Environmental sensor networks are increasingly becoming practical and feasible ways of gathering data across broad landscapes and on a rich collection of animal and plant species. Through a mixture of fixed and mobile sensors, data can be collected across huge areas with relatively low infrastructure costs, using ad hoc networking techniques. The key to these approaches is in developing energy-efficient sensing and computing hardware and communication protocols for interpreting, fusing, and aggregating sensed data and for delivering it to researcher base stations.

The ongoing Princeton ZebraNet project is studying wildlife-tracking systems based on ideas from mobile ad hoc networks. We have designed energy-efficient hardware tracking nodes that integrate computing, wireless communication, and nonvolatile storage, along with GPS and other sensors, and we are building communication-protocol software to aggregate the archived position data and forward it peer-to-peer toward the researcher base station. Our first test deployment was in January 2004 on zebras at Sweetwaters Game Reserve near Nanyuki, Kenya, to study both fine-grained and long-term movement and migration decisions. This chapter describes the biology research goals of ZebraNet, discusses many of the ZebraNet hardware- and software-design decisions, and gives some initial information from our first deployment.

16.1 Introduction

Environmental sensor networks have been a topic of significant research attention in recent years because they offer the potential to get fine-grained experimental observations about natural phenomena that have been thus far very difficult to study [1–4]. Habitat monitoring, with its focus on dynamic interactions within and between a variety of scales, is an ideal application of sensor networks because answering fundamental biodiversity-research questions on animal interactions on landscapes that are changing in response to normal as well as anthropogenic processes requires large amounts of diverse data, collected and correlated across large temporal and spatial scales.
Sensor networks consist of many (tens, hundreds, or more) nodes, each of which combines sensing capabilities with a small computer and storage. Each sensing node also communicates, usually wirelessly, with other nodes or a researcher's base station to share and archive data. Sensing capabilities are quite diverse and might consist of anything from simple temperature or orientation sensors, to cameras, microphones, or GPS systems [5–8]. Sensor networks enable researchers to do continuous, long-term, autonomous sensing of many different aspects of an environmental system.

While sensor networks offer the potential to make great strides forward in biocomplexity research, their application has been limited thus far by significant technical challenges in implementing them. For the most part, environmental sensor-network deployments have involved modest collections of sensors at fixed positions in a relatively small ecological area under study. These include Intel/University of California, Berkeley's deployment on Great Duck Island, Maine, and UCLA's deployment at the James Reserve.

Sensors that are fixed in their position limit the type and amount of data that can be collected by a sensor network. With this in mind, in 2002, we began the ZebraNet project: research on mobile sensor networks for tracking wildlife migrations over large and sparsely populated areas [3]. The ZebraNet research project is an interdisciplinary project combining computer science and biology thrusts. On the computer systems side, we are researching application and protocol issues for building sensor networks that are effective in aggregating data collected over very large (hundreds or thousands of square kilometers) and sparsely populated areas. We are also researching strategies for building power-efficient sensor hardware. In January 2004, ZebraNet hardware prototypes were deployed at the Sweetwaters Game Reserve near the MpalA Research Centre in Kenya [9].

These prototypes are the outcome of the computer systems side of the research and, in turn, drive the biology side of the research, where the nodes enable biologists to study both fine-grained and long-term decision making in certain species (notably zebras).

16.2 Habitat Monitoring and ZebraNet's Biology Goals

16.2.1 Wildlife Tracking and Habitat Research: Background

Research on animal behavior depends critically on having good technology for wildlife tracking. For understanding both the mechanistic and functional basis of many types of animal behavior—especially those associated with foraging, moving, mating, and sociality—individually identifiable animals must be located and followed repeatedly and on a consistent basis. Often detection and subsequent tracking is visual, but for nocturnal species or those moving over large distances (tens of kilometers per day or hundreds of kilometers per season) or even for those whose lifestyle is simply secretive, detection and monitoring must be done remotely, typically by radio tracking.

Ordinarily, radio tracking involves collaring animals with very high frequency (VHF) radio transmitters and using triangulation from a series of radio fixes to find them [10, 11]. If aerial reconnaissance is employed, then large distances can be
more) nodes, each of which store data. Each sensing node is a researcher base station and might consist of cameras, microphones, sensors to do continuous, real-time monitoring.

great strides forward in this area of study and continue to make significant advances, particularly in environmental engineering, computer science, and sensor technology.

amount of data that can be collected varies depending on the specific application. In some cases, data can be collected over very short periods of time (hours), while in other cases, data can be collected over longer periods (days or weeks). To be effective, sensor hardware must be designed to handle large volumes of data and be able to transmit data to a central location.

side of the research team is the development of algorithms and data analysis techniques to make sense of the data collected by the sensors. This is critical for turning raw data into meaningful insights.

16.2.2 Habitat Monitoring and Mobile Sensors

Predators and prey are inextricably linked. Preliminary observations show that zebra density increases with vegetation height and decreases with the abundance of lions and hyenas. But lion abundance also increases with grass height. So, even if zebras appear to avoid lions, nutritional needs and the lure of large amounts of vegetation bring zebras and lions together. Until now, it has been difficult to monitor how predators and prey respond to each other and how the game of hide and seek plays out. Although visual observation is possible, close follows for long periods of time disrupt natural behavior. Using traditional VHF radio fixes for short periods cannot provide sufficient spatial and temporal resolution to determine who is driving the movements of these coupled species. After dark, collecting VHF data is unrealistic.

Tracking devices such as the ZebraNet collars can sample the locations and activities of animals at fine-grained intervals (e.g., 8-minute sampling). By gathering such fine-grained data, we can determine if and when each species' activities initiate changes in the other species' behavior. Truly fine-grained analysis of coupled movements and activities for large and wide-ranging predators and prey has never been attempted before. At the regional, or landscape, level, we can also begin to monitor how zebra movements and change in density are shaped by the bottom-up features of vegetation abundance and quality and the top-down forces of predator abundance and perceived risk. Such research can help guide effective
wildlife-management plans and identify at which trophic level interventions are likely to be most effective.

With the expansion of human settlements and the intensification of land use, as well as the conversion of pastoral lands to horticulture, seasonal migrations of wildlife are being impacted [12]. Curtailment of traditional migration routes, as well as the creation of rich feeding alternatives for herbivores and predators alike, increases the potential for human-wildlife conflict [13–15]. As the project proceeds, it may also be possible to fit collars on livestock and (along with fixed sensors positioned near human settlements) help identify the ways in which humans and wildlife interact.

16.3 ZebraNet System Overview

The ZebraNet system has been designed specifically to support the research of biologists working at the Mpala Research Centre in Kenya [9] by applying the latest sensor-network technology to the field of animal tracking. ZebraNet consists of sensor nodes built into collars on zebras, which take positional readings using a GPS unit and propagate them from zebra to zebra until infrequent communications percolate data to base stations [3]. Given that the zebras are fairly mobile and spread over a large distance, we expect our system to form an extremely sparse network, with roughly one collar per 10 km². Our system goals are as follows:

- **Detailed, accurate position logs.** Take frequent positional readings that are fine-grained enough to give the biologists an accurate view into the daily migration pattern of a set of zebras.
- **High data recovery.** Propagate data to a mobile base station through pairwise communications, with latency being less important than eventual success.
- **Autonomous operation.** Survive long term in the wild with no human contact except for occasional drive-by radio uploads of data.

A ZebraNet hardware node includes GPS, a simple microcontroller CPU, a wireless transceiver, and nonvolatile storage to hold logged data. ZebraNet does not rely on constant communication access to a base station or other nodes. Instead, it uses periodic node discovery and node-to-node communication to propagate data toward the base station in a store-and-forward manner.

The ZebraNet hardware is composed of energy-efficient components ideal for use in mobile sensor networks. A photograph of the most recent ZebraNet hardware (top and bottom) is presented in Figure 16.1. A block diagram of the main system components is given in Figure 16.2. Finally, an overview of the main hardware and software system layers in ZebraNet is depicted conceptually in Figure 16.3. The major functional components on the board are the microcontroller, GPS, external Flash, radio, and battery with solar chargers.

To control the hardware, we selected the Texas Instruments Ultra-Low-Power MSP430F149 16-bit microcontroller. This chip has 2 KB of RAM, 60 KB of internal Flash memory, and two serial interfaces [16]. It runs off an uninterruptible power supply as we expect it to run continuously. The microcontroller operates in a dual-clock configuration. It uses an 8-MHz clock when accessing sensing, storage,
c level interventions are needed. These interventions are driven by the need to control activities in the area, as well as to protect wildlife, including migrating species of tropical birds, and to prevent human activities from impacting wildlife. The ZebraNet system is designed to be scalable and adaptable, allowing for the addition of new nodes and the expansion of the network over time.

16.3.1 Evolution of Our Hardware

Based on the system design parameters, the main design constraint is energy efficiency. ZebraNet nodes are built using low-cost, energy-efficient, off-the-shelf components, including a low-power microcontroller and a GPS sensor. The system is designed to operate in a low-power mode most of the time, consuming only enough power to maintain data integrity and detect changes in the environment. The system is also designed to be flexible, allowing for the addition of new nodes and the expansion of the network over time.
components. From conception to realization, the ZebraNet node has gone through several design iterations. Table 16.1 shows the major design changes between the major iterations of the design. Version 0.1 was a proof-of-concept design [3]. In this version, a few design schemes were tested, such as dual radios, where we included two radios, each with different baud rates and ranges. Cycle stealing was employed to use unused cycles of the GPS processor. However, the power consumption of these schemes proves to be too great.

The major step from Version 0.1 to Version 1 was the move to include a low-power microcontroller that controls all the peripherals and an external 2-Mbit Flash memory to store data. (Version 0.1 scavenged both Flash and CPU cycles from the GPS chip.) Having dedicated processing and storage greatly simplified software development and eased design changes by allowing peripheral changes with only firmware-level updates. Version 1 did not have the solar-charging circuitry but did
include various exploratory designs of high-efficiency power supplies. Their impact on the system performance and efficiency is presented in Section 16.5.

Although similar to Version 1, Version 2 was designed to require half as much board area. The design mostly explored issues with the layout and cross-talk interference of onboard components. The power supplies were improved to lower noise.

Version 3 is a complete system design. The microcontroller-centric design and the power supplies were further improved. We also further increased the Flash capacity to 4 Mbit. A photograph of this most recent version of the sensing
Table 16.1 Design Summary for Different Versions of ZebraNet Hardware Nodes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Power supply</td>
<td>Off-board</td>
<td>Buck-boost and boost converters</td>
<td>Same as version 1</td>
<td>Two buck-boost converters</td>
</tr>
<tr>
<td>Noise reduction</td>
<td>Bypass capacitor</td>
<td>Standard low-FSR capitors</td>
<td>Os-con capacitors</td>
<td>Liquid crystal (LC) post-filters and common mode choke</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>600 MHz</td>
</tr>
<tr>
<td>Radio</td>
<td>Two-radio system</td>
<td>2.4 GHz</td>
<td>900 MHz</td>
<td>Same as version 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPS</td>
<td>Cycle-stealing from onboard CPU</td>
<td>Off-board antenna power</td>
<td>Ultra-low noise linear regulator</td>
<td>False-charging</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data flash</td>
<td>5 Mbit</td>
<td>4 Mbit</td>
<td>4 MHz</td>
<td></td>
</tr>
<tr>
<td>Battery charger</td>
<td>None</td>
<td>Off-board</td>
<td>Off-board</td>
<td></td>
</tr>
<tr>
<td>System weight</td>
<td>1.131g</td>
<td>1.36g</td>
<td>1.38g</td>
<td></td>
</tr>
<tr>
<td>Battery</td>
<td>Ni/A</td>
<td>Li-Ion, 2A-hr, 45g</td>
<td>Li-Ion, 2A-hr, 45g</td>
<td>Li-Ion, 2A-hr, 45g</td>
</tr>
</tbody>
</table>

Hardware is shown in Figure 16.1. It is powered using a lithium-ion battery that is recharged with a solar array. This version was built into collars and deployed in central Kenya in January 2004. We deployed seven nodes on plains zebras at the Sweetwaters Game Reserve near Nanyuki, Kenya. Based on the results from the first deployment of the system, we are currently making improvements with a broad-scale, long-term deployment in mind. More information about our deployment experiences are provided in Section 16.5.

One fundamental aspect of sensor systems is their ability to overcome the numerous constraints imposed by the hardware and the applications’ scenarios. In particular, we list the most important design constraints below:

- **Data and program memory.** The data memory in the microcontroller is only 2 KB. This affects the program behavior in many aspects, especially in data buffering. As they are used to keep system states and to handle large flows of network data, data buffers often consume large amount of memory and, therefore, must be carefully allocated. Additionally, the program memory is only 60 KB. This requires that software programs be concise.

- **Energy.** The energy budget is tight as we use a solar array to recharge the battery and to provide the energy essential to achieve the sensing and communication tasks. As is estimated, we are able to charge the battery fully in 50 hours of daylight. This number can vary in either direction, however, depending on the orientation of the solar cells in relation to the sun. Therefore, efforts must be made to save energy maximally, and resorts must be provided to preserve the system when the energy level is severely low.

- **Device access.** Device accesses must be carefully scheduled to avoid conflicts that are likely to happen due to hardware limitations. For example, due to voltage-regulation challenges, the GPS and the radio should not be turned on at the same time for interference-avoidance purposes. Additionally, the GPS

16.4 The Impala

...As demonstrated operating for this se...
and the Flash share the same serial connection to the microcontroller and, therefore, cannot be accessed simultaneously.

- **Radio packet size**: The physical packet size of our radio hardware is only 64 bytes, an order of magnitude smaller than the Ethernet packet size, for example. This means the multipacket header in the traditional TCP/IP model will entail a significant communication overhead. Therefore, we need a special network protocol that requires a low overhead to accomplish the essential network-communication services.

- **Flash data storage**: For the Flash memory, new data cannot be written to an address before the data currently at that address is erased, and the smallest erasable unit is a 256-byte page. This means writing data to one location will affect data at other locations. Therefore, a global Flash organization is required to achieve efficient data storage.

- **GPS sensing time**: The time for the GPS unit to acquire an accurate position lock is typically 10 to 20 seconds. This considerable delay in data acquisition implies that an asynchronous access and control model is preferred to a synchronous model for operating this sensing device.

### 16.4 The Impala Middleware System

As depicted in Figure 16.3, the ZebraNet hardware is managed by several custom-designed software layers. In particular, the Impala system comprises the main operating system and network services for ZebraNet. It works with a thin, underlying firmware layer to provide a range of services to sensor-network applications. In this section, we discuss Impala's design in more detail.

The Impala operating system and middleware service model is driven by several issues applicable to ZebraNet and to general sensor-network applications as a whole. The first issue is that the long-term sensing and communication tasks of sensor-network applications require dependable scheduling of regular operations. Sensor networks are designed to run for indefinite periods without human intervention. Many sensing and communication operations occur on a predictable timetable. ZebraNet, for example, executes GPS position sensing and wireless radio communication periodically. In addition to these operations, the system must perform many other routine system computations and maintenance. Therefore, Impala must provide clean mechanisms to schedule recurring operations.

A second issue is that sensor-network applications require efficient handling of irregular events. Fundamentally, sensor nodes are event-driven systems. Events such as sensor-data capture and network-data reception occur frequently and are the primary triggers of system computations. An event may result from a single or a sequence of hardware interrupts. Depending on the API, promiscuous hardware interrupts can be made transparent to applications and delivered only as a few types of abstract events. However, an appropriate event abstraction should balance simplifying application programming with maintaining the granularity of application-level processing. Additionally, events may be handled by different components of the system and, therefore, require efficient event filtering and dispatching.
Third, sensor network applications require specialized network support. As data gathering is the primary goal of sensor networks, sensor nodes often use aggressive flooding strategies to maximize the chance of finding a path to the desired destination. The resultant multicasts and broadcasts are common communication patterns. Transmissions must be reliable in scenarios where data integrity is critical. Data can be unreliable, however, in cases where packets can be lost without compromising our goals. For example, peer discovery messages are considered unreliable as the nodes are mobile and may not be in range of the other nodes. Additionally, due to the severe resource constraints and limited hardware capabilities of sensor nodes, efforts must be made to minimize the overhead in communication, buffering, and processing.

The fourth issue is that the complexity of sensor-network systems requires dynamic software adaptation. The scale of sensor-network systems can be on the order of thousands of nodes; therefore, coordinating the communication and computation across the system is complex. Depending on node topology, network connectivity, and node mobility, over its lifetime the system may encounter a number of different scenarios, for each of which a different communication protocol may be appropriate. As such, it is nearly impossible for a single protocol to be appropriate all the time. Some amount of adaptivity is crucial for applications to handle an interesting range of possible parameter values properly.

Finally, the long-term deployment and inaccessibility of sensor-network systems require automatic remote software updates. It is inevitable that software updates will be required during the lifetime of a sensor network. Because sensors are typically deployed in large numbers in inaccessible places, updates must be deployed wirelessly. Therefore, Impala needs to support automatic remote software updates so that new software can be plugged in at any time. ZebraNet offers very clear motivation for remote software updates since we clearly do not want to have to tranquilize and recapture a collection of collared animals each time we need to update the software.

Compared to other sensor operating system work, such as TinyOS [29], Impala’s differences can be summarized in terms of providing additional core services beyond the minimal layer provided by TinyOS. These services include base networking support, as well as hooks for software adaptation and update. By embedding these services into the operating system itself, these services can be provided in a lean and system-oriented manner that lessens the need for application programmers either to custom-write these services themselves or to further layer prewritten services on top of the operating system.

We have built two implementations of Impala. The first implementation was a prototype running on Palm/Compaq iPAQ pocket PC handhelds [17]. Subsequently, we implemented Impala again on the real ZebraNet hardware nodes [18]. This latter implementation and description focuses on its operating system functionalities in hardware/software interfacing, system operation scheduling, event handling, and network communication support.

### 16.4.1 Impala System Layers

Figure 16.5 shows the static view of Impala with three system layers: the uppermost application layer, the Impala layer, and the firmware layer. Services and events are
lized network support. As sensor nodes often use aggre-
gating a path to the desired common communication, data integrity is critical, keys can be lost without messages are considered of the other nodes. Additional hardware capabilities of thread in communication, network systems require work systems can be on the communication and complex topology, network con-
are encounter a number of application protocol may be protocol to be appropriate applications to handle an sensor-network systems. That software updates because sensors are typically must be deployed, update software updates it offers very clear moti-
rant to have to sprinkle we need to update the such as TinyOS [20],ing additional core services include base net-
tation and update. By these services can be pro-
ected for application processes or to further layer implementation was a ds [17]. Subsequently, nodes [18]. This latter item functionalities in event handling, and

layers; the uppermost services and events are

Figure 16.5 Impala system architecture: layers and interfaces

the major interfaces between layers. Through the service interface, the firmware layer exports numerous hardware access and control functions to the Impala layer. The Impala layer, however, protects these firmware functions from direct use by the application layer and only exports the ones needed by applications in a reduced or protected form. It also exports its own network interface to the application layer.

16.4.2 Regular Operation Scheduling

The dynamic view of Impala comprises regular computing and maintenance operations required by the long-term sensing and communication tasks and irregular events incurred by the inherent event-driven attribute of sensor-network applications. For regular operations, Impala acts as an operation scheduler that schedules and coordinates system operations based on application goals, hardware constraints, and energy budget. For irregular events, it acts as an event filter that captures and dispatches events to different system components and initiates chains of processing.

Impala uses timers to trigger various operations. GPS-aided time calibration allows networkwide operation synchronization. Since ZebraNet sensor nodes have ongoing access to global GPS time, sensor nodes can be easily synchronized. This is especially important for network communication in which all nodes need to turn on and off their radios simultaneously and transmit in assigned time slots to avoid collisions. In addition, long-duration events, such as GPS, also influence operation scheduling. Where there may be large variation, a split-transaction approach is used.

The stringent energy budget requires energy conservation whenever possible. The battery capacity of the ZebraNet nodes can support the full level of system activities for one to three days. The solar array can extend this time indefinitely, but solar cell area is limited, so we need to conserve energy whenever possible. Impala
achieves this with two approaches. First, ZebraNet has an 8-minute GPS data sampling interval to ensure that we capture significant movements of zebras while minimizing redundant data records. In between these intervals, Impala works to put system components into low-power modes to save energy. Second, although the energy supply of our system is designed to fulfill the energy consumption under typical conditions, we still need to preserve the system in the case of energy deprivation. Therefore, Impala adapts its operation scheduling to current battery levels. It skips energy-intensive phases such as communication or GPS sensing if the energy level is inadequate.

16.4.3 Event-Handling Model

Impala's event-handling model is designed to address three fundamental issues. First, sensor-network systems require an efficient event-based API. Events originate from hardware interrupts. Dealing with these interrupts not only involves considerable programming efforts but also requires detailed hardware knowledge. Therefore, Impala has an event abstraction that encapsulates miscellaneous hardware interrupts into abstract events to simplify application programming, while maintaining the granularity of application-level processing.

Impala implements four types of abstract events that are essential for ZebraNet applications. An event is generated and enqueued by an event signaler, dequeued and dispatched by Impala's event filter, and processed by an application event handler. Figure 16.6 shows the abstract events and Impala's event-handling components. A network-packet event represents the arrival of a network packet. Impala's network interface generates this type of event after it receives a packet from the radio firmware and examines the validity of the data. A network send-done event represents the completion or failure of a network-message transmission. The network interface generates this type of event after it has completed the transmission or a failure has occurred. An application-timer event represents the time to execute a prescheduled application operation. The timer firmware generates this type of event after the application timer expires. A GPS-data event represents the capture of a GPS position fix. The GPS firmware generates this type of event after it analyzes the information output from the GPS unit and identifies a position fix.

Because concurrency is an important attribute of sensor-network systems, Impala has a hierarchical event-handling model that processes simple hardware interrupts in short or atomic routines and handles complex software events in long or nonatomic routines. This not only achieves concurrency among multiple flows of processing but also allows low-level processing to interleave with and, if needed, override high-level processing.

Finally, event prioritization is desirable in sensor-network systems. Some events are urgent and require immediate processing, such as the network-packet events. Some events are time constrained but are not sensitive to small delays, such as the application-timer events. Other events are highly latency tolerant, such as the GPS-data events. Therefore, event prioritization allows events with different time constraints to be processed in the desired order. As shown in Figure 16.6, Impala's event filter maintains an event queue for each type of event and associates each queue with a priority for event processing.
8 minute GPS data samplings of zebras while minivans, Impala works to purify. Second, although the energy consumption under typical case of energy deprivation, current battery levels. It skips stressing if the energy level is

...fundamental issues. First, API events originate from only involves considerable knowledge. Therefore, efficacious hardware intermingling, while maintaining are essential for ZebraNet event signaler, dequeued an application event handler's event handling component network packet. Impala's receives a packet from the network send-done event trigger transmission. The network interface completes the transmission and sets the time to execute a generates this type of event sent the capture of a UPS event after it analyzes the position fix.

Sensor-network systems, process simple hardware events in complex software events in long among multiple flows of data with add, if needed, network systems. Some events in network-packet events of small delays, such as the IP packet tolerant, such as the events with different timing in Figure 16.6, Impala's event and associates each

![Impala event-handling model](Image)

**Figure 16.6** Impala event-handling model.

### 16.4.4 Communication Characteristics and Impala Networking

The network interface, as a middleware service, is crucial in mobile wireless sensor systems. As in many other mobile WSNs, ZebraNet uses peer-to-peer communication. Unlike many others, the sparse connectivity caused us to choose pairwise store-and-forward routing rather than common path-based approaches. To support the application layer, which studies various store-and-forward routing strategies, Impala's network interface focuses on the networking model within one hop.

The special communication pattern of sensor-network applications like ZebraNet changes the message model. Sensor nodes often use an aggressive flooding strategy to maximize the chance of finding a path to the base station. This leads to the common use of multicast and broadcast protocols. Furthermore, transmissions must be reliable in some cases but, for energy reasons, unreliable transmissions are preferred in other less critical uses. Impala's network services must support these different uses.

Impala uses session-based transport control. A session is a message designated by the application to have network-transaction semantics. Sessions can vary from 1K to 32K bytes, can be unicast, multicast, or broadcast, can transmit data from Flash or from application RAM buffer, and can use reliable or unreliable transmission. We chose connectionless session transmission because connection-oriented approaches seemed a poor match for the unpredictable motion of sensor nodes in our system. Connectionless sessions also reduce computation and communication overhead.

### 16.4.5 Time-slot-based Media Access Control

Since ZebraNet sensor nodes have ongoing access to globally synchronized GPS time, sensor-node activities can be easily synchronized. Impala takes advantage of this time
synchronization and uses simple, round-robin, time-slot-based MAC. We chose this protocol for simplicity (code size and energy) but also because the round-robin nature of the approach means that the MAC layer always knows which nodes should be acknowledging reception in each time slot. This allows simple yet efficient time-out and retransmission mechanisms, as described in the next subsection.

In the time-slot-based MAC, each node is statically assigned a unique time slot for transmission in an iteration. A sensor node uses its time slot both to transmit data packets and to acknowledge previously received packets. Figure 16.7 shows an example of time-slotted transmissions between multiple sensor nodes. ZebraNet only expects tens of nodes, so this non-scalable solution is acceptable and more efficient. In a larger network, one might choose to use a hybrid time or contention algorithm in which a small number of nodes share a time slot.

16.4.6 Impala Evaluations

The Impala middleware system, including the firmware layer, the Impala layer, and a baseline application, has been implemented on the ZebraNet hardware nodes. To evaluate Impala's overhead and performance, we have conducted some preliminary measurements and analysis as described below.

Static memory footprint Since our system faces severe memory constraints, it is important for us to minimize code size and RAM usage. Figure 16.8 shows the program-memory and data-memory footprints of different system layers. For program memory, the network interface requires 5,712 instruction bytes and is the largest component in the Impala layer. The Flash, GPS, and timer modules are the major components in the firmware layer. The application layer is lean because we only implemented one basic application.

For data memory, the network interface in the Impala layer claims 51 bytes of data memory. In the firmware layer, the GPS module requires a 125-byte buffer to receive information from the GPS unit, and the radio module requires a 64-byte buffer to receive packets from the radio.

As depicted in Figure 16.8, our code currently consumes less than one-third of the total program memory, and we statically allocate less than one-sixth of our RAM, which leaves ample memory for dynamic allocation and for future expansions to our system.

Network interface performance We evaluate Impala's network interface in its processing overhead for packet reception and transmission, its communication

![Data sends and related acknowledgments in the time-slot model](image)

Figure 16.7 Data sends and related acknowledgments in the time-slot model

<table>
<thead>
<tr>
<th>B</th>
<th>A</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>acks packet 1-8</td>
<td>acks packet 1-8</td>
<td>acks packet 1-8</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>acks packet 1-4</td>
<td>acks packet 1-4</td>
<td>acks packet 1-4</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>acks packet 1-8</td>
<td>silent</td>
<td>acks packet 9-16</td>
<td></td>
</tr>
</tbody>
</table>

Table 16.2

<table>
<thead>
<tr>
<th>System Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
</tr>
<tr>
<td>Software</td>
</tr>
<tr>
<td>Network and I</td>
</tr>
<tr>
<td>Filter Module</td>
</tr>
<tr>
<td>Radio Firma</td>
</tr>
<tr>
<td>Radio Hardw</td>
</tr>
</tbody>
</table>

Figure 16.8 (a) Packet overhead for a multihop packet filter to define time to on an asset. 16.4. time 1 message data W states.
16.4 The Impala Middleware System

![Program memory footprint](chart)

![Data memory footprint](chart)

Figure 16.8 (a) Program-memory footprint, and (b) data-memory footprint.

Overhead caused by packet headers, and its communication latency in reliable multicast.

For packet reception, Impala propagates an incoming network packet from the radio hardware, to the radio firmware, to the network interface through interrupts and callbacks. Then, the packet is enqueued by the network interface until the event filter dequeues and delivers it to the application. Table 16.2 shows the processing time to receive a network packet in each system layer.

On the transmission side, the network interface provides the applications with an asynchronous operation for network transmission. As we described in Section 16.6.4, sessions can be dropped into the network interface by the application at any time. The time to drop an unreliable broadcast session, such as a peer-discovery message, is 496 cycles. The time to drop a reliable multicast session, such as a GPS data session, is 901 cycles.

When the networking time comes, the network interface will update session states, compute the information to send, copy data between Flash and RAM, and

<table>
<thead>
<tr>
<th>System Layer</th>
<th>Time</th>
<th>Processing Breakdown</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Software</td>
<td>111819 cycles</td>
<td>Packet write to FLASH</td>
<td>10172 cycles</td>
</tr>
<tr>
<td>Network and Event Filter Middleware</td>
<td>1058 cycles</td>
<td>Packet processing by the network interface</td>
<td>1647 cycles</td>
</tr>
<tr>
<td>Radio Firmware</td>
<td>3470 cycles</td>
<td>Processing packet synchronization bytes</td>
<td>8 cycles</td>
</tr>
<tr>
<td>Radio Hardware</td>
<td>255585 cycles</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System Layer</th>
<th>Time</th>
<th>Processing Breakdown</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Software</td>
<td>111819 cycles</td>
<td>Packet write to FLASH</td>
<td>10172 cycles</td>
</tr>
<tr>
<td>Network and Event Filter Middleware</td>
<td>1058 cycles</td>
<td>Packet processing by the network interface</td>
<td>1647 cycles</td>
</tr>
<tr>
<td>Radio Firmware</td>
<td>3470 cycles</td>
<td>Processing packet synchronization bytes</td>
<td>8 cycles</td>
</tr>
<tr>
<td>Radio Hardware</td>
<td>255585 cycles</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
invoke the radio firmware to transfer data to the radio send buffer. All these operations are in parallel with the actual data transmission by the radio hardware. Figure 16.9 shows the time spent on each system component for transmitting a packet.

16.5 Deployment

During January 2004, seven complete nodes were successfully deployed at the 100-km² Sweetwaters Game Reserve in central Kenya. During the deployment, data gathered by the system has shown fine-grained zebra movements that had not been available before ZebraNet.

Figure 16.10 shows an example of collected zebra location data. The data points are plotted over a habitat classification on a Landsat image for the Sweetwaters area. The plot shows how fine-grained the spatiotemporal data is. In addition, the location of the points reveals that, at least for this zebra, her behavior at night is somewhat different from that exhibited during the day. Not only is she moving to slightly more brushy habitats (filled squares) after dark, but she is also moving most rapidly as darkness sets in. During the day, most of her GPS fixes are located in areas of open grassland or light acacia bush (white and light grey regions of Figure 16.10). But at night, a significant number of her sightings occur in the brushy habitats (darker areas in Figure 16.10). Since these areas are also characterized by the steepest slopes, these findings suggest that zebras may be trading off reduced acuity for increased agility relative to predators, who are forced to cut and weave on uneven and sloping surfaces. Since fixes were collected every 8 minutes, meaningful zebra velocities could be computed for the first time with remotely recorded data. At least for this zebra, walking during the day, apart from walking to water late in the

![Figure 16.9 Packet-transmission processing time by system components](image-url)
16.5 Deployment

Figure 16.10 An example of GPS location data collected over a one-week period. The pictured area is roughly 10 km × 10 km. (Source: Nasser Olwero of Mpala Research Centre produced the habitat classification map on which we have plotted the position data.)

morning, was done at a leisurely pace. At dawn and dusk (crepuscular periods), however, her speed increased to very high rates as she changed habitat types.

Figure 16.11 shows a photograph of a collared zebra. For reliability and ruggedness, the radio antenna is embedded between the layers of butyl belting that form the collar. A dipole antenna is formed from a strip of copper tape and braid. Since the radio uses a range of frequencies, the dipole's length is tuned for the center frequency of the radio. To reduce absorption into the animal's neck and thereby improve the radio range, we also include a ground plane of woven conductive cloth separated roughly 1 cm from the antenna by a foam dielectric.

Like the radio antenna, the GPS antenna is also sandwiched between the butyl collar layers. Our design experiments show that it operates quite successfully through the butyl layer. We intended for a v-shaped wedge positioned at the animal's throat to seat the collar upright at all times, but unfortunately, during the deployment, we noticed that the collars would rotate, sometimes misdirecting the GPS antenna.

The photograph also illustrates the proper orientation of the solar modules. Each collar has an array of 14 solar modules, connected in parallel. Each module, in turn, is the series, voltage-controlled connection of three solar cells, as described in Section 16.3. Under optimal conditions, each collar's 14 modules can together generate roughly 100 mA at 5 V under full sun. However, since the solar cells wrap
somewhat around the zebra’s neck, we do not reach full charging efficiency and design for roughly half that amount.

We have learned many design lessons from the deployment. In addition to physical design issues related to collar orientation, some communications issues also have also arisen. For example, energy constraints caused us to design in a 2-hour interval between communication times. We hope in the future to implement more event-driven communication intervals to interrogate nodes and to allow for opportunistic data transfers between collars when they are within range. Decreasing the interval, while maintaining the power profile, however, will require the radio to have a significantly lower power profile.

To accomplish this, we plan to replace the radio with separate receiver and transmitter chips. This will allow more flexibility and control over receive and transmit power consumption than what is available in transceiver modules. In receive mode, the current radio drains approximately 50mA at 5V. In our next version, we plan to achieve a drain of less than 10mA at 3V with chips similar to the Xemics DP1201A [21]. This can be accomplished while maintaining the high receiver sensitivity of $-107$ dBm due to recent advances in low-power single chip receivers and by lowering the baud rate. We also plan to use a lower-frequency band (around 150 MHz) to reduce path loss of the radio and extend the radio range even further. Since zebras spend most of the time grazing with the collar close to the ground, lower frequencies are less affected by ground absorption and by obstructions such as small hills. A lower-power radio will also allow us to use a linear regulator and reduce energy consumption while increasing performance.

GPS accuracy also comes into question as some outliers were found. In addition to the filtering scheme previously discussed, we plan to switch to a GPS with a power
profile that will allow it to be kept on for longer periods of time. In our upcoming version, we plan to use the Xemics XE1600 chipset, which consumes an order of magnitude less energy than our current chip.

In the near future, we also plan on switching microcontrollers within the TI MSP430 family. The MSP430F1611, which will be available soon, has 10 KB of RAM, which will be extremely helpful when we replace our flooding protocol with a more selective routing protocol. According to its data sheet, the new microcontroller consumes 330 μA at a voltage of 2.2V and a clock rate of 1 MHz [22], which is 50 μA more than the MSP430F149 under the same conditions [23]. Although we cannot be sure how much more energy will be conserved once the chip is integrated into our system, we feel that the power savings from reducing transmissions with selective routing protocols will almost certainly conserve more energy than the new microcontroller will expend.

16.6 Related Work

We have developed a system that combines aspects from the sensor-networking community and from the mobile ad hoc–networking community. In this section, we look at how our system compares to new technologies in both of these distinct fields.

Sensing hardware. A number of energy-efficient sensor nodes have been developed in the past few years [24–27]. The two devices that most closely parallel our nodes are Berkeley’s Mica2 mote and UC Irvine’s Medusa-MK2.

The Mica2 mote has a 4-MHz, 8-bit processor and uses the same off-chip Flash memory chip and serial interfaces as our nodes. Medusa-MK2 features a dual-microcontroller scheme, which uses the same processor as the Mica2 in situations that require minimal computational power, and a 40-MHz Advanced RISC Machine (ARM) processor to operate its onboard GPS unit and other attachable high-energy consuming sensors.

However, once deployed, both of the aforementioned nodes are intended to remain close together to form a densely populated network. This allows them to use extremely low-power radios with a very limited range. ZebraNet nodes, on the other hand, are intended to be extremely mobile and distributed over a large area. Having a sparsely populated mobile network demands a more powerful radio with a much larger range.

In addition, due to the high power consumption of our radio and GPS, we cannot hope to run the system continuously for months at a time on one set of batteries. Nor can we reduce the duty cycle of data collection and still achieve our objectives. To compensate, we use a rechargeable battery with solar cells distributed around the collar.

Sensing operating systems and middleware. Various operating systems and middleware layers have emerged to control sensor nodes [28–31]. Two such systems that closely relate to Impala are TinyOS and Mate.

TinyOS, the popular operating system designed to run on the motes, has many low-level characteristics in common with Impala. For example, both Impala and
TinyOS place an emphasis on event handling through hardware interrupts and the utilization of on-the-fly processing to conserve memory.

One big difference between our system and TinyOS is a result of the differences in the nodes on which they will be used. The mote is designed to accommodate a variety of interchangeable sensors. This is reflected in TinyOS's emphasis on concurrency-intensive operations that allow the system to handle multiple flows of data from independent sensors simultaneously. Impala uses a combination of polling and interrupt handling to allow for a similar interleaving of scheduled and unscheduled events. In the ZebraNet system, however, Impala takes advantage of the fact that we have a fixed number of sensors that work in a predetermined fashion by using hardware timers to schedule all major events. This allows us to save a great deal of power through the use of the dual-clock scheme and the timely use of energy-hungry components.

Mate is a virtual machine that lies on top of the operating system and is designed to provide a layer of security and a basis for automatically updating nodes via virally propagated programs. ZebraNet does not need the added security Mate provides, but the ability to perform viral software updates was implemented in the original version of Impala [17] designed for palmtop computers and will be implemented on the ZebraNet nodes in the near future.

Protocols and routing schemes. Our peer-to-peer routing scheme has roots in a number of proposed routing methods designed to make communication in mobile ad hoc networks more efficient. DSR [19] and Ad Hoc on Demand Distance Vector (AODV) [32] send route discovery messages that perform similar roles to our peer-discovery messages. The difference is that DSR and AODV attempt to discover a complete route to a destination, whereas our algorithm only attempts to discover a node's immediate neighbors. This modification is essential to our system because under normal circumstances, there will not be a complete route to the base station; rather, data is expected to propagate slowly from node to node until the base station comes into range.

Directed Diffusion [33] uses a data-centric scheme in which messages are passed through the network in a series of independent neighbor-to-neighbor communications very similar to our peer-to-peer transmissions. However, this scheme would not work well in our system because our network has a very low connectivity and our topology is changing too fast for important messages to arrive in a timely fashion.

Sensing application studies. In addition to ZebraNet, there are other concurrently running efforts to use sensors to monitor wildlife or in other mobile applications. The VAFalcons project places solar-powered satellite transmitters on Falcons and uses satellite telemetry to determine the animals' position [34]. Similarly, the Pacific Ocean Salmon Tracking Project places acoustic tags on juvenile salmon [35]. The sound emitted from the tags is recorded by receivers strategically placed along known migration routes and can be used to reconstruct the exact movements of the fish closely. Telenti's Electronic Shepherd project bears considerable similarity to ZebraNet in its goals but has lower-range radios. All of these projects, moreover, rely on a fixed infrastructure to gather data; none harnesses the sort of peer-to-peer node interactions that ZebraNet uses to improve data collection and connectivity.

References

[1] Cav
[2] Cav
[4] Nan
[5] Fis
[6] Fas

16.7 Conclusion

A state deployed habitats and humid in which MAC layer deployment for nodes relevant r

16.7 Conclusion

This chapter system for fielding techniques have do fine-grained energy. c energy-n GPS and array for one degree tools [tr: stand or various ab African ter has c

16.7 Conclusion

A state deployed habitats and humid in which MAC layer deployment for nodes relevant r
A stationary sensor network composed of motes running TinyOS has been deployed on an uninhabited island off the coast of Maine to monitor the nesting habitats of certain birds, along with environmental conditions such as temperature and humidity [7, 36]. The group conducted a successful multiple-month experiment in which data was collected and transmitted through the network using a CSMA MAC layer to protect against collisions. This deployment, along with a similar deployment at the James Reserve in Idyllwild, California [37], is providing the sensor-network community with a great deal of insight into the numerous issues relevant to a real-world deployment.

16.7 Conclusion

This chapter has described our experiences building and deploying the ZebraNet system for habitat monitoring in Africa. ZebraNet utilizes mobile sensor-networking technologies to monitor zebra migrations on energy-constrained hardware. We have developed a highly mobile, energy-efficient sensing system that determines fine-grained positional data and propagates it through the network. To reduce the energy consumption on the board, we have implemented several system-level energy-management techniques. Even with these methods, the high-power of the GPS and the long system-lifetime target requires that our hardware utilize a solar array for recharge.

One of the most sobering lessons learned in wildlife tracking today is the degree to which the technology limitations of widely available observational tools (tracking collars and other sensors) limit the ability of biologists to understand our environment. The goal of the ZebraNet project is to answer key questions about interspecies interactions and resource impacts in the context of African wildlife. Written at the halfway-point of a multiyear project, this chapter has described the design decisions, system status, and deployment experiences of our project thus far.

References


16.7 Conclusion


[37] Center for Embedded Networked Sensing, Research Infrastructure: James Reserve Local Area Power System and Network Enhancements, at http://research.cens.ucla.edu/portal/page?_pageid=59,13783&_dad=portal&_schema=PORTAL.