The Pennsylvania State University

The Graduate School

Department of Electrical Engineering

STOLEN VEHICLE TRACKING USING SENSOR NETWORKS

A Thesis in

Electrical Engineering

by

Soumya Jain

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2009

The thesis of Soumya Jain was reviewed and approved* by the following:

Sencun Zhu
Assistant Professor, Department of Computer Science and Engineering
Thesis Co-Advisor

David Miller
Professor, Department of Electrical Engineering
Thesis Co-Advisor

Mohsen Kavehrad
W.L. Weiss Professor, Department of Electrical Engineering

Ken Jenkins
Professor, Department of Electrical Engineering
Head of the Department of Electrical Engineering

*Signatures are on file in the Graduate School

ABSTRACT

Wireless sensor networks are typically deployed in ubiquitous and distributed computer system settings to ‘sense and send’ information. Recently, there have emerged a significant set of applications which go beyond collection of data, and involve taking context aware and event based collaborative decisions. We present one such system – a Wireless Sensor Network based Stolen Vehicle Tracking System. Anti-theft systems are deployed widely in vehicles today, but have many shortcomings and have not been very effective in reducing theft rates. To address their limitations, a novel system for theft detection was proposed called the “Sensor Network based Vehicle Anti-theft System (SVATS)”. This research work serves as the next part of this system and extends the functionality to be able to *track* and *recover* the stolen vehicle. The main challenges addressed here are the development of a new, more dependable theft detection mechanism, design of an *intra*-vehicle network (in addition to the inter-vehicle network) towards adding more reliability and security, intelligent theft reporting and security primitives for the system. A complete prototype of the system has been built, using MICA2 motes and other hardware representative of the conceptualized system elements. We also carry out an evaluation and performance analysis of each subsystem.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my advisor Professor Zhu, who provided the over all direction as well as day to day guidance for this research work. For their review, instructive comments and advice, I thank my committee co-chair Professor Miller and committee member Professor Kavehrad. I would also like to thank graduate students Jyoti Bala and Ruan Ge, for their help in the coding and implementation of the system.

Chapter 1

Background and Introduction

Improvements in wireless communication and processor technology have enabled the development of low cost, low power, small sized ‘motes’ which have sensing, computation and radio communication capabilities. When deployed as a network, they can be used in a wide range of civilian and military applications. Typical deployments are directed towards remote area monitoring, collecting data about rare species, industrial parameter monitoring and other defense applications. In a project started two years ago, we have developed a Sensor Network based Vehicle Anti-theft System (SVATS) [1]. SVATS was mainly designed as an automobile theft detection and notification mechanism. This thesis presents the design and implementation of the *next step* – locating and tracking the movement of the stolen vehicle.

In the United States, a motor vehicle is stolen every 26 seconds [2]. The National Insurance Crime Bureau has reported that vehicle theft rates have kept steadily high at about 1.2 million since 1997, despite the widespread use and increasing sophistication of anti-theft devices. These systems can be classified roughly into three types – *lock devices, alarm systems and vehicle tracking / recovery systems*. Lock devices are represented by the typical steering wheel lock which, though cheap are easy to disarm. Alarm systems are by far the most popular anti-theft devices. However, they suffer from very high false positive rates – we have all heard their loud noises. The most sophisticated are the vehicle tracking systems, usually based on wireless transmitters which can be activated along with a GPS device which broadcasts the current location to all tracking devices. The main drawbacks of this system are the high costs involved and the fact that they can be easily disabled. As an illustration of the former reason, the upfront costs are in the range of \$ 500 - \$ 1500, and there are other associated periodic payments. About the latter, if the thief simply shields the antenna or breaks the device, the system will fail.

To address these above mentioned limitations, we designed a sensor network based anti-vehicle theft system. The main concept is that all parked vehicles (say in a parking lot) will contain sensor nodes, which form a sensor network. This network monitors all the cars, with each node monitoring its neighbors and in turn being monitored by them. When a vehicle theft is detected, a theft report is sent hop by hop to the base station from where further action is coordinated, including sending an SMS to the owner of the vehicle. This system relies on the vision that sensor nodes will become cheap and thus be produced in large quantities in the near future [3]. The following section gives an overview of SVATS.

1.1 Overview of Sensor network based Vehicle Anti-theft System

In SVATS each vehicle is equipped with a *master sensor* which is connected to the vehicle power. The master sensors of all the vehicles form a sensor network in the parking lot. These parking lots could be located at shopping centers, residence complexes, university campuses, hospitals, etc. For each parking lot a separate network is formed which is centered about one installed base station.

Each master sensor broadcasts periodic *alive* messages once it has joined the existing network. Vehicle theft detection is achieved by the mutual monitoring and detection of unauthorized movements of neighboring sensors (and thus vehicles) based on the Received Signal Strength Index (RSSI) of these *alive* messages. When the theft is detected it is reported in a multi-hop fashion through the network to the base station (and thus the security office). Further an SMS can be sent automatically to the owner of the vehicle. The following example illustrates the working of SVATS.

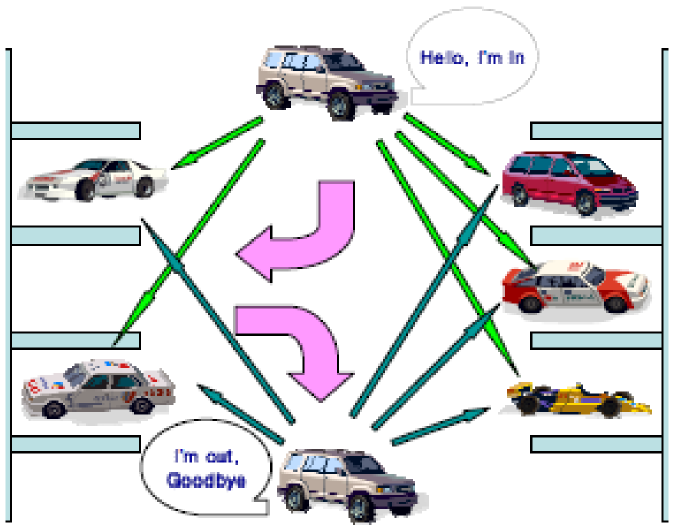


Figure **1-1**: Working of SVATS in a representative parking lot

Let us say Bob enters the parking lot in his car. Using a remote control, he instructs the master sensor to send a ‘join’ message to the network. On receiving this message, the nodes of the existing network add Bob’s master sensor to their list of neighbors who they need to monitor. Further, Bob starts making his own list of neighbors at the same time. After joining the network, Bob’s master sensor sends periodic *alive* messages which indicate his presence and help the neighbors create a signature of his RSSI. When the car leaves, it sends a ‘leave’ message. If it moves without sending a valid ‘leave’ message, the ‘monitors’ will immediately know this based on the change in the RSSI signature formed. After voting amongst themselves (to reduce false positives), they will notify the base station of the theft.

1.2 Introduction to Tracking the Stolen Vehicle

The focus of SVATS is on detecting the theft of the vehicle. Valuable as it is, this system is considerably enhanced if the stolen vehicle can be tracked, thus significantly boosting chances of recovery. This is the contribution of this research, and the design and implementation is presented in this thesis.

First of all, we introduce the concept of slave sensors. The master sensor is envisioned to be connected to the vehicle's power source. This implies that it would be easy to find and could be destroyed by the thief, resulting in the failure of the entire system. To prevent this we design 'slave sensors', which are hidden in the car, have a very passive (and low power) operation and thus can function for a long time on batteries. Their primary function is to monitor the master sensor, and in case of it being destroyed, take action towards reporting this. In case the master is functioning also, they are still useful in reporting the location of the vehicle for tracking.

Secondly, we utilize 'Roadside Access Points'. These are envisioned for various purposes ranging from providing Internet access to cars to highway safety mechanisms [4],[5],[6],[7]. We design an application for the Roadside Access Points, to which the slave sensors and master sensor of the stolen vehicle will report its theft. Thus, by using these theft reports received by the Roadside APs, we can track the stolen vehicle throughout the city. In case these Roadside APs are not directly in range of the stolen vehicle, the sensors could transmit the message to the sensor of neighboring vehicles which could then carry this theft report to the Roadside APs or take other appropriate action. Further, in another model, we could also use routers at people's residences.

We design the complete system as a chain of blocks or sub-systems each with a specific function. This comprises the following blocks –

- *A Motion Sensor based Theft Detection mechanism* – As a more reliable alternative to the RF signal based theft detection, we design a new motion sensor based theft detection mechanism

- *Monitoring System of the Slaves* – This includes the protocol which the slaves follow while monitoring the master sensor achieving targets of extremely low power operation, however keeping reasonably fast theft detection.
- *Protocol after theft is detected* – This involves switching to a secret pre-decided channel and then voting amongst the slaves to confirm that the master has been destroyed or the vehicle has been stolen.
- *Theft Reporting* – This section is concerned with the various protocols for reporting the theft to the AP, keeping in mind interference and avoiding detection by the thief.
- *Interfacing of the slaves to the AP* – The sensors function on either *ZigBee* or 802.15.4 communication protocol stacks, while the roadside APs are expected to use 802.11 or Wi-Fi. This interfacing is also an important block of our system.
- *Security* – Finally, the over all security of the system is considered using pair-wise key based encryption. We use a version of Blundo's Scheme [8] for the key establishment.

In comparison with the existing systems, SVATS and the stolen vehicle tracking system together have many distinctive advantages. Firstly, our system is cheap. Sensor nodes, though currently expensive, are expected to cost under $1 [2]. We do not use any advanced modalities such as a GPS or any powerful sensors. The Roadside APs are also not a requirement of our system, but are already in place for various other functions as mentioned before. Secondly, the entire functionality of our system is distributed. This makes is next to impossible to find a single disabling point, which will lead to complete failure. This is the case with most current devices such as GPS based tracking or alarm systems. Not only is SVATS distributed i.e. any car is monitored by a number of other cars, but the master sensor's monitoring by slave sensors is also distributed. Also, the form factor of the sensors, envisioned to be sized below 1cubic mm [2], makes them easy to hide and thus difficult to destroy. Finally, our system can be deployed incrementally. We have tested it on a very small scale so far, but the infrastructure for this system is coming up rapidly and can be installed in parts testing it at each level.

We have built a complete prototype of our system and tested its performance using actual vehicles. We used Crossbow's MICA2 motes [9] as sensors both as the master sensor and the slave sensors. To represent the Roadside APs we use the very popular Linksys WRT54G [10] home router.

1.3. Overview of Hardware and Software Platforms

This section gives an overview of the hardware and software platforms we use to implement and evaluate our system. A real implementation is an important strength of this research work, as opposed to simulated or analytical results. It has helped in highlighting and thus addressing several important issues, and made us aware of the practicalities involved. We choose the Crossbow MICA2 [9] sensor mote as our hardware platform and use TinyOS and nesC to design and install the software on the mote.

1.3.1. Hardware Platform – The MICA2 mote

Figure **1-2**: The Crossbow MICA 2 sensor mote

The MICA2 sensor mote is a third generation mote module used for enabling low-power, wireless, sensor networks. It comprises a MPR400 processor, which is actually based on the low power Atmel AtMega128L microprocessor and CC1000 radio, with an expansion connector for sensor boards (with various sensing modalities) and LEDs for visual debugging. The radio is a basic 916 MHz transceiver with a tunable frequency. With proper power management and typical applications, the MICA2 can function for greater than a year on two AA size batteries. The 51-pin expansion connector can connect to a variety of sensor boards, which provide sensing of quantities such as pressure, temperature, light, acceleration and magnetic fields.

We use a MIB510 programming board to connect the PC to the mote via a serial cable, which allows programming the mote with the desired software and also reading the flash / debugging information.

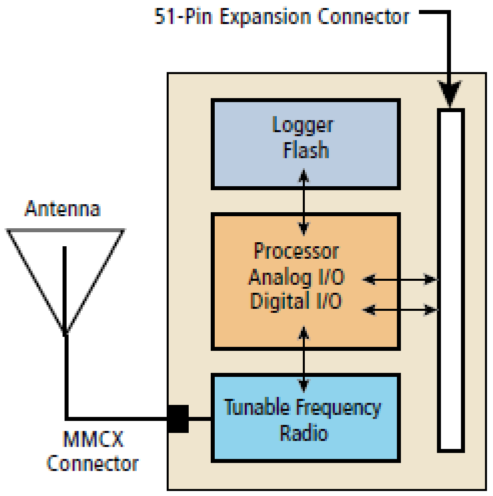


Figure **1-3**: Block diagram of the MPR400 based MICA 2 mote

1.3.2. The Software Platform – TinyOS operating system and nesC programming language

To design the software and install it on the mote, we use the operating system TinyOS and the programming language nesC. A brief description of the important concepts and relevant details are given here.

The TinyOS system, libraries, and applications are written in nesC, a new C-like language for programming structured component-based applications. The nesC language is primarily intended for embedded systems such as sensor networks. nesC has a C-like syntax, but supports the TinyOS concurrency model, as well as mechanisms for structuring, naming, and linking together software components into robust network embedded systems. The principal goal is to allow application designers to build components that can be easily composed into complete, concurrent systems, and yet perform extensive checking at compile time.

TinyOS defines a number of important concepts that are expressed in nesC. Firstly, nesC applications are built out of **components** with well-defined, bidirectional **interfaces**. Secondly, nesC defines a concurrency model, based on **tasks** and **hardware event handlers**, and detects **data races** at compile time.

A nesC application consists of one or more *components* linked together to form an executable. A component **provides** and **uses** *interfaces*. These interfaces are the only point of access to the component and are bi-directional. An interface declares a set of functions called **commands** that the interface provider must implement and another set of functions called **events** that the interface user must implement. For a component to call the commands in an interface, it must implement the events of that interface. A single component may use or provide multiple interfaces and multiple instances of the same interface.

There are two types of components in nesC: **modules** and **configurations**. Modules provide application code, implementing one or more interface. Configurations are used to assemble other components together, connecting interfaces used by components to interfaces provided by others. This is called **wiring**. Every nesC application is described by a **top-level configuration** that *wires* together the components inside.

Finally, we present an overview of the concurrency model. TinyOS executes only one program consisting of selected system components and custom components needed for a single application. There are two threads of execution: *tasks* and *hardware event handlers*. Tasks are functions whose execution is deferred. Once scheduled, they run to completion and do not preempt one another. Hardware event handlers are executed in response to a hardware interrupt, and also run to completion, but they may preempt the execution of a task or some other hardware event handler. Commands and events that are executed as part of a hardware event handler must be declared with the **async** keyword.

Because tasks and hardware event handlers may be preempted by other asynchronous code, nesC programs are susceptible to certain race conditions. Races are avoided either by accessing shared data exclusively within tasks, or by having all accesses within **atomic** statements. The nesC compiler reports potential *data races* to the programmer at compile-time. It is possible the compiler may report a false positive. In this case a variable can be declared with the **norace** keyword.

Chapter 2

Motion Sensor based Vehicle Theft Detection

2.1 Introduction

The concept of using an accelerometer as a movement / motion sensor has been used in a number of applications. These range from detecting theft of portable items (e.g. laptops, televisions, etc) [11] to measuring heartbeats of patients [12]. They are also used in applications where measuring vibrations is important, such as structure monitoring.

With respect to vehicles, there are some products where these accelerometers have been used to detect the movement of a car as well [13] [14]. A two-axis accelerometer model can be used to detect *jacking up* or tilting of a car while getting towed. In our system, we extend this concept to an anti-theft mechanism where unauthorized movement of the car will imply theft, and will be reported immediately to the owner / law authorities.

2.2 Hardware Description

The MICA 2 sensor motes used in our system typically use the MTS310 [15] or MTS300 sensor boards for their sensing modalities. The MTS310 includes a dual-axis accelerometer, which we use as a movement sensor. This component is the MEMS based ADXL202 Analog Devices accelerometer [11]. Its chief features of interest to us are –

- Resolution of 2 mG
- Range of + / - 2 G
- Noise floor of 200 µG / Hz

- Acquisition time of below 1 ms

The analog output from the accelerometer is put through an ADC and read as a voltage in the raw form.

2.3 Implementation of Theft Detection

The master sensor's software module utilizes the TinyOS provided *AccelX* and *AccelY* interfaces and gets sample values at a frequency of 5 Hz. This is found, experimentally, to be sufficient to detect any changes in acceleration, which will indicate movement. It is observed that due to noise effects (at 5 Hz, we expect a floor noise of 1 mG) the value of the accelerometer varies by a small amount even if the vehicle is static. Thus, we use an average of ten consecutive values to form a signature of the static acceleration. This signature is also updated periodically to account for rare, but possible minor changes in the *readings* due to wind or temperature effects.

Once the signature has been formed a confidence interval is calculated about it i.e. limits are set, which if exceeded by the acceleration values, will indicate theft. These limits are also determined experimentally keeping in mind the following probable causes of false positives –

- People accidentally 'pushing' the car from outside
- Passengers shaking the car while entering and being seated

The parameters were set after exhaustive experiments. We used the MTS 310CB [15] sensor board, and used three different master sensors to avoid any errors. These experiments were carried out in a car, and helped in reducing the false positive rate to 0% and detection rate to 100%.

2.4. Description of Software Modules used

The main software modules for this functionality of the code are the *Accel*, the *Timer* and the part of the Master sensor code, which controls these two modules. The function of the *Accel* module is to provide commands which can control the ADC, which is wired into it. The ADC provides an interface to the actual accelerometer raw values, which are read by the *Accel* module. When these readings are read in is decided by the *Timer* module which generates interrupts at periods specified in its invocation.

Besides these core components, *Leds* and *GenericComm* is used for testing, debugging and experimenting with the system. The former controls the three LEDs available on the sensor, which is very useful in visual debugging, while the latter is used as the send and receive message interface, which transmits required 'readings' back to our base station for analysis.

The following figure shows the component graph for this functionality of the system.

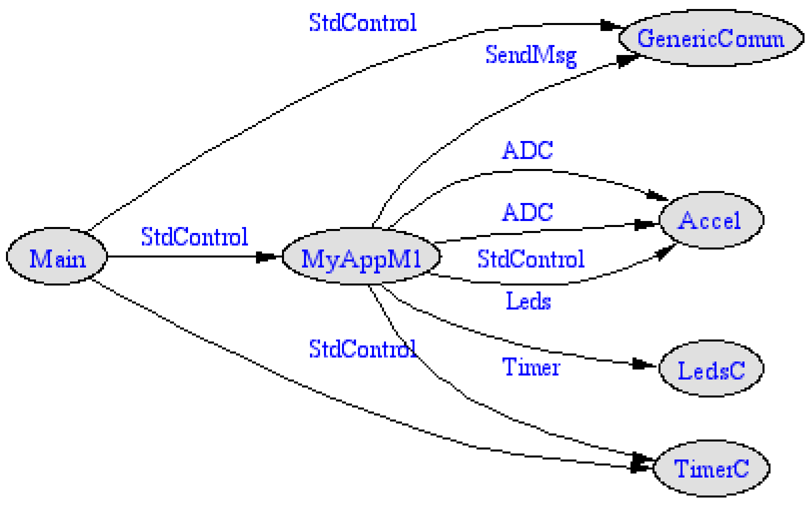


Figure **2-1**: Software Modules of Motion Sensor based Theft Detection

2.5. Comparison between Motion Sensor and RSSI based schemes

It is important to evaluate the relative performance of the RSSI based and the motion sensor (or accelerometer) based schemes. We carry out this comparison based on several important parameters, and discuss how each scheme addresses its limitations.

2.5.1. Detection Rate

This is the most important parameter since the system must be completely reliable. There should no possibility of the vehicle being driven away without the alarm being raised. In this respect, both schemes do very well. The RSSI values are very sensitive to movement of the vehicle and can detect a change within a movement of 2 feet. The accelerometer readings as discussed above are also sensitive enough to detect movement of the vehicle immediately. Experimentally, we find that movement even less than 1 foot is detected. This is expected because while RSSI takes some time to change, the accelerometer will detect any kind of motion instantly.

Quantitatively speaking, both schemes are able to achieve a 100% detection rate.

2.5.2. Detection Speed

The second parameter we study is the speed of detection or response. Here again, both schemes do very well. The RSSI based scheme takes typically 4 -9 [1] seconds to detect the theft, and vote amongst the neighbors to confirm. The accelerometer based scheme however is quicker in the sense that it does not depend on other vehicle's sensors to vote on its decision. As soon as unauthorized motion is detected, the master sensor can decide itself and broadcast a message saying that it is stolen. In that sense, this detection does not take more than 1 second.

This difference may not seem significant in terms of seconds, but if we imagine the actual scenario it might make a huge difference. Instant detection would imply an almost instant reporting, and thus prevention of the theft would be much more likely. This could include steps like closing the parking garage, or moving security personnel to the spot immediately.

2.5.3. False Positive Rate

The problem with most security systems are false positives. In case of vehicle anti-theft systems, this is a major problem, because if false positives happen often enough, the owners typically end up disabling the entire system. Both the schemes fair well on this parameters also. However, the chances of false positives are much higher in the RSSI based scheme than in the motion sensor based scheme.

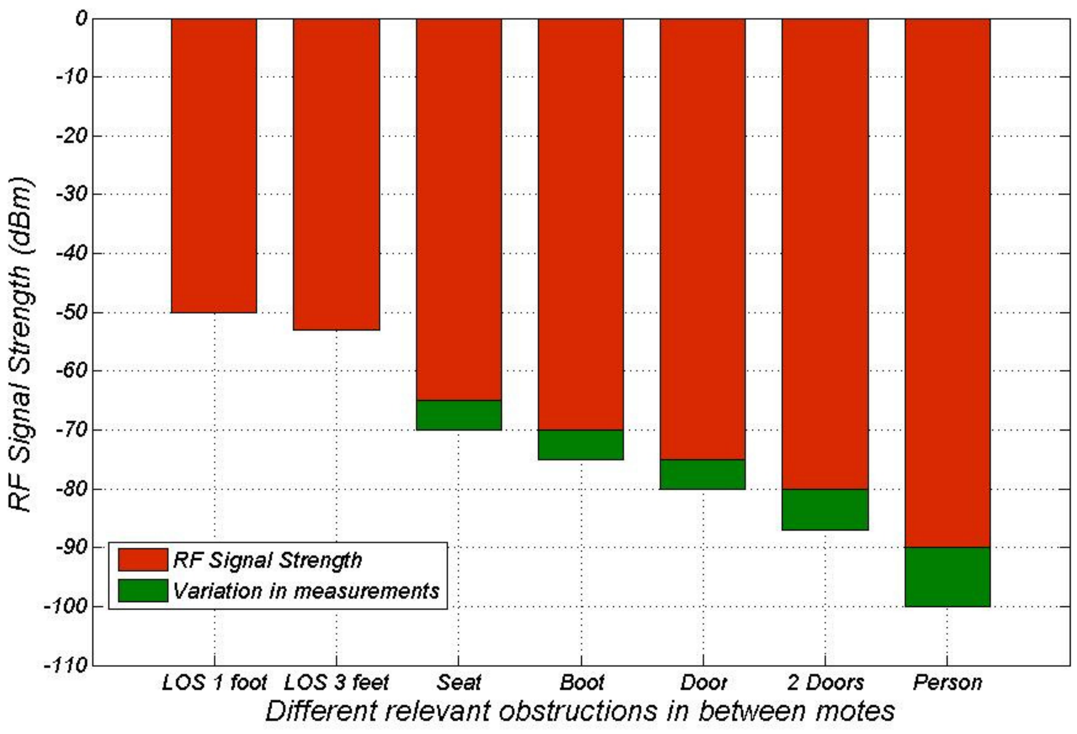


Figure **2-2**: RF Signal Strength measurements with different obstructions

This is because the RF signal could change because of various external parameters. Our experiments show that there is a significant change because of any of the following factors –

- People walking between the sensors
- Vehicle driving between or parked between the sensors
- Any kind of strong electromagnetic or radio interference

This is shown in Figure 2-2. We carried out exhaustive experiments, using two different car models (a sedan and a SUV) and about 10 sensor motes. Also, these measurements were taken at different positions of motes' in the cars and with 10 readings per measurement. The red portion indicates the strength of the signal in dBm. This being a negative quantity, the lower bars on the left end of the bar chart are actually stronger signals than the ones on the right end. Further, we noted that the measurements varied due with positions of motes in the cars. For example, different places in the rear seats gave values which varied by about 3 dBm. We recognize this as variation due to internal fading and other such effects and thus depict this separately on the bar chard instead of averaging the values. The green part of the bar chart thus indicates the amount of variation (minimum reading to maximum reading) in the measurements.

As we can observe, there is almost a 10 to 20 dBm drop if a person walks between the two cars. This will definitely trigger off the RSSI based detection scheme. This is stopped only at the voting stage, after other sensors are able to refute the accusations made by some. This system work well as in it can achieve 0% false positive rates, however, at the cost of a lot of communication overhead, since every time there is an accusation a vote needs to be taken.

On the other hand in the motion sensor based scheme, we need to deal only with the reading of an individual accelerometer. As long as the confidence intervals are set properly, there are no external factors which can cause false positives. The only rare cases are when people accidentally push the vehicle from outside, and we could also account for that in our system. Thus the chances of false positives are much lower. We also achieve a 0% false positive rate, based on experimental results. Further, we note that there is no communication overhead in reducing our false positives.

To sum up, both schemes are able to remove false positives effectively, however the RSSI based scheme requires a lot of communications and sophistication in doing that. The motion sensor based scheme can attain these goals with relative simplicity and minimal overheads both in processing and in messages.

2.5.4. Cost of method

Finally we consider the cost required to enable both systems. The RSSI based scheme definitely provides a cheaper solution. It uses the inherent radio feature of the sensor motes and does not require a motion sensor. The motion sensor based scheme however uses an accelerometer at its core. This will require an additional sensor board and its interfacing devices, such as the ADC.

If we analyze this with respect to costs today however, it is not a big drawback. The accelerometer we are using costs only $ 10 [11] bought as a single piece. The complete sensor board from Crossbow is more expensive though. However, since we envision sensors to be mass produced and become cheap in the future, we expect the sensing functionalities to also become cheap. Thus, though currently the motion sensor is an additional modality which adds cost, in the future we expect that this would not be an issue.

The points discussed above are summarized in the Table **1-1**.

Table **2-1**: Comparison between RSSI and Motion Sensor based schemes

Comparison Parameter	**RSSI based scheme**	**Motion Sensor based scheme**	**Remarks**
Detection Rate	100 %	100 %	Motion Sensor based scheme is more sensitive
Detection Speed	4 – 9 seconds	Less than 1 second	Motion Sensor based scheme is faster
False Positive Rate	0 %	0 %	RSSI based scheme requires considerably more communications and processing over heads
Cost of Scheme	No extra equipment apart from basic radio	Accelerometer required	Though additional cost, accelerometer / sensor board is not very expensive, and could be used for other purposes also

Chapter 3

Monitoring System of the Slave Sensors

3.1. Introduction

This chapter describes the steady state operation of the slave sensor motes (slaves) monitoring the master sensor mote (master). This phase is active once the vehicle has been parked and the master has started broadcasting its periodic *alive* messages to the other vehicle's masters. The slaves ensure that the master has not been destroyed or tampered with by making certain that they receive the periodic *alive* messages.

Section 2 details the design goals and constraints. Section 3 gives an overview of the existing protocols and their suitability for our system. Section 4 presents our solution and evaluates its performance.

3.2. Design Considerations

There are three main concerns in the design of this protocol, which are detailed below –

- *Firstly,* we expect that the slaves should be hidden and not connected to the vehicle's power source. This implies that they must use batteries, which makes energy efficiency a primary concern to ensure very infrequent replacements. A reasonable requirement of how long the slaves *should* function without recharging, given current hardware platforms, is about a year. However, typically sensor motes can only last between 100 to 120 hours if always active [17]. Thus, our first goal is to design a *sleep scheduling algorithm* for the slaves.

- *Secondly*, they must detect the loss of the master very fast. With quick detection, not only will the tracking be more effective, but prevention of the theft (such as barricading the parking lot) might also be possible. However, typically faster detection requires more active monitoring, which implies more energy consumption. Thus, this delay-efficiency trade-off needs to be set to give the best possible performance.
- *Thirdly*, the slaves must operate as passively as possible i.e. send out a minimum number of messages. This is crucial for two reasons – to prevent the pin-pointing of their locations (and thus getting disables) based on RF signal measurements, and to not add to the radio communication which will increase collisions and possible the false positive rate.

3.3. Existing Protocols

Energy efficiency being a major concern in sensor networks, sleep scheduling has been an active area of research. Different mechanisms have different assumptions and priorities such as the detection model, sensing area, transmission range, failure model, time synchronization and location information. This section discusses some commonly used protocols, and some which are the most relevant with respect to our goals and constraints.

The first step is to narrow down the list of existing solutions based on our requirements and assumptions –

- Our system will comprise only a few slaves (typically less than 10), and thus they can be in a non-hierarchical architecture, with no 'cluster-heads' or leaders. Thus, every sensor will assume the same role and responsibility.

- The sensor network deployment strategy is a one-hop network, since it is all intra-vehicular. Thus we can assume that all the nodes are in the transmission range of each other.
- The detection model is deterministic i.e. if the slave is within range of the master it will hear an *alive* message and can conclude that the master is functional. This is in contrast with [18], where the detection probability is proportional to the distance between the source and sensor.
- We do not assume that time synchronization is already implemented. This is not required for the rest of the system, and hence, as far as possible, this protocol also should not require it.
- We do not expect frequent failures of slave motes, and thus do not design taking that into account.
- Lastly, the slaves are stationary with respect to each other and the master, and thus we need to not consider mobility issues.

Based on these guidelines, we evaluate the suitability of various protocols and modifications based on them. The Sponsored Sector mechanism [19] and the MSNL protocol [20] are both based on the principle that a slave would only shut off if there are definitely others to monitor the event. This would have worked well for our case, but the communications overhead is significantly high. Probing Environment and Adaptive Sensing (PEAS) [21] is another popularly used scheme, where a node probes other slaves, and wakes up only if it receives no replies. This scheme however does not ensure balanced energy consumption, and in our system, when the master is destroyed, it is important for all slaves to be ready to 'take action'. Another protocol, ASCENT [22], also suffers from the same drawback.

Apart from the above outlined schemes, there are a few other major ones proposed which give a probabilistic coverage. Among these, a common one is the LDAS [23], based upon the principle that periodic tickets are given to each slave, and with sufficient number of tickets it can sleep off. However, this also uses the extra messages, which our system must avoid.

Finally, we consider Random Independent Scheduling [24]. This protocol involves the slaves making completely independent decisions about their waking up, without dynamic co-ordination with others. In essence, after a set period, a slave wakes up with a probability *p*. This scheme satisfies most of our criteria for a scheduling algorithm, except that it assumes time synchronization. A subset of this scheme would be a deterministic wake up schedule, where $p = 1$. An excellent comparison between these two schemes, both theoretically and using simulations has been done in [25]. This research proves that the both deterministic and probabilistic Independent Scheduling per form more or less at par, if there are constraints on the delay, which in our case there are [see Section 2, point 2].

Thus, our scheme is based on Independent Scheduling. However, the design does not require time synchronization as a separate phase of the operation.

3.4. Deterministic Independent Sleep Scheduling Mechanism

The phase of the slaves monitoring the master starts as soon as the vehicle is parked. The master starts broadcasting periodic *alive* messages, primarily intended for the master sensors of the neighboring vehicles. However, these are also heard by the slaves, in the intra-vehicle network. Based on the periodic receipt of these *alive* messages, the slaves decide that the master is functional.

Initially (when the vehicle is parked), each slave is woken up, using a protocol like Radio Triggered Wake-up [26]. If not feasible, then any other extremely low duty cycle scheduling algorithms could be used, for the slaves when the vehicle is being driven. There is a lot of research in this area.

When the slave now receives its first *alive* message performs the following actions –

- It starts a *wake_up_timer,* which decides the period for which it can enter the low power sleep mode. This timer decides the duty-cycle of the slave. It is meant to fire 'just before' the next *alive* message, which this slave has to 'catch' is due to be broadcasted.
- It then enters the sleep mode, implementation details about which are given at the end of this chapter.

When the *wake_up_timer* fires, the slave performs the following actions –

- It shifts from sleep mode to active mode
- It starts another timer : *timer_no_alive*. The purpose of this timer is to wait some time after the scheduled *alive* has been missed, and then fire. This firing implies that the slave has stopped receiving *alive* messages from the master, and hence deems it to be destroyed. Its actions after this i.e. the next phase, are described in the next chapter.
- It then waits for the *alive* message from the master.
- On receiving an *alive* message it re-starts a *wake_up_timer*, and carries out all the functions described previously.

This scheme meets the design objectives laid out earlier very well. Firstly, by sleeping most of the time, the slaves can conserve most of their energy. Secondly, it is a perfectly passive scheme, in which no messages are sent out by the slaves. Thirdly, its detection speed is very high. Each of these is evaluated below.

3.5. Performance Analysis

3.5.1. Energy Efficiency Analysis

The following analysis shows how much the sleep scheduling extends the battery life of the slaves, as well as the important parameters in the system. The slave is awake only for the time 'just before' the *alive* message is due, and for a negligibly short time after that. Thus, this time is crucial in deciding the duty cycle.

Exhaustive experiments have been conducted to determine the time the slave should be awake, specific to our system. If this time is too low, the slave will miss packets when they are due, and will have to keep awake and waiting till the next *alive* is sent (thus, for a time equal to the master's *alive* sending period). If the time is too high, the slaves are awake unnecessarily, thereby wasting energy. These experiments take into account the following parameters which might affect results –

- Averaged over 10,000 *alive* message packets
- Three master motes were used, and averages taken
- Three slave motes were used, and averages taken

The following graph shows the variation of the first *alive* message loss rate, with the time the slave is active before the message is due –

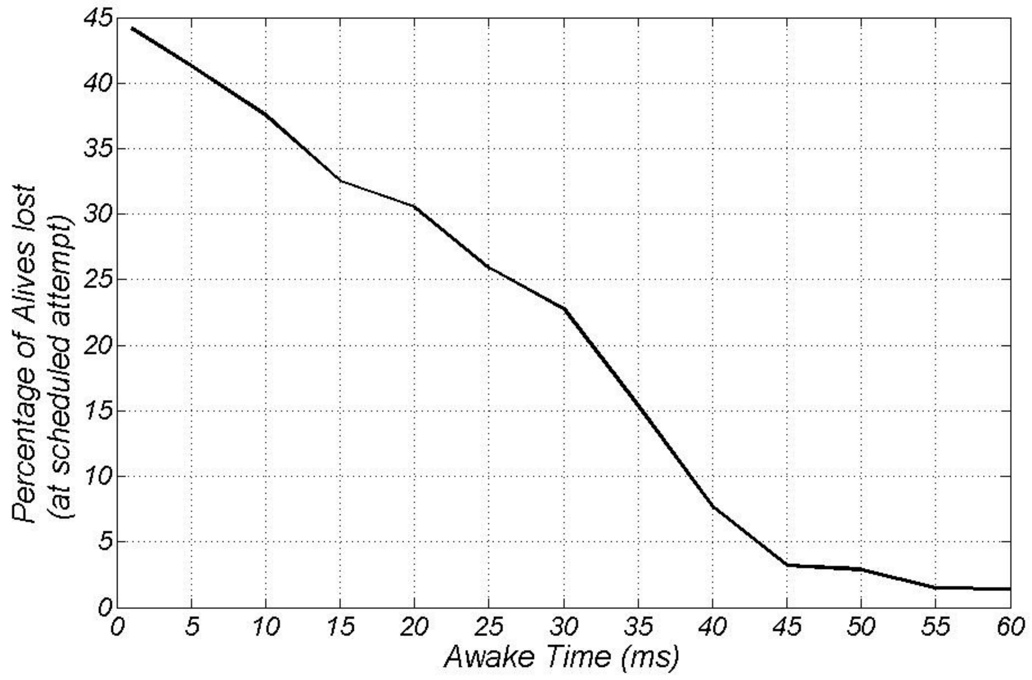


Figure **3-1**: Variation of Packet Loss Rate with Awake Time

As expected, the more time the slave is awake, the more are chances that it will receive the *alive* message the first time itself (i.e. the scheduled one, not the next one). The upper limit of the loss rate is bound to be 50 %, because this extreme case is the one in which the slave wakes up at random times, and waits for one full alive message period, which will average to every alternate message being received. The lower limit will of course be 0 %, but this is observed to be asymptotically approached, due to at least a few packets being lost because of collisions or interference. Based on this graph, we proceed in our analysis of the energy efficiency.

Intuitively, we realize that using a low value of awake time reduces the direct energy consumption. However, if an *alive* packet is missed, then the slave has to be awake and waiting for the next one. This is an order of magnitude larger at least. As an example, a reasonable value for the master's alive message interval is 500ms, while the slaves wake up time is 50ms. Thus, in case of missing a packet, the slave's awake time effectively becomes 550ms, a factor of 11 times.

Thus, we try to make a tradeoff between these parameters, by plotting the effective awake time (based on the *alive* packet loss rate values measured previously) versus the *preset* awake time. To calculate the Effective Awake Time, we calculate the 'expected value' of the awake time, using the measured loss rates as probabilities that the scheduled *alive* message is received.

Let –

N = Number of slave sensors

Awake time of slaves = t_{awake}

Master's alive message period = t_{alive}

Probability of receiving alive message = p

Then,

p = (1 – loss rate)

Effective awake time = $(t_{awake} * p) + (t_{alive} + t_{awake}) * (1-p)$

Using various values of t_{alive} we plot the curves in Figure 3-2. Various values of the t_{alive} are possible because of the implementation details of SVATS. The optimization of that system is expected to give a value of between 200 ms and 500 ms, and thus we plot for these numbers.

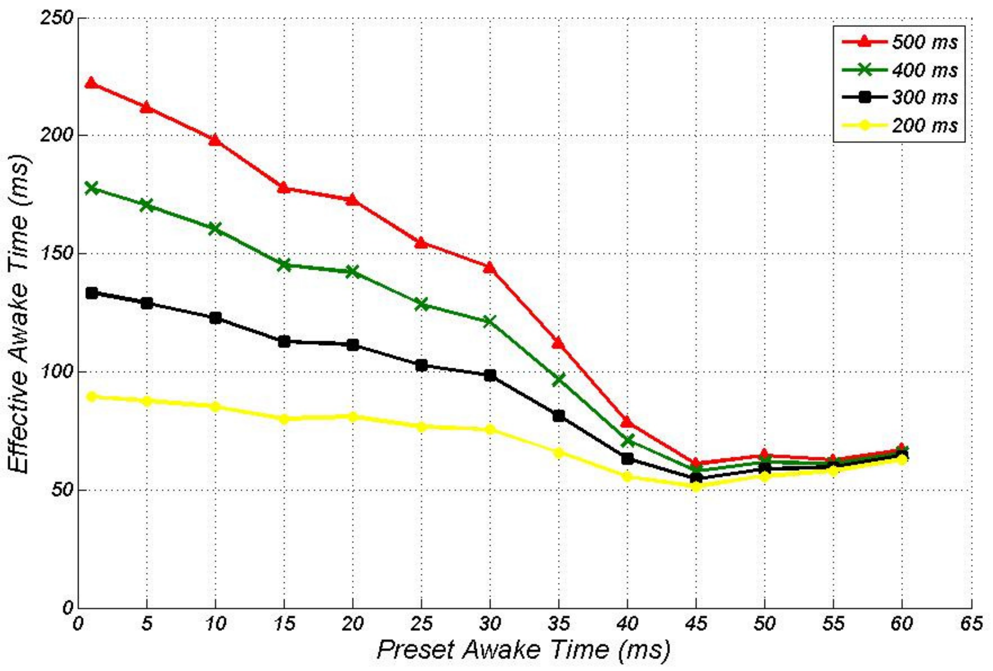


Figure **3-2**: Optimization of the Awake Time

We observe a very clear minimum of the effective awake time at 45ms. The loss rate at this value is about 4 %. If the awake time is decreased, this loss rate goes up and we get higher effective times due to more *alive* messages being received after waiting one period. On the other had if the awake time is increased, the loss rate does go down, but only asymptotically and slowly. Thus, in this case the large preset awake time itself causes the effective time to increase.

Thus we conclude that the optimal time that the slaves should be awake is 45 ms before the next due *alive* message. Further, if the slaves monitor the master in a round robin fashion i.e. each slave receiving the *alive* messages in sequence, we can say that their duty cycle is given by

Duty Cycle $= t_{awake} / (N * t_{alive})$

The number of slaves can be decided based on the following three factors –

- *Required battery life time* - For example, assuming a 500ms *alive* message period, a 45ms awake time period and a round robin detection system, we expect a duty cycle of 2 %. This would result in a life time for the slaves being about 300 days i.e. close to a year.
- *Theft detection time* - The more the number of slaves, the more will be the detection of theft time, since the next phase requires at least a majority of the slaves decide that the master has been destroyed.
- *Cost of sensor motes* - The cost of the system will be an important factor in limiting this value.

3.5.2. Speed of Detection

This factor also depends on the number of slaves, and the design of the next phase. It is thus discussed in entirety in the next chapter.

3.5.3. Passive Operation

One of the significant features of the Deterministic Independent Sleep Scheduling employed is that there are absolutely no messages transmitted by the slaves. The scheme synchronizes as required, and thus there is no communication needed for time synchronization either. Further, since a fresh timer is started (the synchronization is done) with every received *alive* message, we minimize the problems of clock skew and drift.

This ensures that the slaves cannot be detected by RF signal measurements.

3.6. Implementation of Sleep Mode

There are a number of step that have to be taken to ensure that the sensor node is actually in a sleep mode, and will not consume anything more than the minimal required power to keep the timer running.

The first and probably the easiest way would have been to invoke the TinyOS provided *HPLPowerManagement* module. This is claimed to provide a very low power state. However, our experiments and a careful analysis of the actual source code of this module, show that it is not a very good approach. The primary reason for this is a long list of checks that the module performs before it actually shuts down components. This has two problems. Firstly, we found that a lot of times these checks fail. For example, if there is a message received on the radio, which is not even intended for that particular mote, the radio will not shut down. Secondly, these checks cause a delay which also wastes power.

Hence, we decide to write our own code to put the mote into sleep mode. This involves the following components –

- Firstly, we set the radio to the lowest power possible, to minimize leakage current using the command *call CC1000Control.SetRFPower()*
- Secondly, we power down the radio altogether using *call CC1000StdControl.stop()*
- Thirdly, we disable the ADC using *cdi(ADCSRA, ADEN)*
- Fourthly, we disable the analog comparator using *sbi(ACSR, ACD)*
- Fifthly, we put the CPU into low power state, using a series of port configurations
- Finally, we enable timer interrupts to ensure waking up using *sei()*

These steps enable us to put the mote truly into a low power sleep mode. We carried out some experiments to confirm that the current drawn was actually minimal and thus power used was minimal.

Chapter 4

Voting among Slave Sensors and Theft Reporting

This chapter details the protocol that the slave sensors follow after a theft has been detected or the master sensor has been destroyed. There are three main parts of the protocol: firstly, switching to a secret channel, secondly, voting to prevent false positives and thirdly, theft reporting. The following three sections describe each of these parts.

4.1. Switching to a secret channel

The slave sensors are normally in a completely passive state, sending out no packets, but only receiving the *alive* messages from the master sensor to monitor it. This is essential for a number of reasons such as conserving battery lifetime and to prevent the thief from locating and destroying them using RF means. However, once the slave sensor detects that the master has been destroyed or that the car is being stolen, it needs to communicate with other slaves. This communication has two major functions –

- *Preventing false positives* - It is possible that for some reason one of the slaves does not receive the *alive* messages. In this case if it starts reporting theft then it would be a false positive. Hence, we need to employ a voting mechanism to ensure that a minimum quorum of slaves have detected theft / destruction of master and only then start reporting theft.
- *Intelligent Theft Reporting* - All the slaves should not start reporting theft at the same time. They must employ a scheduled and sophisticated reporting system, which is

detailed in Section 4.3. Towards establishing this, it is necessary for them to communicate amongst themselves.

All this communication is carried out on a secret channel. The standard communication channel will not necessarily be a secret. Thus, it is conceivable that a thief can eavesdrop on it or jam it. To add a further layer of security and protection, we design the first step of the protocol to switch to a secret channel.

The MICA 2 uses the CC1000 radio. This radio uses a digital frequency synthesizer to select a particular send/receive channel. Specific control registers are programmed with values according to the channel and FSK separation used. Because of the nature of the synthesizer, it is only capable of reproducing discrete frequencies in the operating range of the device. The manufacturer (ChipCon – now Texas Instruments) provides tools to determine these optimal frequencies and their associated control values, but they are unintuitive and cumbersome to use. The TinyOS stack and our related tools take the guesswork out of tuning the CC1000 for the MICA2 mote. The stack automatically computes the nearest channel for a given frequency and programs the necessary register values (manual tuning). By default however, it uses pre-determined values from a preset table (preset tuning).

Specifically, this dynamic switching to a secret channel is carried out using the CC1000Control interface. The command we use for the channel switching is *call CC1000Control.TuneManual(uint32_t DesiredFreq).* This method is the preferred method of tuning the MICA 2 radio. The control path function *CC1000Control.TuneManual()* takes a desired frequency in Hz, computes the optimal achievable frequency, determines the necessary control register values, programs the CC1000 and calibrates the device. It returns the frequency, in Hz, of the actual channel.

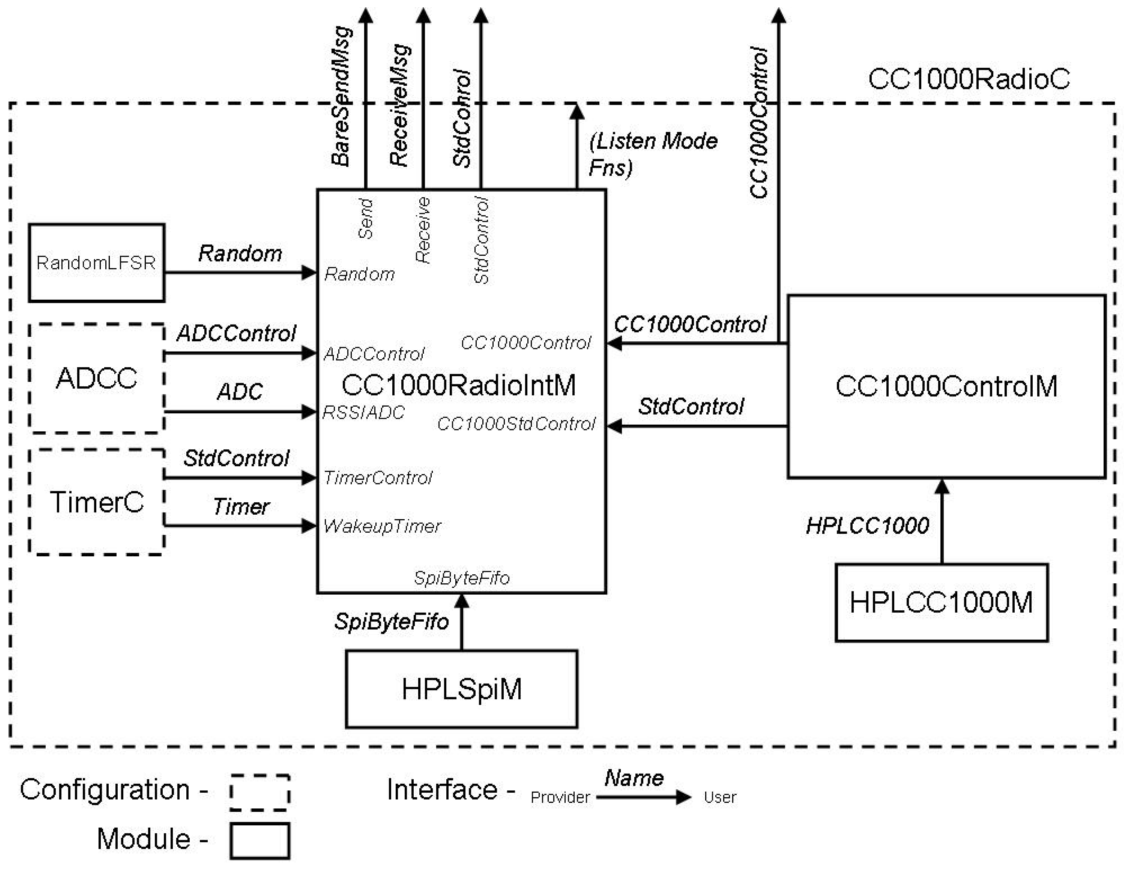


Figure **4-1**: Software stack of the CC1000 Radio

The figure (Figure 4.2) shows the component structure of the CC1000 radio stack. We use the control component, CC1000Control to alter the channels of operation.

Thus, as soon as the slave detects that the master has been destroyed (see Chapter 2) it switches to the secret channel. The other important question here is how the slaves know this secret channel. This is easily addressed by the following two different schemes.

- In a simple scenario, we conceptualize that this secret channel ID is built into the slave sensor motes which are installed in the car.
- The other and somewhat better scheme, which we employ, is that the slaves can decide this on the fly similar to establishing a session key in secure communications.

4.2. Neighbor Discovery and Voting among Slave Sensors

The slave sensors monitor the master sensor by keeping track of its *alive* messages. If a particular slave does not receive these for a certain period of time, it determines that the master has been destroyed. In such a system, it is conceivable that some kind of obstruction or interference might lead to the loss of these messages for a certain slave. In such a case, if this slave starts sending out alarms, in the form of theft report messages, it will be false positive.

False positives are a major flaw in today's alarm systems. It is a common occurrence to see cars sounding their loud alarms because of some one accidentally pushing them or in some cases cars coming too close. It has been observed that the alarms even go off sometimes because of a strong wind. If this occurs often enough, the user tends to switch off the system, which ultimately compromises security.

Thus, to prevent any false positives in our system, we implement a co-operative voting system. The theft reporting phase is entered only if a certain quorum of slaves determines the master to be stolen. For this to be successful, the first step is neighbor discovery. We describe this part of the protocol and the voting part in the next two sections.

4.2.1. Neighbor Discovery among Slaves

As described in Section 4.1, the slaves switch to a secret channel when they determine that the master has been destroyed. After this, they must carry out two major tasks. The first is to determine if at least a quorum number of slaves have also found the master to be destroyed or not. The second is to establish a schedule among them to report theft. The reasons for a schedule and the exact schedule are described in Section 4.3 which details the theft reporting. For now it suffices to say that this functionally imposes the requirement that each slave needs to know the other slaves, as well as the fact that the others know it.

We outline the main goals for this part of the protocol as follows –

- Each slave should discover all the other slaves in the secret channel
- Each slave should know that all the others have discovered it
- The time and number of messages required should be low

We design two protocols for this phase of neighbor discovery. The first is a basic scheme which is not reliable, while the second one adds reliability and some guarantees to the process.

The basic discovery protocol involves each slave sending out hello messages as soon as it switches to the secret channel. The receipt of these hello messages leads to the creation of a neighbor list. Thus, on every received hello message, the slave checks if the node ID of the sender is already on its list. If yes, it silently drops the message. If no, it adds the new slave as its neighbor. The important issue here is when to stop sending out hello messages. We experiment with two schemes.

- The slave stops sending out hello messages when it receives them from everyone else.
- An experimentally determined time later, the slave stops sending out hello messages.

The first scheme here, though somewhat intuitive, does not work very well. The second scheme works really well, but it does not provide any guarantees or reliability.

Towards making this a better system, we implement a reliable neighbor discovery algorithm. The key addition to the basic scheme is the sending of an acknowledgement packet for every received hello message. The first part of the system remains identical to the basic scheme i.e. slaves broadcast out hello messages and the receipt of these helps them create a neighbor list. However, the following two actions are added to their functions.

- On receipt of every hello message, the slave should send an acknowledgement packet back to the sender.

- Secondly, it should record the IDs of the senders of every received acknowledgement message.

The slaves should only cease the broadcasting of hello messages once it has received the acknowledgements from all the other slaves in the secret channel. This discovery mechanism guarantees the following

- Every slave populates its own neighbor list, by receiving hello messages from all others.
- Each slave knows that everyone has the same list, and this is used in further processing to decide the theft reporting schedules.
- If slave A does not receive an acknowledgement from slave B, it does not stop sending hellos. Thus, this protocol guards against unidirectional links also, and ensures the complete discovery.

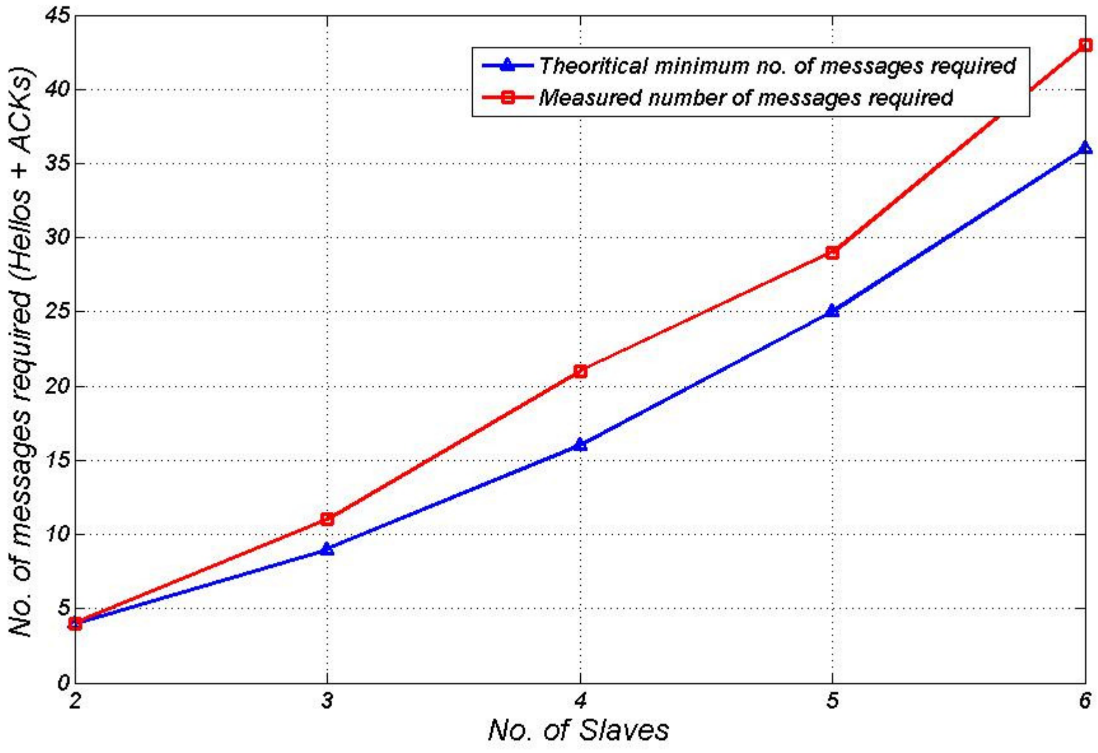


Figure **4-2**: Number of messages required to complete advanced discovery protocol

Figure 4-2 shows the number of messages required by the advanced discovery protocol, to complete the discovery, and ensure that all the slaves have found each other, and know that every other slave has found itself. We first plot the number of messages that is the theoretical minimum, and then plot the actual measured number of messages required as well. The curves prove that our scheme does not exceed the theoretical minimum by a significant number. This ensures that the overhead incurred is not very high and that the scheme is perfectly viable.

It is necessary to note that there are significant synchronization issues in the implementation of this protocol. These mainly include cases where messages arrive one after the other, within a very short space of time. In this case, the receipt of a message is an ‘event’, which preempts all other functions temporarily in the TinyOS concurrency model. This might result in shared global variables changing and the slaves having corrupted neighbor lists and counts. The main feature we use to prevent any of these problems is the *atomic* keyword. In conventional system, *atomic* implies that the task will complete without getting interrupted. However, we observe that is not implemented strictly in TinyOS. A modified version is employed, in which all tasks which get preempted, are resumed from the very same place, with exact same context for shared variables as well. We also use a buffer for incoming messages, which prevents their dropping.

4.2.2. Voting among the Slaves

As discussed before, voting before reporting of theft is essential to prevent false positives. We observe that in our system, a slave only switches to the secret channel if it has detected that the master has been destroyed. Further, once the slave switches to the channel the neighbor discovery protocol discussed above ensures that they are aware of each others presence, and have a common and complete neighbor list.

These steps above can be used directly as a vote. In essence, the presence of a slave in the secret channel itself indicates that it has also detected theft or destruction of the master sensor. Thus, if the neighbor list count is above the quorum, the slave sensors can easily determine that the vote is in favor of the theft or master's destruction event. On the other hand if the slave sensor does not find enough neighbors then it will not enter the theft reporting phase, and it will be a prevented false positive.

4.3. Intelligent Theft Reporting

This section details the protocols for the theft reporting phase. This phase is entered once it has been decided by the slaves that the master has indeed been destroyed and that the vehicle is being stolen. The slaves in this case have the basic responsibility of transmitting theft report messages to the Roadside Access Points. While designing this part of the protocol, we need to keep in mind the following concerns.

- *High success rate* – The probability of each message reaching the roadside AP must be high i.e. the success rate of theft reporting must be high.
- *Minimum number of messages* – A minimum number of messages must be transmitted by each slave. This is important for two reasons: firstly, to prevent the revealing of their location to the thief based on RF methods, and secondly, to conserve battery power.
- *Scheduled theft reporting* – All the slaves should not transmit the messages at the same time. This will not only cause collisions, but also consume more and unnecessary power from their batteries.

We design and implement two schemes, one being a basic scheme and the other an advanced heuristics based scheme. Each of these is described in the following two functions.

4.3.1. Basic Scheme

We envision roadside APs to be at major intersections throughout the city. These access points typically broadcast a beacon message periodically, to announce their presence. The basic scheme dictates that the slave sensors should send out their theft report as soon as they hear a beacon message.

The slave thus sends out only a single message and based only on this, it is very difficult for the thief to detect their presence using RF methods. However, this theft report is not guaranteed to reach the AP. We expect problems such as bad RF conditions, obstructions, interference or collision of the packet with another packet.

To improve on this, we extend this scheme to allow the slave to send a limited threshold number of packets. This increases chances of the theft report reaching the AP, but increases the risk of detection. We determine this threshold experimentally, but recognize that this will have to be tuned depending on a lot of factors such as density of APs, power considerations, RF channel occupancy, etc. One way to set this is to let the slave send as many messages as it an in the in-touch time. The in-touch time is the time the AP is within range of the slave. This time is found to be about 10 seconds at a low speed (around 15mph) of the vehicle. Thus, if the average interval of retransmission time is about 2 seconds, we can set the threshold number of messages to 5.

The basic scheme, even with the improvement, has a major shortcoming in terms of the thief being able to find the location of the slave based on RF techniques, because a fixed number of messages are being transmitted on a periodic average. Towards addressing this issue, we develop the advanced heuristics based scheme which is described next.

4.3.2. Advanced Heuristics based Scheme

Keeping in mind the main goals, we implement two techniques for reliable theft report delivery and minimal messages transmitted. The first involves estimating the wireless channel and based on that sending one message which gets through. The second involves a handshake procedure, in which the slaves acknowledge the APs presence and vice versa. Both of these are described below.

The first technique is based on heuristics will determine the most suitable time to send the theft report, based on the RF signal area being good. We ensure that every beacon sent by the AP has a sequence number. The slave keeps track of the sequence numbers and thus knows how many beacons it has missed. Missing none would imply a good RF signal area, and the slave would immediately send out the theft report. However, if a high number of beacons are missed it implies a bad signal area and thus the slave ought to wait before sending the theft report.

The second technique is based on an adaptive handshake procedure. To help achieve a more reliable delivery, the roadside AP sends an acknowledgement packet to the reporting slave sensor called a *theft ACK*. The reporting slave only stops sending theft messages when it receives this acknowledgement.

In case of loss of the theft report due to interference or collisions, the reporting slave keeps a timer running after sending the theft report message. If this timer expires without receiving a theft ACK, the slave decides that the theft report was lost, and thus retransmits it. The reporting slave has to try to retransmit till it receives a theft ACK, or till it stops receiving beacons from the AP, in which case it is clear that it is no longer in range. This is implemented using the sequence numbers of the beacon. The slave maintains a record of the highest sequence number received. Every time the timer mentioned above expires, the slave first checks if the last received sequence number is higher than the recorded highest i.e. it checks if in the timer duration, whether it has received a new beacon or not. Based on this, it can decide if it still within range of that particular AP or not. Naturally, this timer is stopped when a theft ACK is received.

To minimize the number of theft reports sent, we design an adaptive protocol on the side of the AP. It is clear from the above discussion that the slave will keep sending theft reports till it receives a theft ACK. This implies that the AP should keep receiving duplicate theft report messages while the slave is still in range. To prevent this, the AP will progressively increase its theft ACK sending frequency if and as it keeps receiving further duplicate theft reports. This is implemented using the beacon interval as a reference. The AP will start by sending one theft ACK per beacon interval. If it receives duplicate theft report messages, it will conclude that the theft ACK was lost. Then, it will increase it frequency of sending theft ACKs to two per beacon interval, and so on. In essence, it increases the rate of sending with every duplicate theft report message received. This increases the chances of receipt of a theft ACK by the slaves, thus reducing the number of theft report messages required adaptively. This also helps in achieving the required functionality in poor radio channel conditions.

Chapter 5

Interfacing with the Roadside Access Point

5.1. Introduction

This chapter details the interfacing of the slave sensor mote, which is trying to report theft, with roadside access point (AP). This is a key component of the entire system, since the theft reporting is infeasible without demonstrating that a mote can communicate with a device similar to a router (used as an AP).

For our prototype of the system we use the MICA 2 motes as representative of the slave motes, and a Linksys WRT54G [10] router as a representative of an AP. These choices are based on the current generality of use and popularity of these devices. We believe that they are fairly good representatives of their particular classes of hardware. Further, they are the cheapest products in their classes, and this system being one of the largest deployments of sensor networks and APs, it has to be cheap.

We also utilize the MIB510 programming board and the Stargate Netbridge Gateway [27]. These components can be done away in the finally deployed system, but help us in building and analyzing our prototype.

5.2. Overview of Hardware

5.2.1. The Access Point – Linksys WRT54G

The Linksys WRT54G is a Wi-Fi router, mainly intended as a residential gateway. It uses 802.11 b/g and also has ports for 802.3 Ethernet. Released in December 2002, it has 4 port network switch build in and two detachable antennas. Its typical maximum range of communication is 200 feet, and being the cheapest (less than 50 dollars), is a good candidate for a roadside access point.

Figure **5-1**: The Cisco Linksys WRT54G Router

A key feature of this router is that its firmware source code has been released under the GPU GPL, and thus can be modified to suit requirements or replaced altogether. We implement the latter option i.e. we replace the original firmware with the DD-WRT [28] solution. This is a free Linux (OpenWRT project) based firmware and allows many enhancements and modifications.

Figure **5-2**: The Crossbow Stargate Netbridge Gateway

5.2.2. Stargate Netbridge Gateway

The Stargate Netbridge Gateway is a Crossbow product, designed as a bridge between the mote tier of devices and the Internet at large. It is a modified Cisco NSLU2 router, with Crossbow software (free and open source) installed on it. It features one Ethernet port and two USB 2.0 ports. The device is further equipped with 8MB of program flash, 32MB of RAM and uses one of the USB ports to plug in a system disk. It runs the Debian Linux (2.6.18 kernel) operating system. A photograph of the Netbridge Gateway is show in Figure 5-2.

5.3. Setup of Interfacing

Figure 5-3 shows the block diagram of the interfacing part of the system.

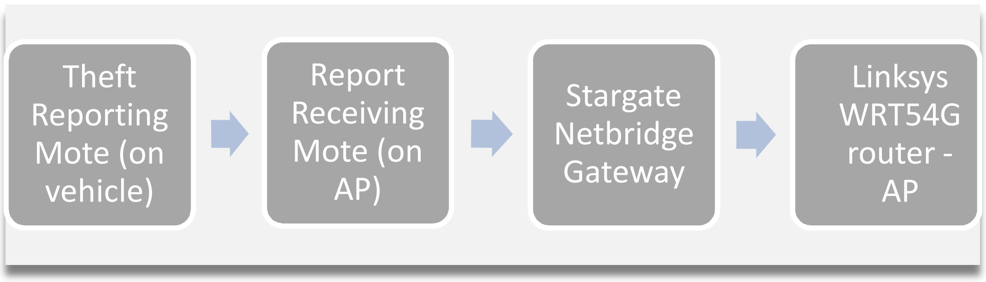

Theft Reporting Mote (on vehicle)
Report Receiving Mote (on AP)
Stargate Netbridge Gateway
Linksys WRT54G router - AP

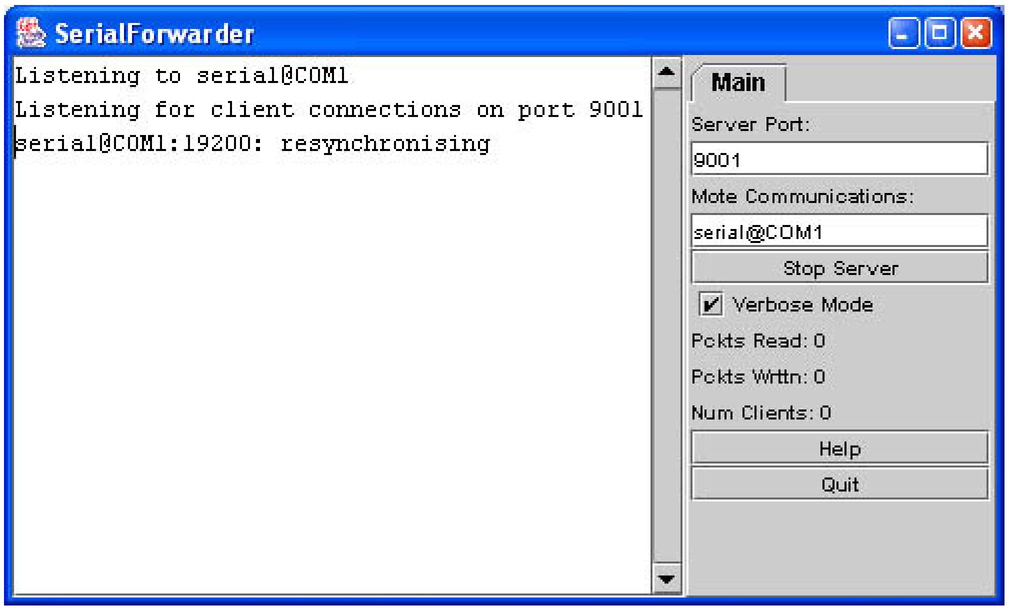
Figure **5-3**: Block Diagram of the Interfacing

The first step is connecting a MICA 2 mote to a MIB510 programming board. This mote serves as the "base station" to the slave motes or the receiver of the tracking messages. Its function is simply to receive the tracking messages, and forward them through the MIB510 programming board to the next block i.e. the Stargate Netbridge.

The Stargate Netbridge receives the tracking messages from the mote on the MIB510, on one of its USB ports. We use a serial to USB converter cable (PL2303) which helps connect the USB port on the Netbridge to the serial port of the MIB510. We installed the Serial Forwarder program on the Netbridge, which is available as part of the TinyOS package. This program is to receive packets on a given port, and forward them (as raw or parsed) to a specified port.

The Netbridge's Serial Forwarder thus forwards packets to port 9001, which is read by the AP. The AP is configured to simply read this port and forward the data unchanged, via TCP/IP protocol, to the PC connected. For analysis purposes, we have connected a PC, but in the actual scenario we need not use this since the purpose is to send the data to the roadside AP only. The PC itself runs Serial Forwarder and reads packets coming in from the router and then takes action depending on the messages coming in.

5.4. Serial Forwarder Documentation



```
Listening to serial@COM1
Listening for client connections on port 9001
serial@COM1:19200: resynchronising
```

Server Port: 9001
Mote Communications: serial@COM1
[✓] Verbose Mode
Pckts Read: 0
Pckts Wrttn: 0
Num Clients: 0

Figure **5-4**: Screenshot of the Serial Forwarder GUI

The figure above shows a screen capture of the Serial Forwarder GUI. This code is written in Java and is loosely based on the PPP in HDLC like framing described in RFC 1662. In fact it uses the same framing and escape sequences, and a 16 bit CRC. The protocol always assumes Address-and-Control field compression. The implementation understands the Protocol IDs in Table 2-1. All other protocols are rejected.

Table **4-1**: Protocols accepted by the Serial Forwarder

ID	VALUE	DESCRIPTION
PROTO_ACK	64	An acknowledgement packet, sent to the host backend. It consists of a one byte payload which is the token of the corresponding packet being acknowledged.
PROTO_PACKET_ACK	65	A data packet, which expects and acknowledgement. The first byte of the payload contains the token which must be sent back with the acknowledgement.
PROTO_PACKET_NOACK	66	A data packet that does not require a subsequent acknowledgement packet.
PROTO_UNKNOWN	255	In case the receiver receives an unknown or unsupported type of packet, it sends this packet.

Chapter 6

System Security

The final component of our system is the security. By security here we mean the defenses of the system against attackers with capabilities such as eavesdropping on communications, trying to decipher the networking protocols and possibly injecting false messages. We address most important problems using a variation of Blundo's scheme [8] for key distribution, and then encryption and decryption of all messages based on these keys. We note that our security design is not comprehensive i.e. it is not resilient to very sophisticated attacks, but is sufficient to repel most commonly conceived ones. We discuss these attacks and possible solutions in the last part of this chapter as future work.

First we outline our key distribution mechanism. The following four steps clearly show our implementation of Blundo's scheme –

1. The server randomly chooses a symmetric polynomial $P(x_1, x_2 \dots x_t)$ in t variables of degree k over a predetermined GF(q).
2. To each $User_i$ the server gives the polynomial $f_i(x_2 \dots x_t) = P(i, x_2, x_3 \dots x_t)$, that is the polynomial obtained by evaluating $P(x_1, x_2 \dots x_t)$ at $x_1 = i$.
3. If the users $User_{j1}$ to $User_{jt}$ want to setup a conference key then each user $User_{ji}$ evaluates $f_{ji}(x_2, x_3 \dots x_t)$ at $(x_2, x_3 \dots x_t) = (j_1, j_2 \dots j_{i-1}, j_{i+1} \dots j_t)$.
4. The conference key for this group then is $P(j_1, j_2 \dots j_t)$.

We use this mechanism in two ways. Firstly, we use it to establish a group key between the master and the slaves. This key is used for all communications between the master and slaves and for all communications between the slaves as well. Secondly, we implement a special case of this general group key distribution protocol as a pair-wise key establishment protocol, between the theft reporting slaves and the roadside APs.

The next part of the security design is the actual usage of these keys to encrypt and decrypt the messages being sent and received. We use the TinySec protocol [27] to apply link layer encryption and decryption. The TinySec protocol is designed as a user friendly API, and thus it is easy to supply the keys we have generated using Blundo's scheme as an input to it. Specifically, the Rivest Cipher (RC5) encryption mechanism is used.

To analyze the design of the security system we list and discuss the four main issues addressed as follows.

- *Message secrecy* – This is clearly achieved since each message is encrypted with a group key or pair-wise key, depending on the parties involved in the communication. No eavesdropper can receive the message, and then decipher any information from it. Further, since all the messages are the same length after encryption, it is not even possible to judge the type of messages from their lengths. Our scheme gives complete and perfectly secure communications in this aspect.
- *Message integrity* – We achieve a reasonable level of security as far as this parameter is considered. The nodes decide on a particular field which it requires to have a certain value. On decryption of the message, if the node finds that field intact and as expected, only then does it accept the message as not having been tampered with. Since the messages are encrypted by secure keys, tampering it is not an easy task for an attacker in the first place. Further standard link layer checks such as CRC ensure protection against corruption of messages. If a higher level of security is desired, primarily from attackers who are capable of compromising the network from inside, we could use a hashing of the message and include that at the end of each message. Since hashing is a one way operation, if the message is tampered with, the hash will not match the one originally sent. However, we leave this for future work, since addressing all these issues will require a separate design to ensure that overheads do not affect the system drastically.

- *Authentication* – The keys established also ensure that the messages are coming from the desired senders. Further, we expect that the sensors will be issued by a central trusted key distribution authority, which will ensure that they are issued to trusted members of the community whose records have been verified and so on.

These safeguards are enough against most attacks. However, more sophisticated attacks such as replay attacks can still be mounted. Against this, we need to incorporate *nonces* or counters which grow with each message. However, it is important to note that this needs to be done within the severe memory constraints of the mote and the limited part of the RAM it can devote to the neighbor tables and associated data structures. There are also other internal node compromises which need to be dealt with, and we are designing solutions for currently.

Chapter 7

Conclusions and Future work

This final chapter outlines the conclusions and details the future work to be done. It also highlights some limitations of the system which need to be worked upon. This work extends the capabilities of the Sensor network based Vehicle Anti-theft System (SVATS) to add a novel tracking system, which enables locating and tracking the movements of the stolen vehicle through the city. We address technical issues related to this application and have built a complete working prototype of the system.

Firstly, a new vehicle theft detection mechanism was developed. This motion sensor based system was designed and implemented, and shown to over come some significant inefficiencies and drawbacks of the earlier designed RSSI based scheme. The highlights of this section of the system are a 100% detection rate, an almost instant detection, a 0% false positive rate, and no communication or processing overheads as compared to the RSSI based scheme. Further, the 'tuning' or parameter setting phase is much easier because it involves only a single reliable source of readings, instead of the RSSI based scheme which relies on a number of neighbors and the reading of the signal strengths by them.

Secondly, we recognize that since the master sensor is connected to the vehicle's power source, it is possible that the thief might destroy it. To address this situation, we design a slave sensor tier, which monitors the master sensor. The efficient and reliable design of the slave sensor system forms an important part of this project. Since the slave sensors are not powered by the vehicle, but are low power battery driven hidden sensors, they need an energy efficient sleep schedule. We survey commonly used methods, and due to limitations in each, we design a specific Deterministic Independent Sleep Scheduling on our own. Based on our implementation and experiments, we optimize our system to consume the minimum energy and analyze the performance of our system with respect to various parameters.

Thirdly, a discovery and voting system is enabled, whereby after a theft is detected the slaves switch to pre-decided secret channel and vote amongst themselves to reduce any chances of false positives. If the quorum number of votes is not received, the alarm is considered false. The switching to secret channel is done dynamically and adds an extra layer of security, especially from jamming and eavesdropping.

Fourthly, an intelligent theft reporting schedule is developed. We first propose and a basic scheme and then an advanced heuristics based scheme to enable reliable and quick theft report delivery to the access point, while minimizing the number of messages required. Fifthly, we interface the roadside AP to our mote tier and the implementation details, both of software and hardware, are detailed earlier. Lastly, we develop the basic security primitives of the system which ensure authentication, integrity and secrecy.

We evaluate the relevant parts of the system and present the results at the end of each chapter. Overall, our system works very well and reliably, and serves a cheap, difficult to disable and sophisticated stolen vehicle tracking system.

As part of future work, we are working a number of issues.

- We are making the system more reliable and robust by upgrading the entire system to the MICA Z mote platform which has a better and more powerful radio. This involves porting the code to this new platform and re-testing the range and motion sensor calibration.
- Next, we are working on analyzing the various possible attacks and developing security measures against them. These would include Denial of Service attacks, Replay attacks, node compromise scenarios and user privacy issues.

- Finally, intend to deploy a large scale sensor network, running both SVATS and our tracking system. This would involve considerable experimental design, and execution. Also, we intend to collect valuable data about failures, performance issues and scalability problems.

References

[1]. Hui Song, Sencun Zhu and Guohong Cao – "SVATS: A Sensor network based Vehicle Anti-Theft System" – *IEEE INFOCOMM, April 2008*.

[2]. Insurance Information Institute – "Updates on Auto Thefts, May 2007" - http://www.iii.org/media/hottopics/insurance/test4/

[3]. J. Kahn, R. Katz and K. Pister – "Next century challenges: mobile networking for smart dust" – *MobiCom 1999*.

[4]. V. Bychkovsky, B. Hull, A. Miu, H. Balakrishnan, and S. Madden - "A measurement study of vehicular internet access using in situ wi-fi networks" - *ACM Mobicom*, 2006.

[5]. B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden - "Cartel: A distributed mobile sensor computing system" - *ACM Sensys*, 2006.

[6]. D. Jiang, V. Taliwal, A. Meier, and W. Holfelder - "Design of 5.9 GHz DSRC-Based Vehicular Safety Communication" - *IEEE Wireless Communications Magazine*, October 2006.

[7]. J. Zhao and G. Cao - "VADD: Vehicle-Assisted Data Delivery in Vehicular Ad Hoc Networks" - *IEEE INFOCOM*, April 2006.

[8]. C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro and M. Yung - "Perfectly-Secure Key Distribution for Dynamic Conferences" - *Lecture Notes in Computer Science*, vol. 740, pp. 471–486, 1993.

[9]. Crossbow MICA 2 mote – datasheet available at www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf

[10]. Linksys by Cisco WRT54G Router – data sheet available at http://www.linksysbycisco.com/US/en/support/WRT54G/

[11]. Analog Devices ADXL202 Accelerometer – data sheet available at http://www.analog.com/en/mems-and-sensors/imems-accelerometers/adxl202/

[12]. L. Grajales and I.V. Nicolescu - "Wearable Multisensor Heart Rate Monitor" – *IEEE BSN 2006*

[13]. US Patent 6950011 - Vehicle theft detection device and method

[14]. Anti-theft alarm device for vehicle – Patent number 4885572

[15]. Crossbow MTS310 Sensor Board – data sheet available at http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0047-01_B_MTS.pdf

[16]. V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh - "Simulating the power consumption of large-scale sensor network applications" - *SenSys, 2004*.

[17]. G. Xing, C. Lu, R. Pless, and J. A. O'Sullivan – "CoGrid: an efficient coverage maintenance protocol for distributed sensor networks" - *Proceedings of IPSN 2004,* April 2004

[18]. D. Tian and N. D. Georganas – "A coverage-preserving node scheduling scheme for large wireless sensor networks" *–ACM WSNA 2002*

[19]. P. Berman, G. Calinescu, C.Shah, and A. Zelikovsly – "Efficient energy management in sensor networks" - *Ad Hoc and Sensor Networks*, *Nova Science Publishers, 2005*.

[20]. F. Ye, G. Zhong, J. Cheng, S. Lu, and L. Zhang – "PEAS: A robust energy conserving protocol for long-lived sensor networks" - *Proceedings of the 23rd International Conference on Distributed Computing Systems(ICDCS '03),* 2003

[21]. A. Cerpa and D. Estrin – "ASCENT: Adaptive self-configuring sensor networks topologies" - *IEEE INFOCOM*, *June 2002.*

[22]. K. Wu, Y. Gao, F. Li, and Y. Xiao – "Lightweight deployment-aware scheduling for wireless sensor networks" - *ACM/Kluwer Mobile Networks and Applications (MONET) Special Issue on"Energy Constraints and Lifetime Performance in Wireless Sensor Networks" December 2005.*

[23]. S. Kumar, T. H. Lai, and J. Balogh - "On k-coverage in a mostly sleeping sensor network" - M*obicom 2004*

[24]. V.P. Sadaphal and B.N. Jain – "Random and Periodic Sleep Schedules for Target Detection in Sensor Networks" – *Journal of Computer Science and Technology, May 2008*

[25]. L. Gu and J. A. Stankovic - "Radio-triggered wake-up for wireless sensor networks" - *Real-Time System*, *2005.*

[26]. Crossbow Stargate Netbridge – data sheet available at - *http://www.xbow.com/Products/productdetails.aspx?sid=275*

[27]. C. Karlof, N. Sastry and D. Wagner – "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks" – *SenSys 2004.*