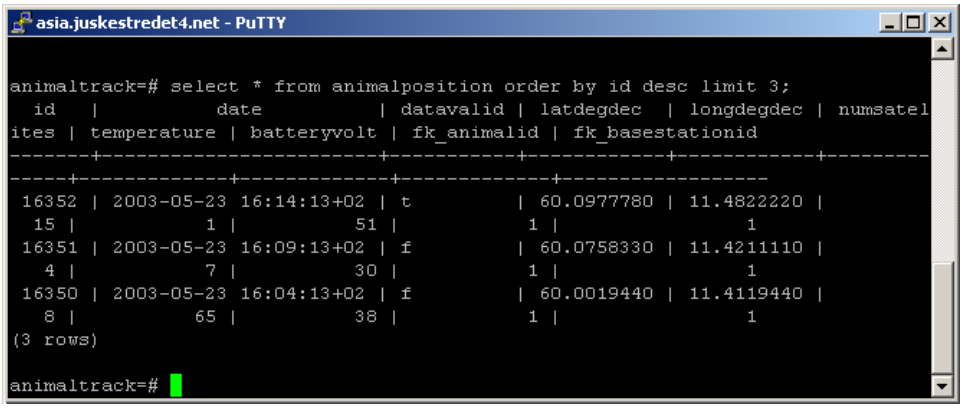


Figure 5.4: *Data received by the server is stored in the database*

5.4.2 Alarm Monitor Data

In Figure 5.5 we see the data stored in the *RegisteredAlarm* table described in Section 4.5.3. The alarms are generated by the Alarm Monitor, and any alarm modules that triggers will insert a record in this table. Because of time constraints no proper alarm modules was developed, but a couple of simple test modules showed the design to work. It is one of these alarms that is registered in the table.

Figure 5.5: *Registered alarms in the database*

Each farmer is able to choose which alarms he want the system to monitor. Figure 5.6 shows two records, stating that the farmer with internal ID 1 has chosen two alarms. Configurable arguments related to the alarm is also stored in the *ChosenAlarm* table, together with how the farmer wants to be notified.

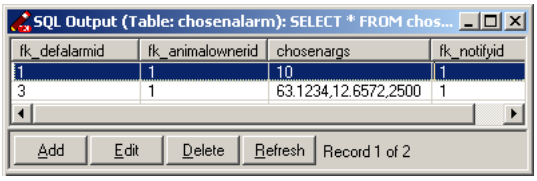


Figure 5.6: *Alarms chosen by the users of the system*

Table *DefinedAlarm* stores all alarms made available to the farmers by the system. The *comment* field is supposed to contain a descriptive comment about the defined alarm module, and may contain HTML data. Each modules sourcecode is stored in the *Pythoncode* field, although the two latest does not contain any code. There was unfortunately not enough time to develop and test these modules, but they were registered in the database to better illustrate the web interface for choosing and configuring alarms as shown in Figure 5.11 and 5.12.

Figure 5.7: *Table with all available alarms*

5.5 Web Application

5.5.1 Login Screen

Figure 5.8 is the webpage the farmer will get when he or she wants to login to the system.

5.5.2 Farmers list of Animals

Once logged in, the farmer can get a list of his animals, with information about their last position, and when it was registered, as shown in Figure 5.9. The farmer may select an individual animal and view more detailed information (Section 5.5.3).

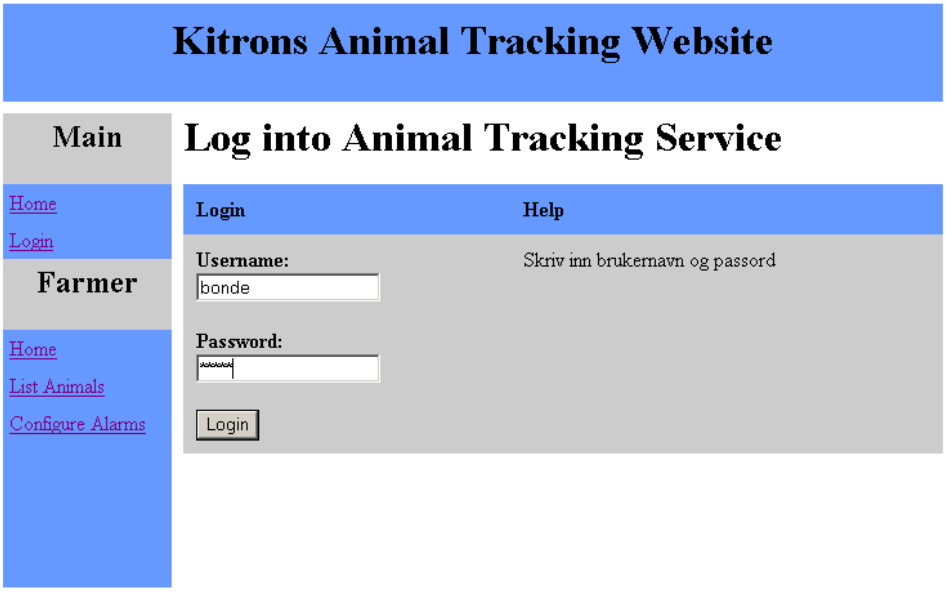


Figure 5.8: *The Web Applications login interface*

Notice also that the data received and processed in Section 5.2 and stored in the database as shown in Figure 5.4 is visible in the first row of the table.

5.5.3 List of stored information for one animal

Figure 5.10 shows the detailed information for the animal with ID 1. All registered positions for this animal is shown, with 15 rows per page, sorted descending by date.

5.5.4 List of chosen and available alarms

The webpage depicted in Figure 5.11 allows the farmer to configure the Alarm Monitor. At the top, a table lists the alarms already chosen by the farmer, and lets him remove them from his list or reconfigure them. The bottom table shows the alarms that is available in the system, but not chosen by the farmer.

5.5.5 Configure an Alarm

Figure 5.12 shows the interface a farmer has for adding alarms to his profile. The same interface is also used to reconfigure an already chosen alarm. The number of input fields for arguments to the alarm is dynamically adapted to the number

Figure 5.9: *Web page listing all animals owned by a farmer*

of arguments specified in the table *DefinedAlarm*, shown in Figure 5.7, and the default values is filled in if a new alarm is to be added. If the interface is used to reconfigure an alarm, the chosen arguments is filled in.

Figure 5.10: *All stored position data for a single animal*

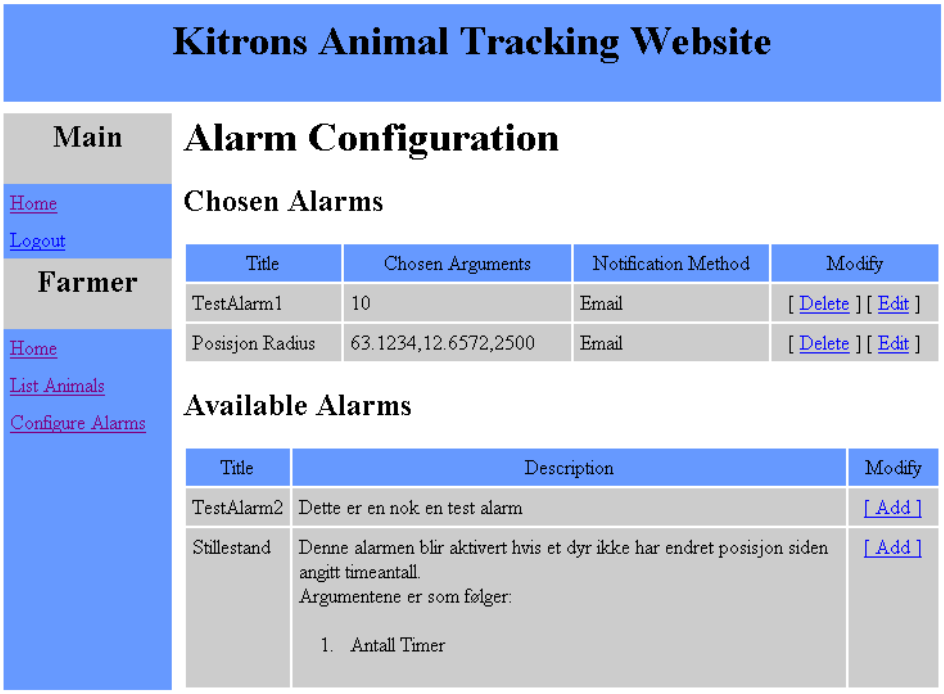


Figure 5.11: *List of chosen and available alarms*

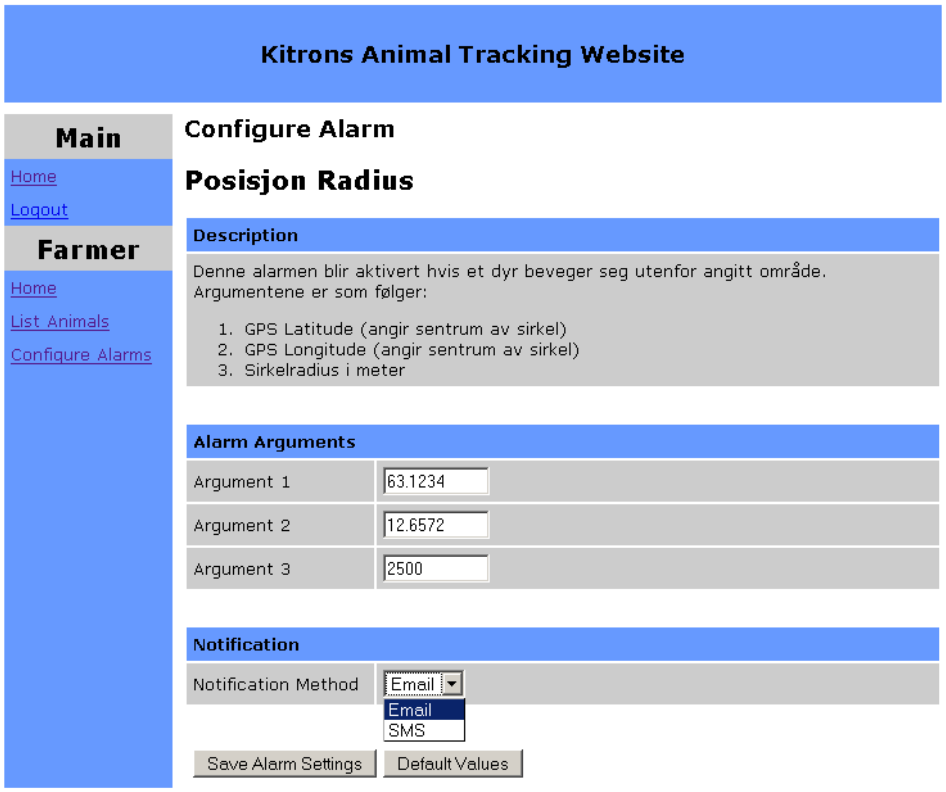


Figure 5.12: *Interface for configuring an alarm*

6. DISCUSSION

6.1 Introduction

In the previous chapter we presented our results. This chapter will contain discussion on various elements of the results and comparisons with elements of the requirements in Chapter 3.

6.2 Radio-collar and Base station Emulator

This component was not part of the requirement, but was developed to allow us to test the other components, because Kitron Development did not have a radio-collar or a base station that we could use for testing purposes. It was the first component to be developed, and it had to be finished early in the project to be able to test the other component.

It was developed for the first specification, as seen in Appendix A.1, but after meeting with Kitron and receiving the second specification (Appendix A.2) we reimplemented large parts of it to conform to this specification. It does not follow the latest recevied specification (Appendix A.3).

Because of the constant changes and the limited time allocated for this component, more time was spent reimplementing the specifications, than making the Emulator produce anything else than pseudo-random data. This is evident in some of the screenshots in Chapter 5.

6.3 Animal Tracking Server

The multithreaded synchronous socket interface of the server was tested with the Radio-collar Emulator continously for three days. Emulating three base stations, with one animal each. Each animal sent its position every second, totaling over 750.000 positions. Neither the server, nor the database had any problems handling this amount of data.

The result of the test was as expected, because the server by default is able to handle ten concurrent connections, and as mentioned we ran the test with three base

stations – each with its own connection. Ten concurrent connections is not an absolute value, but is what we have used as the number of threads available through the threadpool. The number of threads in the threadpool is configureable, and can be adjusted if the amount of base stations exceed the default value. Tests indicate that more than 50 concurrent connections is acheiveable with out degrading performance, but this is highly dependent on the hardware, as well as the underlying OS, used.

Our server breaks the requirements related to message handling, in that it does not analyze all the position data contained within. If a message is received that contains a position that is not received earlier it will not be entered into the database, because the server only looks at the current position message, not the two previous. There is also no checking performed to make sure that an identical position message received from two base stations is not stored in the database. The reason for this is that the Emulator we developed could not generate this data, and as mentioned earlier, we ha no more time to spend on implementing these features in the Emulator.

As specified in the requirement (Section 3.3.2), no security mechanisms were implemented on the socket interface of the server. This is not recommendable, especially if the position data is to be transmitted using GPRS, which uses IP over the public Internet. We will discuss this further in Chapter 7.

Unfortunately the specifications for the base station were altered several times during the project, and the latest specification was received 2003-04-25. By that time the server component was already feature freezed, because the other components relied on it. It was therefore developed early in the project. Because of this the server does not communicate according to the latest specification (Appendix A.3).

6.4 Alarm Monitor

The Alarm Monitor is the component that was developed last, and is the least tested. Because of time constraints no reasonable Alarm Module was developed, and tested. On the other hand, the Alarm Monitor is highly modular, with little business logic contained in the main process.

The test module we implemented and used for testing, indicated that the Alarm Monitor functions properly. Still, this component needs further testing to be included in the complete system. In addition to not completing any proper modules, we did not implement a notification system.

6.5 Database

The database functions well with the other prototype components. Our chosen DBMS, PostgreSQL, supports large amounts of data, and tests indicate that the database scales well when increasing the number of concurrent connections on the server. This is because the server uses a database connection pool to limit the number of physical connections to the database. One drawback of using PostgreSQL is that it has no builtin support for replication, which limits scalability to some extent. There exist an open source project [43], called *pgreplication*, that provides partial replication support, but it is made for the old revision, v6.4.2, and not yet brought up-to-date with the newer revisions v7.2 and v7.3.

We created two database designs, one which was implemented, and a second which was an updated version given to Kitron Development late in the project. The implemented database design was designed according to talks with Kitron and the two first specifications received (Appendix A.1 & A.2). Our revised database model was designed according to the last received specification (Appendix A.3). The *Basestation* table may need some more work, to separate the static data from the data that is more dynamic, and whose variations should be recorded. As the table is design now, all data is considered static.

6.6 Web Application

Our web application adheres quite closely to the requirements specified in Section 3.6. A web interface for listing all animals was developed. As seen in Figure 5.9, in the rightmost column, the possibility of illustrating the animals position on a map was discussed. After consultation with our supervisor we abandoned this task, because Kitron Development did not want to use a third party for this service, which made the task of map visualization too comprehensive.

We performed a study of the most common web application vulnerabilities, and incorporated necessary measures to limit the possibilty of such attacks. In addition we configured the web application server to use server authenticated SSL, and thus encrypting the data transferred between the server and the clients.

The requirements stated that administrative interfaces to perform maintenance tasks was to be developed. We were unfortunately not able to do this within the specified time frame. The software design of the web application makes it rather extensible, so adding an administration section would only require time.

7. FURTHER STUDIES

7.1 Introduction

In this chapter we will propose improvements and changes, as well as areas which should be studied further.

7.2 Radio-collar and Base station Emulator

The emulator should be brought up to date with the latest specification, and then it should be modified to generate data with a bit more lifelikeness. An emulator that followed the current specification would make developing and testing a revised Animal Tracking Server easier.

The data generated by the emulator is correct in terms of the specification in Appendix A.2, but some of the values are not particulary realistic. The position data is correct to some extent, but it does not take the speed of the animals into account, and thus the simulated animals seem to cover a rather large distance.

An updated emulator should also generate invalid data, like duplicate and missing position messages, to enable the testing of these requirements of the server.

7.3 Animal Tracking Server

The server needs to be brought up to date with the latest specification. In addition, handling of duplicated or dropped position messages should be implemented, which should be rather straightforward if the Emulator is able to generate "correct" invalid data.

Securing the communication between the Animal Tracking Server and the base stations should be studied further. Kitron Development felt it was unnecessary in the prototype, but if the server is to be used directly with the base stations, communicating using TCP/IP over the Internet, securing the communication is very important. The server might not be the most obvious choice for a hacker, and one could ask why a hacker would want to compromise the system, but history have shown that hackers really do not need a reason beyond actually managing to compromise the system.

"Security through obscurity" is a term applied by hackers to most OS vendors' favorite way of coping with security holes – namely, ignoring them, documenting neither any known holes nor the underlying security algorithms, trusting that nobody will find out about them and that people who do find out about them won't exploit them. This kind of security should be regarded as void, and a more proactive security policy should be adapted.

We recommend that the Animal Tracking Server and the base stations is fitted with a cryptographical security mechanism, like SSL, IPsec or some other VPN solution. With digital certificates authenticating both the server and the base stations and an encrypted transmission channel, the data could be trusted to a much higher degree.

7.4 Alarm Monitor

The Alarm Monitor component was the last component to be developed, and because of time constraints it was not properly completed. Tests with our small test-module indicated that the design functioned as intended, but further testing should be performed. The modularity specified in the requirements was acheived by storing the modules as source code in the database, and a proper Alarm Monitor module would be easy to implement.

Support for notifying the farmer of an alarm needs to be implemented. This could possibly require some redesign of the database, especially if this function also needs modularity. The problem would be to store the data needed for a proper notification to be made, for instance the cellular phone number if an SMS is to be sent.

This could be done by using the same kind of design used for the alarm modules. One table containing a set of defined notification methods with the amount of arguments needed to perform this notification, and one table that stores the notification methods the farmer have chosen for the particular alarm, together with the user-dependent arguments needed for notification.

7.5 Database

PostgreSQL proved to be more than able to handle the workload put on it during our project, but has some short comings when it comes to replication, and thus scalability through replication.

All the components use database APIs that abstract the used DBMS, so replacing the DBMS would not require much code change. As mentioned in Section 4.5.2 we were supposed to test the system using MS-SQL Server, but because of theft of the server it was not possible. This should be done, to see how well MS-SQL Server

would handle the workload, and also to test the replication features that MS-SQL Server has. Testing other DBMS', like MySQL, might also be of interest.

We developed two different database models, one which was implemented in our system, and one that was an updated version that we provided Kitron Development with, for their testing. Unfortuneatly there were no time to update and test all the other components with the latest database design. The updated database model should also be revised, to handle the data concerning the base stations better. In addition, an Access Control List (ACL) table should be added, to enable differentiated access for the users stored in the database.

7.6 Web Application

Communication between the farmers web browser, and the web application is secured using SSL. The farmer is also able to authenticate the server, because the server provides him with a SSL certificate. We generated a so-called self signed certificate, which provides the same degree of security, but will give the user warnings because the certificate is not issued by a trusted third party, like Verisign [41]. If the web application should be put into production use, a proper SSL certificate should be obtained.

If a higher degree of accountability and authentication of the user is desirable, it is possible to demand certificates from the users. This would allow the server to authenticate the users with a high degree of confidence.

The interfaces presented to the farmer could be modified to contain even more information. Especially if the database model is updated. In addition many minor useability features could be implemented. Like allowing the user to specify what columns he wants the data sorted by, and similar small – but useful – features.

We were unable to implement an administrative interface within the time limits of the project, this ought be performed. To be able to implement this, the database must be given a table that contains information about what access the different users of the system should have. Optionally the administration interface could use client authenticated SSL which would eliminate the need for changes to the database model.

8. CONCLUSION

In this thesis we have designed, implemented and tested a part of Kitron Development's Animal Tracking System. During our work we have evaluated security, scalability and cost. Part of the implementation suffered under the consequences of working in parallel with a development team at Kitron. This meant frequent changes in the specifications and subsequent modification of already implemented code. A consequence of this was that some of the components was not implemented according to the last received specification. None the less, our components interact with each other and perform the tasks required, although some of the components differ from the specification. These aberrations are easy to mend given the needed time. During this project we have shown that it is absolutely possible to develop a working system.

We have also presented a study of different communication technologies between the base station and our server. The cost study we performed showed that prices for transmitting relatively small amounts of data using either GSM or GPRS is well within the limit that Kitron Development wanted. We also found that the transmission speed would not pose any problem, regardless of transmission technology. Even though the study showed that GSM was the cheapest alternative, when transmitting only position data. It also showed that almost every change that can be made to the system would influence GSM more than GPRS. We expect that the system will be altered and expanded in the future, making GPRS the best suited transmission alternative.

Our contribution combined with the work done at Kitron will hopefully help both farmers and researchers working with grazing animals, enabling them to collect live data from the animals without physical precense.

BIBLIOGRAPHY

[1] (1996) Husdyr på utmarksbeite. [Online]. Available: http://www.dyrebeskyttelsen.no/artikler/963_2.shtml 1

[2] Statens kartverk, pressesenter. [Online]. Available: http://www.statkart.no/IPS/?module=Articles;action=Article.publicShow;ID=776 2

[3] M. S. Ali. Enterprice wireless technology. [Online]. Available: http://www.enterprisewireless.co.uk/page.cfm/Link=143 5

[4] K. Holley and I. Doig, *Digital cellular telecommunications system (Phase 2+) (GSM); Alphabets and language-specific information (GSM 03.38 version 7.2.0 Release 1998)*, ETSI, July 1999. [Online]. Available: http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=6821 9

[5] L. Pesonen. (1999) Gsm interception. [Online]. Available: http://www.net-security.sk/telekom/phreak/radiophone/gsm/gsm-secur/gsm-secur.html 11

[6] D. Margrave. Gsm security and encryption. [Online]. Available: http://www.net-security.sk/telekom/phreak/radiophone/gsm/gsm-secur/gsm-secur.html 12, 14

[7] B. Hall. (2001) Beej's guide to network programming - using internet sockets. [Online]. Available: http://www.ecst.csuchico.edu/~beej/guide/net/html/ 15

[8] J. Postel, "RFC 768: User datagram protocol," Aug. 1980, status: STANDARD. See also STD0006. [Online]. Available: {ftp://ftp.internic.net/rfc/rfc768.txt},{ftp://ftp.math.utah.edu/pub/rfc/rfc768.txt} 15

[9] ——, "RFC 793: Transmission control protocol," Sept. 1981, status: STANDARD. See also STD0007. [Online]. Available: {ftp://ftp.internic.net/rfc/rfc793.txt},{ftp://ftp.math.utah.edu/pub/rfc/rfc793.txt} 15

[10] S. Rushing. Asynchronous socket programming. [Online]. Available: http://squirl.nightmare.com/medusa/async_sockets.html 16

[11] S. Low. The world of select(). [Online]. Available: http://www.lowtek.com/sockets/select.html 16

[12] G. Travis. (2003) Use select for high-speed networking. [Online]. Available: http://www.javaworld.com/javaworld/jw-04-2003/jw-0411-select.html 16

[13] K. E. Hickman, "The SSL protocol," Netscape Communications Corp., Tech. Rep., Feb 1995. [Online]. Available: http://wp.netscape.com/eng/security/SSL_2.html 17

[14] J. Bradley, "The sslp reference implementation project," Master's thesis, University of Bristol, 1995. [Online]. Available: http://www.cs.bris.ac.uk/~bradley/publish/SSLP/cover.html vii, 18

[15] R. Laboratories. (2000) RSA laboratories' frequently asked questions about today's cryptography, version 4.1. [Online]. Available: http://www.rsasecurity.com/rsalabs/faq/5-1-2.html 17

[16] The OpenSSL project. [Online]. Available: http://www.openssl.org/ 18

[17] stunnel.org. Stunnel – universal SSL wrapper. [Online]. Available: http://www.stunnel.org/ 18, 33

[18] IP security protocol charter. [Online]. Available: http://www.ietf.org/html.charters/ipsec-charter.html 18

[19] S. Kent and R. Atkinson, "RFC 2402: IP authentication header," Nov. 1998, obsoletes RFC1826. Status: PROPOSED STANDARD. [Online]. Available: ftp://ftp.internic.net/rfc/rfc1826.txt,ftp://ftp.internic.net/rfc/rfc2402.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc1826.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc2402.txt 18

[20] ——, "RFC 2406: IP Encapsulating Security Payload (ESP)," Nov. 1998, obsoletes RFC1827. Status: PROPOSED STANDARD. [Online]. Available: ftp://ftp.internic.net/rfc/rfc1827.txt,ftp://ftp.internic.net/rfc/rfc2406.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc1827.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc2406.txt 19

[21] D. Maughan, M. Schertler, M. Schneider, and J. Turner, "RFC 2408: Internet Security Association and Key Management Protocol (ISAKMP)," Nov. 1998, status: PROPOSED STANDARD. [Online]. Available: ftp://ftp.internic.net/rfc/rfc2408.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc2408.txt 19

[22] N. Ferguson and B. Schneier, "A cryptographic evaluation of IPsec," CounterPane, 3031 Tisch Way, Suite 100PE, San Jose, CA 95128, USA, Tech. Rep., 2000. [Online]. Available: http://www.counterpane.com/ipsec.html 19

[23] M. Nash. Expresso database connection pooling. [Online]. Available: http://www.jcorporate.com/html/products/expresso/dbpooling.html 23

[24] W. A. Klimke, "Web application development using open source and java technologies," Master's thesis, Massachusetts Institute of Technology, 2001. 24

[25] S. Bakken, A. Aulbach, E. Schmid, J. Winstead, L. T. Wilson, R. Lerdorf, A. Zmievski, and J. Ahto, *PHP Manual*, 2003. [Online]. Available: http://www.php.net/manual/en/ 25

[26] J. Parshall. Active server pages history. [Online]. Available: http://www.jimparshall.net/asp/asp_history.aspx 26

[27] J. Byous. Java technology: An early history. [Online]. Available: http://java.sun.com/features/1998/05/birthday.html 26

[28] Apache jakarta tomcat homepage. [Online]. Available: http://jakarta.apache.org/tomcat/index.html 27, 48

[29] Javaserver pages technology. [Online]. Available: http://java.sun.com/products/jsp/ 27

[30] J. Hunter and W. Crawford, *Java Servlet Programming*, 2nd ed. O'Reilly, 2001. 27

[31] Bugtraq. [Online]. Available: http://www.securityfocus.com/archive/1 28

[32] CERT. Cert advisory ca-2000-02 malicious html tags embedded in client web requests. [Online]. Available: http://www.cert.org/advisories/CA-2000-02.html 29, 30

[33] *PostgreSQL 7.2 User's Guide*, The PostgreSQL Global Development Group. [Online]. Available: http://www.postgresql.org/docs/view.php?version=7.2&idoc=0&file=user.html 29

[34] G. van Rossum and P. S. Foundation. Python homepage. [Online]. Available: http://www.python.org 38, 40

[35] J. Nibbens. Jon's python modules - dbpool module. [Online]. Available: http://jonpy.sourceforge.net/dbpool.html 39

[36] P. Team. PostgreSQL homepage. [Online]. Available: http://www.postgresql.org 43

[37] The mono project. [Online]. Available: http://go-mono.org/index.html 47

[38] Apache jakarta struts homepage. [Online]. Available: http://jakarta.apache.org/struts/index.html 48

[39] Velocity manual (english). [Online]. Available: http://jakarta.apache.org/velocity/user-guide.html 49

[40] T. Sturm, J. von Voss, and M. Boger, "Generating code from uml with velocity templates," *Lecture Notes In Computer Science*, vol. 2460, pp. 150–161, 2002. 49

[41] Verisign homepage. [Online]. Available: http://www.verisign.com/ 50, 65

[42] Comodo group homepage. [Online]. Available: http://www.comodogroup.com/ 50

[43] The pgreplication project. [Online]. Available: http://gborg.postgresql.org/project/pgreplication/projdisplay.php 62

[44] J. Hultgreen and R. I. Sivertsen, "Radio system for reliable data transfer," Master's thesis, Høyskolen i Agder, Grimstad, 2002.

[45] J. Han and M. Kamber, *Data Mining Concepts and Techniques*. Morgan Kaufmann Publishers, 2001.

[46] C. C. Fleming and B. von Halle, *Handbook of Relational Database Design*. Addison-Wesley Publishing Company, 1989.

[47] Database theory. [Online]. Available: http://www.frick-cpa.com/ss7/Theory_TOC.asp

[48] B. Schneier, *Applied Cryptography*, 2nd ed. John Wiley & Sons, Inc., 1996.

[49] CCITT. Security architecture for open systems interconnection for ccitt applications. [Online]. Available: http://fag.grm.hia.no/IKT7000/litteratur/paper/x800.pdf

[50] Nasa programs, gps. [Online]. Available: http://samadhi.jpl.nasa.gov/msl/Programs/gps.html

[51] C. Date, *An introduction to database systems*, 7th ed. Addison Wesley Longman, Inc., 2000.

[52] P. Neff, "Web application security," 2002. [Online]. Available: http://www.patrice.ch/en/computer/web/articles/2002/web_security.pdf

[53] J. E. Nakkestad, K. Halvorsen, J. K. Lohne, and K. Pedersen, "Wap-basert værstasjon," Master's thesis, Høyskolen i Agder, Grimstad, 2000.

[54] B. T. Mynatt, *Software Engineering with Student Project Guidance*. Prentice-Hall, 1990.

[55] J. L. Whitten, L. D. Bentley, and K. C. Dittman, *Systems Analysis and Design Methods*, 5th ed. McGraw-Hill Irwin, 2001.

[56] Telenor. Telenor mobil GPRS priser. [Online]. Available: http://telenormobil.no/bedrift/priser/tjenester/gprs

[57] ——. Telenor mobil GSM bare data priser. [Online]. Available: http://telenormobil.no/bedrift/priser/abonnement/gsmbaredata/#Datatrafikk

[58] Netcom, “Prisliste netcom bedriftsavtale.”

ABBREVIATIONS

AH	IP Authentication Header
ASP	Active Server Pages
BTS	Base Transceiver Station
DBMS	DataBase Management System
ESP	IP Encapsulation Security Payload
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communications
HLR	Home Location Register
HSCSD	High Speed Circuit Switched Data
HTML	HyperText Markup Language
IP	Internet Protocol
JSP	Java Server Pages
MS	Mobile Station
MSC	Mobile services Switching Center
NAVSTAR	Navigation System with Timing And Ranging
NOK	Norske kroner (Norwegian currency)
ORDBMS	Object Relational DataBase Management System
RDBMS	Relational DataBase Management System
SA	Security Association
SGSN	Serving GPRS Support Node
SIM	Subscriber Identity Module
SMS	Short Message Service
SRES	Signed RESponse
SSL	Security Socket Layer
TCP	Transfer Control Protocol
UDP	User Datagram Protocol
VHF	Very High Frequency
XSS	Cross Site Scripting

APPENDIX

A. SOFTWARE DESIGN DESCRIPTION FOR BASESTATION

A.1 First Revision

VHF Meldingsformat

Meldingene fra dyreenheten skal bestå av en rekke ledd med forskjellig informasjon.

Meldingene fra dyreenhet til basestasjon skal ha følgende format beskrevet nedenfor. Meldingen har en fast bredde på 192 bit (24 bytes). Bit 0 er det bit'et som sendes først, bit 191 sendes sist. Alle meldingsdeler er orientert slik at mest signifikante bit sendes først.

Bit	Length	Beskrivelse	Verdi
0-7	8	Preamble byte 0	0x00
8-15	8	Preamble byte 0	0x00
16-23	8	Sync byte 0	0xC4
24-31	8	Sync byte 1	0xD7
32-39	8	Meldingstype/Versjon	0x01
40-79	40	ID nummer. Hex kodet decimal	0 – 1 099 511 627 775
80-84	5	Klokke: Time	0-23
85-90	6	Klokke: Minutt	0-59
91-96	6	Klokke: Sekund	0-59
97	1	Gyldige data (1). Ikke gyldige (0)	0 | 1
98-104	7	Breddegrad: Grader	0-89
105-110	6	Breddegrad: Minutter	0-59
111-124	14	Breddegrad: Minutter desimal	0-9999
125	1	Nord (1) / Sør (0) indikering	0 | 1
126-134	9	Lengdegrad: Grader	0-360
135-140	6	Lengdegrad: Minutter	0-59
141-154	14	Lengdegrad: Minutter desimal	0-9999
155	1	Øst (1) / Vest (0) indikering	0 | 1
156-162	7	Temperatur sensor 1: (Grader celcius)	-64- +64
163	1	Temperatur sensor 1: 0.5 decimal	0 | 1
164-170	7	Temperatur sensor 2: (Grader celcius)	-64- +64
171	1	Temperatur sensor 2: 0.5 decimal	0 | 1
172-175	4	Reservert	0b0000-0b1111
176-191	16	16-bit CRC [1])	0x0000-0xFFFF

[1]) CRC generator polynom er som følger: $X^{16}+X^{15}+X^{14}+X^{12}+X^5+X^3+X+1$. Startverdi er 0x0000. Den resulterende verdien er til slutt skiftet 1 gang til venstre.

A.2 Second Revision

Created	Last saved	Printed	Revision	Issue	Document no.	Reference
2003-02-14	2003-02-22		1		SPS002751	

Prepared by	Date	Approved by	Date	Checked by	Date
Erik Hardeng	14.02.2003				

Radiobjelle, systemspesifikasjon

Innhold

Document Change Record

Rev.	Date	Pages affected	Change Description
1	14.02.2003	Alle	Første versjon

1 Innledning

1.1 Formål

Prosjektet ”Radiobjelle” er et prosjekt for utvikling av en elektronisk sauebjelle som kan brukes for lokalisering og overvåking av husdyr på beite. Høsten 2002 ble det laget en prototypløsning for å teste basisegenskaper ve systemet. Våren 2003 blir det utviklet og produsert en 1000-serie av systemet. Denne systemspesifikasjonen inneholder spesifikasjoner for 1000-serien.

1.2 Definisjoner og forkortelser

Forkortelse	Forklaring	Kommentar
NSG	Norsk sau- og geitalslag	
GPS	Global Positioning System	Satellittnavigasjonssystem
VHF	Very High Frequencies	Brukes gjerne om frekvenser fra 100 – 300 MHz
CRC	Cyclic Redundancy Check	Beregnet tverrsum som avslører om det er feil i en datamengde
NMEA	National Maritime Electronics Association	Standardiseringsorgan i USA som har laget standard for GPS-posisjoner
Dyreenhet	Selve ”radiobjellen”, klaven med elektronikk som dyrene har rundt halsen	
Basestasjon	Radiomottageren og datamaskinen som tar i mot meldinger fra dyreenhetene	
Sentral	Den sentrale datamaskinen som tar i mot meldinger fra basestasjonene	

1.3 Referanser

RAP002573 Radiobjelle sluttrapport 2002 Endelig 16122002.doc

2 Dokumentoversikt

Dette dokumentet gir en generell systemspesifikasjon for prosjektet. Alle andre dokumenter skal være i henhold til de spesifikasjonene som står i dette dokumentet.

Dokumentet vil komme i nye versjoner i løpet av produktutviklingen.

Kapittel 3 gir en bakgrunn for prosjektet. Kapittel 4 inneholder systemspesifikasjon for prosjektet. Vedleggene i kapittel 5 beskriver GPS-nøyaktighet, meldingsformat og alarmer.

3 Bakgrunn for prosjektet

Radiobjelleprosjektet er beskrevet I tidligere rapporter, inkludert RAP2573, se henvisning ovenfor.

1000-serien skal brukes til å teste systemet under realistiske driftsforhold.

4 Systemspesifikasjon

Systemet består av tre deler, en dyreenhet, en basestasjon og en sentral. Det finnes normalt et stort antall dyreenheter for hver basestasjon, mens det bare er én sentral.

4.1 Generelle spesifikasjoner

Prinsippet for systemet er først og fremst å kunne lokalisere radiobjellen (dyreenheten). I tillegg skal radiobjellen overføre data til basestasjonen.

Det er bare kommunikasjon fra dyreenheten til basestasjonen, ikke tilbake.

Emne	**Spesifikasjon**	**Kommentar**
Lokalisering	Dyreenheten finner posisjon med GPS og sender denne til basestasjonen	
Data fra sensorer	Dyreenheten sender data fra sensorer til basestasjonen, styrt av fastsatte regler.	
Meldinger	Meldingene til basestasjonen har et standardisert format	Se vedlegg 2
Behandling av data	Basestasjonen tar i mot meldinger fra dyreenhetene og korrigerer posisjonene med avvik fra egen GPS	
Lagring av data	Basestasjonen har kontakt med en felles sentral datamaskin (”sentralen”) gjennom GSM. Alle mottatte meldinger sendes videre til sentralen og lagres der.	
Brukertilgang	Brukere har tilgang til mottatte meldinger gjennom sentralen.	
Alarmer	Sentralen gir alarm til brukerne dersom det kommer inn alarmer eller dersom den finner at posisjonsdata gir grunnlag for alarm.	
Frekvens	Dyreenhetene og basestasjonen bruker en felles frekvens	Det er ikke avklart hvordan flere nett arbeider sammen. En naturlig løsning er å reservere et sett med enkeltfrekvenser som kan distribueres mellom systemer slik at samme frekvens gjenbrukes langt unna den første

Kollisjonssikring	Dyreenhetene har ikke kollisjonssikring når de sender meldinger. Alle meldinger har feilsikringskode (CRC).	Hver melding er svært kort, gjerne 0, 5s. I løpet av 6 timer er det teoretisk plass til ca. 1800 x 6 sendere, 10800 sendere. Ett system vil sjelden håndtere mer enn kanskje 1000 dyr.

4.2 Dyreenhet

Dyreenheten plasseres på dyret når dyret sendes ut på beite.

Emne	**Spesifikasjon**	**Kommentar**
Lokalisering	Dyreenheten bruker en GPS-mottager for å finne posisjon.	
GPS-mottager	Finner posisjon i hht GPS-standard	
Radiosender	Sender posisjon og andre data	Se egne spesifikasjoner nedenfor
Meldinger	Meldingene til basestasjonen har et standardisert format, se vedlegg 2	
Styringsenhet	Kontrollerer GPS-mottageren og radiosenderen og styrer disse etter fastsatte regler.	
Spenning	3,3 V	Regulert fra ca. 4,5 V
Strømtrekk	Ikke fastlagt	Typisk 1 mA
Kraftforsyning	3 stk alkaliske batterier	Levetid ca. 2 år.
Temperaturområde	Ikke bestemt	Forslag: - 10 til + 70 grader

Radiosenderen har disse spesifikasjonene, se også: NOT002441 Forslag til enkel og nøyaktig FM-sender.doc

Parameter	**Spesifikasjon**	**Kommentar**
Frekvens	Ca 80 MHz eller ca 140 MHz	Ikke fastlagt, diskuteres med NPT
Effekt	Antagelig ≤ 500 mW	Nok for rekkevidde, gunstig mht lisens
Modulasjon	FSK, ≤ 25 kHz	Ikke fastlagt, NPT kan ha synspunkter
Frekvensvalg	Stillbar via uP	Ikke endelig fastlagt
Justering	Ingen	Ikke ønsket pga masseproduksjon
Temperatur	-10 grader til +60 grader	Ikke fastlagt, må tåle frost og direkte sol
Pris	< 20 kr	Ikke fastlagt, men lav pris er generelt viktig

Styringsenheten har regler for styring av GPS-mottageren og radiosenderen. Aktuelle regler er:

Posisjon	Sendes 4 ganger pr døgn	Ikke endelig bestemt
Innsanking	Posisjon sendes oftere i en bestemt periode, f.eks. etter 1/9.	Forslag: Posisjon 4 ganger pr time.
Stillstand	Posisjon sendes oftere dersom posisjonen	Se detaljert beskrivelse i vedlegg 1.

	ikke endrer seg etter en viss tid.	
Hviletid	Dyrene kan hvile lengre tid noen ganger, f.eks. om natten, uten at det gis alarm	

Meldingene til basestasjonen har et bestemt format som lett kan utvides, se vedlegg 2. Meldingene skal inneholde følgende informasjon:

Modul	**Format**	**Kommentar**
Identifikasjon	12-sifret serienummer fra øremerke	
Posisjon	NMEA-0183	
Tidsangivelse	Dato og tid	

I tillegg kan meldingene inneholde følgende informasjon men det er ikke noe krav.

Alarm	00-99, se vedlegg 3	Plass for 100 forskjellige alarmer
Sensorer	Informasjon fra forskjellige sensorer:	Det er ikke bestemt å ha sensorer i første utgave av dyreenheten.
Utetemperatur	Temperatur i grader celsius	
Dyrets temperatur	Temperatur i grader celsius fra en sensor	
Dyrets puls	Puls i antall pulser pr minutt	
Støynivå	Støynivå i dB(A)	Oppløsningen kan være svært grov, f.eks. bare 50 – 60 – 70 db.
Sabotasje	Klaven er åpnet uten tillatelse	Basert på skjult bryter i klaven
Manuell	Manuelt utløst alarm	Enkel mulighet for å be om hjelp dersom annen radiokommunikasjon ikke er mulig
TimeToFix	Tiden det tar fra GPS varmstartes til første gyldige posisjon returneres	For å kunne få statistikk for hvor mye GPS egentlig er på. Hvis GPS er oppe og ikke får fix, går ned igjen og så opp, tas også tiden for forrige periode med, slik at tiden blir fra siste gyldige posisjon til neste med aktiv GPS. Anta 1024 (10bit) sekunder. 1024 betyr >= 1024 sekunder.

OBS: Disse alarmene og meldingene er ikke en del av spesifikasjonen, men kan inkluderes senere.

Alle alarmer skal normalt sendes umiddelbart. Deretter sendes normalt nye alarmer med korte intervaller for å gjøre sporing lettere.

4.3 Basestasjon

Basestasjonen tar i mot meldinger fra dyreenhetene. Samtidig har basestasjonen sin egen GPS-mottager som finner posisjonen. Siden basestasjonen står i ro vil basestasjonen hele tiden kunne regne ut det aktuelle feilsignalet for GPS, ved å sammenligne øyeblikksverdien med et gjennomsnitt av et stort antall tidligere målinger. De mottatte posisjonsverdiene korrigeres så med feilsignalet. Deretter lagres posisjonen og eventuelle data i en enkel database.

Emne	Spesifikasjon	Kommentar
Mottager	Følsom mottager med fast frekvens.	Motorola GM950 er foreslått
Antenne	Rundstrålende	Antagelig horisontal polarisering
Modem	1200 baud	Forslag: Etic MR1200
Styringsenhet	Portabel PC	Modell ikke bestemt
GPS	Lokal GPS-mottager for å finne posisjonsfeil	Forslag: u-Blox EV-kit
GSM	Wavecom Fastrack 1200	
Database	Ikke bestemt	
Kraftforsyning	Ikke bestemt	Nettdrift er foretrukket løsning, men batteridrift bør være mulig
Temperaturområde	Ikke bestemt	Forslag: - 10 til + 70 grader, men vi kan godta begrensninger i prøveperioden.

Se også: NOT002422 Forslag til enkel basestasjon.doc

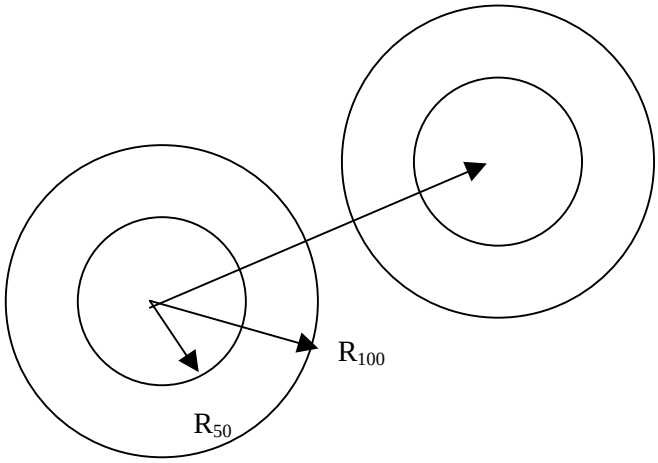
5 Vedlegg

5.1 Vedlegg 1: GPS-nøyaktighet

En GPS-mottager måler riktig posisjon med en feilradius på omtrent 5 meter. Vi kaller denne verdien for R_{50}. Feilradius vil si at f.eks. 50 % av alle målinger faller innenfor en radius på 5 m. u-Blox oppgir følgende tall for CEP, Circular Error of Positioning, ved kontinuerlig måling (horisontal feil):

CEP	Radius
50%eqv.CEP (2D)	2.1 m
Sigma-1 (68.3%)	2.8 m
Sigma-2 (95.5%)eqv.R95	4.9 m
Sigma-3 (99.7%)	7.9 m

La oss da anta at en feilradius på 10 m dekker 100 % av alle målinger. Vi kaller denne radien for R_{100}.



For å være svært sikre på at dyret har beveget seg mellom to målinger må vi derfor vente tiden t, til dyret har flyttet seg 2 x R_{100}. Farten blir da:

$$(2 \times R_{100}) / t$$

Vi må anslå en passende tid for t. Et anslag kan være 10 – 30 minutter. Det gir en gjennomsnittelig minstefart på 20 m / 10 minutter (hhv 20 m / 30 minutter) = 120 m / h = 0,12 km/h (hhv 0,04 km/h).

Det har ingen mening i å måle posisjonen oftere så lenge usikkerheten i posisjonen er så stor.

Et alternativ hadde vært å sende posisjonen oftere og overlate til basestasjonen, som korrigerer for GPS-feil, å avgjøre om dyret står i ro. Da blir imidlertid strømtrekket for dyreenheten en del høyere.
En passende regel for stillstandsalarm er: Ved to påfølgende tilfeller av stillstand gis alarm.

1. Styringsenheten slår på GPS og måler posisjonen.
2. Dersom posisjonen har forandret seg mindre enn 2 x R_{100} etter forrige måling settes et flagg.
3. Posisjonen måles på nytt etter 10 (hhv 30) minutter og det gis alarm dersom flagget settes på ny.

5.2 Vedlegg 2: Meldingsformat

Meldingene fra dyreenheten skal bestå av en rekke ledd med forskjellig informasjon.

Meldingene fra dyreenhet til basestasjon skal ha følgende format beskrevet nedenfor. Meldingen har en fast lengde på **TBD** bit (**TBD** bytes). Bit 0 er det bit'et som sendes først, bit nn sendes sist. Alle meldingsdeler er orientert slik at mest signifikante bit sendes først.

Bit nr			Bit lengde	Byte	Beskrivelse
0	-	7	8	1	Preamble byte 0
8	-	15	8	2	Preamble byte 1
16	-	23	8	3	Preamble byte 2
24	-	31	8	4	Preamble byte 3
32	-	39	8	5	Preamble byte 4
40	-	47	8	6	Preamble byte 5
48	-	55	8	7	Sync byte 0
56	-	63	8	8	Sync byte 1
64	-	71	8	9	Meldingstype/Versjon
72	-	83	12	10 4/8	Adresse/ID. Hex kodet decimal
84	-	88	5	11 1/8	Klokke: Time
89	-	94	6	11 7/8	Klokke: Minutt
95	-	100	6	12 5/8	Klokke: Sekund
101	-	101	1	12 6/8	Gyldige data (1). Ikke gyldige (0)
102	-	108	7	13 5/8	Breddegrad: Grader
109	-	114	6	14 3/8	Breddegrad: Minutter
115	-	128	14	16 1/8	Breddegrad: Minutter desimal
129	-	129	1	16 2/8	Nord (1) / Sør (0) indikering
130	-	138	9	17 3/8	Lengdegrad: Grader
139	-	144	6	18 1/8	Lengdegrad: Minutter
145	-	158	14	19 7/8	Lengdegrad: Minutter desimal
159	-	159	1	20	Øst (1) / Vest (0) indikering
160	-	163	4	20 4/8	Antall satellitter mottatt
164	-	170	7	21 3/8	Temperatur sensor (Grader celsius)
171	-	176	6	22 1/8	Batterispenning (Volt)
177	-	183	7	23	Reservert
Foregående posisjon (aktuell posisjon - 1)					
184		188	5	23 5/8	Klokke: Time
189		194	6	24 3/8	Klokke: Minutt
195		200	6	25 1/8	Klokke: Sekund
201		201	1	25 2/8	Gyldige data (1). Ikke gyldige (0)
202		208	7	26 1/8	Breddegrad: Grader
209		214	6	26 7/8	Breddegrad: Minutter
215		228	14	28 5/8	Breddegrad: Minutter desimal
229		229	1	28 6/8	Nord (1) / Sør (0) indikering
230		238	9	29 7/8	Lengdegrad: Grader
239		244	6	30 5/8	Lengdegrad: Minutter
245		258	14	32 3/8	Lengdegrad: Minutter desimal
259		259	1	32 4/8	Øst (1) / Vest (0) indikering
260		263	4	33	Antall satellitter mottatt
Foregående posisjon (aktuell posisjon - 2)					
264		268	5	33 5/8	Klokke: Time
269		274	6	34 3/8	Klokke: Minutt
275		280	6	35 1/8	Klokke: Sekund
281		281	1	35 2/8	Gyldige data (1). Ikke gyldige (0)
282		288	7	36 1/8	Breddegrad: Grader
289		294	6	36 7/8	Breddegrad: Minutter
295		308	14	38 5/8	Breddegrad: Minutter desimal
309		309	1	38 6/8	Nord (1) / Sør (0) indikering
310		318	9	39 7/8	Lengdegrad: Grader
319		324	6	40 5/8	Lengdegrad: Minutter
325		338	14	42 3/8	Lengdegrad: Minutter desimal
339		339	1	42 4/8	Øst (1) / Vest (0) indikering
340		343	4	43	Antall satellitter mottatt
Slutt på melding					
344		359	16	45	16-bit CRC 1)

[1]) CRC generator polynom er som følger: $X^{16}+X^{15}+X^{14}+X^{12}+X^5+X^3+X+1$. Startverdi er 0x0000. *Den resulterende verdien er til slutt skiftet 1 gang til venstre.(Fjernes!)*

A.3 Third Revision

SW Design Description for basestasjon

Created	Last saved	Printed	Revision	Issue	Document no.	Reference
2003-03-25	2003-04-24	2003-04-25	PA5		BSK002798	

Prepared by	Date	Approved by	Date	Checked by	Date
ABR	2003-03-05				

SW Design Description for basestasjon

Innhold

Document Change Record

Rev.	Date	Pages affected	Change Description
PA1	31.03.2003	Alle	
PA2	03.04.2003	Alle	Klar for granskning
PA3	22.04.2003	Alle	Oppdatert etter granskning
PA4	23.04.2003	Alle	Opdatert etter granskning
PA5	24.04.2003	Alle	Opdatert etter granskning

1 Innledning

1.1 Hensikt

Prosjektet "Radiobjelle" er et prosjekt for utvikling av en elektronisk klave som kan brukes for lokalisering og overvåking av husdyr på beite. Den elektroniske klaven sender posisjonsmeldinger til en basestasjon som er plassert i beiteområdet. Basestasjonen sender meldingene videre til en sentral. Hensikten med dette dokumentet er å beskrive hvordan programvaren i basestasjonen skal implementeres.

1.2 Gyldighetsområde

Denne spesifikasjonen gjelder for en førsteutgave (prototyp/testutgave) av basestasjonen, som skal brukes for å samle erfaringsdata sommersesongen 2003. Dokumentet er et internt Kitron-dokument.

1.3 Referanser

REF. NR.	DOKUMENTNUMMER OG -NAVN
1	SPS002751 Radiobjelle – Systemspesifikasjon.doc
2	SPS002779 Kravspesifikasjon for basestasjon

1.4 Definisjoner og forkortelser

FORKORTELSE	FORKLARING
NSG	Norsk sau- og geitalslag
VHF	Very High Frequency
GSM	Global System for Mobile Communication
GPRS	General Packet Radio Service
GPS	Global Positioning System
S/N	Serial Number
FEC	Forward Error Correction
uC	Mikrokontroller
ECC	Error Correction Code
AT-kommando	Attention command set, kommandoer utviklet for modemer
SIM-kort	(subscriber identification module). Et smartkort i GSM-telefonen som inneholder informasjon om abbonementet.
PIN-kode	(Personal Identification Number). Kode for personlig identifisering.
FIFO	First in first out
NMEA	National Marine Electronics Association.

1.5 Terminologi

TERM	FORKLARING
Basestasjon	Enhet plassert ute i felt. Denne enheten samler inn data fra dyr på beite via et VHF-samband og sender data videre til en sentral

Basestasjon SW	Med dette menes den samlede softwaren som styrer basestasjonen, uavhengig av hvor den er implementert (uController, PC, GSM-/GPRS-modem)
Dyreenhet	Radioenhet som sitter på dyret og sender posisjonsdata til en basestasjon
Sentral	Sentral kommunikasjonsenhet som kontrollerer flere basestasjoner. Posisjonsdata og alarmer som kommer inn fra basestasjoner blir lagret i sentralen

2 Konsept

Basestasjonen tar imot meldinger fra dyreenhetene i området omkring basestasjonen via et VHF-samband. Basestasjonen sender meldingene videre til en sentral via et trådløst modem (GSM/GPRS). Programvaren som styrer basestasjonen implementeres på en ATMega128 mikrokontroller. Basestasjonen består av et sett moduler, som vist i figur 1:

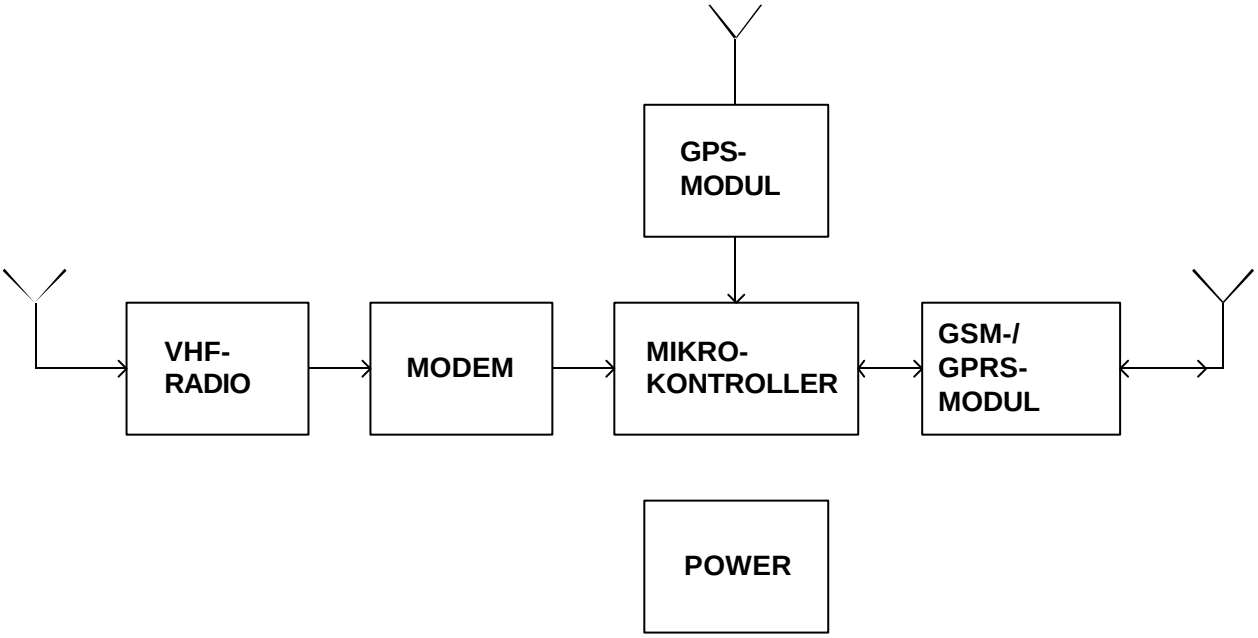


Figure 1: Forenklet blokkskjema for basestasjonen

3 Design

3.1 Generell beskrivelse

Basestasjonen skal kunne ta imot posisjonsmeldinger som kommer inn fra dyreenheter via et VHF-samband og lagre disse meldingene. Basestasjonen skal så kunne ringe opp sentralen via et GSM-modem og sende posisjonsmeldingene til sentralen. Basestasjonen skal videre kunne bli ringt opp og motta meldinger fra sentralen.

3.2 Software Data Flow Diagram

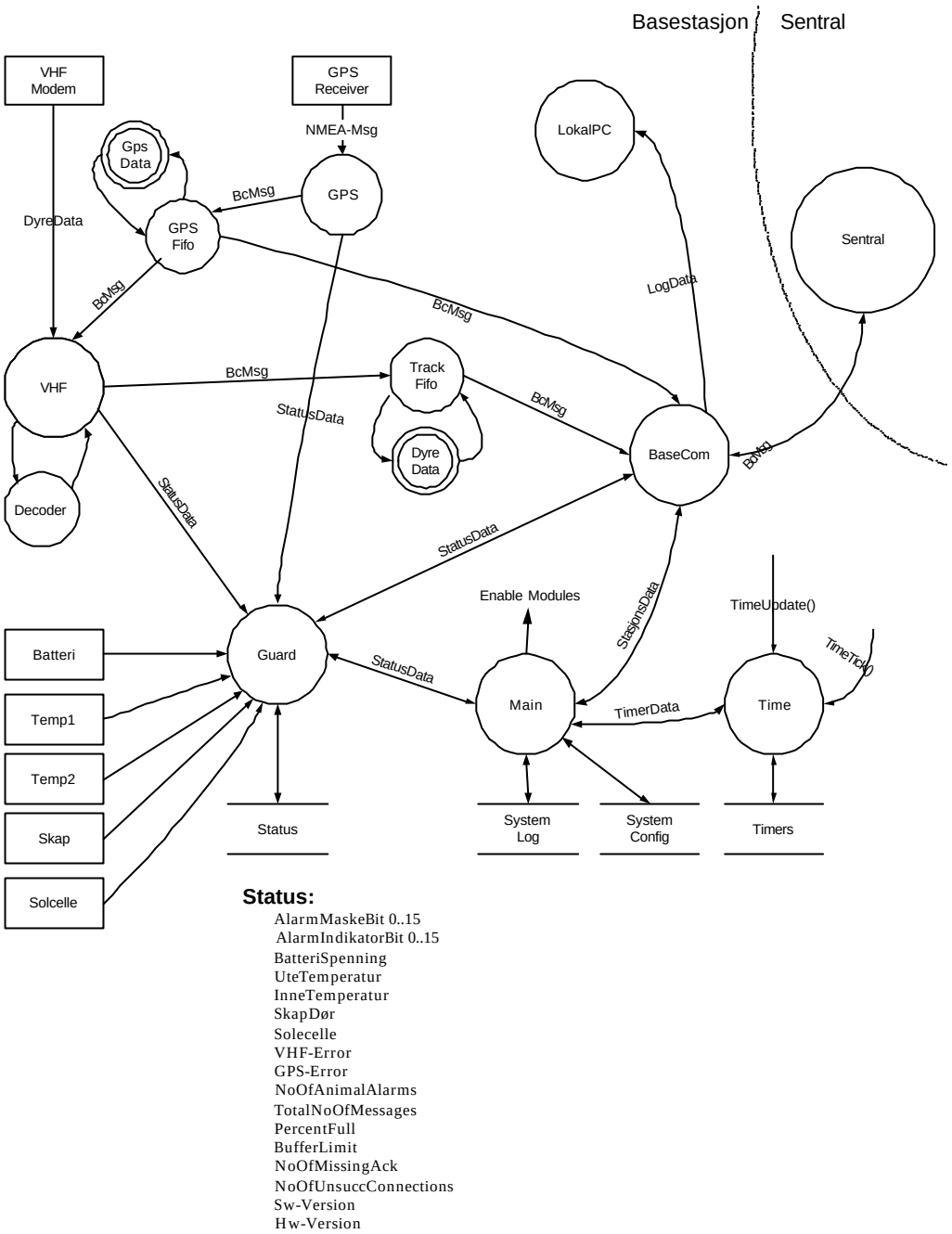


Figure 2:Software Data Flow Diagram

4 Software moduler

4.1 Oversikt over moduler

Programvaren i basestasjonen er delt opp i moduler. Interaksjonene mellom disse modulene er skissert i Software Data Flow Diagram i forrige kapitel. Basestasjonen er delt opp i følgende moduler :

- GPS, tar inn data fra basestasjonens GPS-mottaker.
- GPS Fifo, håndterer buffferet hvor GPS-data fra basestasjonens GPS-mottaker lagres.
- VHF, tar inn data som kommer inn fra dyreenhetene via VHF sambandet.
- Dekoder, dekoder meldinger som kommer inn via VHF-sambandet.
- Decoder, håndterer dekoding og evnt feilkorrigering av data som kommer inn fra dyreenhetene.
- Guard, overvåker sensorer og data som kommer inn og avgjør om det skal sendes alarmer til sentralen.
- Track Fifo, håndterer bufferet hvor meldinger fra dyreenheter mellomlagres før de sendes videre til sentralen.
- Main, hovedprogrammet.
- Time, håndterer timere.
- BaseCom, håndterer interface mot basestasjonens GSM-modem og kommunikasjon med sentralen.
- Lokal PC. Dette er en egen PC-modul som kan kople seg til basestasjonen via GSM-modemet eller via en en serieport (RS-232) på basestasjonen. Dette programmet kan lagre informasjon som kommer fra basestasjonen, presentere posisjonsdata på kart, samt simulere en enkel sentral som kan kommunisere med basestasjonen.

4.2 Funksjonalitet

4.2.1 Generelt om kommunikasjon mellom basestasjon og sentral

Kommunikasjon mellom basestasjon og sentral skal baseres på et oppringt samband vha av et GSM-modem som er plassert i basestasjonen. Det er bestemt at kommunikasjonen mellom basestasjon og sentral skal baseres på at sentralen alltid er master. Sentralen skal kunne kontakte en basestasjon, for så å be om å få informasjon om status, hente ned lagrede meldinger osv. Basestasjonen vil ringe opp senralen i de tilfeller hvor det er detektert en alarmsituasjon på basestasjonen. I dette tilfelle er det altså basestasjonen som ringer opp og etablerer forbindelsen, men så snart forbindelsen er etablert vil sentralen ta over kommandoen. Sentralen vil da starte kommunikasjonen ved å be om basestasjonens status. Statusmeldingen vil inneholde informasjon om evnt. alarmsituasjoner og sentralen vil ut fra denne informasjonen se hvorfor basestasjonen ringer. Det er deretter sentralen som kopler ned forbindelsen igjen.

4.2.2 Alarmer

Som nevnt så vil basestasjonen ringe opp sentralen i de tilfeller hvor en alarmsituasjon er detektert i basestasjonen. Følgende alarmsituasjoner er identifisert:

- Batteri alarm (batterispenning under et angitt nivå).
- Tamper,dvs sol-celle brudd alarm (ikke implementert i HW).
- Dør åpnet alarm.
- GPS feil alarm (ingen kontakt med GPS-mottaker).
- VHF feil alarm.

- Meldingsbuffer grense alarm, dvs meldingsbufferet er mer enn 60% fullt.
- Alarmmelding fra dyreenhet i meldingsbuffer.

Sentralen kan ved å bruke meldingen 'Basestasjon alarm på/av' bestemme hvilke situasjoner som skal føre til en alarm. Man kan f.eks velge at basestasjonen ikke skal ringe opp når det kommer en alarmmelding fra en dyreenhet ved å 'skru' denne alarmen av.

4.2.3 Håndtering av GPS-data

Basestasjons hovedfunksjonalitet er å ta imot posisjonsmeldinger fra dyreenhetene og sende disse videre til sentralen. I den forbindelse skal basestasjonen legge ved posisjonsdata fra basestasjonens egen GPS-mottaker i meldingene som sendes til sentralen. Disse posisjonsdataene skal være hentet inn ved samme tispunkt som posisjonsdataene som kommer i meldingene fra dyreenhetene. I og med at meldingene som kommer inn fra dyrenhetene kan være forsinket, så må basestasjonen lagre posisjonsdata fra egen GPS-mottaker slik at man kan finne frem posisjonsdata for korresponderende tidspunkt når en forsinket melding kommer inn. I den forbindelse er det satt følgende krav til funksjonalitet Ref[2]:

- En korrekt VHF-melding (dvs at dekoding går OK) skal sendes videre til sentralen i sin opprinnelige form. GPS-data med korresponderende tidsangivelse fra basestasjonens egen GPS-mottaker skal legges ved i meldingen.
- Dersom en VHF-melding inneholder feil (dvs at dekoding ikke går OK), så skal den sendes videre i sin opprinnelige form. GPS-data fra basestasjonens egen GPS-mottaker her og nå skal legges ved i meldingen.

4.2.4 Håndtering av meldinger i FIFO-bufferet

Meldinger som kommer inn fra dyreenhetene blir lagret i et buffer og meldingene blir deretter sendt videre til sentralen ved behov. Følgende prinsipper gjelder:

- Hvis bufferet går fullt (f.eks. pga. at basestasjonen ikke får kontakt med sentralen ved behov) så overskrives de eldste meldingene i bufferet.
- Meldingene lagres i RAM og vil bli slettet dersom prosessoren restarter.

5 Kommunikasjon (protokoller)

5.1 Protokoll mot basestasjonens GPS-mottaker

Dataene som kommer fra den interne GPS-mottakeren har følgende format:

Name of output data	GGAsentence.. Global Positioning System Fix Data
Purpose	Time, position, positioning status, DOP value, DGPS info. etc. output
Format	$GPGGA,hhmmss,ddmm.mmmm,N,dddmm.mmmm,E,n,nn,nn.n,xnnnnn.n,M,xnnn.n,M,ss,nnn n* hh[CR][LF]
Description	$GPGGA(Fixed) hhmmss = UTC time(Hour, Minute, Second) ddmm.mmmm = Latitude(degree, minutes,1/ 10000 minutes) N = N : the north latitude, S : the south latitude dddmm.mmmm = Longitude(degree, minutes,1/ 10000 minutes) E = E : the East longitude, W : the West longitude

	n = Positioning status (0 : Not positioned yet / 1 : Independently positioning / 2 : DGPS positioning) nn = Number of satellites used nn.n = HDOP value nnnn.n = Altitude above sea level(-999.9~ 9999.9) M = Unit of altitude above sea level(m) nnnn.n = Geoid height(-999.9~ 9999.9) M = Unit of geoid height(m) ss = Elapsed time after DGPS data acquisition(Unit : Sec.) nnnn = Identification number of DGPS * hh Astarisk (*) followed by two digits of checksum

De data som skal leses ut av GPS-mottakeren, lagres og sendes videre til sentral er:

- UTC time
- Latitude (inkl. Nord/Sør indikering)
- Longitude (inkl Øst/Vest indikereing)
- Positioning status
- Number of satellites used
- Altitude above sea level (skal være med i 'Basestajon GPS-data' melding)

Se også meldingsfomat for meldingene 'basestasjon GPS-data' og 'posisjonsmelding fra dyreenhet' for å se hvordan disse data 'pakkes' før de lagres og videresendes.

5.2 Protokoll mot dyreenhet

Meldingene fra dyreenhet til basestasjon skal ha følgende format beskrevet nedenfor. Meldingen har en fast lengde på 36 bytes. De to neste tabellene beskriver meldingsformatet og disse tabellene er hentet fra Ref[1].

Bit nr			Bit lengde	MeldingsByte	Databyte	Beskrivelse	Verdi
0	-	*7*	*8*	*1*		*Preamble byte 0*	*0xFF*
8	-	*15*	*8*	*2*		*Preamble byte 1*	*0xFF*
16	-	*23*	*8*	*3*		*Preamble byte 2*	*0xFF*
24	-	*31*	*8*	*4*		*Preamble byte 3*	*0xFF*
32	-	*39*	*8*	*5*		*Sync byte 0*	*0xC4*
40	-	*47*	*8*	*6*		*Sync byte 1*	*0xD7*
48	-	55	8	7	1	Meldingstype/Versjon	0x01
56	-	63	8	8	2	Reservert	0x00-0xFF
64	-	75	12	9 4/8	3 4/8	Adresse/ID. Hex kodet decimal	0 - 4095
76	-	80	5	10 1/8	4 1/8	Klokke: Time	0-23
81	-	86	6	10 7/8	4 7/8	Klokke: Minutt	0-59
87	-	92	6	11 5/8	5 5/8	Klokke: Sekund	0-59
93	-	93	1	11 6/8	5 6/8	Gyldige data (1). Ikke gyldige (0)	0 | 1
94	-	100	7	12 5/8	6 5/8	Breddegrad: Grader	0-89
101	-	106	6	13 3/8	7 3/8	Breddegrad: Minutter	0-59
107	-	120	14	15 1/8	9 1/8	Breddegrad: Minutter desimal	0-9999
121	-	121	1	15 2/8	9 2/8	Nord (1) / Sør (0) indikering	0 | 1
122	-	130	9	16 3/8	10 3/8	Lengdegrad: Grader	0-360
131	-	136	6	17 1/8	11 1/8	Lengdegrad: Minutter	0-59
137	-	150	14	18 7/8	12 7/8	Lengdegrad: Minutter desimal	0-9999
151	-	151	1	19	13	Øst (1) / Vest (0) indikering	0 | 1
152	-	155	4	19 4/8	13 4/8	Antall satellitter mottatt	0-15
156	-	162	7	20 3/8	14 3/8	Temperatur sensor (Grader celsius)	-63 ? +63
163	-	168	6	21 1/8	15 1/8	Batterispenning (Volt)	3.0 - 6.5
169	-	169	1	21 2/8	15 2/8	Alarm Status : Høy bevegelse	1 | 0
170	-	170	1	21 3/8	15 3/8	Alarm Status : Liten/ingen bevegelse	1 | 0
171	-	175	5	22	16	Reservert	0b00000-0b11111
Foregående posisjon (aktuell posisjon - 1)							
176		180	5	22 5/8	16 5/8	Klokke: Time	0-23
181		186	6	23 3/8	17 3/8	Klokke: Minutt	0-59
187		192	6	24 1/8	18 1/8	Klokke: Sekund	0-59
193		193	1	24 2/8	18 2/8	Gyldige data (1). Ikke gyldige (0)	0 | 1
194		200	7	25 1/8	19 1/8	Breddegrad: Grader	0-89
201		206	6	25 7/8	19 7/8	Breddegrad: Minutter	0-59
207		220	14	27 5/8	21 5/8	Breddegrad: Minutter desimal	0-9999
221		221	1	27 6/8	21 6/8	Nord (1) / Sør (0) indikering	0 | 1
222		230	9	28 7/8	22 7/8	Lengdegrad: Grader	0-360
231		236	6	29 5/8	23 5/8	Lengdegrad: Minutter	0-59
237		250	14	31 3/8	25 3/8	Lengdegrad: Minutter desimal	0-9999
251		251	1	31 4/8	25 4/8	Øst (1) / Vest (0) indikering	0 | 1
252		255	4	32	26	Antall satellitter mottatt	0-15
Foregående posisjon (aktuell posisjon - 2)							
256		260	5	32 5/8	26 5/8	Klokke: Time	0-23
261		266	6	33 3/8	27 3/8	Klokke: Minutt	0-59
267		272	6	34 1/8	28 1/8	Klokke: Sekund	0-59
273		273	1	34 2/8	28 2/8	Gyldige data (1). Ikke gyldige (0)	0 | 1
274		280	7	35 1/8	29 1/8	Breddegrad: Grader	0-89
281		286	6	35 7/8	29 7/8	Breddegrad: Minutter	0-59
287		300	14	37 5/8	31 5/8	Breddegrad: Minutter desimal	0-9999
301		301	1	37 6/8	31 6/8	Nord (1) / Sør (0) indikering	0 | 1
302		310	9	38 7/8	32 7/8	Lengdegrad: Grader	0-360
311		316	6	39 5/8	33 5/8	Lengdegrad: Minutter	0-59
317		330	14	41 3/8	35 3/8	Lengdegrad: Minutter desimal	0-9999
331		331	1	41 4/8	35 4/8	Øst (1) / Vest (0) indikering	0 | 1
332		335	4	42	36	Antall satellitter mottatt	0-15

Tabell 1 Koding av data i meldingen

I de tilfeller hvor et datafelt ligger fordelt over flere byte (f.eks Breddegrad: Minutter desimal) så vil de mest signifikante bitene ligge i den første byten (Big Endian).

I tabellene under er det skissert hvordan feilkorrigerings data blir flettet inn i de opprinnelige dataene. Merk at Databyte 1 i tabellen under er Databyte 1 (eller Meldingsbyte 7) i tabellen over. Preamble og sync er følgelig ikke en del av dataene som behandles med feilkorrigerende data. Det er disse 64 byte som sendes videre fra basestasjon til sentral.

C1ECC-R1A	C1ECC-R1B	**Databyte 1**	**Databyte 2**	**Databyte 3**	**Databyte 4**	**Databyte 5**	**Databyte 6**
C1ECC-R2A	C1ECC-R2B	**Databyte 7**	**Databyte 8**	**Databyte 9**	**Databyte 10**	**Databyte 11**	**Databyte 12**
C1ECC-R3A	C1ECC-R3B	**Databyte 13**	**Databyte 14**	**Databyte 15**	**Databyte 16**	**Databyte 17**	**Databyte 18**
C1ECC-R4A	C1ECC-R4B	**Databyte 19**	**Databyte 20**	**Databyte 21**	**Databyte 22**	**Databyte 23**	**Databyte 24**
C1ECC-R5A	C1ECC-R5B	**Databyte 25**	**Databyte 26**	**Databyte 27**	**Databyte 28**	**Databyte 29**	**Databyte 30**
C1ECC-R6A	C1ECC-R6B	**Databyte 31**	**Databyte 32**	**Databyte 33**	**Databyte 34**	**Databyte 35**	**Databyte 36**
C2ECC-C1A	C2ECC-C2A	C2ECC-C3A	C2ECC-C4A	C2ECC-C5A	C2ECC-C6A	C2ECC-C7A	C2ECC-C8A
C2ECC-C1B	C2ECC-C2B	C2ECC-C3B	C2ECC-C4B	C2ECC-C5B	C2ECC-C6B	C2ECC-C7B	C2ECC-C8B

Tabell 2 Fordeling av data og ECC bytes. C1ECC-Rxy er for C1 ECC (rekker). C2ECC-Cxy er C2 ECC for kolonner.

1	2	**3**	**4**	**5**	**6**	**7**	**8**
9	10	**11**	**12**	**13**	**14**	**15**	**16**
17	18	**19**	**20**	**21**	**22**	**23**	**24**
25	26	**27**	**28**	**29**	**30**	**31**	**32**
33	34	**35**	**36**	**37**	**38**	**39**	**40**
41	42	**43**	**44**	**45**	**46**	**47**	**48**
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Tabell 3 Rekkefølge for sending av de forskjellige bytes. 1 sendes først, deretter 2 osv.. til og med 64.

5.3 Protokoll mot modem

Basestasjonen skal kunne etablere forbindelse med sentralen ved å ringe den opp og basestasjonen skal kunne kople ned forbindelsen etter at data er overført. Videre skal basestasjonen kunne bli oppringt av sentralen og denne forbindelsen skal kunne koples ned etter at data er overført.. Det implementere en enkel protokoll som håndterer oppkopling og nedkopling av dataforbindelsen via GSM-modemet. En enkel feilhåndtering implementeres for å gjøre protokollen robust. Protokollen mot modemet er basert på AT-kommandoer.

Modemene som benyttes er av type Wavecom Fasttrack M1203A og Wavecom Integra. I modemene må det sitte et SIM-kort med data-abbonement samt at PIN-koden på SIM-kortet er deaktivert.

Kommandoene som benyttes i implementasjonen er:

Kommando	Funksjon
ATD<telefon-nummer>	Modemet ringer opp <telefon-nummer>
ATA	Modemet svarer på anrop
ATH	Modemet legger på
+++	Modemet skifter fra data-modus til kommando-modus. Brukes for å avslutte en dataforbindelse
RING	Modemet blir ringt opp og bruker kommandoen RING for å informere om dette.
BUSY	Modemet ringer opp et modem som er opptatt (linjen er opptatt)
NO ANSWER	Modemet ringer opp et modem men brukeren av dette modemet svarer ikke.

NO CARRIER	Modemet forsøker å ringe opp et modem som det ikke oppnås kontakt med (modemet er slått av el.)

I de følgende figurene er det skissert opp noen scenarioer som BaseCom modulen må kunne håndtere.

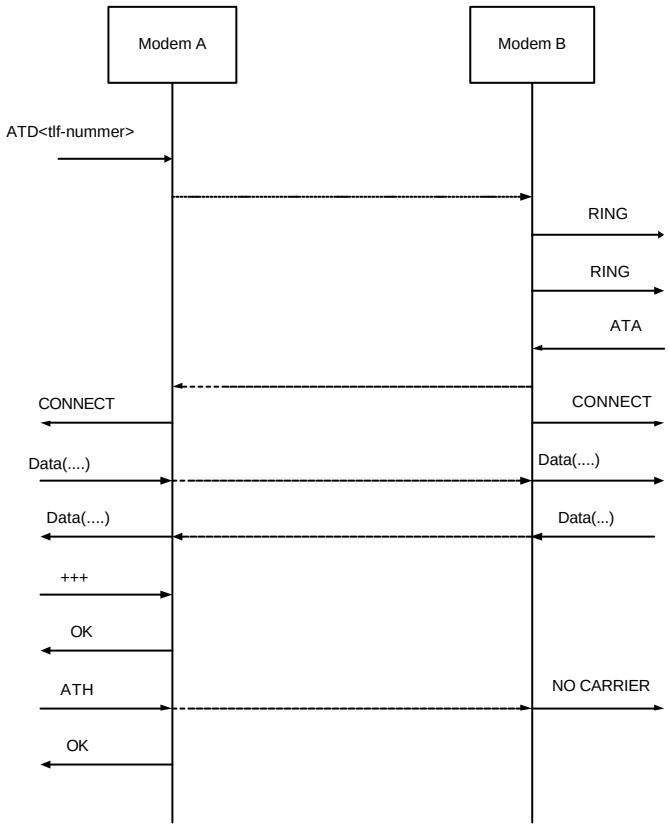


Figure 3: Modem A ringer modem B. Modem B svarer. Modem A kopler ned

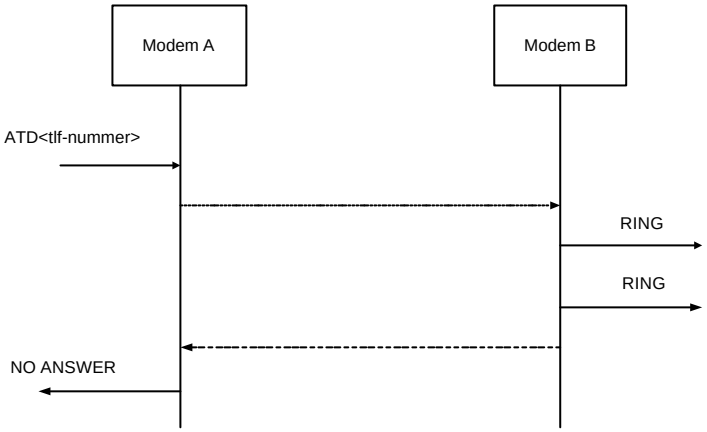


Figure 4: Modem A ringer modem B. Brukeren av modem B svarer ikke

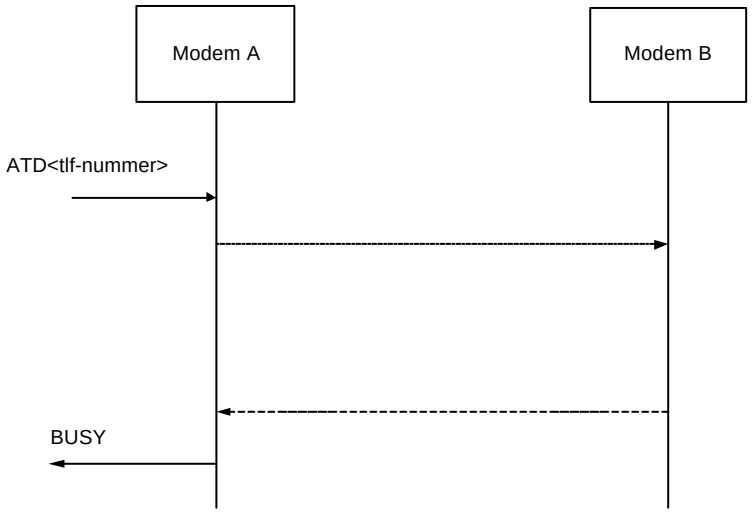


Figure 5: Modem A ringer modem B. Modem B er opptatt

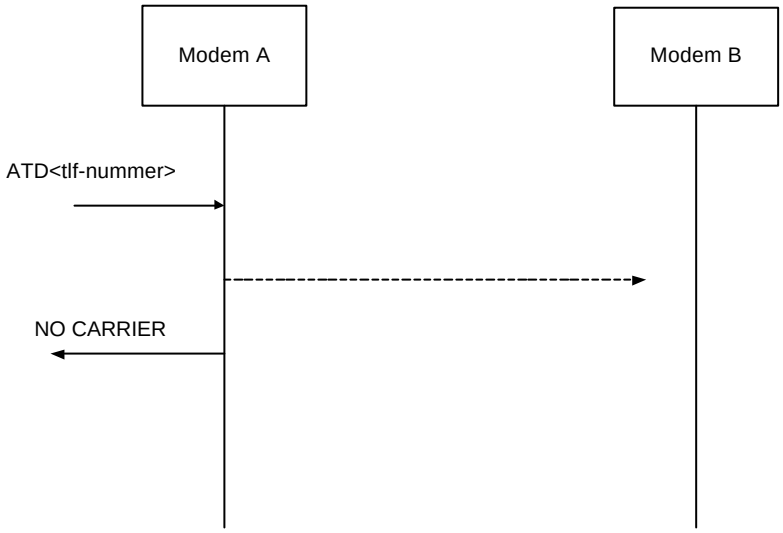


Figure 6: Modem A ringer modem B. Det oppnår ikke kontakt

5.4 Protokoll mellom basestasjon og sentral

Kommunikasjon mellom basestasjon og sentral skal baseres på et oppringt samband vha av et GSM-modem som er plassert i basestasjonen. Dette vil gi en direkte kommunikasjonskanal på 9600 bit/s. I de følgende avsnitt beskrives en protokoll for kommunikasjonen mellom basestasjon og sentral.

5.4.1 Prinsipp for protokoll

Hovedprinsippet for kommunikasjon mellom sentral og basestasjon er at sentralen alltid skal være master, dvs at sentralen alltid styrer kommunikasjonen (sentralen spør og basestasjonen svarer). Både basestasjon og sentral kan etablere en oppringt forbindelse men det er sentralen som avgjør når den skal koples ned.
Protokollen mellom basestasjon og sentral implementeres ved at det defineres en meldingsprotokoll på applikasjonsnivå. Dette øverste laget i protokollen kalles BaseCom-laget (basestation protokoll) og vil ha følgende generelle meldingsformat:

BaseCom-Meldings Format:

MeldingsType/ID (1 byte)	'spare' byte = 0 (1 byte)	Meldings Lengde (1 byte)	Data (x antall bytes)

Den første byte inneholder informasjon til mottaker om hvilke type melding som kommer. Den andre byte er en 'spare'-byte som foreløpig ikke benyttes. Den 3. byte er lengden på meldingen (lengden på meldingen er fra og med byte0 som er Meldings Type/ID og til og med siste byte i Data-feltet). Data-feltet vil variere i lengde avhengig av hvilken melding som sendes. I de tilfeller hvor et tall må fordeles over flere byte så kommer den mest signifikante byte først (Big Endian)

Meldingene på applikasjonsnivå, dvs på BaseCom-laget vil bli pakket inn i en 'container' før de sendes. Denne 'containeren' blir definert som TransCom-laget (Transfer protokoll) og vil ha følgende generelle meldingsformat:

TransCom-Meldings Format:

START_OF_PACKET = STX = 2 1(byte)	Pakke-lengde (1 byte)	BaseCom-Melding (y antall bytes)	CRC-sjekksum (2 bytes)

Den første byte inneholder et fast tegn som indikerer begynnelsen på en pakke, den andre byte inneholder lengden på pakken (lengde på pakken fra og med START_OF_PACKET og til og med siste byte i CRC-sjekksum). BaseCom-meldingen vi variere i lengde avhengig av hvilke melding som blir sendt. CRC-sjekksummen er på 2 byte. CRC-sjekksummen genereres ved å benytte CCITT 16bit algorithm (X^16 + X^12 + X^5 + 1). CRC-sjekksumen utgjør 2 bytes, og det er mest signifikante byte som kommer først (Big Endian)

For å kunne sende meldinger over en serieport og videre via et GSM-modem bruker vi et ATCom-lag (AT-kommando protokoll) og et SerialCom-lag (Serieport protokoll) til å etablere kommunikasjonkanalen mellom basestasjon og sentral. SerialCom laget inneholder funksjoner for å sende og motta data over en serieport. ATCom-laget inneholder funksjoner for å kommunisere med modemet ved hjelp av AT-kommandoer.

Den følgende tabell gir en oversikt over de forskjellige lagene.

MODUL	G R E N S E S N I T T :	F U N K S J O N :
BaseCom (BC)	BOOL BCReceiveMsg() void BCExtractMsgInfo() WORD BCExtractStationID() void BCExtractStatusInfo(); void BCBuildGetStatus() void BCBuildStatus() void BCBuildNoPosData() void BCBuildAckMsg()	Meldingsprotokoll mellom basestasjon og senter. Det er dette laget som bygger opp meldingene, håndterer sekvenskontroll med retransmisjoner osv.
TransCom(TC)	void TCCrcEncode() WORD TCCrcDecode() void TCBuildMsg() WORD TCFindMsg()	Inn- ut-pakking av meldinger med CRC. CRC-sjekksummen benyttes til å identifisere en mottatt pakke på mottaker-siden. Når man har identifisert det som kan være en pakke sjekker man

	WORD TCReceiveMsg();	CRC-sjekksummen, hvis denne er OK er pakken mottatt.

ATCom (AC)	void ACBuildDialUp();
	void ACBuildATA();
	void ACBuildDisConn()
	void ACBuildATH()
	bool ACSearchForRING();
	bool ACSearchForOK()
	bool ACSearchForCONNECT()
	bool ACSearchForNOCARR();
	bool ACSearchForNOANSW()
	bool ACSearchForBUSY();
	int GetATComRxBuffer();
	void FlushATCom();
	BOOL ACScann();
	Int ACReceive ()
		Modem AT-kommandoer
		Opp- og ned-kobling av modem, etablering av forbindelse.

SerialCom (SC)	BOOL SerialComRead() Int SerialCom1ScannBuffer() BOOL SerialComWrite() int SerialComRxBytes() int SerialComTxSpace(); void SerialComDBugMsg() void SerialFlushRx() void SerialFlushTx()	Platformavhengig på laveste nivå. Håndterer åpning og lukking av serie-port, samt det å sende/motta data på serie-porten.

Implementasjonen av BaseCom,TransCom og ATCom skal være mest mulig generell slik at koden kan benyttes uavhengig av operativsystem og programvareplattform (mest mulig av koden skrives i ANCI C)

Som tidligere nevnt skal basestasjonen kunne etablere forbindelse med sentralen, samt at sentralen skal kunne etablere forbindelse med basestasjonen. I begge tilfellene er det sentralen som skal være master, dvs den son styrer sekvensen og kopler ned forbindelsen når den har fått all informasjon. Basestasjonen bare svarer på requester fra sentralen. I de tilfeller hvor det er basestasjonen som ringer opp og etablerer forbindelsen, så vil sentralen ta over kontrollen når forbindelsen er etablert og det er sentralen som kopler ned.

Basestasjonen vil kople opp forbindelse med sentralen når:

- Det har oppstått en alarmsituasjon på basestasjonen

Sentralen vil kople opp forbindelse med sentralen når:

- Sentralen ønsker å hente status fra basestasjonen.
- Sentralen ønsker å hente inn posisjonsmeldinger fra basestasjonen.

5.4.2 Meldinger fra sentral til basestasjon

5.4.2.1 Kvittering for mottatt data (med feilmelding)

Melding som sendes når sentralen har mottatt melding fra basestasjon og denne meldingen krever kvittering. Meldingen brukes også når melding er mottatt fra basestasjon og denne meldingen ikke er godkjent . I dette

tilfelle benyttes <feilkode>. Denne meldingen er definert som melding 1. Denne meldingen benyttes også fra basestasjon til sentral.

Byte0 = 0x01 // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = 0x07 //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikante byte
Byte4 = <basestasjon ID/nr> //minst signifikante byte
Byte5 = <mottatt meldings type> // meldingstype det sendes kvittering for
Byte6 = <feilkode>

Feilkode:
OK = 0
Feil basestasjon ID = 1
Mottatt ugyldig meldings type = 2

5.4.2.2 Request basestasjon status

Melding som sendes for å innhente informasjon om status på basestasjon. Denne meldingen er definert som melding 2. Denne meldingen benytter ikke basestasjon ID/nr som en parameter fordi i de tilfeller hvor basestasjonen ringer opp sentralen så vet ikke sentralen hvilke basestasjon som har ringt opp. Basestasjon ID/nr vil bli returnert i Status meldingen.

Byte0 = 0x02 // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = 0x03 //pakke lengde

Denne melding forventer melding 6 'Basestasjon status' som svar hvis alt er OK, evnt. melding 1 'Kvittering' (med feilmelding) hvis noe går galt.

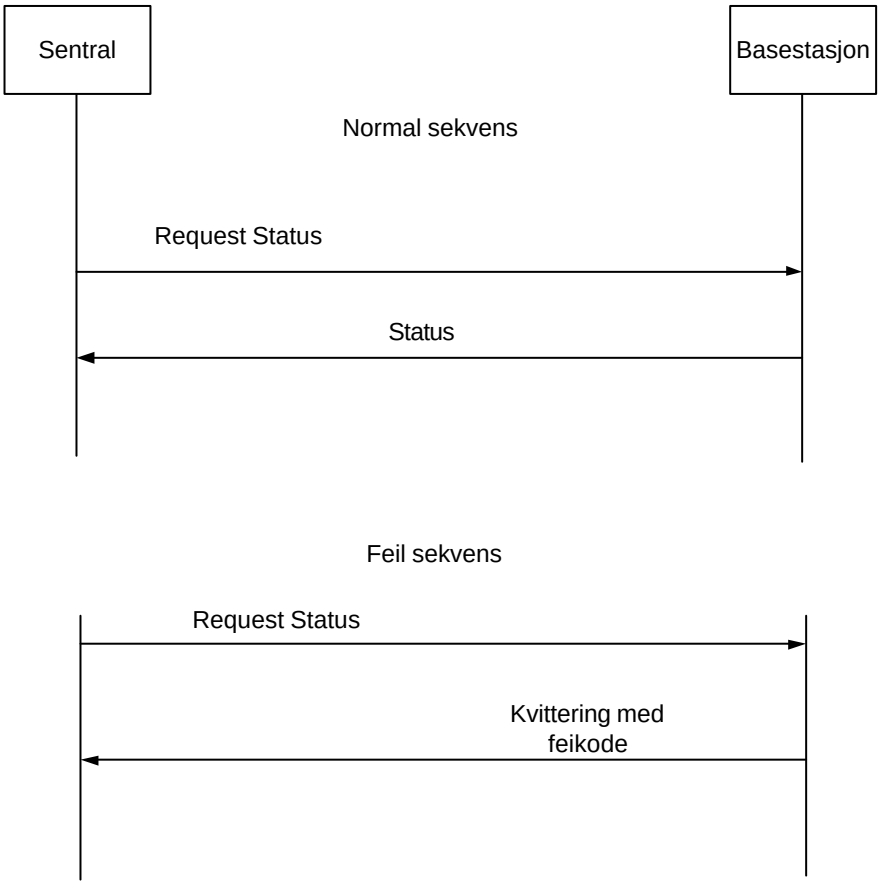


Figure 7: Request status fra basestasjon

5.4.2.3 Request basestasjon konfigurasjon

Melding som sendes for å innhente informasjon om basestasjonens konfigurasjon (f.eks. telefonnr til sentralen). Meldingen ber om å få en parameter. Denne meldingen er definert som melding 3.

Byte0 = 0x03 // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = 0x06 //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikante byte
Byte4 = <basestasjon ID/nr> // minst signifikante byte
Byte5 = <parameter nr> //identitet til parameter, feks at parameter 1 er telefonnr til sentralen

Denne melding forventer melding 7 'Basestasjon konfigurasjon' som svar hvis alt er OK, evnt. melding 1 'Kvittering'(med feilmelding) hvis noe går galt.

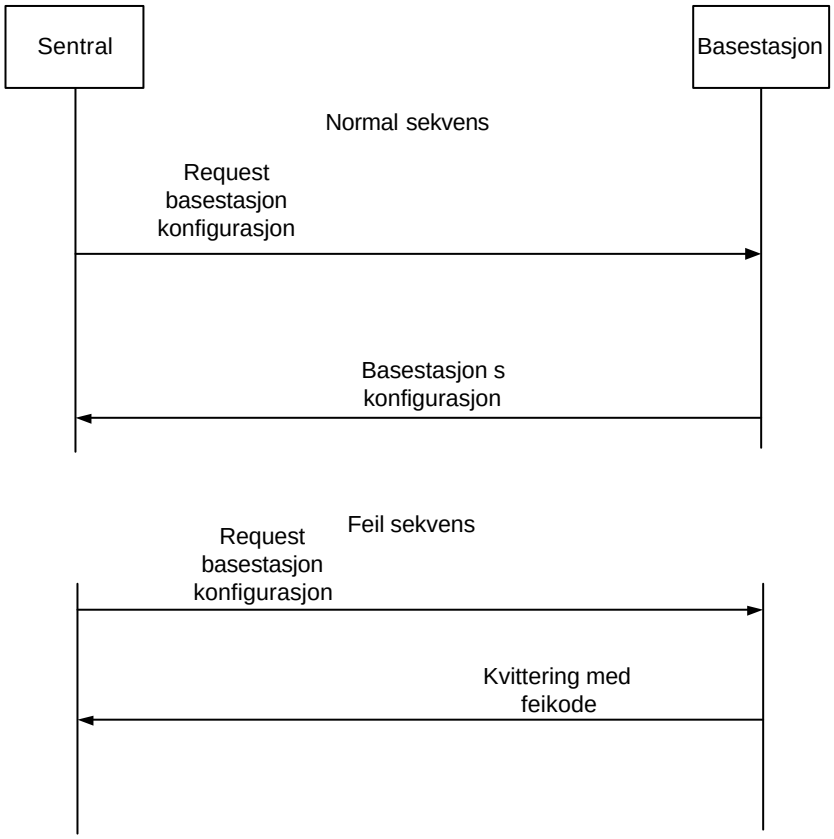


Figure 8: Request konfigurasjon fra basestasjon

5.4.2.4 Forandre basestasjon konfigurasjon.

Melding som sendes for å forandre en konfigurasjonsparameter i basestasjonen. Parameteren sendes som en tekst-string og denne pakken kan ha variabel lengde ettersom parameteren kan bestå av et ulikt antall tegn. Stringen er Null-terminert. Denne melding er definer som melding 4

Byte0 = 0x04 // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = <lengde> //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikante byte
Byte4 = <basestasjon ID/nr> // minst signifikante byte
Byte5 = <parameter nr> //identitet til parameter, feks at parameter 1 er telefonnr til sentralen
Byte6 = parameter byte0
Byte7 = parameter byte1
..
ByteK+6 = parameter byteK

Denne melding forventer melding 7 'Basestasjon konfigurasjon' som svar hvis alt er OK, evnt. melding 1 'Kvittering'(med feilmelding) hvis noe går galt.

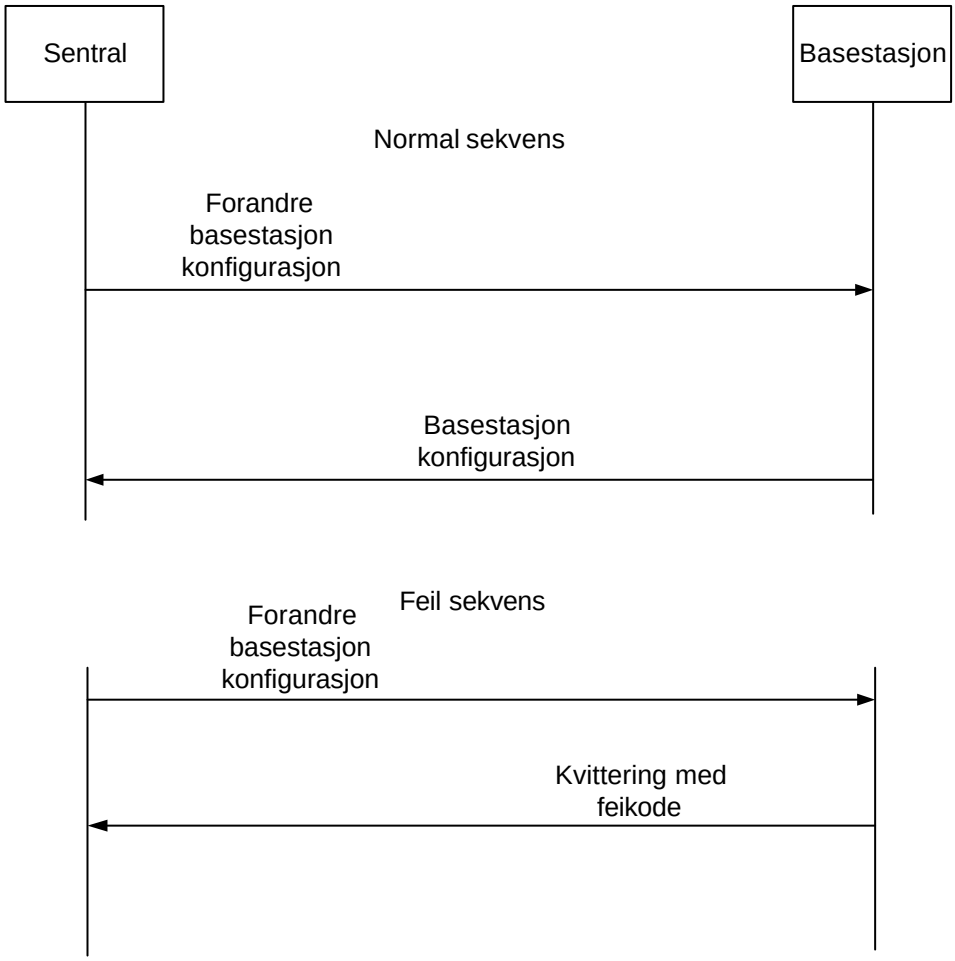


Figure 9: Forandre basestasjonen konfigurasjon

5.4.2.5 Basestasjon alarm på/av.

Melding som sendes for å slå alarmer på/av i basestasjonen. Denne melding er definert som melding 5.
Byte5 og Byte6 utgjør en 16 bit maske med informasjon om hvilke alarmer som skal slås på og slås av (1 = på, 0 = av). Det at en alarm slås på betyr at basestasjonen ringer opp sentralen hvis den aktuelle alarmindikatoren blir aktivert (f.eks dør åpnes i basestasjonen). Det at en alarm slås av betyr at basestasjonen ikke varsler hvis den aktuelle alarmindikatoren blir aktivert.

Byte0 = 0x05 // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = 0x07 //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikante byte
Byte4 = <basestasjon ID/nr> //minst signifikante byte
Byte5 = <alarm bit-maske> //mest signifikante byte
Byte6 = <alarm bit-maske> //minst signifikante byte

Bit0 i Byte6 og Byte8 = Meldingsbuffer grense alarm
Bit1 i Byte6 og Byte8 = Alarmmelding fra dyreenhet i meldingsbuffer
Bit2 i Byte6 og Byte8 = dør åpnet alarm
Bit3 i Byte6 og Byte8 = batteri alarm
Bit4 i Byte6 og Byte8 = tamper,dvs sol-celle brudd alarm

Bit5 i Byte6 og Byte8 = GPS feil alarm
Bit6 i Byte6 og Byte8 = VHF feil alarm

Hvis f.eks Bit2 = 1 så betyr det at dør åpnet alarm skal skrus PÅ.
Hvis f.eks Bit2 = 0 så betyr det at dør åpnet alarm skal skrus AV.

Denne melding forventer melding 'Basestasjon status' som svar hvis alt er OK, evnt. melding 1 'Kvittering'(med feilmelding) hvis noe går galt.

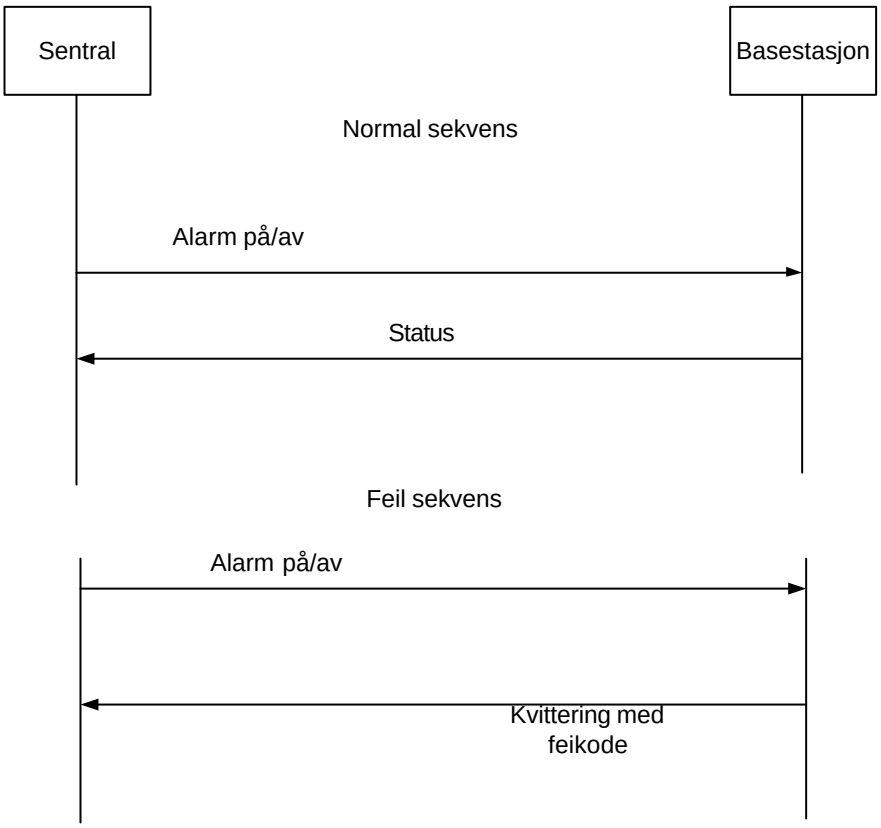


Figure 10: Basestasjon alarm på/av

Figure 11: Nedlasting av fil til basestasjon

5.4.2.6 Restart av basestasjon

Melding som sendes når basestasjonen skal restartes, feks etter at ny SW er overført til basestasjonen. Man kan restarte kjørende program, eller restarte ved å switche til ny SW. Denne melding er definert som melding 6.

Byte0 = 0x06 // meldings type

Byte1 = 0x00 // 'spare' byte
Byte2 = 0x06 //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikante byte
Byte4 = <basestasjon ID/nr> //minst signifikante byte
Byte5 = <swich SW> //

Forklaring Byte5
bit0 =1 restart med switch av SW, bit0 = 0 restart kjørende versjon
bit1 =1 restart HW

Denne melding forventer melding 1 'Kvittering' (uten feilmelding) som svar hvis alt er OK, evnt melding 1 'Kvittering'(med feilmelding) hvis noe går galt.

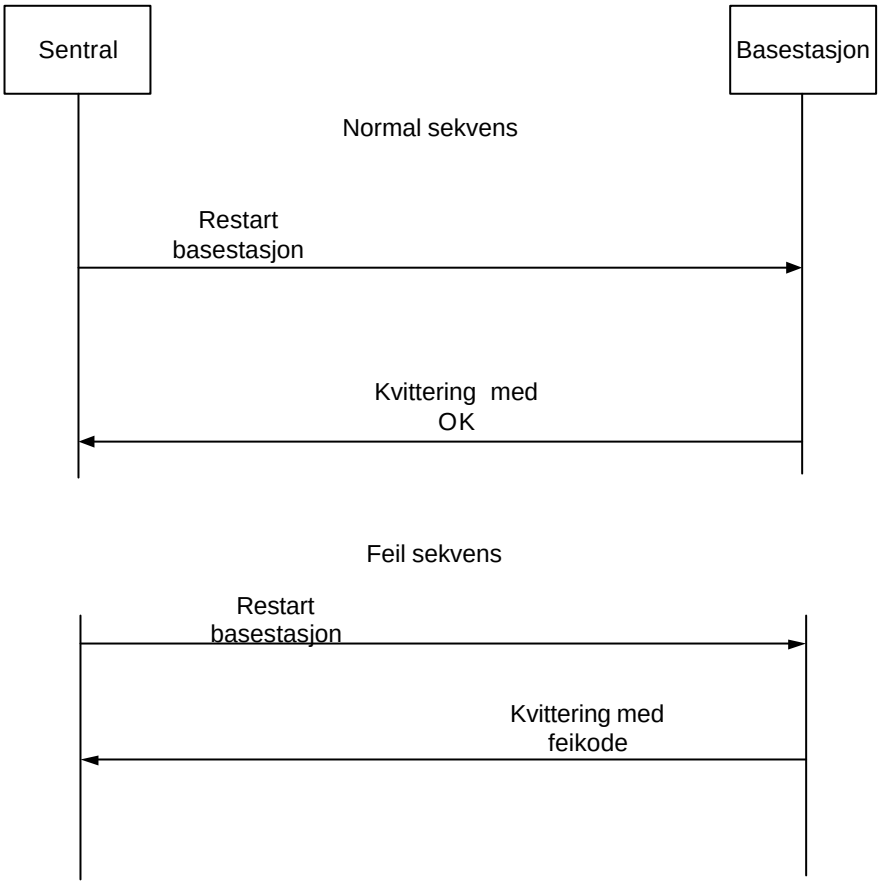


Figure 12: Restart basestasjon

5.4.2.7 Request posisjonsmeldinger fra basestasjon

Denne meldingen sendes når sentralen ønsker å få tilsendt posisjonsmeldingene som ligger lagret i basestasjonen. Denne meldingen er definert som melding 7.

Byte0 = 0x07 // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = 0x05 //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikante byte

Byte4 = <basestasjon ID/nr> //minst signifikante byte

Denne meldingen forventer enten ’Posisjonsmelding fra dyrenehet’ eller ’Ingen Posisjonsmeldinger lageret’ eller ’Kvittering’ (med feilmelding) som svar.

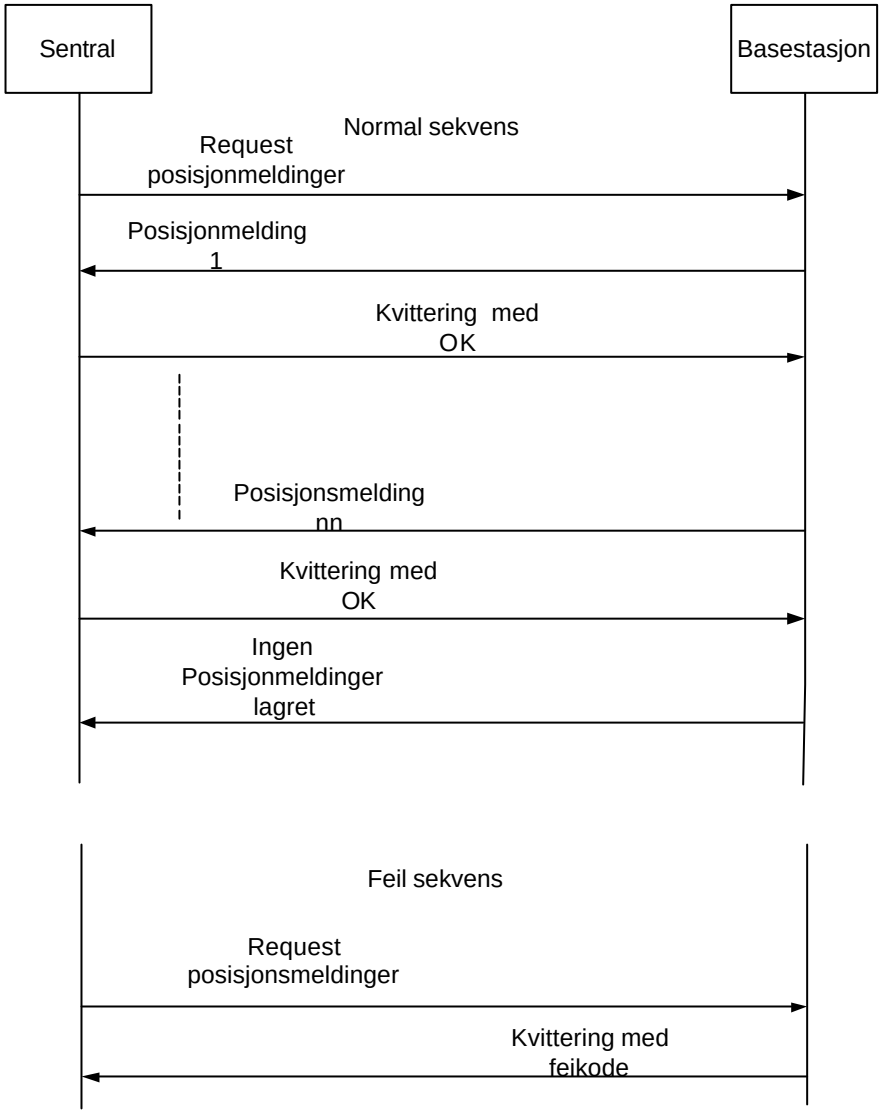


Figure 13: Request posisjonsmeldinger

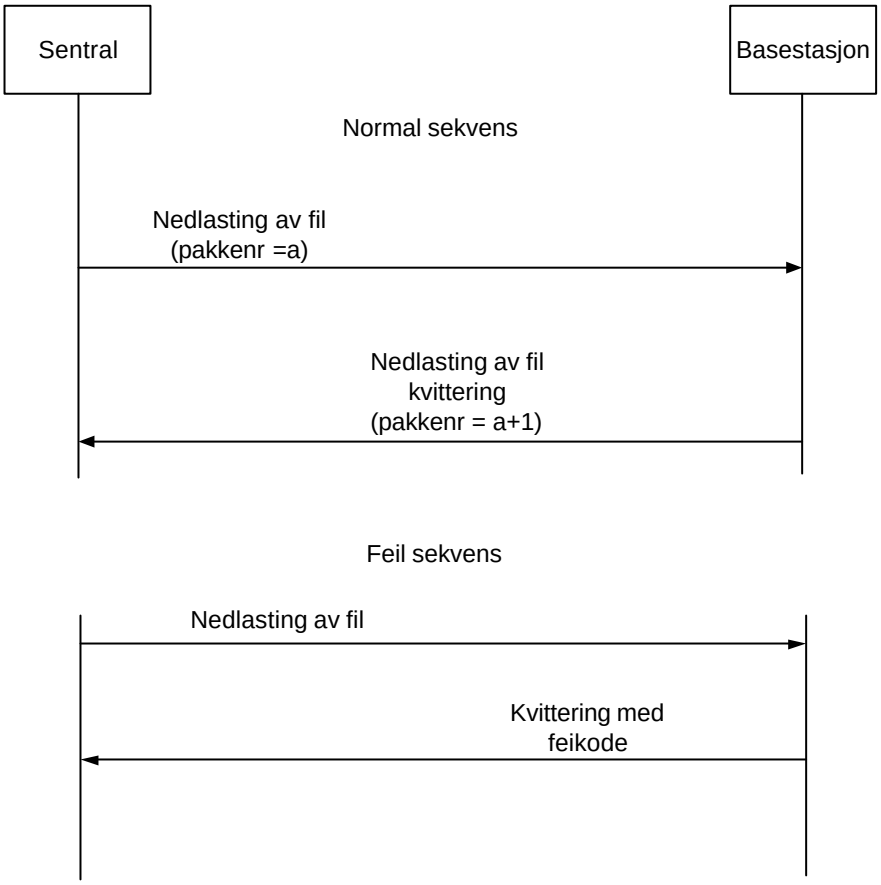
5.4.2.8 Nedlasting av fil til basestasjon

Melding som sendes når sentralen skal laste ned en fil, f.eks ny versjon av SW, til basestasjonen. Denne melding er definert som melding 8.

Byte0 = 0x08 // meldings type
Byte1 = 0x00 // ‘spare’ byte
Byte2 = <lengde> //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikane byte

Byte4 = <basestasjon ID/nr> //minst signifikante byte
Byte5 = <fil type> //f.eks nytt program = 1
Byte6 = <totalt antall meldinger> //antall meldinger for å overføre filen. (max 65535),mest signifikante byte
Byte7 = <totalt antall meldinger> //antall meldinger for å overføre filen. (max 65535),minst signifikante byte
Byte8 = <pakke nr> //denne pakken nr i sekvensen (1-65535), mest signifikante byte
Byte9 = <pakke nr> //denne pakkens nr i sekvensen (1-65535), minst signifikante byte
Byte10 = <fildata> //neste 128 byte er fildata
..
..
Byte138 = <fildata>

Denne meldingen forventer melding 11 'Nedlasting av fil til basestasjon kvittering' som svar hvis alt er OK, evnt melding 1 'Kvittering'(med feilmelding) hvis noe går galt.

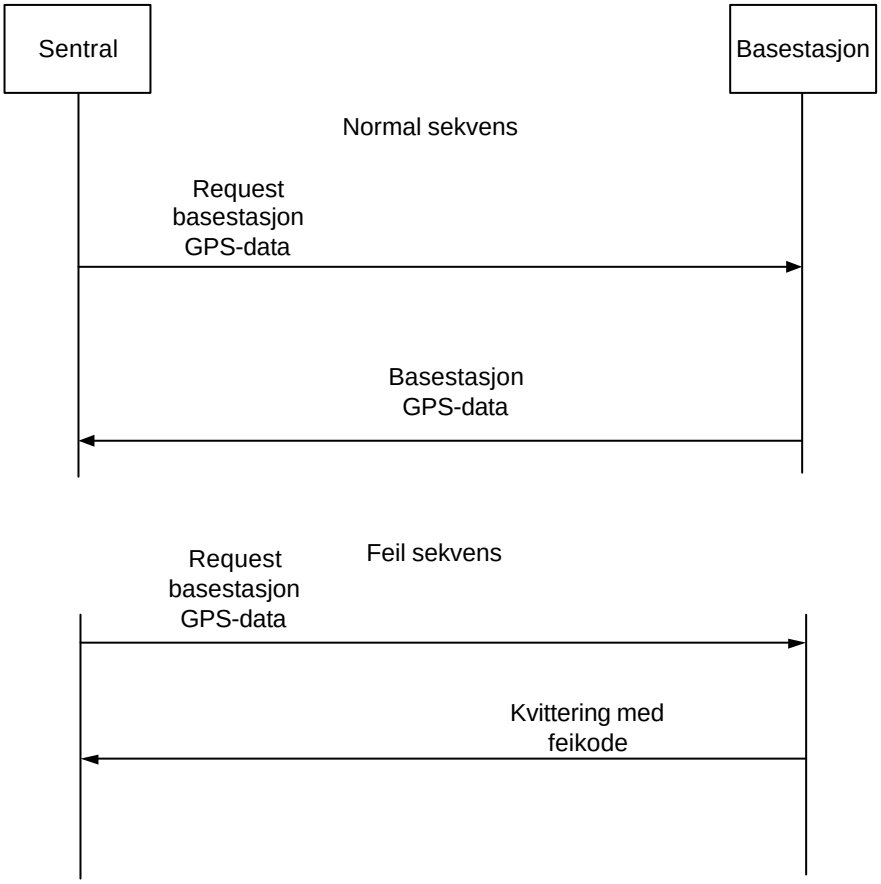


5.4.2.9 Request basestasjon GPS-data

Denne meldingen sendes når sentralen ønsker å få tilsendt basestasjonens GPS-data, dvs data fra basestasjonens egen GPS-mottaker. Denne meldingen er definert som melding 17.

Byte0 = 0x11 // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = 0x05 //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikante byte
Byte4 = <basestasjon ID/nr> //minst signifikante byte

Denne meldingen forventer melding 16 'Basestasjon GPS-data' som svar hvis alt er OK, evnt melding 1 'Kvittering'(med feilmelding) hvis noe går galt.



5.4.3 Meldinger fra basestasjon til sentral

5.4.3.1 Basestasjon status

Melding som sendes når sentralen har bedt om å få informasjon om status på basestasjon. Status som returneres er utetemperatur, innetemperatur, batterispenning, hvilke alarmer som er på/av, hvilke sensorer som er aktivert/passiv, basestasjones GPS-data (posisjon, høyde over havet, antall satelitter) etc. Denne meldingen er definert som melding 9.

Byte0 = 0x09 // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = 0x28 //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikante byte
Byte4 = <basestasjon ID/nr> //minst signifikante byte
Byte5 = <alarm på/av bit-maske> //mest signifikante byte
Byte6 = <alarm på/av bit-maske>> //minst signifikante byte
Byte7 = <alarm indikator>//mest signifikante byte
Byte8 = <alarm indikator>//minst signifikante byte
Byte9 = <batteri spenning > // batterispenning multiplisert med 10
Byte10 = <utetemperatur> // utetemperatur (hele grader), signed char, bit7 = 1/0 = +/-

Byte11 = <innetemperatur> //innetemperatur(hele grader), signed char, bit7 = 1/0 = +/-
Byte12 = <antall ikke kvittert/kvittering med feilkode fra sentral> // mest signifikante byte
Byte13 = <antall ikke kvittert/kvittering med feilkode fra sentral> //minst signifikante byte
Byte14 = <antall mislykkede oppkoplinger> // mest signifikante byte
Byte15 = <antall mislykkede oppkoplinger> //minst signifikante byte
Byte16 = <sw versjon> //versjon applikasjons program
Byte17 = <sw versjon> //versjon boot program
Byte18 = <hw versjon> //versjon hardware
Byte19 = <antall lagrede posisjonmeldinger> //mest signifikante byte
Byte20 = <antall lagrede posisjonsmeldinger>//minst signifikante byte
Byte21 = <antall prosent fylt buffer for posisjonsmeldinger>
Byte22 = <antall lagrede alarmmeldinger fra dyreenhet>//mest signifikante byte
Byte23 = <antall lagrede alarmmeldinger fra dyreenhet>>//minst signifikante byte

Angående byte5 – byte8. Alarm på/av bit maske viser hvilke alarmer som er slått på/av (1 = på, 0 = av), dvs hvilke feilsituasjoner som skal rapporteres til sentralen. Alarm indikator bit'ene viser hvilke feil/situasjoner som er detektert (1 = feil/aktiv, 0 = ingen feil/passiv)

Bit0 i Byte6 og Byte8 = Meldingsbuffer grense alarm
Bit1 i Byte6 og Byte8 = Alarmmelding fra dyreenhet i meldingsbuffer
Bit2 i Byte6 og Byte8 = dør åpnet alarm
Bit3 i Byte6 og Byte8 = batteri alarm
Bit4 i Byte6 og Byte8 = tamper,dvs sol-celle brudd alarm
Bit5 i Byte6 og Byte8 = GPS feil alarm
Bit6 i Byte6 og Byte8 = VHF feil alarm

Denne melding forventer ikke kvittering

5.4.3.2 Basestasjon konfigurasjon

Melding som sendes når sentralen har bedt om å få informasjon om basestasjonens konfigurasjon (f.eks. telefonnr til sentralen). Meldingen sender en parameter. Parameteren sendes som en tekst-string og denne pakken kan ha variabel lengde ettersom parameteren kan bestå av et ulikt antall tegn. Stringen er 0-terminert. Denne meldingen er definert som melding 10.

Byte0 = 0x0A // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = <lengde> //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikante byte
Byte4 = <basestasjon ID/nr> //minst signifikante byte
Byte5 = <parameter nr> //identitet til parameter, feks at parameter 1 er telefonnr til sentralen
Byte6 = parameter byte0 //ascii-verdi for 1. tegn
Byte7 = parameter byte1 //ascii-verdi for 2. tegn
..
ByteK+6 = parameter byteK // ascii-verdi

Denne meldingen forventer ikke svar.

5.4.3.3 Posisjonsmelding fra dyreenhet

Melding som sendes når sntralen ber om å få posisjonsmeldinger fra dyreenhet. Denne melding er definert som melding 11. Meldinger inneholder først data fra basestasjonens GPS-mottaker og deretter kommer data fra dyreenheten.

Byte0 = 0x0B // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = 0x55 //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikante byte
Byte4 = <basestasjon ID/nr> //minst signifikante byte
Byte5 = <klokke,time> // 0-23, fra basestasjonens GPS mottaker
Byte6 = <klokke,minutt> // 0-59, fra basestasjonens GPS mottaker
Byte7 = <klokke,sekund >// 0-59, fra basestasjonens GPS mottaker,
Byte8 = <basestasjon breddegrad,grader> //0-89
Byte9 = <basestasjon breddegrad, minutter> //0-59
Byte10 = <basestasjon breddegrad, minutter desimal> //0-9999, mest signifikante byte
Byte11 = <basestasjon breddegrad, minutter desimal> //0-9999, minst signifikante byte
Byte12 = <Nord(1)/Sør(0) indikering>
Byte13 = <basestasjon lengdegrad,grader> //0-360, mest signifikante byte
Byte14 = <basestasjon lengdegrad,grader> //0-360, minst signifikante byte
Byte15 = <basestasjon lengdegrad, minutter> //0-59
Byte16 = <basestasjon lengdegrad, minutter desimal> //0-9999, mest signifikante byte
Byte17 = <basestasjon lengdegrad, minutter desimal> //0-9999, minst signifikante byte
Byte18 = <Øst(1)/ Vest (0) indikering >
Byte19 = <Gyldige(1)/ugyldige(0) data>
Byte20 = <basestasjon, antall satelitter>//
Byte21 = <fra dyreenhet> //her begynner første byte fra dyreenheten
Byte22 = <fra dyrenehet> //data fra dyreenhet
Byte23 = <fra dyreenhet> // data fra dyreenhet..
..
..
Byte84 = <fra dyreenhet> // siste byte fra dyreenhet

Innholdet i byte5 – byte20 er data fra basestasjonens GPS-mottaker.

Innholdet i byte21 – byte84 er data fra dyreneheten. Det er totalt 64 bytes hvorav 36 bytes er posisjonsdata etc fra dyreenheten og 28 bytes er feilkorrigeringsdata. Se tabell for fordeling av data og ECC bytes samt tabell for koding av melding fra dyreenhet.

Denne melding forventer melding 1 'Kvittering' (uten feilmelding) som svar hvis alt er OK, evnt melding 1 'Kvittering'(med feilmelding) hvis noe går galt.

5.4.3.4 Alarmmelding fra dyreenhet

Melding som sendes når sentralen ber om å få posisjonsmeldinger fra dyreenhet. Denne melding er definert som melding 12. Meldinger inneholder først data fra basestasjonens GPS-mottaker og deretter kommer data fra dyreenheten. Meldingen er den samme som melding 'Posisjonsmelding fra dyreenhet' med den forskjellen at et av alarm-bitene er satt (se tabell for koding av data i melding fra dyreenhet).

Byte0 = 0x0C // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = 0x55 //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikante byte
Byte4 = <basestasjon ID/nr> //minst signifikante byte
Byte5 = <klokke,time> // 0-23, fra basestasjonens GPS mottaker
Byte6 = <klokke,minutt> // 0-59, fra basestasjonens GPS mottaker
Byte7 = <klokke,sekund >// 0-59, fra basestasjonens GPS mottaker,
Byte8 = <basestasjon breddegrad,grader> //0-89
Byte9 = <basestasjon breddegrad, minutter> //0-59
Byte10 = <basestasjon breddegrad, minutter desimal> //0-9999, mest signifikante byte
Byte11 = <basestasjon breddegrad, minutter desimal> //0-9999, minst signifikante byte
Byte12 = <Nord(1)/Sør(0) indikering>
Byte13 = <basestasjon lengdegrad,grader> //0-360, mest signifikante byte
Byte14 = <basestasjon lengdegrad,grader> //0-360, minst signifikante byte
Byte15 = <basestasjon lengdegrad, minutter> //0-59
Byte16 = <basestasjon lengdegrad, minutter desimal> //0-9999, mest signifikante byte
Byte17 = <basestasjon lengdegrad, minutter desimal> //0-9999, minst signifikante byte
Byte18 = <Øst(1)/ Vest (0) indikering >
Byte19 = <Gyldige(1)/ugyldige(0) data>
Byte20 = <basestasjon, antall satelitter>//
Byte21 = <fra dyreenhet> //her begynner første byte fra dyreenheten
Byte22 = <fra dyrenehet> //data fra dyreenhet
Byte23= <fra dyreenhet> // data fra dyreenhet
..
..
..
Byte84 = <fra dyreenhet> // siste byte fra dyreenhet

Innholdet i byte5 – byte20 er data fra basestasjonens GPS-mottaker.

Innholdet i byte21 – byte84 er data fra dyreneheten. Det er totalt 64 bytes hvorav 36 bytes er posisjonsdata etc fra dyreenheten og 28 bytes er feilkorrigeringsdata. Se tabell for fordeling av data og ECC bytes samt tabell for koding av melding fra dyreenhet.

Denne melding forventer melding 1 'Kvittering' (uten feilmelding) som svar hvis alt er OK, evnt melding 1 'Kvittering'(med feilmelding) hvis noe går galt.

5.4.3.5 Feil i posisjonsmelding fra dyreenhet

Melding som benyttes for å sende posisjonsmeldinger fra dyreenhet i de tilfeller hvor det er identifisert feil ved meldingen fordi meldingen ikke lar seg dekode. Meldingen fra dyreneheten blir da i sin helhet lagt inn i en melding og sendt til sentralen slik at den senere kan analyseres. Denne melding er definert som melding 13.Denne meldingen vil sendes sammen med vanlige posisjonsmeldinger når sentralen ber om å få overført posisjonsmeldinger.

Byte0 = 0x0D // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = 0x55 //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikante byte
Byte4 = <basestasjon ID/nr> //minst signifikante byte
Byte5 = <klokke,time> // 0-23, fra basestasjonens GPS mottaker
Byte6 = <klokke,minutt> // 0-59, fra basestasjonens GPS mottaker
Byte7 = <klokke,sekund >// 0-59, fra basestasjonens GPS mottaker,
Byte8 = <basestasjon breddegrad,grader> //0-89
Byte9 = <basestasjon breddegrad, minutter> //0-59
Byte10 = <basestasjon breddegrad, minutter desimal> //0-9999, mest signifikante byte
Byte11 = <basestasjon breddegrad, minutter desimal> //0-9999, minst signifikante byte
Byte12 = <Nord(1)/Sør(0) indikering>
Byte13 = <basestasjon lengdegrad,grader> //0-360, mest signifikante byte
Byte14 = <basestasjon lengdegrad,grader> //0-360, minst signifikante byte
Byte15 = <basestasjon lengdegrad, minutter> //0-59
Byte16 = <basestasjon lengdegrad, minutter desimal> //0-9999, mest signifikante byte
Byte17 = <basestasjon lengdegrad, minutter desimal> //0-9999, minst signifikante byte
Byte18 = <Øst(1)/ Vest (0) indikering >
Byte19 = <Gyldige(1)/ugyldige(0) data>
Byte20 = <basestasjon, antall satelitter>//
Byte21 = <fra dyreenhet> //her begynner første byte fra dyreenheten
Byte22 = <fra dyrenehet> //data fra dyreenhet
Byte23 = <fra dyreenhet> // data fra dyreenhet
..
..
..
Byte84 = <fra dyreenhet> // siste byte fra dyreenhet

Innholdet i byte5 – byte20 er data fra basestasjonens GPS-mottaker.

Innholdet i byte21 – byte84 er data fra dyreneheten. Det er totalt 64 bytes hvorav 36 bytes er posisjonsdata etc fra dyreenheten og 28 bytes er feilkorrigeringsdata. Se tabell for fordeling av data og ECC bytes samt tabell for koding av melding fra dyreenhet.

.

Denne melding forventer melding 1 'Kvittering' (uten feilmelding) som svar hvis alt er OK, evnt melding 1 'Kvittering'(med feilmelding) hvis noe går galt.

5.4.3.6 Ingen meldinger lagret

Denne meldingen blir sendt i de tilfeller hvor sentralen har bedt opp å få sendt over posisjonsmeldinger fra basestasjonen og basestasjonen ikke har noen lagrede meldinger. Meldingen benyttes også som siste melding i en sekvens av posisjonmeldinger for å informere sentralen om at det ikke er flere posisjonsmeldinger lagret i basestasjonen. Denne meldingen er definert som melding 14.

Byte0 = 0x0E // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = 0x05 //pakke lengde
Byte3 = <basestasjon ID/nr> //mest signifikante byte

Byte4 = <basestasjon ID/nr> //minst signifikante byte

Denne meldingen forventer ikke svar.

5.4.3.7 Nedlasting av fil kvittering

Melding som sendes til sentralen når basestasjonen har mottatt en pakke med fildata og basestasjonen er klar til å motta neste pakke med fildata fra sentralen. Denne melding er definert som melding 15.

Byte0 = 0x0F // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = 0x07 //pakke lengde
Byte3 = <basestasjon ID/nr>
Byte4 = <basestasjon ID/nr>
Byte5 = <nr på neste pakke> //nr på neste pakke i sekvens, hvis pakke nr 50 er mottatt så returneres 51
Byte6 = <nr på neste pakke>

Denne melding forventer ikke svar.

5.4.3.8 Basestasjon GPS-data

Denne meldingen benyttes når sentralen ber om å få overført basestasjonens GPS-data. Meldingen benyttes også til å overføre GPS-data fra GPS-modulen til VHF-modulen. Meldingene mellomlagres i et eget buffer (GPS Fifo). Denne meldingen er definert som melding 16.

Byte0 = 0x10 // meldings type
Byte1 = 0x00 // 'spare' byte
Byte2 = 0x19 //pakke lengde
Byte3 = <basestasjon ID/nr>
Byte4 = <basestasjon ID/nr>
Byte5 = <klokke,time> // 0-23, fra basestasjonens GPS mottaker
Byte6 = <klokke,minutt> // 0-59, fra basestasjonens GPS mottaker
Byte7 = <klokke,sekund >// 0-59, fra basestasjonens GPS mottaker,
Byte8 = <basestasjon breddegrad,grader> //0-89
Byte9 = <basestasjon breddegrad, minutter> //0-59
Byte10 = <basestasjon breddegrad, minutter desimal> //0-9999, mest signifikante byte
Byte11 = <basestasjon breddegrad, minutter desimal> //0-9999, minst signifikante byte
Byte12 = <Nord(1)/Sør(0) indikering>
Byte13 = <basestasjon lengdegrad,grader> //0-360, mest signifikante byte
Byte14 = <basestasjon lengdegrad,grader> //0-360, minst signifikante byte
Byte15 = <basestasjon lengdegrad, minutter> //0-59
Byte16 = <basestasjon lengdegrad, minutter desimal> //0-9999, mest signifikante byte
Byte17 = <basestasjon lengdegrad, minutter desimal> //0-9999, minst signifikante byte
Byte18 = <Øst(1)/ Vest (0) indikering >
Byte19 = <Gyldige(1)/ugyldige(0) data>
Byte20 = <basestasjon, antall satelitter>//
Byte21 = <basestasjon høyde over havet>// mest signifikante byte ,høyde over havet multipisert med 10

Byte22 = <basestasjon høyde over havet>// nest mest signifikante byte, høyde over havet multipisert med 10
Byte23 = <basestasjon høyde over havet>//nest minst signifikante byte, høyde over havet multipisert med 10
Byte24 = <basestasjon høyde over havet>//minst signifikante byte, høyde over havet multipisert med 10

Byte19 - Byte22 er basestasjonens høyde over havet multiplisert med 10. Data fra GPS-mottaker har verdier –999,9 ~ 9999,9 slik at tallet som ligger lagret i meldingen har verdier –9999 ~ 99999.

5.4.4 Interne meldinger i basestasjonen

I noen tilfeller der det skal overføres data fra en modul til en annen internt i basestasjen vil det være hensiktsmessig å overføre disse ved å bruke det meldingkonseptet som er innført for kommunikasjon mellom basestasjon og sentral.

Meldingen 'Basestasjon GPS-data' benyttes som en internmelding for å overføre GPS data fra GPS-modulen til VHF-modulen.

5.4.5 Oversikt over meldinger i systemet

Følgende tabell gir en oversikt over meldingene som er definert i systemet:

Meldingsnummer	Meldingsnavn	Beskrivelse
1	Kvittering for mottatt data	Begge veier
2	Request basestasjon status	Fra sentral til basestasjon
3	Request basestasjon konfigurasjon	Fra sentral til basestasjon
4	Forandre basestasjon konfigurasjon	Fra sentral til basestasjon
5	Basestasjon alarm på/av	Fra sentral til basestasjon
6	Restart av basestasjon	Fra sentral til basestasjon
7	Request posisjonsmeldinger fra basestasjon	Fra sentral til basestasjon
8	Nedlasting av fil til basestasjon	Fra sentral til basestasjon
9	Basestasjon status	Fra basestasjon til sentral
10	Basestasjon konfigurasjon	Fra basestasjon til sentral
11	Posisjonsmelding fra dyreenhet	Fra basestasjon til sentral
12	Alarmmelding fra dyreenhet	Fra basestasjon til sentral
13	Feil i posisjonsmelding fra dyreenhet	Fra basestasjon til sentral
14	Ingen meldinger lagret	Fra basestasjon til sentral
15	Nedlasting av fil kvittering	Fra basestasjon til sentral
16	Basestasjon GPS-data	Fra basestasjon til sentral, også internmelding i basestasjonen
17	Request basestasjon GPS-data	Fra sentral til basestasjon

B. CORRESPONDENCE

B.1 Request about database design

Subject: Database
From: Ole Jørgen Lium <ole.jorgen.lium@kitron.com>
Date: Wed, 23 Apr 2003 12:33:00 +0200
To: "'Yngve F. Johansen'" <yfjohans@siving.hia.no>, 'Oeystein Vinningland' <ovinning@siving.hia.no>

Hei.

Hvordan går det med dere? Fikk dere noen påskeferie?

Vi er nå igang med å programmere basestasjonen, og kommer til å begynne
testing ganske snart.
I denne forbindelse må vi ha en database, og det beste vil vel være å
benytte deres databasemodell.
Kan dere legge denne inn på serveren vår?

Har dere prøvd å logge dere inn på xxx.xxx.xxx.xxx?

mvh
Ole Jørgen Lium

B.2 Last specification received

Subject: RE: Mottak av meldingstype 0x08
From: Ole Jørgen Lium <ole.jorgen.lium@kitron.com>
Date: Fri, 25 Apr 2003 13:44:05 +0200
To: "'yfjohans@siving.hia.no'" <yfjohans@siving.hia.no>

Her er svaret jeg fikk av Arild Bråthen.
Han er ansvarlig for design dokumentet.

mvh
Ole Jørgen Lium

-----Original Message-----
From: Arild Bråthen
Sent: 25. april 2003 12:27
To: Ole Jørgen Lium
Cc: Per Vang
Subject: RE: Mottak av meldingstype 0x08

Hei.
Gode spørsmål, jeg vet ikke hvilke versjon av design-dokumentet gutta i
Grimstad har men her er versjon PA5 som har vært gransket og som skal settes
i rev A.
I dette dokumentet går det tydligere frem det som det her spørres om, men
jeg skal ta en oppsummering av dette med meldingstype 0x08 'Posisjonsmelding
fra dyreenhet' (denne meldingennger har nå nr 11 etter at vi har
omstrukturert litt, regner med at det ikke skaper problemer)

1. Basestasjonen registrerer at bufferet med posisjonsmeldinger (dyredata)
er iferd med å gå fullt (i dette bufferet ligger meldinger av type 11, 12 og
13)
2. Basestasjon ringer opp sentral.
3. Sentral registrerer at en basestasjon har koplet seg opp. Sentralen
overtar rollen som 'master' og sender meldingen 'Request basestasjon
status'.
4. Sentralen ser av status-meldingen at det ligger medlinger i bufferet og
starter nedlasting av meldinger ved å sende medlingen 'Request posisjonsmeldinger fra basestasjon'.
5. Basestasjon sender første posisjonsmelding, og venter på 'Kvittering for
mottatt data'.
6.Når kvitteringen kommer, sletter basestasjonen meldingen og sender neste,

venter på kvittering osv. Når det er tomt sender basestasjonen meldingen
'Ingen medlinger lagret'.
7. Sentralen kopler ned.

Denne sekvensen kan også startes ved at det er sentralen ringer opp.

Kvitteringsmeldingen er som du sier melding nr 1 med OK (og meldingstype det
sendes kvittering for).

Meldingen 'Feil i posisjonsmelding fra dyreenhet' (er nå melding 13)er som
du sier ment å lagres som rådata i sentralen for å gjøre det mulig og
analysere dem senere.

Legg også merke til medling nr 12 'Alarmmelding fra dyreenhet'. Denne er
helt lik melding 11 'Posisjonsmelding fra dyreenhet', men her er det satt
noen alarmbit i meldingen (se den oppdaterte tabellen for hvordan data fra
dyreenhet er kodet.

Melding 11 ,12 og 13 kan komme vilkårlig til sentralen i nedlastingssekvensen jeg beskrev da disse meldingene ligger i samme buffer.

Her er dokumentet, som sagt så er det endringer i meldingnr og i data i
meldingene. Dette dokumentet (som har rev PA5) vil snart bli satt i rev A og
dere vil da få et eksemplar. Men jeg regner ikke med at det blir flere
endringer.

Mvh Arild Bråthen

-----Original Message-----
From: Ole Jørgen Lium
Sent: 25. april 2003 06:35
To: Arild Bråthen
Subject: FW: Mottak av meldingstype 0x08

Svarer du meg, så svarer jeg studentene...

Ole Jørgen

-----Original Message-----
From: Yngve F. Johansen [mailto:yfjohans@siving.hia.no]
Sent: 25. april 2003 03:04
To: Ole Jørgen Lium

Subject: Mottak av meldingstype 0x08

Hei,

Har jeg forstått dokumentasjonen hvis jeg antar at meldingstype 8
(posisjonsmelding fra dyreenhet) kan komme som en strøm fra basestasjon?
Og avsluttes av meldingstype 0x0F?

Og kvitteringsmeldinger (type 0x01) uten feilmelding som det refereres
til i dokumentet, er det meldingstype 0x01 med feilkode 0 (OK)?

Et siste spørsmål, meldingstype 0x09 (feil i posisjonsmelding), kan de
også komme i en strøm, ala 0x08 (hvis så er tilfelle...)? Skal disse
dataene lagres sammen med andre posisjonsmeldinger, eller skal kun
rådataene lagres for senere analyse?

Vennlig hilsen
Yngve F. Johansen

C. SOURCE CODE FOR THE PROTOTYPE (CD)