Security and Deployment Issues in a Sensor Network

Mike Chen
mikechen@cs.berkeley.edu

Weidong Cui
wdc@eecs.berkeley.edu

Victor Wen
vwen@cs.berkeley.edu

Alec Woo
awoo@cs.berkeley.edu

11 December, 2000

Abstract

We are facing an growing demand in deploying large scale sensor networks in the real world. However, there are still many obstacles before we can proceed. Security is a major concern as we need to be sure that the data we receive is authentic and confidential, and have not been tampered with. This is especially difficult because sensors have very limited resources and we cannot guarantee physical security of the sensors. Limited bandwidth is another problem as we scale up the number of sensors. We need to ensure that we achieve fairness and do not starve any sensors. We present our design and implementation to address these issues, and describe our deployment of a real sensor network and the problems that we encountered. Our security protocol works well on sensors with 8KB of memory and 4MHz 8-bit Atmel processors. Our adaptive transmission scheme achieves fairness among sensors in the network and prevents starvation. We also implemented two applications: people tracking and light sensing.

1 Introduction

There are several obstacles that impede the successful deployment of large-scale sensor networks in the real world, especially for sensors that have extremely limited resources.

Security If we want to trust sensor data that we receive, we need to be able to authenticate the source so that bogus and malicious sensors can not inject false data. We also need to be able to verify data integrity to detect data modification. In addition, if we want the data to be confidential, we need to employ techniques such as encryption so no one else will be able to read the data. Current security protocols do not extend well to PDAs such as Palm Pilots, let along sensors which have orders of magnitude less resources. Our goal is to design an efficient protocol that provide authentication and confidentiality under the constraints of code size, CPU, and memory size.

Network bandwidth When scaling up the number of sensors, it is necessary to use network bandwidth efficiently and fairly. Techniques like aggregation in the network and compression enable us to get as much data as possible. Fairness is also important as we want to collect data from all sensors and avoid starvation. It is especially important for sensors near the base stations, where much network traffic needs to be routed in addition to the sensor readings from those sensors.

Power consumption One metric of the usefulness of a sensor network is its operating lifetime. Obviously, the longer it operates the better. The wireless interface on the sensors in particular consumes a relatively large amount of energy. We can minimize its power consumption by lowering its duty cycle and perform adaptive power tuning of the radio.

In this paper, we present our approaches to address these three issues, and discuss our experience in deploying a 15-mote sensor network and building two sensor applications: people tracking and light sensing. We focus on sensors that have extremely limited resources: small amount of memory, low computation capability, poor bandwidth, and limited energy resources. The sensor platform we has 8KB of flash memory, a 4MHz 8-bit Atmel processor, and a 900MHz radio interface.

In Section 2, we describe our security protocol andits performance on resource-limited sensors. In Section 3, we present our adaptive networking algorithm. In Section 4, we discuss the design and implementation of our people tracking application and the light sens-

ing application. In Section 5, we discuss future work that would further simplify deployment. In Section 6, we state our conclusions.

2 Security

2.1 Motivations

Given that tiny sensors are supposed to be commodity devices, it is potentially easy for malicious attackers to overlay foreign sensor network on top of existing one. If there is no authentication or confidentiality in communication between motes and base stations, then there is no mechanism to protect legitimate network from being taken over by hostile one (e.g. a hostile base station could broadcast a better route thus legitimate sensor nodes will send their data to the malicious base station).

To secure legitimate sensor network, we propose a base station-to-mote confidentiality and authentication protocol, and source authentication protocol adapted from TESLA [9].

2.2 Protocols

These protocols are based on share-key cryptography algorithm. It would have been more convenient to bootstrap with public-key algorithm for symmetric key setup. However, due to hardware constraint of our sensor nodes, we chose shared-key cryptography because it is much much less computational inexpensive. The choice of alogrithm (we chose RC5) and its implementation is also strongly influenced by the hardware constraints of the tiny motes.

2.2.1 Base Station To Mote Confidentiality and Authentication

Authentication The base station-to-mote authentication protocol hinges on an 8-byte Message Authentication Code (MAC) included in every packet mote sends to the base station. The MAC is calculated based on RC5's encrypting function and not easily reversed. Since, only $mote_i$ and the base station share the secret key for $mote_i$, the base station can verify that the message is authenticate from $mote_i$ by calculating the MAC of the message and compare it against with the MAC in the packet. Given this, the base station can verify that $mote_i$ is the actual originator of the message. This protocol gives the base station some assurance that the message comes from where it claims. Furthermore, MAC also doubly functions as Cyclic Redundancy Code (CRC). If MAC check fails, then either the packet was sent maliciously or it suffered bit errors in the channel. In either case, the payload should not be trusted. In our design, the RC5 module will pass up the packet to application with error flag bit set. The application will then decide whether to accept the data in the payload or not.

Confidentiality By running RC5 in output-feedback mode (OFB), we can also achieve base station-to-mote confidentiality of communication. Under this scheme, the mote uses its secret key and some initialization vector (IV) to calculate a pad. The plaintext is then XOR'ed with the pad to produce the ciphertext. One nice feature of OFB is that since it is stream cipher in nature, the ciphertext is the same size as plaintext, not some multiple of block size. This property is nice since the bandwidth is constrained and mote should send as much useful data in a packet as possible.

For every packet sent to base station, the actual payload is encrypted (if the application so wishs). This encrypted payload, along with application handler ID, sequence number and source ID are then MAC'ed to provide authentication of the message. The *confidentiality* and *authentication* between base station and mote establishes a secure communication channel between mote and base station. The protocol for secure communication channel shown below:

$$
\begin{aligned}
X &= \{payload, seqno\}_{K_{M\_BS}} \\
M \rightarrow BS &: X(src, dst, AMhandler, X)_{K_{M\_BS}} \\
(\ldots) &: MAC \\
\{\ldots\} &: Encryption
\end{aligned}
$$

2.2.2 TESLA: Source Authentication Protocol

Overview The basic idea is that the sender first generates a sequence of secret keys, $\{K_j\}$ where each key K_j is an element of a hash chain. By successively applying a one-way function g (e.g. a cryptographic hash function such as MD5 [11]) to a randomly selected seed, K_N, we can obtain a chain of keys, $K_j = g(K_{j+1})$. Because g is a one-way function, anybody can compute forward (backward in time), e.g. compute $K_0, \ldots, K_j$ given K_{j+1}, but nobody can

compute backward (forward in time), e.g. compute K_{j+1} given only $K_0, \ldots, K_j$, due to the one-way generator function. This is similar to the S/Key one-time password system [2].

The time is divided into intervals. Each sender then associates its key sequence with the sequence of the time interval, with one key per time interval. In time interval t, the sender uses the key of the current interval, K_t, to MAC packets in that interval. The sender will then reveal the key K_t after a delay of δ_r after the end of the time interval t. The key disclosure time delay δ_r is on the order of a few time intervals, as long as it is greater than any reasonable round trip time plus the maximum synchronization error between the sender and the receivers.

When the receiver receives the packets with the MAC, it saves the arrival time for each packet. Note that the receiver and the sender need an approximate time synchronization. For the purpose of this approach, the time only needs to be loosely synchronized, e.g. the synchronization error may be on the order of multiple seconds. This level of approximate time synchronization is easy to achieve in practice. Please refer to [4, 9] for more details on time synchronization.

Then after the sender discloses its key for interval i, the receiver can then authenticate previous saved packets from the same interval.

Issues There are several issues in implementing Tesla on motes.

- How do sender (in this case, the base station) and receiver(s) to share an initial Tesla key?
- Memory is limited on motes for buffering packets.

The first problem can be solved by having the receiver(s) sending a message to base station, asking for the current Tesla key, via the secure channel. Since the base station's reply is encrypted (well it doesn't need to) and authenticated (very important), the receiver(s) can bootstrap into the Tesla group.

The second challenge is harder to tackle. We propose to store the buffered packet to EEPROM on-board. Thus giving us another 512 bytes of storage. Furthermore, we can fine-tune the Tesla disclosure interval, δ_r, so that limited buffering is needed.

2.3 Implementation Challenges and Implications

We chose RC5 as our basic share-key algorithm because of its low memory requirement (both code size and table size) and its relatively high encrypting performance. We looked at and rejected Rijndael (the new AES standard) [6] because it requires 1.4 KB of table for an unoptimized version and 14 KB of table for optimized version. Due to our severely constrained mote hardware, it seems RC5 is a better choice as the cryptography algorithm. Our initial implementation of RC5 (taken from openssl source code [7]) resulted in almost 8 KB of program text size. Thus our mote will be able to perform RC5 and nothing else. The resulting code size is due in part to the 8 bit nature of the Atmel processor and generality in the openssl source [1]. Through careful hand optimizations and replacing some critical functions (particularly the 32-bit rotate function) with hand-coded assembly, we were able to reduce the code size to 1.6 KB and achieve 10x speed increase. The resulting implementation uses 8-byte key (64 bits, which provides good security while relatively inexpensive computationally), and the encryption is limited to 8 rounds only [2].

For encryption, we implemented OFB mode so that only encryption routines are needed for confidentiality. The encryption key and IV are used to produce an encryption pad. This pad and the plaintext are XOR'ed together to produce ciphertext. Since $A \oplus B \oplus B = A$, the receiver can produce the same pad, it can recover A by XORing the ciphertext with the pad.

To perform MAC function, we iteratively encrypts and XORs chunks of plaintext to produce the final MAC. This MAC function also bases on the basic encryption routine, further reusing the code and reduce total code size.

2.4 Performance

2.4.1 Server

The implementation of RC5 on desktop is written in Java. It is a direct translation of the C implementation for Mote, with modifications made for the Java

[1] Many operations in RC5 requires 32-bit quantities. On an 8-bit processor, it would require 4 registers and 4 instructions (at least) for every equivalent 32-bit operation

[2] For details of RC5 encryption algorithm, please refer to [12].

Packet Size (byte)	MAC (MB/s)	Encrypt (MB/s)
8	5.525	4.00
16	11.11	7.69
32	16.67	9.09
64	16.67	10.00
128	20.0	10.00

Table 1: Testing Platform: Pentium III (Coppermine core) 650MHz, 640MB of RAM, Linux 2.2.14, IBM JDK 1.3 with JIT enabled.

Packet Size (byte)	MAC (MB/s)	Encrypt (MB/s)	Key setup (no./s)
15	0.043	0.25	250
16	0.053	0.27	250

Table 2: RC5 operation throughput on Mote.

language. The performance is given in Table 1. Given that a packet is only 30 bytes long, a typical PC can authenticate approximately 0.58 million packets per second. Encryption takes a little bit longer, but the server can still encrypt (or decrypt) 0.35 million packets. Thus we imagine that cryptography operations will not be the scalability bottleneck. Instead, key lookup and key setup and storage of expanded keys (which occupies 72 bytes per key)) will be.

2.4.2 Mote

The performance of RC5 on mote is given in Table 2. Given the radio bandwidth of 10Kbps, the mote can encrypt (or decrypt) and authenticate every message it receives. In fact, the limiting factor is not computation power. Rather it is the memory requirement. The storage of key and buffering of unauthenticated messages (for Tesla) will take up to 200 bytes out of 512 bytes of available RAM (refer to Table 3). Storage is the real constraint in our cryptographic protocols.

Module	RAM size (bytes)
RC5	80
Tesla	120
Encryption/MAC	20

Table 3: Storage requirements for RC5 and Tesla.

3 Networking Support

The ultimate goal of sensor network is to allow backend servers to collect data from sensors, scattered around an open field or inside a building, and process such information to derive interesting applications like location tracking or environmental control. In this regime of computing, information flow rather than information processing is the most critical component. In fact, all sensor network applications will require uses of the network. Unfortunately, the ad-hoc characteristics and the dynamics of wireless connectivity in sensor network impose great obstacles in deploying real applications. Therefore, the operating system should provide a set of adaptive, self configurable network protocols for ease of application development. Work in TinyOS [3] has provided us an unreliable data propagation channel called Active Messages and a simple source based routing framework to create a self-maintaining ad-hoc network.

3.1 Active Messages

An active message simply comprises of the destination of the message, an identifier for the message handler at the destination, and the application specific data unit. A set of error detection and forward correction schemes can be used to combat for interference. Such schemes include 16 bit CRC, voting over redundant data, and single-bit-error-correction double-bit-error-detection-scheme. Data retransmission is currently not supported by the system. In fact, we believe that since traffic in sensor network is real time information from continuous sampling, retransmission is often not necessary and may be energy inefficient.

3.2 Ad hoc Routing

The goal of the ad hoc routing protocol is to let each sensor be capable of discovering a network route to a nearby base station. The current protocol will attempt to form a minimum spanning tree topology with the base station being the root of the tree which periodically broadcasts a one hop route to its nearby neighbors. Neighboring nodes listening for the broadcast will set its route accordingly and can route directly to the base station. Subsequently, these neighboring nodes will append themselves into the route and broadcast a two hop route. Nodes that receive these two hop routes but fail to hear the base station broadcast can route packets through these base station neighbors and they can further extend and propagate these routes with an addition of to their neighbors. These route propagations will eventually flood the entire network and each node will be capa-

ble of establishing a route to the base station given it can hear at least one neighbor. To maintain the minimum spanning tree topology, each node, based on what it can receive, will use route with the minimum number of hops from the base station. Although base station is the origin of the tree, each node has no notion of which base station it is routing to. Therefore, such a scheme can also be deployed in the case of the presence of multiple base stations.

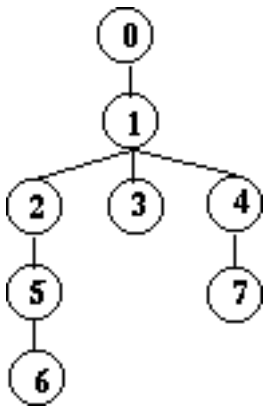

Figure 1: A spanning tree topology in which all nodes are attempting to send packets at a rate of 4 packet/s to node 0 through the routes denoted by the edges in the graph.

3.3 Adaptive Transmission Control

Given we are in a multi-hop network, our goal is to enable global communication from nodes to base station with a hope of achieving some degree of fairness among each node.

A node with a high packet transmission rate will dominant the wireless channel and decrease the chances of routing its children packets which subsequently hinders the probability of its children's packets from reaching the base station. It also creates an unfairness issue in channel allocation among neighboring nodes. Therefore, an adaptive scheme must be employed for a node to control its own transmission rate to allow its neighbors and its children to achieve somehow a fair chance of reaching the base station while maintaining an a moderate level of channel utilization.

The adjustment of transmission rate can be based on the probability of successful transmission signalled by acknowledgements. That is, if transmission is not successful, it is natural to decrease the transmission rate to allow others to send while a successful transmission should reinforce an increase to maximize channel utilization. The adaptation can take a linear increase and multiplicative decrease approach based on whether an acknowledgment is received or not. The upper bound of the transmission rate is the aggregate transmission rate of all applications on a node. Packets that are being routed will not follow this scheme since the goal is to have each node to adapt to the amount of routing traffic. One interesting characteristics of multi-hop network is that the action of routing by the parent acts as an acknowledgement to the child. That is, acknowledgments are free in this regime.

The effectiveness of such an adaptive scheme can be illustrated by simulating the network topology shown in Figure 1. The goal is to allow each node in the network to have a fair chance to communicate to the base station. The adaptive scheme controls the transmission rate by controlling a transmission probability variable. Setting the transmission probability to 0.5 will set the transmission rate to be 50% of the aggregate send rate of all applications. Therefore, by increasing the send probability by an amount α, we can linear increase the send rate. To multiplicative decrease the transmission rate, we simply decrease the send probability by a factor β. With an $\alpha = 0.03$ and $\beta = 2$, we arrive with a simulation result summarized in Table 4.

Node ID	Actual Send Rate	Base Station Receive Rate	% Received By Node 0
	(packet/s)	(packet/s)	%
1	1.725	1.707	99
2	0.935	0.6732	72
3	1.025	0.6765	66
4	1.065	0.6496	61
5	1.28	0.6571	61.7
6	2.84	1.391	49
7	1.355	0.664	49

Table 4: Simulation result for the topology shown in Figure 1

Given the bandwidth of 10kbps in the simulation and a packet size of 34 bytes, a node can send only send at most 11 packets/s after taking acknowledgments into account. Since the application at each node is sending at a rate of 4 packets/s, it is clear that none of the nodes can send to the base station at such a rate. In Table 4, it is clear that the rate control mechanism can successfully adapt the transmission rate at each node to a point where every node, including node 6 and 7 which are far away from the base station,

can have some degree of success in reaching the base station.

4 Applications and Deployment Issues

4.1 Implementation of Motivating Applications

To explore possible functionalities of motes, we implement a couple of prototype applications on a small sensor network test bed deployed in the fourth floor of Soda Hall. The first application is light sensing, which is used to monitor light strength in several rooms in Soda Hall. The second one, object tracking, is aimed at locating objects within an accuracy of respective rooms. Both of these applications provide a web interface which enable users to obtain information on the Internet. (http://nighthawk.cs.berkeley.edu:8080/tracking/)

4.1.1 System architecture

Considering the scalability of future systems, we design a system architecture that can support multiple base stations by decoupling applications from base stations. The system architecture is shown in Figure 2.

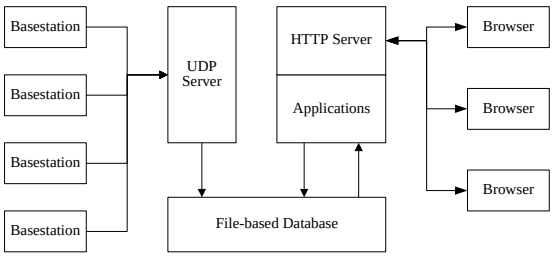

Figure 2: The architecture of the prototype system

The work process of the system is as follows:

1. motes keep sending packets to base stations over radio links
2. base stations parse these packets, remove control bytes (e.g. packet header and tail), then forward these packets to the UDP server
3. the UDP server analyzes UDP packets and save data to the file-based database
4. users can visit the applications by submitting CGI requests to the HTTP server
5. applications access to the file-based database, retrieve and process data, and then return corresponding information back to users

Currently, we manually manage part of the file-based database like naming motes and recording motes distributed information (i.e., valid mote IDs, mappings from motes to rooms, mapping from motes to objects, etc.).

The system is built on FreeBSD 2.2.7 with apache 2.3.14 and is implemented on PERL 5.005.

4.1.2 Light Sensing

Light sensing is used to monitor light strength changes in a building. The very mechanism can be applied to temperature sensing as well. Current implementation is only a prototype to evaluate sensor network's performance and is built based on the system architecture discussed above.

The UDP packet format for light sensing is shown in Figure 3. Byte 0 is source mote ID. Byte 1 is used to discriminate packets for different applications. Byte 2 is a cycling number (from 0 to 255) to label the order of application-specific packets sent from each mote. Byte 4 is light strength (255: lightest; 0: darkest). Byte 3, 5-19 are not used for now. We are considering the possible mechanism of aggregation to increase the efficiency of packets without losing real time property. When UDP server receives a packet, it will check the application type label first. If it's a light sensing packet, then the server will save sequence number and light strength as well as a timestamp into corresponding log file of that mote.

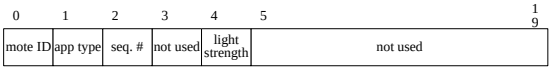

Figure 3: Packet format for light sensing

Right now the light sensing application supports monitoring of the light strength of a specific room at any given time point. It provides two granularities of time intervals for statistics: one is every hour; the other is every 5 minutes. Figure 4 is a sample demonstration, as shown in the browsers, of the statistics of the light strength of Room 473 in Soda Hall in both last two hours and last twenty-four hours from

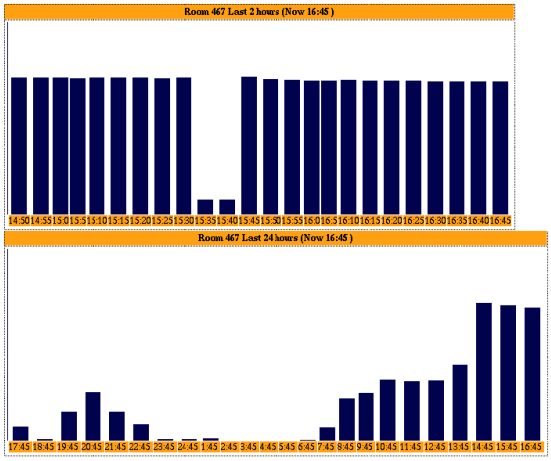

Figure 4: An example of the output of light sensing (This is the graph showing the changes of the light strength in Room 467 Soda Hall in the last two hours and last twenty=four hours before 16:45 on 12/10/00.

a given time point. Sharp changes in that figure are caused either by shutting down the light in that room in night or by packet errors. One of our future works is to decrease packet error rates.

4.1.3 Object Tracking

Many people are doing research on in-building people/object tracking. [8, 5] Compared to their research, one of the main features of our work on object tracking using motes is the limitations of motes' functionality. For instance, motes can not measure radio strength accurately. Orientation effect on motes' radio propagation is not well studied. On the other hand, motes are cheaper and can be deployed more easily than sensors and instruments used in [x] and [x]. Given these features of motes, we make the goal of our object tracking application as locating objects in a scale of different rooms.

The format of the UDP packets for object tracking is shown in Figure 5. The contents for the first 4 bytes are the same as those in packets for light sensing. Routing trace information is saved in bytes 4-8. In each byte is the ID of the most which is in the route path from the source mote to the base station. The following 12 bytes (bytes 9-18) are used to save the IDs of neighbor motes of the source mote and corresponding radio strengths between each neighbor to the source mote. The neighbors information is collected based on this mechanism. Each mote in the

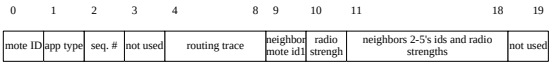

Figure 5: Packet format for light sensing

sensor network sends out packets periodically. So for each mote, it can know who are its neighbors by listening to packets from other motes. Right now we just use the neighbors information to infer the location of the object that takes the source mote. A reason why we don't use the information of radio strengths is that they are not accurate. They are always either 255 or 0.

A sample topology of sensor network is shown in Figure 6. There are two kinds of motes. One is landmark motes, whose locations are fixed. The other is moving motes, which are attached to respective objects. Since base stations are fixed, they also play as landmarks. Given the goal of object tracking application, we design an algorithm for inferring the locations of moving motes, which is sort of straightforward and intuitive. Generally speaking, for every moving mote, if we know it is close to some landmark motes, then we can infer that it should be around a specific location. Moreover, if the moving mote does not have any landmark neighbor but some moving motes, then we can infer its location by inferring the location of its moving neighbors. Now we just support two level recursion for inference, which means that if we can not infer the location of the moving neighbors of a given moving mote, then the inference algorithm will return with a failure.

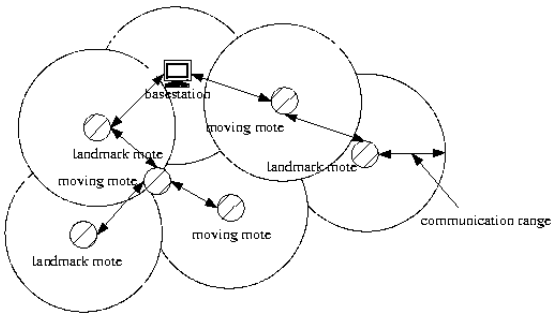

Figure 6: A sample topology of a sensor network. Every mote can only receive packets from neighboring motes which are in its communication range.

According to this algorithm, we can see that communication range of motes play a very important role. Actually there is a tradeoff. If we make the communication range very large, then the precision of

the inferred location will be very low because each mote can hear other motes even they are far from it. On the other hand, if we make the communication range very small, then the chance that a mote cannot send packets in the sensor network will be high. Right now we do some manually tune of the communication range of moving motes by adjusting the antenna lengths of them. Adaptive tune of communication ranges of motes in a sensor network to improve the performance of object tracking is a very interesting topic for future research. For current version of motes, we cannot adaptively tune their radio strengths to change their communication ranges.

To solve the problem of error packets, we maintain a log file of valid mote IDs, which can help the inference algorithm remove error neighbor IDs. For future work, we would add some rules to locate errors when some error neighbor IDs happen to be some valid one but far from the source mote's real location.

The object tracking application supports tracking of specific object's location at any given time point. Users just need to visit the web page and input the object's name (now it is a person's name) and a time, and then they will get the location information of that object at that time. However, we implement a simple mechanism to do access control for protecting privacy. For some specific objects, only people who know these objects' control numbers can track them. When this mechanism is applied to the real world, it can support the case like only your boss and project partners can track you.

4.2 Deployment Issues

4.2.1 Development Platform

The networked sensor prototypes, designed and developed by the Smartdust [10] and TinyOS [3] research group, have provided us a valuable testbed for real deployment of our applications. We believe these prototypes are very representative since they entail typical networked sensor constraints of limited computation power, storage, and energy. The processor is an ATMEL [1] 4MHz, 8 bit microprocessor with 8kB of program and 512 byte of data memory. The radio is a single channel RF transceiver operating at 916MHz. They are power limited by the attached battery and consumes in an order of 10mA in its active state. With hardware analog to digital conversion built into the processor and software I2C [13] support, a heterogenous set of sensors such as light, temperature, humidity, pressure, acceleration, and magnetic field can be integrated into these prototypes.

Application development on networked sensors has been greatly simplified by the component based design in TinyOS and networking support discussed in 3. An application comprises a tree of components, with each levels in the tree constitutes an abstraction down to the hardware. A rich set of components for accessing the network and sensors has already been provided by TinyOS. Therefore, application author simply composes a tree of components that it needs (e.g. networking, light sensor) for its application and concentrates on implementing the application itself. Since TinyOS takes an event driven programming approach, each components is capable of issuing events and accepting commands. An application should also follow this state machine model by processing events such as incoming messages or incoming requested sensor readings and issuing commands to send messages or request sensor readings. The path of event and command propagations are defined by the tree of components. Any computations such as averaging can be done by posting a task which is non-blocking and runs to completion. Our experience suggests that most applications under TinyOS can be accomplished in less than one hundred lines of code.

4.2.2 Energy Consumption

Energy is probably the most precious resource on a networked sensor. Table 5 shows the current consumption of various components on our sensor prototype. [3]

Component	Active (mA)	Idle (mA)	Inactive (μA)
MCU core (AT90S8535)	5	2	1
MCU pins	1.5	-	-
LED (each)	4.6	-	-
Photocell	.3	-	-
Radio (RFM TR1000)	12 tx	-	5
Radio (RFM TR1000)	4.5 rx	-	5
Temp (AD7416)	1	0.6	1.5
Co-proc (AT90LS2343)	2.4	.5	1
EEPROM (24LC256)	3	-	1

Table 5: Current per hardware component of our sensor prototype.

At peak load, the entire system consumes about 19.5mA of current. With our prototype powered by a lithium battery rated at around 575mAh, it is expected to run for about 30 hours. Our experience

from real experiment suggests that it actually run for about 20 hours as the processor fails to operate at a voltage below 2.7V which constitutes about 80% of the battery capacity. With two AA alkaline batteries rated at 2850mAh each, it is expected to run for 300 hours or 12.5 days. If every components stays in idle mode, the system can extend to run for 120 days. Therefore, the operation duty cycle for each component must be tuned in order to achieve a right balance between application demand and life time of the device. At the current implementation, we tune the radio duty cycle to be 10% while it is sampling for the preamble of a packet and resumes to 100% once it has sensed an incoming packet.

5 Future Work

Our deployment experience has opened up lots of interesting issues that we are motivated to continue exploring:

1. *Authenticated Data Aggregation* An interesting challenge arises when data aggregation is performed in the intermediate nodes of the routing tree. Since the intermediate nodes does not have the shared-key to validate data from their children, it cannot authenticate the message and aggregate the data. We believe this problem can be solved by public-key cryptography system. With public-key system, the authentication problem reduces to validating the certificates of children nodes via a trusted third party.

2. *Secure Peer-to-Peer Communication* Under our current scheme, communications between peering motes are not secure. To establish peer-to-peer secure channel, we believe well-known solution such as Kerberos [14] is suffice to setup shared secret between the communicating parties and enable secure communication. Although we have not encountered applications that require this capability, as the need arises, such scheme can be implemented.

3. *Life Monitoring Capability* Having the battery life monitoring capability is probably one of the most useful feature in the case of deployment. It is not possible to rely solely on the analog to digital converter inside the processor since its reading is relative to the power supply voltage. We are currently looking to use an adjustable band-gap voltage reference device to implement this task.

4. *Data Aggregation* To deploy a scalable sensor network, some form of application dependent data aggregation should be done along routes to the base station while maintaining a certain degree of error tolerance. Though data aggregation may not be applicable in all cases, some variants of packet concatenation along routes to the base station may as well achieve similar aggregation behavior.

5. *Adaptive Radio Transmission Range Control* Physical node placement to achieve desirable node density without any dead spot can be extremely tedious, especially for large scale deployment. Having the ability for each node to adapt its transmission signal strength based on a desirable node density rather than careful placement of nodes by human would be yet another desirable feature. Most importantly, when nodes fail, the same mechanism can be used for neighboring nodes to increase their transmission strength to route over failure nodes to achieve a self healing effect.

6. *Location Based Addressing with Random MAC ID* Our current deployment of 15 nodes is not a large scale study. In fact, our MAC identifier for each sensor node only allows 255 different nodes. Thousands of sensors may each require a unique identifier if deployment is done this way. An interesting approach is to use location based address combined with a small random identifier. For example, most applications may only need temperature reading in a specific location and do not care which particular sensor is reporting the reading. Such a scheme also naturally provides a hierarchical addressing scheme based on geographical scope. Since radio transmission range is relatively small, a small random identifier is adequate for a low probability of identifier conflict within a local region.

7. *Security of sending packets from base stations to the UDP server* Now the packets from base stations to the UDP server are in the clear. Since some information in the packets are related to privacy, e.g. the packets for object tracking, we are thinking about encrypting these packets in the future.

8. *Improvement of current inference algorithm* Learning algorithms are widely applied to applications in sensor networks. The current version of our inference algorithm for object tracking is kind of simple. As the functionalities of motes are getting stronger and stronger, we can

collect more information from different perspectives, e.g., radio strength, antenna orientation, etc. Then, we can import Kalman Filter algorithm to improve the performance of object tracking.

9. *Database optimization* Now we implement these applications by using file-based database and PERL. The speed of these applications is not impressive. As the size of the sensor network grows larger, we need to optimize the structure of the database to increase the applications' speed.

6 Conclusion

Security and deployment issues are critical in real-world large scale sensor networks. We have designed and implemented a security protocol that provides confidentiality and authentication using only RC5 shared key decryption primitive. Our results show that code size is 1.6KB and performance is adequate even for a mote with only 8KB of memory and a 4MHz 8-bit processor.

We have also designed and implemented an adaptive transmission and routing scheme to address fairness in the network, so we can better scale up the number of motes without starvation.

Our two demo application are people tracking and light sensing, and we can successfully track people at room-level resolution and can monitor light levels. The application have web-based query interface to make them easier to use.

In terms of deploying the actual sensor network, we used AA batteries instead of the small lithium button batteries for better battery life, and we slowed down the send rate to reduce energy consumption. We also manually tuned the transmission range of the motes to give the tracking application better resolution. We believe that using combinations of self-monitoring and adaptive algorithms will further simplify the deployment process.

From our experience in setting up a real sensor network in our building, we learned that the solutions that we have used are helpful, but are only a tip of the iceberg (no pun intended). There is still much research that we need to work on to be able to deploy a large scale sensor network.

References

[1] Atmel, Inc. AT90S4434/LS4434/S8535/LS8535 *Preliminary (Complete) Datasheet.*

[2] Neil M. Haller. The S/KEY one-time password system. In *ISOC*, 1994.

[3] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister. System architecture directions for networked sensors, 2000.

[4] David L. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. Internet Request for Comments, March 1992. RFC 1305.

[5] Anit Chakraborty N. B. Priyantha and Hari Balakrishnan. The cricket location-support system. In *Proceedings of ACM MOBICOM*, pages 32–43, 2000.

[6] NIST. Advanced encryption standard (aes) development effort. `http://csrc.nist.gov/encryption/aes/`, October 2000.

[7] OpenSSL. The OpenSSL project. `http://www.openssl.org/`, 2000.

[8] V. N. Padmanabhan P. Bahl. Radar: An in-building rf-based user location and tracking system. In *Proceedings of IEEE Infocom*, pages 775–784, 2000.

[9] Adrian Perrig, Ran Canetti, J.D. Tygar, and Dawn Xiaodong Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, May 2000.

[10] K. S. J. Pister, J. M. Kahn, and B. E. Boser. Smart dust: Wireless networks of millimeter-scale sensor nodes, 1999.

[11] Ronald L. Rivest. The MD5 message-digest algorithm. Internet Request for Comments, April 1992. RFC 1321.

[12] Bruce Schneier. *Applied Cryptography (Second Edition)*. John Wiley & Sons, 1996.

[13] Philips Semiconductors. The I^2C-bus specification, version 2.1. `http://www-us.semiconductors.com/acrobat/various/I2C_BUS_SPECIFICATION_%3.pdf`, 2000.

[14] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. pages 191–202, Winter 1988.