System Architecture for Wireless Sensor Networks

by

Jason Lester Hill

B.S. (University of California, Berkeley) 1998
M.S. (University of California, Berkeley) 2000

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERISY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor David E. Culler, Chair
Professor Kris Pister
Professor Paul Wright

Spring 2003

The dissertation of Jason Lester Hill is approved:

Chair	Date
	Date
	Date

University of California, Berkeley

Spring 2003

System Architecture for Wireless Sensor Networks

Jason Lester Hill

Abstract

System Architecture for Wireless Sensor Networks

by

Jason Lester Hill

Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor David Culler, Chair

In this thesis we present and operating system and three generations of a hardware platform designed to address the needs of wireless sensor networks. Our operating system, called TinyOS uses an event based execution model to provide support for fine-grained concurrency and incorporates a highly efficient component model. TinyOS enables us to use a hardware architecture that has a single processor time shared between both application and protocol processing. We show how a virtual partitioning of computational resources not only leads to efficient resource utilization but allows for a rich interface between application and protocol processing. This rich interface, in turn, allows developers to exploit application specific communication protocols that significantly improve system performance.

The hardware platforms we develop are used to validate a generalized architecture that is technology independent. Our general architecture contains a single central controller that performs both application and protocol-level processing. For flexibility, this controller is directly connected to the RF transceiver. For efficiency, the controller is

supported by a collection of hardware accelerators that provide basic communication primitives that can be flexibility composed into application specific protocols.

The three hardware platforms we present are instances of this general architecture with varying degrees of hardware sophistication. The Rene platform serves as a baseline and does not contain any hardware accelerators. It allows us to develop the TinyOS operating system concepts and refine its concurrency mechanisms. The Mica node incorporates hardware accelerators that improve communication rates and synchronization accuracy within the constraints of current microcontrollers. As an approximation of our general architecture, we use Mica to validate the underlying architectural principles. The Mica platform has become the foundation for hundreds of wireless sensor network research efforts around the world. It has been sold to more than 250 organizations.

Spec is the most advanced node presented and represents the full realization of our general architecture. It is a 2.5 mm x 2.5 mm CMOS chip that includes processing, storage, wireless communications and hardware accelerators. We show how the careful selection of the correct accelerators can lead to orders-of-magnitude improvements in efficiency without sacrificing flexibility. In addition to performing a theoretical analysis on the strengths of our architecture, we demonstrate its capabilities through a collection of real-world application deployments.

_____

Professor David Culler
Dissertation Committee Chair

Table of Contents

List of Figures:

Chapter 1: Introduction

The emerging field of wireless sensor networks combines sensing, computation, and communication into a single tiny device. Through advanced mesh networking protocols, these devices form a sea of connectivity that extends the reach of cyberspace out into the physical world. As water flows to fill every room of a submerged ship, the mesh networking connectivity will seek out and exploit any possible communication path by hopping data from node to node in search of its destination. While the capabilities of any single device are minimal, the composition of hundreds of devices offers radical new technological possibilities.

The power of wireless sensor networks lies in the ability to deploy large numbers of tiny nodes that assemble and configure themselves. Usage scenarios for these devices range from real-time tracking, to monitoring of environmental conditions, to ubiquitous computing environments, to *in situ* monitoring of the health of structures or equipment. While often referred to as wireless sensor networks, they can also control actuators that extend control from cyberspace into the physical world.

The most straightforward application of wireless sensor network technology is to monitor remote environments for low frequency data trends. For example, a chemical plant could be easily monitored for leaks by hundreds of sensors that automatically form a wireless interconnection network and immediately report the detection of any chemical leaks. Unlike traditional wired systems, deployment costs would be minimal. Instead of having to deploy thousands of feet of wire routed through protective conduit, installers

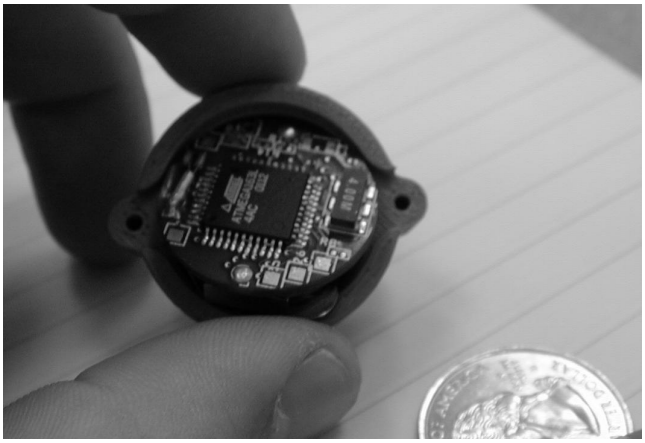

Figure 1-1: DOT – Wireless sensor network device designed to be the approximate size of a quarter. Future devices will continue to be smaller, cheaper and longer lasting.

simply have to place quarter-sized device, such as the one pictured in Figure 1-1, at each sensing point. The network could be incrementally extended by simply adding more devices – no rework or complex configuration. With the devices presented in this thesis, the system would be capable of monitoring for anomalies for several years on a single set of batteries.

In addition to drastically reducing the installation costs, wireless sensor networks have the ability to dynamically adapt to changing environments. Adaptation mechanisms can respond to changes in network topologies or can cause the network to shift between drastically different modes of operation. For example, the same embedded network performing leak monitoring in a chemical factory might be reconfigured into a network designed to localize the source of a leak and track the diffusion of poisonous gases. The network could then direct workers to the safest path for emergency evacuation.

Current wireless systems only scratch the surface of possibilities emerging from the integration of low-power communication, sensing, energy storage, and computation.

Generally, when people consider wireless devices they think of items such as cell phones, personal digital assistants, or laptops with 802.11. These items costs hundreds of dollars, target specialized applications, and rely on the pre-deployment of extensive infrastructure support. In contrast, wireless sensor networks use small, low-cost embedded devices for a wide range of applications and do not rely on any pre-existing infrastructure. The vision is that these devise will cost less that $1 by 2005.

Unlike traditional wireless devices, wireless sensor nodes do not need to communicate directly with the nearest high-power control tower or base station, but only with their local peers. Instead, of relying on a pre-deployed infrastructure, each individual sensor or actuator becomes part of the overall infrastructure. Peer-to-peer networking protocols provide a mesh-like interconnect to shuttle data between the thousands of tiny embedded devices in a multi-hop fashion. The flexible mesh architectures envisioned dynamically adapt to support introduction of new nodes or expand to cover a larger geographic region. Additionally, the system can automatically adapt to compensate for node failures.

The vision of mesh networking is based on strength in numbers. Unlike cell phone systems that deny service when too many phones are active in a small area, the interconnection of a wireless sensor network only grows stronger as nodes are added. As long as there is sufficient density, a single network of nodes can grow to cover limitless area. With each node having a communication range of 50 meters and costing less that $1 a sensor network that encircled the equator of the earth will cost less that $1M.

Figure 1-2: Possible deployment of ac-hoc wireless embedded network for precision agriculture. Sensors detect temperature, light levels and soil moisture at hundreds of points across a field and communicate their data over a multi-hop network for analysis.

An example network is shown in Figure 1-2. It depicts a precision agriculture deployment—an active area of application research. Hundreds of nodes scattered throughout a field assemble together, establish a routing topology, and transmit data back to a collection point. The application demands for robust, scalable, low-cost and easy to deploy networks are perfectly met by a wireless sensor network. If one of the nodes should fail, a new topology would be selected and the overall network would continue to deliver data. If more nodes are placed in the field, they only create more potential routing opportunities.

There is extensive research in the development of new algorithms for data aggregation [1], ad hoc routing [2-4], and distributed signal processing in the context of wireless sensor networks [5, 6]. As the algorithms and protocols for wireless sensor network are developed, they must be supported by a low-power, efficient and flexible hardware platform.

This thesis focuses on developing the system architecture required to meet the needs of wireless sensor networks. A core design challenge in wireless sensor networks is coping with the harsh resource constraints placed on the individual devices. Embedded

processors with kilobytes of memory must implement complex, distributed, ad-hoc networking protocols. Many constraints derive from the vision that these devices will be produced in vast quantities and must be small and inexpensive. As Moore's law marches on, each device will get smaller, not just grow more powerful at a given size. Size reduction is essential in order to allow devices to be produced as inexpensively as possible, as well as to be able to allow devices to be used in a wide range of application scenarios.

The most difficult resource constraint to meet is power consumption. As physical size decreases, so does energy capacity. Underlying energy constraints end up creating computational and storage limitations that lead to a new set of architectural issues. Many devices, such as cell phones and pagers, reduce their power consumption through the use specialized communication hardware in ASICs that provide low-power implementations of the necessary communication protocols [7] and by relying on high-power infrastructure. However, the strength of wireless sensor networks is their flexibility and universality. The wide range of applications being targeted makes it difficult to develop a single protocol, and in turn, an ASIC, that is efficient for all applications. A wireless sensor network platform must provide support for a suite of application-specific protocols that drastically reduce node size, cost, and power consumption for their target application.

The wireless sensor network architecture we present here includes both a hardware platform and an operating system designed specifically to address the needs of wireless sensor networks. TinyOS is a component based operating system designed to run in resource constrained wireless devices. It provides highly efficient communication

primitives and fine-grained concurrency mechanisms to application and protocol developers. A key concept in TinyOS is the use of event based programming in conjunction with a highly efficient component model. TinyOS enables system-wide optimization by providing a tight coupling between hardware and software, as well as flexible mechanisms for building application specific modules.

TinyOS has been designed to run on a generalized architecture where a single CPU is shared between application and protocol processing. We detail three generations of wireless nodes and a host of application deployments that have proven the capabilities of our general system architecture. Figure 1-3 lays a picture and timeline of several "mote" generations. The Mica platform has been produced in the largest quantities – over 5000 Mica nodes have been produced and distributed to over 250 companies and research organizations from around the country. The Mica platform includes a low power transceiver, a power management subsystem, extended storage and an embedded microcontroller.

The most advanced hardware platform we present is a single-chip CMOS device that integrates the processing, storage and communication capabilities to form a complete system node. This single chip node – called Spec – measures just 2.5 mm x 2.5 mm, contains a microcontroller, transmitter, ADC, general purpose I/O ports, UART, memory and encryption engine. The tiny chip only needs to be supported by a 32 KHz watch crystal, an off-chip inductor and a power supply, a battery and a 4 MHz clock. The Spec node represents the coming generation of wireless sensor nodes that will be manufactured for pennies and deployed in the millions.

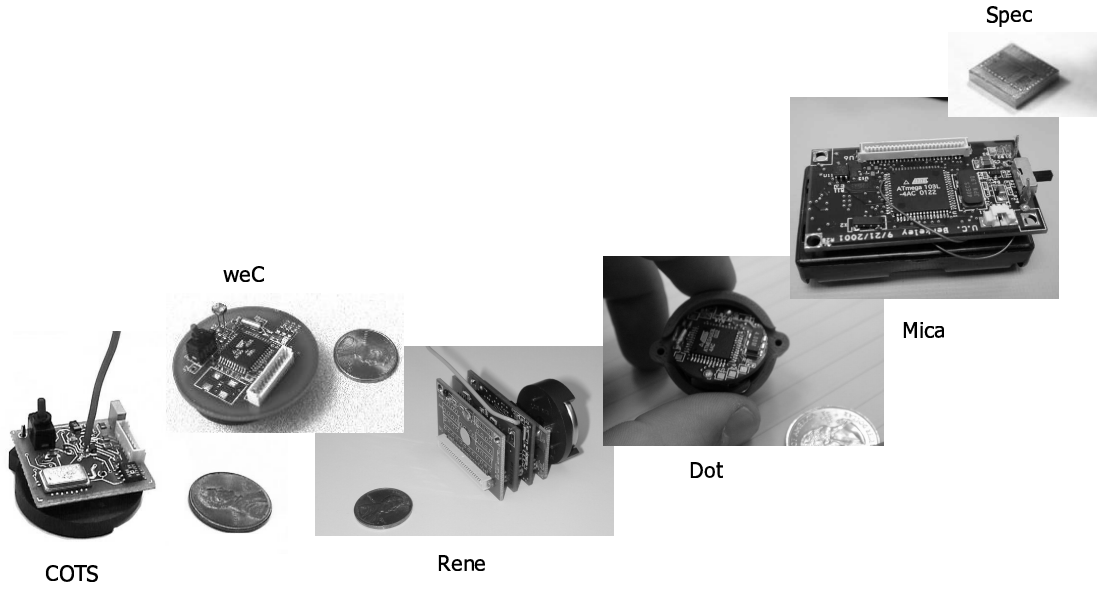

Figure 1-3: Design lineage of Mote Technology. COTS (Common off the shelf) prototypes lead to the weC platform. Rene then evolved to allow sensor expansion and enabled hundreds of compelling applications. The Dot node was architecturally the same as Rene but shrunk into a quarter-sized device. Mica – discussed in depth in this thesis – made significant architectural improvements in order to increase performance and efficiency. Spec represents the complete integrated CMOS vision.

Both the Mica and Spec node are used to substantiate our claim that optimal system architecture for wireless sensor networks is to have a single central controller directly connected to a low-power radio transceiver through a rich interface that supports hardware assistance for communication primitives. In contrast to having a hierarchical partition of hardware resources dedicated to specific functions, a single shared controller performs all processing. This allows for the dynamic allocation of computation resources to the set of computational tasks demanded by the system. The layers of abstraction typically achieved through hardware partitioning can instead be achieved through the use of a highly efficient software-based component model. Software abstractions allow for a wider scope of cross-layer optimizations that can achieve orders of magnitude improvements is system performance. The power and viability of this architecture is demonstrated through a collection benchmarks performed on real-world hardware and in application level deployments.

The main contributions of this work are (1) a general architecture that meets the strict efficiency and flexibility requirements of wireless sensor networks, (2) an implementation of the architecture using current microcontroller and low-power radio technology, (3) an operating system that compliments the hardware capabilities and provides support for sensor network applications, (4) an integrated single chip CMOS hardware platform for use in wireless sensor networks and (5) a demonstration of the flexibility of this architecture on several novel demonstration applications.

This thesis is organized in 9 chapters. Chapter 2 presents three key application scenarios and an overview of the requirements for wireless sensor networks. It is intended to provide the background necessary for a general understanding of the issues discussed in later chapters. Additionally, Chapter 2 provides a general outline of the capabilities of modern hardware building blocks. It culminates in describing a first-generation sensor node used as a baseline for comparison. Chapter 3 presents the overall architecture for the TinyOS operating system and a list of included components. TinyOS is an event based operating system with a highly efficient component model. It plays a critical role in exposing the capabilities of the hardware architectures we present. Chapter 4 discusses the critical design issues that must be addressed by a wireless sensor network platform. We discuss the shortcomings of traditional wireless architectures and present a generalized architecture that addresses these issues. Our generalized architecture is designed to allow for flexibility without sacrificing efficiency.

In Chapter 5 we present the Mica platform – an off-the-shelf approximation of our generalized architecture. Mica represents what is possible with commercial hardware and allows us to validate the basic principles of our generalized architecture. Mica has

bee successfully used in hundreds of real-world sensor network deployments. Chapter 6 presents the single chip Spec node where communication, computation, and storage are combined onto a 2.5 mm x 2.5 mm die. Spec is a full realization of the generalized architecture presented. Spec represents the future of wireless sensor network hardware. Chapter 7 presets an overview and analysis of several demonstration applications. This analysis underscores the performance impact of the architecture presented. Additionally the applications presented demonstrate the flexibility and validate the platform.

Chapter 8 provides a survey of related work. This includes a summary of research efforts that have been layered on top of the system we present as well as other wireless activities. Chapter 9 summarizes the thesis and concludes with a prediction of future technological trends.

Chapter 2: Wireless Sensor Networks

The concept of wireless sensor networks is based on a simple equation:

Sensing + CPU + Radio = Thousands of potential applications

As soon as people understand the capabilities of a wireless sensor network, hundreds of applications spring to mind. It seems like a straightforward combination of modern technology.

However, actually combining sensors, radios, and CPU's into an effective wireless sensor network requires a detailed understanding of the both capabilities and limitations of each of the underlying hardware components, as well as a detailed understanding of modern networking technologies and distributed systems theory. Each individual node must be designed to provide the set of primitives necessary to synthesize the interconnected web that will emerge as they are deployed, while meeting strict requirements of size, cost and power consumption. A core challenge is to map the overall system requirements down to individual device capabilities, requirements and actions. To make the wireless sensor network vision a reality, an architecture must be developed that synthesizes the envisioned applications out of the underlying hardware capabilities.

To develop this system architecture we work from the high level application requirements down through the low-level hardware requirements. In this process we first attempt to understand the set of target applications. To limit the number of applications that we must consider, we focus on a set of application classes that we believe are representative of a large fraction of the potential usage scenarios. We use this set of

application classes to explore the system-level requirements that are placed on the overall architecture. From these system-level requirements we can then drill down into the individual node-level requirements. Additionally, we must provide a detailed background into the capabilities of modern hardware.

After we present the raw hardware capabilities, we present a basic wireless sensor node. The Rene node represents a first cut at a system architecture, and is used for comparison against the system architectures presented in later chapters.

2.1 Sensor network application classes

The three application classes we have selected are: environmental data collection, security monitoring, and sensor node tracking. We believe that the majority of wireless sensor network deployments will fall into one of these class templates.

2.1.1 Environmental Data Collection

A canonical environmental data collection application is one where a research scientist wants to collect several sensor readings from a set of points in an environment over a period of time in order to detect trends and interdependencies. This scientist would want to collect data from hundreds of points spread throughout the area and then analyze the data offline [8, 9]. The scientist would be interested in collecting data over several months or years in order to look for long-term and seasonal trends. For the data to be meaningful it would have to be collected at regular intervals and the nodes would remain at known locations.

At the network level, the environmental data collection application is characterized by having a large number of nodes continually sensing and transmitting

data back to a set of base stations that store the data using traditional methods. These networks generally require very low data rates and extremely long lifetimes. In typical usage scenario, the nodes will be evenly distributed over an outdoor environment. This distance between adjacent nodes will be minimal yet the distance across the entire network will be significant.

After deployment, the nodes must first discover the topology of the network and estimate optimal routing strategies [10]. The routing strategy can then be used to route data to a central collection points. In environmental monitoring applications, it is not essential that the nodes develop the optimal routing strategies on their own. Instead, it may be possible to calculate the optimal routing topology outside of the network and then communicate the necessary information to the nodes as required. This is possible because the physical topology of the network is relatively constant. While the time variant nature of RF communication may cause connectivity between two nodes to be intermittent, the overall topology of the network will be relatively stable.

Environmental data collection applications typically use tree-based routing topologies where each routing tree is rooted at high-capability nodes that sink data. Data is periodically transmitted from child node to parent node up the tree-structure until it reaches the sink. With tree-based data collection each node is responsible for forwarding the data of all its descendants. Nodes with a large number of descendants transmit significantly more data than leaf nodes. These nodes can quickly become energy bottlenecks [11, 12].

Once the network is configured, each node periodically samples its sensors and transmits its data up the routing tree and back to the base station. For many scenarios, the

interval between these transmissions can be on the order of minutes. Typical reporting periods are expected to be between 1 and 15 minutes; while it is possible for networks to have significantly higher reporting rates. The typical environment parameters being monitored, such as temperature, light intensity, and humidity, do not change quickly enough to require higher reporting rates.

In addition to large sample intervals, environmental monitoring applications do not have strict latency requirements. Data samples can be delayed inside the network for moderate periods of time without significantly affecting application performance. In general the data is collected for future analysis, not for real-time operation.

In order to meet lifetime requirements, each communication event must be precisely scheduled. The senor nodes will remain dormant a majority of the time; they will only wake to transmit or receive data. If the precise schedule is not met, the communication events will fail.

As the network ages, it is expected that nodes will fail over time. Periodically the network will have to reconfigure to handle node/link failure or to redistribute network load. Additionally, as the researchers learn more about the environment they study, they may want to go in and insert additional sensing points. In both cases, the reconfigurations are relatively infrequent and will not represent a significant amount of the overall system energy usage.

The most important characteristics of the environmental monitoring requirements are long lifetime, precise synchronization, low data rates and relatively static topologies. Additionally it is not essential that the data be transmitted in real-time back to the central

collection point. The data transmissions can be delayed inside the network as necessary in order to improve network efficiency.

2.1.2 Security Monitoring

Our second class of sensor network application is security monitoring. Security monitoring networks are composed of nodes that are placed at fixed locations throughout an environment that continually monitor one or more sensors to detect an anomaly. A key difference between security monitoring and environmental monitoring is that security networks are not actually collecting any data. This has a significant impact on the optimal network architecture. Each node has to frequently check the status of its sensors but it only has to transmit a data report when there is a security violation. The immediate and reliable communication of alarm messages is the primary system requirement. These are “report by exception” networks.

Additionally, it is essential that it is confirmed that each node is still present and functioning. If a node were to be disabled or fail, it would represent a security violation that should be reported. For security monitoring applications, the network must be configured so that nodes are responsible for confirming the status of each other. One approach is to have each node be assigned to peer that will report if a node is not functioning. The optimal topology of a security monitoring network will look quite different from that of a data collection network.

In a collection tree, each node must transmit the data of all of its decedents. Because of this, it is optimal to have a short, wide tree. In contrast, with a security network the optimal configuration would be to have a linear topology that forms a Hamiltonian cycle of the network. The power consumption of each node is only

proportional to the number of children it has. In a linear network, each node would have only one child. This would evenly distribute the energy consumption of the network.

The accepted norm for security systems today is that each sensor should be checked approximately once per hour. Combined with the ability to evenly distribute the load of checking nodes, the energy cost of performing this check becomes minimal. A majority of the energy consumption in a security network is spent on meeting the strict latency requirements associated with the signaling the alarm when a security violation occurs.

Once detected, a security violation must be communicated to the base station immediately. The latency of the data communication across the network to the base station has a critical impact on application performance. Users demand that alarm situations be reported within seconds of detection. This means that network nodes must be able to respond quickly to requests from their neighbors to forward data.

In security networks reducing the latency of an alarm transmission is significantly more important than reducing the energy cost of the transmissions. This is because alarm events are expected to be rare. In a fire security system alarms would almost never be signaled. In the event that one does occur a significant amount of energy could be dedicated to the transmission. Reducing the transmission latency leads to higher energy consumption because routing nodes must monitor the radio channel more frequently.

In security networks, a vast majority of the energy will be spend on confirming the functionality of neighboring nodes and in being prepared to instantly forward alarm

announcements. Actual data transmission will consume a small fraction of the network energy.

2.1.3 Node tracking scenarios

A third usage scenario commonly discussed for sensor networks is the tracking of a tagged object through a region of space monitored by a sensor network. There are many situations where one would like to track the location of valuable assets or personnel. Current inventory control systems attempt to track objects by recording the last checkpoint that an object passed through. However, with these systems it is not possible to determine the current location of an object. For example, UPS tracks every shipment by scanning it with a barcode whenever it passes through a routing center. The system breaks down when objects do not flow from checkpoint to checkpoint. In typical work environments it is impractical to expect objects to be continually passed through checkpoints.

With wireless sensor networks, objects can be tracked by simply tagging them with a small sensor node. The sensor node will be tracked as it moves through a field of sensor nodes that are deployed in the environment at known locations. Instead of sensing environmental data, these nodes will be deployed to sense the RF messages of the nodes attached to various objects. The nodes can be used as active tags that announce the presence of a device. A database can be used to record the location of tracked objects relative to the set of nodes at known locations. With this system, it becomes possible to ask where an object is currently, not simply where it was last scanned [13].

Unlike sensing or security networks, node tracking applications will continually have topology changes as nodes move through the network. While the connectivity

between the nodes at fixed locations will remain relatively stable, the connectivity to mobile nodes will be continually changing. Additionally the set of nodes being tracked will continually change as objects enter and leave the system. It is essential that the network be able to efficiently detect the presence of new nodes that enter the network.

2.1.4 Hybrid networks

In general, complete application scenarios contain aspects of all three categories. For example, in a network designed to track vehicles that pass through it, the network may switch between being an alarm monitoring network and a data collection network. During the long periods of inactivity when no vehicles are present, the network will simply perform an alarm monitoring function. Each node will monitor its sensors waiting to detect a vehicle. Once an alarm event is detected, all or part of the network, will switch into a data collection network and periodically report sensor readings up to a base station that track the vehicles progress. Because of this multi-modal network behavior, it is important to develop a single architecture that and handle all three of these application scenarios.

2.2 System Evaluation Metrics

Now that we have established the set of application scenarios that we are addressing, we explore the evaluation metrics that will be used to evaluate a wireless sensor network. To do this we keep in mind the high-level objectives of the network deployment, the intended usage of the network, and the key advantages of wireless sensor networks over existing technologies. The key evaluation metrics for wireless sensor

networks are lifetime, coverage, cost and ease of deployment, response time, temporal accuracy, security, and effective sample rate. Their importance is discussed below.

One result is that many of these evaluation metrics are interrelated. Often it may be necessary to decrease performance in one metric, such as sample rate, in order to increase another, such as lifetime. Taken together, this set of metrics form a multidimensional space that can be used to describe the capabilities of a wireless sensor network. The capabilities of a platform are represented by a volume in this multidimensional space that contains all of the valid operating points. In turn, a specific application deployment is represented by a single point. A system platform can successfully perform the application if and only if the application requirements point lies inside the capability hyperspace.

One goal of this chapter is to present an understanding of the tradeoffs that link each axis of this space and an understanding of current capabilities. The architectural improvements and optimizations we present in later chapters are then motivated by increasing the ability to deliver these capabilities and increasing the volume of the capability hypercube.

2.2.1 Lifetime

Critical to any wireless sensor network deployment is the expected lifetime. The goal of both the environmental monitoring and security application scenarios is to have nodes placed out in the field, unattended, for months or years.

The primary limiting factor for the lifetime of a sensor network is the energy supply. Each node must be designed to manage its local supply of energy in order to maximize total network lifetime. In many deployments it is not the average node lifetime

that is important, but rather the minimum node lifetime. In the case of wireless security systems, every node must last for multiple years. A single node failure would create a vulnerability in the security systems.

In some situations it may be possible to exploit external power, perhaps by tapping into building power with some or all nodes. However, one of the major benefits to wireless systems is the ease of installation. Requiring power to be supplied externally to all nodes largely negates this advantage. A compromise is to have a handful of special nodes that are wired into the building's power infrastructure.

In most application scenarios, a majority of the nodes will have to be self-powered. They will either have to contain enough stored energy to last for years, or they will have to be able to scavenge energy from the environment through devices, such as solar cells or piezoelectric generators [14, 15]. Both of these options demand that that the average energy consumption of the nodes be as low as possible.

The most significant factor in determining lifetime of a given energy supply is radio power consumption. In a wireless sensor node the radio consumes a vast majority of the system energy. This power consumption can be reduced through decreasing the transmission output power or through decreasing the radio duty cycle. Both of these alternatives involve sacrificing other system metrics.

2.2.2 Coverage

Next to lifetime, coverage is the primary evaluation metric for a wireless network. It is always advantageous to have the ability to deploy a network over a larger physical area. This can significantly increase a system's value to the end user. It is important to keep in mind that the coverage of the network is not equal to the range of the wireless

communication links being used. Multi-hop communication techniques can extend the coverage of the network well beyond the range of the radio technology alone. In theory they have the ability to extend network range indefinitely. However, for a given transmission range, multi-hop networking protocols increase the power consumption of the nodes, which may decrease the network lifetime. Additionally, they require a minimal node density, which may increase the deployment cost.

Tied to range is a network's ability to scale to a large number of nodes. Scalability is a key component of the wireless sensor network value proposition. A user can deploy a small trial network at first and then can continually add sense points to collect more and different information. A user must be confident that the network technology being used is capable of scaling to meet his eventual need. Increasing the number of nodes in the system will impact either the lifetime or effective sample rate. More sensing points will cause more data to be transmitted which will increase the power consumption of the network. This can be offset by sampling less often.

2.2.3 Cost and ease of deployment

A key advantage of wireless sensor networks is their ease of deployment. Biologists and construction workers installing networks cannot be expected to understand the underlying networking and communication mechanisms at work inside the wireless network. For system deployments to be successful, the wireless sensor network must configure itself. It must be possible for nodes to be placed throughout the environment by an untrained person and have the system simply work.

Ideally, the system would automatically configure itself for any possible physical node placement. However, real systems must place constraints on actual node

placements – it is not possible to have nodes with infinite range. The wireless sensor network must be capable of providing feedback as to when these constraints are violated. The network should be able to assess quality of the network deployment and indicate any potential problems. This translates to requiring that each device be capable of performing link discovery and determining link quality.

In addition to an initial configuration phase, the system must also adapt to changing environmental conditions. Throughout the lifetime of a deployment, nodes may be relocated or large physical objects may be placed so that they interfere with the communication between two nodes. The network should be able to automatically reconfigure on demand in order to tolerate these occurrences.

The initial deployment and configuration is only the first step in the network lifecycle. In the long term, the total cost of ownership for a system may have more to do with the maintenance cost than the initial deployment cost. The security application scenario in particular requires that the system be extremely robust. In addition to extensive hardware and software testing prior to deployment, the sensor system must be constructed so that it is capable of performing continual self-maintenance. When necessary, it should also be able to generate requests when external maintenance is required.

In a real deployment, a fraction of the total energy budget must be dedicated to system maintenance and verification. The generation of diagnostic and reconfiguration traffic reduces the network lifetime. It can also decrease the effective sample rate.

2.2.4 Response Time

Particularly in our alarm application scenario, system response time is a critical performance metric. An alarm must be signaled immediately when an intrusion is detected. Despite low power operation, nodes must be capable of having immediate, high-priority messages communicated across the network as quickly as possible. While these events will be infrequent, they may occur at any time without notice. Response time is also critical when environmental monitoring is used to control factory machines and equipment. Many users envision wireless sensor networks as useful tools for industrial process control. These systems would only be practical if response time guarantees could be met.

The ability to have low response time conflicts with many of the techniques used to increase network lifetime. Network lifetime can be increased by having nodes only operate their radios for brief periods of time. If a node only turns on its radio once per minute to transmit and receive data, it would be impossible to meet the application requirements for response time of a security system.

Response time can be improved by including nodes that are powered all the time. These nodes can listen for the alarm messages and forward them down a routing backbone when necessary. This, however, reduces the ease of deployment for the system.

2.2.5 Temporal Accuracy

In environmental and tracking applications, samples from multiple nodes must be cross-correlated in time in order to determine the nature of phenomenon being measured. The necessary accuracy of this correlation mechanism will depend on the rate of

propagation of the phenomenon being measured. In the case of determining the average temperature of a building, samples must only be correlated to within seconds. However, to determine how a building reacts to a seismic event, millisecond accuracy is required.

To achieve temporal accuracy, a network must be capable of constructing and maintaining a global time base that can be used to chronologically order samples and events. In a distributed system, energy must be expended to maintain this distributed clock. Time synchronization information must be continually communicated between nodes. The frequency of the synchronization messages is dependent on the desired accuracy of the time clock. The bottom line is maintenance of a distributed time base requires both power and bandwidth.

2.2.6 Security

Despite the seemingly harmless nature of simple temperature and light information from an environmental monitoring application, keeping this information secure can be extremely important. Significant patterns of building use and activity can be easily extracted from a trace of temperature and light activity in an office building. In the wrong hands, this information can be exploited to plan a strategic or physical attack on a company. Wireless sensor networks must be capable of keeping the information they are collecting private from eavesdropping.

As we consider security oriented applications, data security becomes even more significant. Not only must the system maintain privacy, it must also be able to authenticate data communication. It should not be possible to introduce a false alarm message or to replay an old alarm message as a current one. A combination of privacy

and authentication is required to address the needs of all three scenarios. Additionally, it should not be possible to prevent proper operation by interfering with transmitted signals.

Use of encryption and cryptographic authentication costs both power and network bandwidth [16, 17]. Extra computation must be performed to encrypt and decrypt data and extra authentication bits must be transmitted with each packet. This impacts application performance by decreasing the number of samples than can be extracted from a given network and the expected network lifetime.

2.2.7 Effective Sample Rate

In a data collection network, effective sample rate is a primary application performance metric. We define the effective sample rate as the sample rate that sensor data can be taken at each individual sensor and communicated to a collection point in a data collection network. Fortunately, environmental data collection applications typically only demand sampling rates of 1-2 samples per minute. However, in addition to the sample rate of a single sensor, we must also consider the impact of the multi-hop networking architectures on a nodes ability to effectively relay the data of surrounding nodes.

In a data collection tree, a node must handle the data of all of its descendents. If each child transmits a single sensor reading and a node has a total of 60 descendants, then it will be forced to transmit 60 times as much data. Additionally, it must be capable of receiving those 60 readings in a single sample period. This multiplicative increase in data communication has a significant effect on system requirements. Network bit rates combined with maximum network size end up impacting the effective per-node sample rate of the complete system [5].

One mechanism for increasing the effective sample rate beyond the raw communication capabilities of the network is to exploit in-network processing. Various forms of spatial and temporal compression can be used to reduce the communication bandwidth required while maintaining the same effective sampling rate. Additionally local storage can be used to collect and store data at a high sample rate for short periods of time. In-network data processing can be used to determine when an "interesting" event has occurred and automatically trigger data storage. The data can then be downloaded over the multi-hop network as bandwidth allows.

Triggering is the simplest form of in-network processing. It is commonly used in security systems. Effectively, each individual sensor is sampled continuously, processed, and only when a security breach has occurred is data transmitted to the base station. If there were no local computation, a continuous stream of redundant sensor readings would have to be transmitted. We show how this same process can be extended to complex detection events.

2.3 Individual node evaluation metrics

Now that we have established the set of metrics that will be used to evaluate the performance of the sensor network as a whole, we can attempt to link the system performance metrics down to the individual node characteristics that support them. The end goal is to understand how changes to the low-level system architecture impact application performance. Just as application metrics are often interrelated, we will see that an improvement in one node-level evaluation metric (e.g., range) often comes at the expense of another (e.g., power).

2.3.1 Power

To meet the multi-year application requirements individual sensor nodes must be incredibly low-power. Unlike cell phones, with average power consumption measured in hundreds of milliamps and multi-day lifetimes, the average power consumption of wireless sensor network nodes must be measured in micro amps. This ultra-low-power operation can only be achieved by combining both low-power hardware components and low duty-cycle operation techniques.

During active operation, radio communication will constitute a significant fraction of the node's total energy budget. Algorithms and protocols must be developed to reduce radio activity whenever possible. This can be achieved by using localized computation to reduce the streams of data being generated by sensors and through application specific protocols. For example, events from multiple sensor nodes can be combined together by a local group of nodes before transmitting a single result across the sensor network.

Our discussion on available energy sources will show that a node must consume less that 200 uA on average to last for one year on a pair of AA batteries. In contrast the average power consumption of a cell phone is typically more than 4000 uA, a 20 fold difference.

2.3.2 Flexibility

The wide range of usage scenarios being considered means that the node architecture must be flexible and adaptive. Each application scenario will demand a slightly different mix of lifetime, sample rate, response time and in-network processing. A wireless sensor network architecture must be flexible enough to accommodate a wide

range of application behaviors. Additionally, for cost reasons each device will have only the hardware and software it actually needs for a given the application. The architecture must make it easy to assemble just the right set of software and hardware components. Thus, these devices require an unusual degree of hardware and software modularity while simultaneously maintaining efficiency.

2.3.3 Robustness

In order to support the lifetime requirements demanded, each node must be constructed to be as robust as possible. In a typical deployment, hundreds of nodes will have to work in harmony for years. To achieve this, the system must be constructed so that it can tolerate and adapt to individual node failure. Additionally, each node must be designed to be as robust as possible.

System modularity is a powerful tool that can be used to develop a robust system. By dividing system functionality into isolated sub-pieces, each function can be fully tested in isolation prior to combining them into a complete application. To facilitate this, system components should be as independent as possible and have interfaces that are narrow, in order to prevent unexpected interactions.

In addition to increasing the system's robustness to node failure, a wireless sensor network must also be robust to external interference. As these networks will often co-exist with other wireless systems, they need the ability to adapt their behavior accordingly. The robustness of wireless links to external interference can be greatly increased through the use of multi-channel and spread spectrum radios. It is common for facilities to have existing wireless devices that operate on one or more frequencies. The

ability to avoid congested frequencies is essential in order to guarantee a successful deployment.

2.3.4 Security

In order to meet the application level security requirements, the individual nodes must be capable of performing complex encrypting and authentication algorithms. Wireless data communication is easily susceptible to interception. The only way to keep data carried by these networks private and authentic is to encrypt all data transmissions. The CPU must be capable of performing the required cryptographic operations itself or with the help of included cryptographic accelerators [16].

In addition to securing all data transmission, the nodes themselves must secure the data that they contain. While they will not have large amounts of application data stored internally, they will have to store secret encryption keys used in the network. If these keys are revealed, the security of the network could crumble. To provide true security, it must be difficult to extract the encryption keys of from any node.

2.3.5 Communication

A key evaluation metric for any wireless sensor network is its communication rate, power consumption, and range. While we have made the argument that the coverage of the network is not limited by the transmission range of the individual nodes, the transmission range does have a significant impact on the minimal acceptable node density. If nodes are placed too far apart it may not be possible to create an interconnected network or one with enough redundancy to maintain a high level of reliability. Most application scenarios have natural node densities that correspond to the

granularity of sensing that is desired. If the radio communications range demands a higher node density, additional nodes must be added to the system in to increase node density to a tolerable level.

The communication rate also has a significant impact on node performance. Higher communication rates translate into the ability to achieve higher effective sampling rates and lower network power consumption. As bit rates increase, transmissions take less time and therefore potentially require less energy. However, an increase in radio bit rate is often accompanied by an increase in radio power consumption. All things being equal, a higher transmission bit rate will result in higher system performance. However, we show later that an increase in the communication bit rate has a significant impact on the power consumption and computational requirement of the node. In total, the benefits of an increase in bit rate can be offset by several other factors.

2.3.6 Computation

The two most computationally intensive operations for a wireless sensor node are the in-network data processing and the management of the low-level wireless communication protocols. As we discuss later, there are strict real-time requirements associated with both communication and sensing. As data is arriving over the network, the CPU must simultaneously control the radio and record/decode the incoming data. Higher communication rates required faster computation.

The same is true for processing being performed on sensor data. Analog sensors can generate thousands of samples per second. Common sensor processing operations include digital filtering, averaging, threshold detection, correlation and spectral analysis.

It may even be necessary to perform a real-time FFT on incoming data in order to detect a high-level event.

In addition to being able to locally process, refine and discard sensor readings, it can be beneficial to combine data with neighboring sensors before transmission across a network. Just as complex sensor waveforms can be reduced to key events, the results from multiple nodes can be synthesized together. This in-network processing requires additional computational resources.

In our experience, 2-4 MIPS of processing are required to implement the radio communication protocols used in wireless sensor networks. Beyond that, the application data processing can consume an arbitrary amount of computation depending on the calculations being performed.

2.3.7 Time Synchronization

In order to support time correlated sensor readings and low-duty cycle operation of our data collection application scenario, nodes must be able to maintain precise time synchronization with other members of the network. Nodes need to sleep and awake together so that they can periodically communicate. Errors in the timing mechanism will create inefficiencies that result in increased duty cycles.

In distributed systems, clocks drift apart over time due to inaccuracies in timekeeping mechanisms. Depending on temperature, voltage, humidity, time keeping oscillators operate at slightly different frequencies. High-precision synchronization mechanisms must be provided to continually compensate for these inaccuracies.

2.3.8 Size & Cost

The physical size and cost of each individual sensor node has a significant and direct impact on the ease and cost of deployment. Total cost of ownership and initial deployment cost are two key factors that will drive the adoption of wireless sensor network technologies. In data collection networks, researchers will often be operating off of a fixed budget. Their primary goal will be to collect data from as many locations as possible without exceeding their fixed budget. A reduction in per-node cost will result in the ability to purchase more nodes, deploy a collection network with higher density, and collect more data.

Physical size also impacts the ease of network deployment. Smaller nodes can be placed in more locations and used in more scenarios. In the node tracking scenario, smaller, lower cost nodes will result in the ability to track more objects.

2.4 Hardware Capabilities

Now that we have identified the key characteristics of a wireless sensor node we can look at the capabilities of modern hardware. This allows us to understand what bit rate, power consumption, memory and cost we can expect to achieve. A balance must be maintained between capability, power consumption and size in order to best address application needs. This section gives a quick overview of modern technology and the tradeoffs between different technologies. We start with a background of energy storage technologies and continue through the radio, CPU, and sensors.

2.4.1 Energy

Just as power consumption of system components are often expressed in milliamps, batteries are generally rated in milliamp-hours (mAh). In theory a 1000 mAh battery could support a processor consuming 10 mA for 100 hours. In practice this in not always true. Due to battery chemistry, voltage and current levels vary depending on how the energy is extracted from a battery. Additionally, as batteries discharge their voltage drops. If the system is not tolerant to a decrease in voltage it may not be possible to use the full rated capacity of a battery. For example, a 1.5 V alkaline battery is not considered empty by the manufacturer until it is outputting only .8 V [18].

2.4.1.1 Battery technologies

There are three common battery technologies that are applicable for wireless sensor networks – Alkaline, Lithium, and Nickel Metal Hydride. An AA Alkaline battery is rated at 1.5 V, but during operation it ranges from 1.65 to .8 V as shown in Figure 2-1 and is rated at 2850 mAh. With a volume of just 8.5 cm^3, it has an energy density of approx 1500 Joules/cm^3. While providing a cheap, high capacity, energy source, the major drawbacks of alkaline batteries are the wide voltage range that must be tolerated and their large physical size. Additionally, lifetimes beyond 5 years cannot be achieved because of battery self-discharge. The shelf-life of an alkaline battery is approximately 5 years.

Lithium batteries provide an incredibly compact power source. The smallest versions are just a few millimeters across. Additionally, they provide a constant voltage

Battery Characteristics

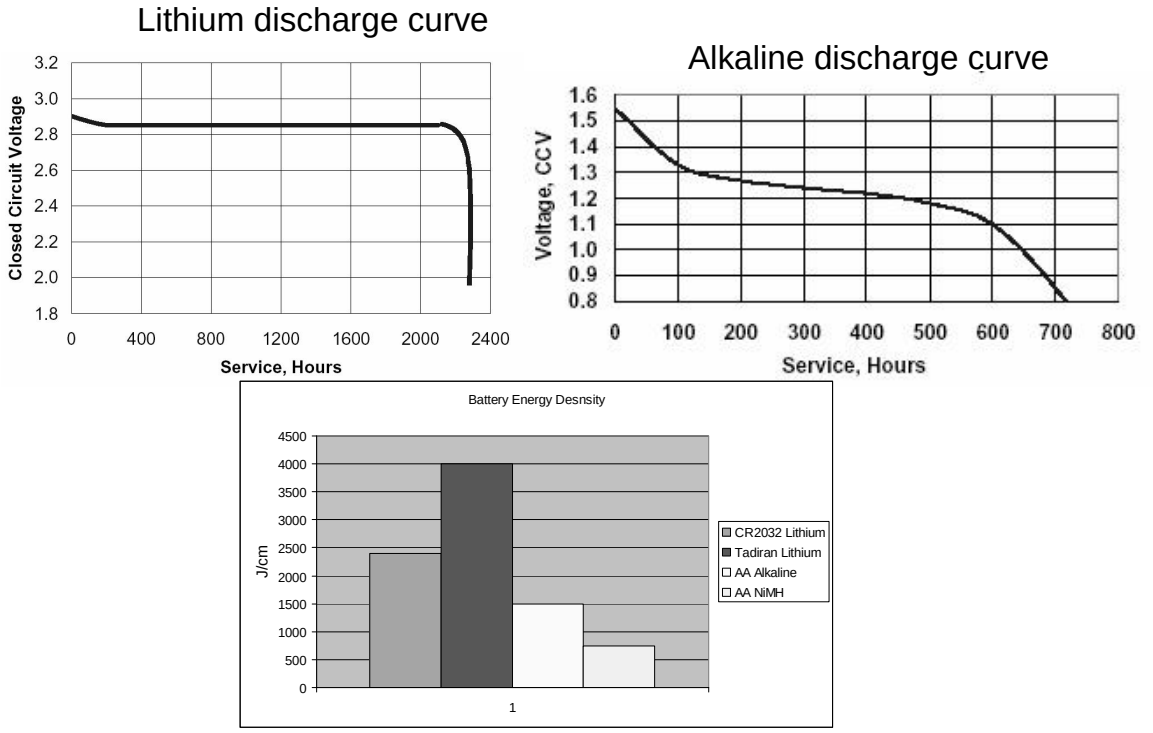

Figure 2-1: Battery characteristics for Lithium, Alkaline and NiMH batteries. The discharge characteristics of alkaline batteries make it essential to design a system to tolerate a wide range of input voltages.

supply that decays little as the battery is drained. Devices that operate off of lithium batteries do not have to be as tolerant to voltage changes as devices that operate off of alkaline batteries. Additionally, unlike alkaline batteries, lithium batteries are able to operate at temperatures down to -40 C. The most common lithium battery is the CR2032 [19]. It is rated at 3V, 255 mAh and sells for just 16 cents. With a volume of 1 cm^3, it has and energy density of 2400 J/cm^3. In addition to traditional lithium batteries, there are also specialized Tadiran lithium batteries that have densities as high as 4000 J/cm^3 and tolerate a wide temperature range. One of the drawbacks of lithium batteries is that they often have very low nominal discharge currents. A D-cell size Tadiran battery has a

nominal discharge current of just 3 mA. This is compared to an alkaline AA battery's nominal discharge rate of 25 mA.

Nickel Metal Hydride batteries are the third major battery type. They have the benefit of being easily rechargeable. The downside to rechargeable batteries is a significant decrease in energy density. An AA size NiMH battery has approximately half the energy density of an alkaline battery at approximately 5 times the cost. Before considering the use of NiMH batteries it is important to note that they only produce 1.2 V. Because many system components require 2.7 volts or more, they it may not be possible to operate directly off of rechargeable batteries.

2.4.1.2 Expected lifetime calculation

While it appears easy to quickly look at a battery's rated capacity, compare it to the systems energy consumption and calculate the system lifetime, there are several additional factors to consider.

In looking at real systems, it is important to look at how power supplies decay over time. Figure 2-1 shows the voltage versus time plot of an AA battery drained at a 500 mW. The graph shows that the battery quickly falls from the 1.5 V starting voltage and ends at just .8 V.

If a theoretical system was built with components requiring 2.7 V and consumed 250 mW, it would only last for 100 minutes off of 2 AA batteries. However, if the system components were selected to operate off of voltages down to 2.0 volts, it would last approximately 5 times as long off of the same power source. A seemingly unimportant CPU parameter results in a 5x difference in system lifetime.

2.4.1.3 Voltage Regulation

In order to avoid issues associated with varying battery voltages, voltage regulation techniques can be used. In general, voltage regulators take in varying input voltages and produce a stable, constant output voltage. Standard regulators require the input voltage be greater than the desired output voltage. Boost converters can output voltages that are higher than the input voltage.

There are two major drawbacks to using voltage regulation. The most obvious is the inefficient conversion process. High efficiency voltage converters can boast of efficiencies up to 90%. However, these are generally tuned for systems consuming hundreds of milliamps. The efficiencies fall drastically as the current consumption of the system is reduced.

Secondly, the quiescent current consumption – or consumption when no current is being output – of regulators can be relatively high. The current consumption of a standard regulator is between 15 and 50 uA. While irrelevant in a system consuming hundreds of milliamps, for ultra-low power devices, this small current drain can dominate overall energy usage.

The inefficiencies associated with voltage conversion and regulation combined with the high variation in output voltage of alkaline batteries makes it highly advantageous to build sensor nodes out of components that are tolerant to a wide voltage range. Selection of components that can operate over a range of (2.1-3.3V) can significantly improve system performance.

2.4.1.4 Renewable Energy

An alternative to relying on batteries with enough energy to last for years is to use renewable energy. Modern solar cells can produce up to 10 mW per square inch in direct sunlight. If stored properly, the energy collected during the day can be enough energy to last through the night. In indoor lighting environments between 10 and 100 uW per square inch can be produced depending on the type of lighting. For solar powered application scenarios, the key to successfully harnessing solar energy lies in the ability to store the energy.

Modern ultra-capacitors represent an option for energy storage. Ultra capacitors are low-voltage capacitors ranging between 1 and 6 Farads. When charged to 3 V they contain enough energy to send several hundred radio packets. While they can be charged and drained easily, one of their biggest drawbacks is that they have internal leakage rates of 20 to 50 uA when charged to 3 V. In low duty cycle operations this will quickly dominate the energy consumption of the nodes.

Rechargeable batteries can also be used to store solar energy. NiMH batteries can be trickle charged directly off of a solar cell. During periods of bright sunlight, energy could be slowly transferred back into the battery. 66% of the energy put into a NiMH battery turns into usable stored energy [20]. While the initial recharging process is less efficient than storing energy in an ultra-cap, rechargeable batteries have a significantly lower self discharge rate. If the energy is going to be used over several hours, the reduced self-discharge rate will outweigh the charging inefficiency.

2.4.1.5 Idealized AA Battery

For the remainder of this thesis we use a pair of idealized AA batteries in order to estimate system lifetime. For the idealized battery we assume that the battery discharges from 1.6 V down to .8 V in a linear fashion while delivering 2250 mA hours of charge. We use 20% less than the rated capacity of AA to ensure that our estimates are not too high.

We use this idealized battery to generate an energy budget and analyze the impact of architectural alternatives. If we assume that the network must last for one year and that the system is able to operate down to 2.0V, it can have an average power consumption of just 192 uA (approximately 500 uW). This current must be divided across all system components and functions. We can keep this figure in mind as we continue to evaluate system components.

2.4.2 Radios

The radio subsystem is the most important system on a wireless sensor node since it is the primary energy consumer in all three of our application scenarios. Modern low-power, short range transceivers consume between 15 and 300 milliwatts of power when sending and receiving. A key hardware observation is that low power radios consume approximately the same amount of energy when in receive or transmit mode. This energy is consumed if the radio is on, weather or not it is receiving actual data. The actual power emitted out of the antenna only accounts for a small fraction of the transceiver's energy consumption. A significant fraction goes to internal operation. Because of this, the overall cost of radio communication can easily be dominated by the receiver power consumption – a metric that is often ignored in wireless studies.

Additionally, if the receiver is left on 100% of the time during periods of intermittent communication, the receiver will be the single largest energy consumer in the system. People commonly make the mistake of assuming that reception is free.

2.4.2.1 Transmission Range

The transmission range of a wireless system is controlled by several key factors. The most intuitive factor is that of transmission power. The more energy put into a signal, the farther it should travel. The relationship between power output and distance traveled is a polynomial with an exponent of between 3 and 4 (non-line of sight propagation) [21]. So to transmit twice as far through an indoor environment, 8 to 16 times as much energy must be emitted.

Other factors in determining range include the sensitivity of the receiver, the gain and efficiency of the antenna and the channel encoding mechanism. In general, wireless sensor network nodes cannot exploit high gain, directional antennas because they require special alignment and prevent ad-hoc network topologies. Omni-directional antennas are preferred in ad-hoc networks because they allow nodes to effectively communicate in all directions.

Both transmission strength and receiver sensitivity are measured in dBm. Typical receiver sensitivities are between -85 and -110 dBm. (The dB scale is a logarithmic scale where a 10 dB increase represents a 10x increase in power. The baseline of 0 dBm represents 1 milliwatt, so 1 watt is 30 dBm.) Transmission range increases can be achieved by either increasing sensitivity or by increasing transmission power. When transmitting at 0 dBm, a receiver sensitivity of -85 dBm will result in an outdoor free space range of 25-50 meters, while a sensitivity of -110 dBm will result in a range of 100

to 200 meters. The use of a radio with a sensitivity of -100 dBm instead of a radio with -85 dBm will allow you to decrease the transmission power by a factor of 30 and achieve the same range.

2.4.2.2 Modulation Type

A key characteristic of any RF device is the modulation mechanism. Most radios communicate information by modulating a RF carrier signal. The standard modulation mechanisms are amplitude modulation and frequency modulation. Amplitude modulation is the simplest to encode and decode, but it is the most susceptible to noise. Because the data is encoded in the strength of the transmission, external noise is added to the signal. In contrast, frequency based modulation is less susceptible to noise because all data is transmitted at the same power level.

2.4.2.2.1 Frequency Agile Radios

Most transceivers on the market today use a VCO (Voltage Controlled Oscillator) based radio architecture and have the ability to communicate at a variety of carrier frequencies. Not only does this give them the potential to be robust to interfering signals, it also allows the radios to comply with FCC part 15 regulations. In the 902-928 MHz band, radios must channel-hop across at least 25 channels in order to be allowed to transmit at up to 250 mW or 24 dBm. If only a single channel is used, the transmission must remain slightly below 1 milliwatt.

An additional reason to use a frequency agile radio is that it allows for the use of FM or frequency modulation based data communication. FM is more robust to

interference and fading than the simpler AM based modulation. Fixed frequency radios are limited to the use of AM based data encoding schemes.

2.4.2.2.2 Frequency Hopping Spread Spectrum

To increase a system's ability to tolerate external interference, spread spectrum transmission techniques can be used. Spread spectrum communication is a technique that increases channel reliability and tolerance to noise by spreading the signal to be communicated over a wide range of frequencies. An interference source may corrupt a small portion of the data, but a majority of the information transmitted will be received. Frequency hopping (FH) is a common form of spread spectrum. The carrier frequency is continually changed based on a pseudorandom algorithm. On each hop, the data is transmitted on a single carrier frequency but over time the data will be spread across the frequency spectrum. Dwell times – the duration each channel is used – range from hundreds of microseconds to tens of milliseconds. (.0001s->.1s)

The major drawback of frequency hopping systems when applied to wireless sensor networks is the overhead of maintaining channel synchronization and the effort to discover the current hopping sequence. As nodes attempt to discover each other they must attempt to search all possible channel locations. This is an expensive, time consuming operation that is not compatible with low duty cycle networks. It can be seen how this leads to high power consumption in Bluetooth devices.

2.4.2.2.3 Direct Sequencing Spread Spectrum

In contrast to frequency hopping systems, direct sequencing spread spectrum radios communicate on a single carrier frequency. Instead of hopping carriers, the signal

to be communicated is directly spread over a wide frequency band by multiplying the signal by a higher rate pseudorandom sequence. Typical DSSS radios have spreading ratios between 15 and 100. This means that the radio must transmit between 15 and 100 times more data than is actually contained in the signal.

During reception, the received signal is passed through a correlator that reconstructs the original input signal. The main downside to the use of DSSS is the overhead associated with the spreading codes and the cost of the decorrelation. To achieve the same effective bit rate as a standard radio, the transmission rate must be increased by 15x-100x. This bit rate increase translates into an increase in radio power consumption and a decrease in receiver sensitivity. The decrease in sensitivity is then offset by the processing gain associated with the correlation, ideally resulting in a comparable communication range.

2.4.2.3 Power Consumption

To illustrate the power consumption of modern low power transceivers, we simply outline the characteristics of two commercial radios, the RF Monolithics TR1000 [22] and the Chipcon CC1000 [23].

The TR1000 radio consumes 21 mW of energy when transmitting at 0.75 mW. During reception, the TR1000 consumes 15 mW and has a receive sensitivity of -85 dBm. In contrast, at optimum input supply voltage, the CC1000 consumes 50 mW to transmit at 3 mW and consumes 20 mW to have a receive sensitivity of -105 dBm. When transmitting at the same .75 mW as the TR1000, the CC1000 consumes 31.6 mW. In practice TR1000 provides an outdoor, line-of-site communication range of up to 300 feet compared to 900 feet for the CC1000.

To put these numbers in perspective we can refer back to our idealized AA battery. On our idealized power source, the CC1000 can transmit for approximately 4 days straight or remain in receive mode for 9 days straight. In order to last for one year, the CC1000 must operate at a duty cycle of approximately 2%.

2.4.2.4 Bit Rate

Unlike many high performance data networks, wireless sensor networks do not require high bit rates. 10-100 Kbps of raw network bandwidth is sufficient for many applications. Radio bandwidth has a more significant impact on node power consumption and lifetime than it does on application capabilities.

As bit rates increase, transmission times decrease. As the highest instantaneous energy consumer it is essential that the radio remain off as much as possible. By increasing the bit rate without increasing the amount of data being transmitted, the radio duty cycle is decreased. The CC1000 and the RFM TR1000 both offer bit rates of approximately 100 Kbps.

2.4.2.5 Turn-on Time

In performing an analysis of a radio's power consumption there are several factors to consider in addition to the peak power consumption. A radio's ability to quickly enter and exit low power sleep modes is extremely important for efficient operation in wireless sensor networks. All usage scenarios assume that the radio will only be on occasionally. Before and after a transmission, time and energy must be spent configuring and powering up the radio. Ideally this should be a quick, cheap, operation. If a system has to react to emergency events within seconds, the radio must be powered at least once per second. If

a radio's turn-on-to-receive time is more than a few tens of milliseconds, it quickly becomes impossible to achieve the required duty cycles of less that 1 percent. Anything beyond a 5 ms turn-on time would be unacceptable for most applications.

One of the drawbacks to multi-channel radios that use a VCO based frequency synthesizer is that the VCO must stabilize prior to transmission or reception. Additionally the VCO is often locked to a high frequency crystal that must also stabilize. The CC1000 radio from Chipcon requires 2 ms for the primary crystal to stabilize and 200 us for the high frequency VCO. In contrast, the RFM radio that uses a fixed frequency SAW filter can be turned on and be ready to receive in just a few hundred microseconds – more than ten times faster.

2.4.3 Processor

Modern microcontrollers integrate flash storage, RAM, analog-to-digital converters and digital I/O onto a single integrated circuit that costs between $1 and $5. Their tight integration makes them ideal for use in deeply embedded systems like wireless sensor networks. When selecting a microcontroller family, some of the key requirements are energy consumption, voltage requirements, cost, support for peripherals and the number of external components required.

2.4.3.1 Power

Power consumption of microcontrollers varies greatly from family to family. Standard 8 or 16 bit microcontrollers consume between .250 to 2.5 mA per MHz. This 10x difference between low-power and standard microcontrollers can have a significant

impact on system performance. While it is obvious that selecting the lowest power controller is essential, peak power consumption is not the most important metric.

A second significant power measurement for a microcontroller is the sleep mode power consumption. The CPU is a major contributor to the node's standby power consumption. During idle periods, the CPU will stop execution and enter a low-power sleep state. The processor only needs to maintain its memory and maintain time synchronization so it can properly wake-up when necessary. Sleep mode current consumption varies between 1 uA [24] and 50 uA across controller families. Despite being a difference of just a few uA, this 50x variation can have a more significant impact on node performance than milliamp differences in peak power consumption. This is because the CPU is expected to be idle 99.9% of the time. A microcontroller that burns 50 uA in sleep mode will have a maximum lifetime of 4 years on our idealized power source even if it never leaves the sleep mode. In contrast the lifetime of a controller that consumes 1 uA in sleep mode will be completely dominated by battery self discharge and active power consumption.

2.4.3.2 Wake-up Time

In addition to the current consumption of the sleep mode, the transition requirements for entering and exiting sleep modes are also significant. Some microcontrollers require as much as 10 ms to enter and resume from the sleep state. In contrast, some controllers have wake-up times of just 6 us. The wake-up delay is used to start and stabilize system clocks. The faster a controller can enter and leave the low-power state, the more often the sleep state can be used and the more responsive the node will be.

With a quick wake-up time, it is not necessary to remain awake through short periods of inactivity. For example, during a message transmission, the controller can enter the sleep mode between each bit of the transmission. This can reduce the effective active current consumption of the controller significantly. Even at a 9600 bps communication rate, a 1 ms sleep time makes entering the sleep mode between each bit impossible.

2.4.3.3 Voltage Requirements

The operating voltage range of a low-power microcontroller can also have significant impact on system performance. Traditional low-voltage microcontrollers operate between 2.7V and 3.3 V. However, new generations of low-power controllers are appearing that operate down to 1.8 V. As discussed in our analysis of energy sources, a wide voltage tolerance allows devices to be more efficiently operated off of Alkaline batteries.

2.4.3.4 Speed

The main duties of the microcontroller are to execute the communication protocols, control the radio, perform data encryption, interact with sensors and perform data processing. Most of these operations place relatively low demands on the speed of the controller. Current designs demand CPU speeds between 1 and 4 MHz. This range is consistent with most microcontrollers on the market today. Most controllers are fully static with the ability to operate between 0 and 8 MHz.

One feature of several controller families is the ability to dynamically change the operating frequency. However, CMOS power consumption obeys the equation

"$P=CV^2F$". This relation states that the power consumption of the controller scales linearly with frequency. However, the execution time is inversely proportional to frequency. The impact of changing the frequency is negligible because it is offset by the change in execution time resulting in no change in energy usage. If proper techniques are used to effectively enter low power sleep states there is not a significant advantage to reducing or increasing the CPU speed.

The critical variable in determining the necessary computational speed for a wireless sensor node is the amount of data analysis and in-network processing that must be performed. The CPU must be capable of meeting the real-time deadlines demanded by the data processing.

2.4.3.5 Memory

In general, sensor nodes only require small amounts of storage and program memory. Data is only stored long enough for it to be analyzed and then transmitted through the network to the base station. In general, modern flash-based microcontrollers contain between 1 and 128 KB of on-chip program storage. This can be used as both program memory and as temporary data storage. Additionally they contain between 128 and 32KB of data ram that can be used for program execution.

Modern flash technology currently produces memory storage densities upwards of 150 KB per square millimeter in a .25 micron process [25]. This is significantly higher than Intel's recent SRAM density record of 60 KB per square millimeter set using a 90 nm process [26]. While it is currently only possible to achieve these storage densities on chips that are dedicated to performing one specific function, we can conclude that there is

a significant memory density advantage for flash over SRAM. In order to achieve the smallest, lowest cost device, SRAM should be used only when necessary.

In addition to considering the size impact of flash versus SRAM, we should also consider the power impact. SRAM requires more energy to retain data over time, but does not require as much energy for the initial storage operation. Flash, on the other hand, is a persistent storage technology that requires no energy to maintain data. In contrast, AMD SRAM chips require 2188 nA per megabit for data retention [25]. However, a flash write operation requires 4 us to complete compared to .07 us for SRAM – both consuming 15 mA. If data is to be stored for long periods of time it is more efficient to use flash instead of SRAM. However, for storage operation to overcome the cost of performing the flash write operation it must be stored for at least 7.6 hours. Any shorter and the initial cost of storage is not outweighed by the decreased cost of retention.

2.4.3.6 Peripheral support

In addition to storage and computation, microcontrollers are specifically designed to interact with external devices. Microcontrollers generally contain a collection of specialized external interfaces. These range from general purpose digital I/O pins to digital communication interfaced to analog inputs and outputs.

Standard digital I/O lines are included on all controllers as the base line interface mechanism. However, to interface directly with analog sensors many controllers have an internal analog-to-digital converter. An internal converter allows for precise control of sample timing and easy access to sample results. If not present, an external converter must be used.

The digital communication interfaces come into play when interacting with digital sensors, other peripheral chips, or for serial communications over a radio or RS-232 transceiver. There are 3 standard communication protocols supported by digital communication primitives: UART, I2C and SPI. Both I2C and SPI use synchronous protocols with explicit clock signals while UART provides an asynchronous mechanism.

2.4.4 Sensors

The last decade has seen an explosion in senor technology. There are currently thousands of potential sensors ready to be attached to a wireless sensing platform. Additionally, advances in MEMS and carbon nano-tubes technology are promising to create a wide array of new sensors. They range from simple light and temperature monitoring sensors to complex digital noses. Figure 2-2 outlines a collection of common micro-sensors and their key characteristics.

2.4.4.1 Interfaces

There are two general ways to interface with sensors that can be used in sensor

Commonly avilable sensors

	Current	Discrete Sample Time	Voltage Requiremet	Manufacturer
Photo	1.9 mA	330 uS	2.7-5.5V	Taos
Temperature	1 mA	400 mS	2.5-5.5V	Dallas Semiconductor
Humitidy	550 uA	300 mS	2.4-5.5V	Sensirion
Pressure	1 mA	35 mS	2.2V-3.6V	Intersema
Magnetic Fields	4 mA	30 uS	Any	Honeywell
Acceleration	2 mA	10 mS	2.5-3.3V	Analog Devices
Acoustic	.5 mA	1 mS	2-10 V	Panasonic
Smoke	5 uA	--	6-12 V	Motorola
Passive IR (Motion)	0 mA	1 mS	Any	Melixis
Photosynthetic Light	0 mA	1 mS	Any	Li-Cor
Soil Moisture	2 mA	10 mS	2-5 V	Ech2o

Figure 2-2: Power consumption and capabilities of commonly available.

networks: analog and digital. Analog sensors generally provide a raw analog voltage that corresponds to the physical phenomena that they are measuring. Generally these produce a continual waveform that must be digitized and then analyzed. While seemingly straightforward to integrate, raw analog sensors often require external calibration and linearization. It is common for the sensor to have non-linear response to stimuli. The host controller must then compensate in order to produce a reading in meaningful units. For example, a raw reading of 1.873 Volts from an accelerometer must be translated into a corresponding acceleration. Depending on the characteristics of the sensor this can be a complex process. In many cases the translation may depend on other external factors such as temperature, pressure, or input voltage.

A second difficulty in interfacing with raw analog sensors is that of scale. Each sensor will have different timing and voltage scales. The output voltage will generally contain a DC offset combined with a time-varying signal. Depending on the ratio of signal to DC component, an array of amplifiers and filters may be required to match the output of the sensor to the range and fidelity of the ADC being used to capture it.

Digital sensors have been developed to remove many of these difficulties. They internally contain a raw analog sensor but provide a clean digital interface to it. All required compensation and linearization is performed internally. The output is a digital reading on an appropriate scale. The interface to these sensors is via one of a handful of standard chip-to-chip communication protocols.

2.4.4.2 Sample Rates and Turn-on times

In looking at the power consumption of a sensor, one of the most important factors is how quickly a sensor can be enabled, sampled, and disabled. In most cases

sensors are capable of producing thousands of samples per second. However, we are generally only interested in a few samples per minute. It is essential that the sensor be able to enter and exit a low-power state quickly.

The power consumption of a sensor is equally dependent on the amount of time it takes to read the sensor as it is to the current consumption. For example, if a sensor takes 100 ms to turn on and generate a reading and consumes just 1 mA at 3 V, it will cost 300 uJ per sample. This is the same amount of energy as a sensor that consumes 1000 mA of current at 3 V but takes just 100 us to turn on and sample.

2.4.4.3 Voltage requirements

A seemingly low-power, easy to interface sensor can quickly become a major hassle if its voltage requirements do not match the capabilities of the system. Some sensors require +/- 6 V. Special voltage converters and regulators have to be added to systems operating off of AA or lithium batteries in order to use this sensor. The power consumption and turn-on times of the regulation and conversion circuitry must be included in the total energy budget for the sensor.

2.5 Rene Design Point

Now that we have provided a background of hardware capabilities, we present a baseline networked sensor platform. The Rene design point is used as a comparison point in order to evaluate the more advanced architecture developed over the subsequent chapters. A photograph for the Rene hardware including sensor and programming boards appear in Figure 2-3. It consists of a microcontroller with internal flash program memory, data SRAM and data EEPROM, connected to a set of actuator and sensor

devices, including LEDs, a low-power radio transceiver, a digital temperature sensor, a serial port, and a small coprocessor unit and an I/O connector that allows a suite of sensors to be attached. The basic design is based on the weC mote, but was redesigned to serve as an experimental and developments platform. The modifications included a rich connector interfaced that allowed for stackable sensor and power boards. This allowes for application specific sensor and power boards.

2.5.1.1 CPU

The processor on the Rene platform is the ATMEL 90LS8535 [27]. It is an 8-bit Harvard architecture with 16-bit addresses. It provides 32 8-bit general purpose registers and runs at 4 MHz and 3.0 V. The system is very memory constrained: it has 8 KB of flash as the program memory, and 512 bytes of SRAM as the data memory. The MCU is designed such that a processor cannot write to instruction memory; our prototype uses a coprocessor to perform that function. Additionally, the processor integrates a set of timers and counters which can be configured to generate interrupts at regular time intervals. In conjunction with a 32 KHz external crystal, the counters can be used to keep precise time. Additionally, it also contains an internal analog to digital converter for reading analog sensor inputs.

2.5.1.2 Storage

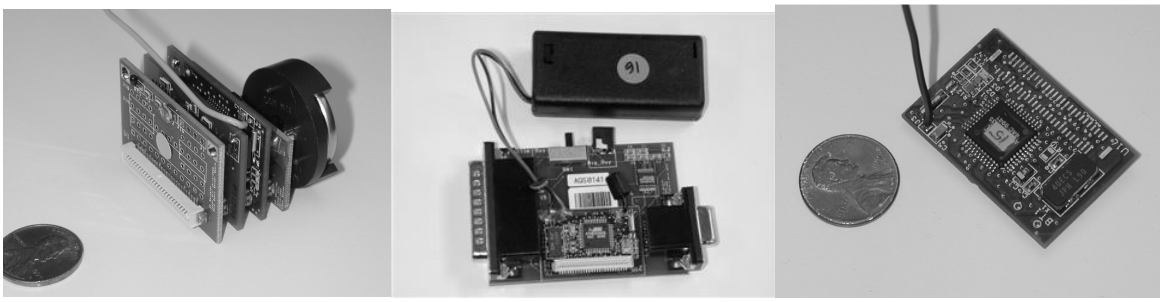

Figure 2-3: Pictures of the Rene wireless sensor network platform.

Besides the storage contained inside the main microcontroller, the Rene hardware platform also contains an external EEPROM capable of holding 32 kilobytes of data. This can be used to temporarily record sensor readings or to act as storage for additional programs.

The CPU interacts with the EPROM via a two wire I2C [28] data communication bus. The I2C bus is a standard communication mechanism in embedded systems. The bus architecture allows for a collection of 8 peripherals to be controlled by a single master via two communication lines and a ground connection.

2.5.1.3 RF Subsystem

The radio is the most important component. It consists of an RF Monolithics 916.50 MHz transceiver (TR1000) [22], antenna, and collection of discrete components to configure the physical layer characteristics such as signal strength and sensitivity. As configured, it operates in an ON-OFF key mode at speeds up to 19.2 Kbps but is only driven at 10 Kbps. Control signals configure the radio to operate in either transmit, receive, or power-off mode. The radio contains no buffering so each bit must be

serviced by the controller on time. Additionally, the transmitted value is not latched by the radio, so jitter at the radio input is propagated into the transmitted signal.

2.5.1.4 Coprocessor

A coprocessor is included on the platform to assist in reprogramming. In this case, it is a very limited controller (AT90LS2343 [29], with 2 KB flash instruction memory, 128 bytes of SRAM and EEPROM) that uses I/O pins connected to an SPI controller. SPI is a synchronous serial data link, providing high speed full-duplex connections (up to 1 Mbit) between various peripherals. The coprocessor is connected in a way that allows it to reprogram the main microcontroller. The sensor node can be reprogrammed by transferring data from the network into external EEPROM. Then the coprocessor can reset the main controller and reprogram it with the new program. This is required because the main CPU cannot program its own memory.

2.5.1.5 I/O Interface

Three LEDs represent analog outputs connected through a general I/O port; they may be used to display digital values or status. A photo-sensor represents an analog input device with simple control lines. In this case, a control line enables and disables the sensor by cutting the power drain through the photo resistor when not in use. The input signal can be directed to the internal ADC in continuous or sampled modes.

2.5.2 Baseline Power Characteristics

The AT908535 microcontroller consumes just 5 mA when operating at 4 MHz. Additionally, it has two low-power states – idle and sleep. When in the idle state, the

controller consumes just 3 mA. The idle state can be entered and exited with just one instruction penalty, making it useful for brief periods of inactivity between computations. In contrast, the sleep sate consumes just 10 uA of current, but requires 2 milliseconds to enter and exit. The sleep state can only be used in conjunction with disabling the radio.

The minimum pulse width for the RFM radio is 52 us. Thus, at 3V and 7 mA, it takes 1.1 uJ of energy to transmit a single bit. Alternatively, it costs approximately 0.5 uJ to receive a bit. During this time, the processor can execute 208 cycles (roughly 100 instructions) and can consume up to .8 uJ. A fraction of this instruction count is devoted to bit level processing. The remainder can go to higher level processing (byte-level, packet level, application level) amortized over several bit times. Unused time can be spent in power-down mode or on application processing.

While listening for incoming communications, the radio will continually consume 5 mA regardless of what is being received. In addition, the CPU must actively search the incoming bit stream for a valid transmission. The CPU will continually be switching between its 3 mA idle state and its 5 mA active. Because the CPU must inspect each arriving bit, it is not possible to enter a low-power sleep mode while scanning for the start of a packet. Even if we ignore the active power consumption, the node will be consuming approximately 8 mA to search for valid data.

2.6 Refined Problem Statement

The potential usage scenarios for wireless sensor networks are countless. In order to reduce the problem space, we have chosen to focus on three distinct application scenarios that we believe are representative of a large fraction of the potential applications. These three scenarios – environmental data collection, security monitoring,

and node tracking – allow us to extract the evaluation metrics that can be used to measure the performance of a wireless sensor network. From these system-level evaluation metrics we have drilled down into the individual node capabilities that will support them.

Our objective is to develop a wireless sensor node architecture that addresses the node-level requirements that we have set forth. In looking at several of the system-level metrics, it is apparent that a balance must be obtained between range, bit rate, duty cycle and power consumption in order to create a system that meets both lifetime and performance requirements. We develop a generalized node architecture and evaluate it in depth over a series of instantiations at technology points ranging from off-the-shelf integrations to a full custom ASIC.

In the next chapter we will look into the software architecture that plays a critical role in realizing the system capabilities. While often ignored, the software subsystem plays as important a roll as any radio, processor or battery. The TinyOS operating system presented will provide a concurrency and flexibility mechanism that allow the development of a generalized wireless sensor node architecture that is substantially different from traditional wireless devices.

Chapter 3: Software Architecture for Wireless Sensors

A critical step towards achieving the vision behind wireless sensor networks is the design of a software architecture that bridges the gap between raw hardware capabilities and a complete system. The demands placed on the software of wireless sensor networks are numerous. It must be efficient in terms of memory, processor, and power so that it meets strict application requirements. It must also be agile enough to allow multiple applications to simultaneously use system resources such as communication, computation and memory. The extreme constraints of these devices make it impractical to use legacy systems. TinyOS is an operating designed explicitly for networked sensors.

TinyOS draws strongly from previous architectural work on lightweight thread support and efficient network interfaces. Included in the TinyOS system architecture is an Active Messages communication system. We believe that there is a fundamental fit between the event based nature of network sensor applications and the event based primitives of the Active Messages communication model.

In working with wireless sensor networks, two issues emerge strongly: these devices are concurrency intensive - several different flows of data must be kept moving simultaneously, and the system must provide efficient modularity - hardware specific and application specific components must snap together with little processing and storage overhead. We address these two problems in the context of current network sensor

technology and the tiny micro threaded OS. Analysis of this solution provides valuable initial directions for architectural innovation.

3.1 Tiny Microthreading Operating System (TinyOS)

Small physical size, modest active power load, and micro standby power load are must be provided by the hardware design. However, an operating system framework is needed that will retain these characteristics by managing the hardware capabilities effectively, while supporting concurrency-intensive operation in a manner that achieves efficient modularity and robustness. Existing embedded device operating systems do not meet the size, power and efficiency requirements of this regime. These requirements are strikingly similar to that of building efficient network interfaces, which also must maintain a large number of concurrent flows and juggle numerous outstanding events [30]. In network interface cards, these issues have been tackled through physical parallelism [31] and virtual machines [32]. We tackle it by building an extremely efficient multithreading engine. As in TAM [32] and CILK [33], TinyOS maintains a two-level scheduling structure, so a small amount of processing associated with hardware events can be performed immediately while long running tasks are interrupted. The execution model is similar to finite state machine models, but considerably more programmable.

TinyOS is designed to scale with the current technology trends supporting both smaller, tightly integrated designs, as well as the crossover of software components into hardware. This is in contrast to traditional notions of scalability that are centered on scaling up total power/resources/work for a given computing paradigm. It is essential

that software architecture plans for the eventual integration of sensors, processing and communication.

In order to enable the vision of single-chip, a low cost sensor node, TinyOS combines a highly efficient execution model, component model and communication mechanisms.

3.2 TinyOS Execution Model

To provide the extreme levels of operating efficiency required in wireless sensor networks, TinyOS uses event based execution. The event model allows for high levels of concurrency to be handled in a very small amount of space. In contrast, a thread based approach requires that stack space be pre-allocated for each execution context. Additionally, the context switch overhead of threaded systems is significantly higher than those of an event-base system. Context switch rates as high as 40,000 switches per second are required for the base-band processing of a 19.2 Kbps communication rate. This is twice every 50 us, once to service the radio and once to perform all other work. The efficiency of an event-based regime lends itself to these requirements.

3.2.1 Event Based Programming

In an event based system, a single execution context is shared between unrelated processing tasks. In TinyOS, each system module is designed to operate by continually responding to incoming events. When an event arrives, it brings the required execution context with it. When the event processing is completed, it is returned back to the system. Researchers in the area of high performance computing have also seen that event

based programming must be used to achieve high performance in concurrency intensive applications [34, 35].

In addition to efficient CPU allocation, event-based design results in low power operation. A key to limiting power consumption is to identify when there is no useful work to be performed and to enter an ultra-low power state. Event-based systems force applications to implicitly declare when they are finished using the CPU. In TinyOS all tasks associated with an event are handled rapidly after each event is signaled. When an event and all tasks are fully processed, unused CPU cycles are spent in the sleep state as opposed to actively looking for the next interesting event. Eliminating blocking and polling prevents unnecessary CPU activity.

3.2.2 Tasks

A limiting factor of an event based program is that long-running calculations can disrupt the execution of other time critical subsystems. If an event were to never complete, all other system functions would halt. To allow for long running computation, TinyOS provides an execution mechanism called ***tasks***. A task is an execution context that runs to completion in the background without interfering with other system events. Tasks can be scheduled at any time but will not execute until current pending events are completed. Additionally, tasks can be interrupted by low-level system events. Tasks allow long running computation to occur in the background while system event processing continues.

Currently task scheduling is performed using a simple FIFO scheduling queue. While it is possible to efficiently implement priority scheduling for tasks, it is unusual to

have multiple outstanding tasks. A FIFO queue has proven adequate for all application scenarios we have attempted to date.

3.2.3 Atomicity

In addition to providing a mechanism for long-running computation, the TinyOS task primitive also provides an elegant mechanism for creating mutually exclusive sections of code. In interrupt-based programming, data race conditions create bugs that are difficult to detect. In TinyOS, code that is executed inside of a task is guaranteed to run to completion without being interrupted by other tasks. This guarantee means that all tasks are atomic with respect to other tasks and eliminates the possibility of data race conditions between tasks.

Low-level system components must deal with the complexities associated with reentrant, interrupt based code in order to meet their strict real-time requirements. Normally, only simple operations are performed at the interrupt level to integrate data with ongoing computation. Applications can use tasks to guarantee that all data modification occurs atomically when viewed from the context of other tasks.

3.3 TinyOS Component Model

In addition to using the highly efficient event-based execution, TinyOS also includes a specially designed component model targeting highly efficient modularity and easy composition. An efficient component model is essential for embedded systems to increase reliability without sacrificing performance. The component model allows an application developer to be able to easily combine independent components into an application specific configuration.

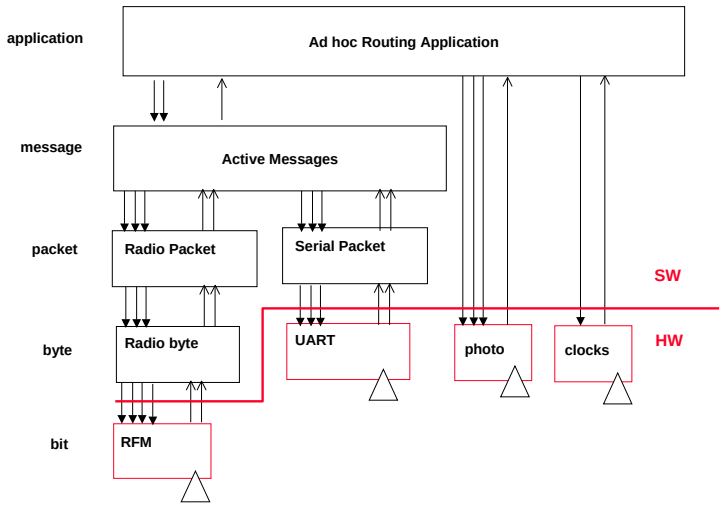

Figure 3-1: Component graph for a multi-hop sensing application.

In TinyOS, each module is defined by the set of commands and events that makes up its interface. In turn, a complete system specification is a listing of the components to include plus a specification for the interconnection between components. The TinyOS component has four interrelated parts: a set of command handlers, a set of event handlers, an encapsulated private data frame, and a bundle of simple tasks. Tasks, commands, and event handlers execute in the context of the frame and operate on its state. To facilitate modularity, each component also declares the commands it uses and the events it signals. These declarations are used to facilitate the composition process. As shown in Figure 3-1, composition creates a graph of components where high level components issue commands to lower level components and lower level components signal events to the higher level components. The lowest layer of components interacts directly with the underlying hardware. A specialized language, NESC, has been developed to express the component graph and the command/event interfaces between components. In NESC,

multiple command and events can be grouped together into interfaces. Interfaces simplify the interconnection between components.

In TinyOS, storage frames are statically allocated to allow the memory requirements of the complete application to be determined at compile time. The frame is a specialized C Structure that is statically allocated and directly accessible only to the component. While TinyOS does not have memory protection, variables cannot be directly accessed from outside of a component. In addition to allowing the calculation of maximum memory requirements, pre-allocation of frames prevents the overhead associated with dynamic allocation and avoids pointer related errors. This savings manifests itself in many ways, including execution time savings because variable locations can be statically compiled into the program instead of accessing state via pointers.

In TinyOS, commands are non-blocking requests made to lower level components. Typically, a command will deposit request parameters into its local frame and conditionally post a task for later execution. It may also invoke lower commands, but it must not wait for long or indeterminate latency actions to take place. A command must provide feedback to its caller by returning status indicating whether it was successful or not, e.g., buffer overrun. Event handlers are invoked to deal with hardware events, either directly or indirectly. The lowest level components have handlers that are connected directly to hardware interrupts, which may be external interrupts, timer events, or counter events. An event handler can deposit information into its frame, post tasks, signal higher level events or call lower level commands. A hardware event triggers a fountain of processing that goes upward through events and can bend downward through

commands. In order to avoid cycles in the command/event chain, commands cannot signal events. Both commands and events are intended to perform a small, fixed amount of work, which occurs within the context of their component's state.

Tasks perform the primary work in a TinyOS application. They are atomic with respect to other tasks and run to completion, though they can be preempted by events. Tasks can call lower level commands, signal higher level events, and schedule other tasks within a component. The run-to-completion semantics of tasks make it possible to allocate a single stack that is assigned to the currently executing task. This is essential in memory constrained systems.

Tasks simulate concurrency within each component, since they execute asynchronously with respect to events. However, tasks cannot block or spin wait or they will prevent progress in other components. While events and commands approximate instantaneous state transitions, task bundles provide a way to incorporate arbitrary computation into the event driven model.

3.3.1 Component Types

In general, components fall into one of three categories: hardware abstractions, synthetic hardware, and high level software components. Hardware abstraction components map physical hardware into the TinyOS component model. The RFM radio component (shown in lower left corner of Figure 3-1) is representative of this class. This component exports commands to manipulate the individual I/O pins connected to the RFM transceiver and signals events informing other components about the transmission and reception of bits. Its frame contains information about the current state of the

component, (the transceiver is in sending or receiving mode, the current bit rate, etc.). The RFM consumes hardware interrupts that are transformed into either the RX bit events or into the TX bit events, depending on the mode of operation. There are no tasks within the RFM because the hardware itself provides the concurrency. This model of abstracting over the hardware resources can scale from very simple resources, like individual I/O pins, to quite complex ones, like encryption accelerators.

Synthetic hardware components simulate the behavior of advanced hardware. A good example of such component is the Radio Byte component (see Figure 3-1). It shifts data into or out of the underlying RFM module and signals when an entire byte has completed. The internal tasks perform simple encoding and decoding of the data. Conceptually, this component is an enhanced state machine that could be directly cast into hardware. From the point of view of the higher levels, this component provides an

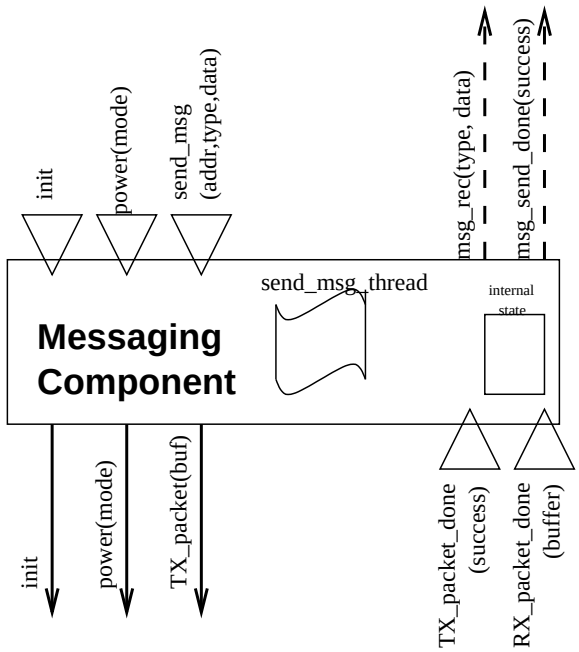

Figure 3-2: AM Messaging component graphical representation.

interface and functionality very similar to the UART hardware abstraction component: they provide the same commands and signal the same events, deal with data of the same granularity, and internally perform similar tasks (looking for a start bit or symbol, perform simple encoding, etc.).

The high level software components perform control, routing and all data transfers. A representative of this class is the messaging module presented in Figure 3-2. It performs the function of filling in a packet buffer prior to transmission and dispatches received messages to their appropriate place. Additionally, components that perform calculations on data or data aggregation fall into this category.

3.3.2 Enabling the migration of hardware/software boundary

The TinyOS component model allows for easy migration of the hardware/software boundary because the event based software model it uses is complementary to the underlying hardware structure. Additionally, the use of fixed size, pre-allocated storage is a requirement for hardware based implementations. Migration from software to hardware is particularly important for networked sensors where the system designers will want to explore the tradeoffs between the scale of integration, power requirements, and the cost of the system. Chapter 6 presents a customized design that pushes several system components from the Rene node across the hardware/software boundary.

```
module AMStandard
{
  provides {
    interface StdControl as Control;
    interface CommControl;

    // The interface are as parameterised by the active message id
    interface SendMsg[uint8_t id];
    interface ReceiveMsg[uint8_t id];

    // How many packets were received in the past second
    command uint16_t activity();
  }

  uses {
    // signaled after every send completion for components which wish to
    // retry failed sends
    event result_t sendDone();

    interface StdControl as UARTControl;
    interface BareSendMsg as UARTSend;
    interface ReceiveMsg as UARTReceive;

    interface StdControl as RadioControl;
    interface BareSendMsg as RadioSend;
    interface ReceiveMsg as RadioReceive;
    interface Leds;
    //interface Timer as ActivityTimer;
  }
}
```

Figure 3-3: NESC component file describing the external interface to the AMStandard messaging component.

3.3.3 Example Components

A typical component including a frame, event handlers, commands and tasks for a message handling component is pictured in Figure 3-2. Like most components, it exports commands for initialization and power management. Additionally, it has a command for initiating a message transmission, and signals events on the completion of a transmission or the arrival of a message. In order to perform its function, the messaging component issues commands to a packet level component and handles two types of events: one that indicates a message has been transmitted and one that signals that a message has been received.

Pictorially, we represent the component as a bundle of tasks, a block of state (component frame), a set of commands (upside-down triangles), a set of handlers (triangles), solid downward arcs for commands they use, and dashed upward arcs for events they signal. Figure 3-3 shows how all of these elements are explicit in the component code. Since the components describe both the resources they provide and the resources they require, connecting them together is simple.

The programmer matches the signatures of events and commands required by one component with the signatures of events and commands provided by another component. Communications across the components takes the form of a function call, which has low overhead and provides compile time type checking.

3.3.4 Component Composition

In order to support the modular nature of TinyOS, we have developed a set of tools that assist the developer to link components together. In TinyOS, components are linked together at compile time to eliminate unnecessary runtime overhead. To facilitate composition, each component's external interface is described at the start of each component file. In these files, a component lists the set of commands it accepts and the events it handles as well as the set of events it signals and the commands it uses. Logically we view each of the inputs and outputs of a component as an I/O pin as if the component were a physical piece of hardware. This complete description of a component's upper and lower interface is used by our compile time tools to automatically generate component header files. Figure 3-4 contains an example component file for a simple application that blinks the LEDs. In order actually compose individual components into a complete application; TinyOS originally used description (.desc) files

```
/**
 * Implementation for Blink application.  Toggle the red LED when the
 * clock fires.
 **/
module BlinkM {
  provides {
    interface StdControl;
  }
  uses {
    interface Clock;
    interface Leds;
  }
}
implementation {
  /**
   * the state of the red LED (on or off)
   */
  bool state;

  /**
   * Initialize the component.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.init() {
    state = FALSE;
    call Leds.init();
    return SUCCESS;
  }


  /**
   * Start things up.  This just sets the rate for the clock component.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.start() {
    return call Clock.setRate(TOS_I1PS, TOS_S1PS);
  }

  /**
   * Halt execution of the application.
   * This just disables the clock component.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.stop() {
    return call Clock.setRate(TOS_I0PS, TOS_S0PS);
  }


  /**
   * Toggle the red LED in response to the <code>Clock.fire</code> event.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  event result_t Clock.fire()
  {
    state = !state;

    if (state) {
      call Leds.redOn();
    } else {
      call Leds.redOff();
    }

    return SUCCESS;
  }
}
```

Figure 3-4: BlinkM.nc application component that blinks the system LED's once per second.

that contained a listing of components to be used as well as the logical connections between the components. The success of the TinyOS component model led to the development a specialized language called NESC that helps facilitate the expression of these logical connections [36].

The application description files can be thought of as a parts lists and a wiring diagram. These description files are divided into two sections. The first section directly lists the modules to include in the application. The second section of the file lists the connections between each component. This wiring diagram takes the form of lists of ports that are logically connected. Each line of the file contains lists a pair of module interfaces that are to be connected together with a wiring network. Figure 3-5 contains and example description file for a simple NESC application.

At compile time, the component description files and the application description files are preprocessed in order to create a net list that links the components together. Just as in hardware, a net in our system can have multiple sources and sinks. This is automatically handled by the NESC compiler. For example, a single event can be handled by multiple components. At compile time, code will be automatically generated

```
configuration Blink {
}
implementation {
  components Main, BlinkM, ClockC, LedsC,
DustKernelWrapper;

  Main.StdControl -> BlinkM.StdControl;
  BlinkM.Clock -> ClockC;
  BlinkM.Leds -> LedsC;
}
```

Figure 3-5: NESC application configuration file that wires together the Blink application.

to send the event to as many places as necessary. The output of the NESC compiler is a standard C file that contains all of the components in the application as well as all of the necessary linkage information.

3.3.5 Application Walk Through

In order to show what applications look like in TinyOS, we walk through a simple application that is included in the TinyOS 1.0 release. The application, BLINK, simply turns on and off the LED's of the system in a binary counting fashion. While this application could easily be written without the help of the TinyOS models, it demonstrates some of the core concepts in TinyOS.

Figure 3-4 shows the module description for the Blink application's main component (from BlinkM.nc). From the module declaration, you can see that the BlinkM component provides the TinyOS StdControl interface and uses the Clock interface and the LEDs interface. This interface file completely describes what functionality needs to be provided by other parts of the system for this component to function correctly. In particular, this component needs to be controlled with the StdControl interface, and it needs to have access to the clock interface and the LEDs. The actual code can then be written to fill in this component without knowledge of what will be providing the necessary support.

All variables declared inside the boundaries of the module are locally scoped private variables. They are placed inside the component's private frame. In this application the module has a single variable called state. This variable contains the current value of the binary counter being displayed on the LED's. The second section of

the code contains the implementation of the StdControl interface. This interface is designed to handle initialization. Functions are declared by specifying the type of function (command/event), the return type, the name of the interface they belong to, the individual function name, and the argument list. The init function is a command that returns a result_t and accepts no arguments and is part of the StdControl interface. It performs all of the necessary initialization for this component. Additionally, it must initialize all sub-components that are used. In this example, the init function issues a single command to initialize the LED. The invocation of commands are performed using the 'call' keyword. Similarly, events are signaled using the signal keyword. The keywords call and signal explicitly state when component boundaries are crossed. This information is used by the NESC compiler in full-program analysis to optimize away component boundaries.

Note that the application never holds onto an execution context. Both the initialization routine and the event handlers release the execution context quickly. This is an essential property of a TinyOS application. By not holding onto an execution context, a single context can be shared across multiple concurrent activities. If this application were to be implemented in a threaded execution model, as opposed to the event based TinyOS model, this simple function would require a dedicated execution context. It would block until it was time to update the counter. This would consume a significant amount of resources when idle. In contrast, this application simply consumes a single byte of memory while idle.

In addition to writing the individual application level components, the application developer must also assemble a collection of TinyOS components into a complete

application. Figure 3-5 shows the application configuration file for this application. The first section of the file contains the list of components that need to be included. In this case, the complete application consists of the Main component, the BlinkM component, the ClockC component and the LedC component. The second section of the file then contains the connections between the components. The first line of this section states that the Main component's StdControl interface is connected to the BlinkM component's StdControl interface. The use of this component model allows applications to be easily reconfigured.

3.4 AM Communication Paradigm

A key piece of TinyOS is the communication model it supports. Instead of scaling a PC based communication models down, we believe that it is beneficial to tailor the communication system to meet the needs of these devices. To accomplish this, we have used the Active Messages communication model as primary building blocks for networking in TinyOS.

3.4.1 Active Messages Overview

Active Messages (AM) is a simple, extensible paradigm for message-based communication widely used in parallel and distributed computing systems [35]. Each Active Message contains the name of an application-level handler to be invoked on a target node upon arrival and a data payload to pass in as arguments. The handler function serves the dual purpose of extracting the message from the network and either integrating the data into the computation or sending a response message. The network is modeled as a pipeline with minimal buffering for messages. This eliminates many of the buffering

difficulties faced by communication schemes that use blocking protocols or special send/receive buffers. To prevent network congestion and ensure adequate performance, message handlers must be able to execute quickly and asynchronously.

Although Active Messages has its roots in large parallel processing machines and computing clusters, the same basic concepts can be used to the meet the constraints of networked sensors. Specifically, the lightweight architecture of Active Messages can be leveraged to balance the need for an extensible communication framework while maintaining efficiency and agility. More importantly, the event based handler invocation model allows application developers to avoid busy-waiting for data to arrive and allows the system to overlap communication with other activities such as interacting with sensors or executing other applications. It is this event centric nature of Active Messages which makes it a natural fit for TinyOS.

3.4.2 Tiny Active Messages Implementation

In bringing Active Messages out of the high performance parallel computing world and down into this low power design regime, we have attempted to preserve the basic concepts of integrating communication with computation and matching communication primitives to hardware capabilities. The overlap of computational work with application level communication is essential. Execution contexts and stack space must never be wasted because applications are blocked, waiting for communication. The Active Messages communication model can be viewed as a distributed event model where networked nodes send each other events. While quite basic, we believe that all applications can be built on top of this primitive model.

In order to make the Active Messages communication model a reality, certain primitives must be provided by the system. We believe that the three required primitives are: best effort message transmission with acknowledgements, addressing, and dispatch. More demanding applications may need to build more functionality on top of these primitives, but that is left for the applications developer to decide. By creating the minimal kernel of a communication system, all applications will be able to build on top of it. Additionally, it is likely that there will be a large variety of devices with different physical communication capabilities and needs. By building the communication kernel as separate components in the TinyOS component model, developers can pick and choose which implementations of the basic components they need. This can take the form of selecting from a collection of delivery components that perform different levels of error correction and detection. However, by providing a consistent interface to communication primitives, application developers can easily transfer their applications to different hardware platforms.

Just as there will be various implementations of the core components for the developer to choose from, various other extensions will be available, such as reliable delivery. This is similar to the design of Horus [37], which attempted to have modular PC-based communication protocols where application developers could choose from a variety of building blocks including encryption, flow control, and packet fragmentation. It is extremely advantageous to be able to customize protocols when dealing with network sensors due to their extreme performance constraints and their large diversity of physical hardware.

3.4.3 Buffer swapping memory management

Fundamental to the TinyOS communication system is its storage model. As data arrives over the radio, it must be stored into a memory buffer. The active messages dispatch layer then delivers the buffer up to the applications. In many cases, applications will wish to keep the message buffers that are delivered to them. In the case of the multi-hop communication, applications would need to keep the buffer long enough for it to be transmitted to the next node.

This situation is problematic in systems that do not support dynamic memory allocation because the radio subsystem must continually acquire new buffers to deliver to the applications. To handle this, TinyOS requires that each application return an unused message buffer to the radio subsystem each time a message is delivered. The radio simply maintains one extra buffer to receive the next message into. After reception the buffer is transferred up to the application. Then a free buffer is handed back by the application component to the radio system. This free buffer is then filled by the next message.

Through this buffer swapping scheme, a fixed set of pre-allocated messaging buffers can be used. If an application needs the ability to store several messages simultaneously, it simply allocates extra message buffers inside its own private frame. As messages arrive that need to be preserved, the application can return the pre-allocated buffers. It must maintain which buffers are empty and which are storing useful data. At all times, the application maintains control of a fixed number of buffers. The actual buffers are traded throughout the system as communication occurs.

3.4.4 Explicit Acknowledgement

While TinyOS only provides best-effort message delivery, we have found that it is important to receive feedback as to whether a transmission was successfully received by the next intended recipient. This simple piece of feedback greatly simplifies application level routing and reliable delivery algorithms.

In TinyOS 1.0, each messages transmission is followed by an immediate, synchronous acknowledgement that is transmitted from the receiver back to the transmitter. This acknowledgement is transmitted by an addressed recipient if a packet successfully passes a CRC check.

This form of packet acknowledgement is included at the physical layer of the TinyOS communication stack because it is significantly less expensive than providing the same functionality at the application level. Instead of transmitting a complete packet that contains an acknowledgement, as well as the information required to match the acknowledgement with the packet it is acknowledged, the TinyOS acknowledgement mechanism simply transmits a special sequence immediately after the packet is received.

In addition to reducing the amount of energy it takes to acknowledge a packet, this mechanism also simplifies the task of processing the acknowledgement. Because the timing of the acknowledgement is synchronous with respect to the packet transmission time, applications do not have to wait for a timeout to occur prior to determining that a transmission failed. The transmitting application knows the fate of the packet immediately after transmission. This allows the application to immediately initiate a retransmission or to search for a new recipient.

3.5 Components Contained in TinyOS

There are several system level components that are included in TinyOS. Additionally, there are a collection of example applications that demonstrate the usages of the TinyOS system. Figure 3-6 contains a listing of the components contained in the

TinyOS 1.0 System Components	
BapM.nc	Beaconless ad-hoc routing protocol
Counter.nc	Continually incrementing counter
IdentC.nc	Component to retrieve and set node identity
IntToLedsM.nc	Display an integer value on the LEDs
IntToRfmM.nc	Report an integer value over the radio
OscopeM.nc	Component that continually samples and reports and ADC channel
ResetC.nc	System reset component
RfmToIntM.nc	Receive and integer over the radio
Route	Multi-hop routing suite
SenseToInt.nc	Generate and iteger valued sensor reading from the ADC
TinyAlloc.nc	Memory allocator
TinyDB	Database application for TinyOS
ADCM.nc	Interface to the ADC
AMStandard.nc	Active Messages implemetnation
ByteEEPROM.nc	EEProm access componet
CRCPacket.nc	CRC packet calculator
ClockC.nc	Timing componet
CrcFilter.nc	CRC packet filter
GenericComm.nc	Generic communication stack for general use
I2CPacketC.nc	I2C protocol implementation
LedsC.nc	LED interface
LoggerM.nc	Interface to log data to the off-chip flash
NoCRCPacket.nc	Packet componet without CRC calculation
PotM.nc	Signal strength control
RFM.nc	TR1000 Radio interface
RadioCRCPacket.nc	TR1000 Packet interface with CRC
RadioNoCRCPacket.nc	TR1000 Packet interface without CRC
RandomLFSR.nc	Random Number Generator
SecDedRadioByteSignal.nc	Forward error correction component
TimerM.nc	Multi-application timer module
UARTComm.nc	UART communicaitons stack
UART.nc	UART interface module
UARTNoCRCPacket.nc	UART communication stack without CRC
VoltageM.nc	Battery voltage measurement component

Figure 3-6: Listing of components contained in TinyOS 1.0. Application developers select from this collection of system-level components to create the desired application attributes.

current TinyOS release.

3.6 TinyOS Model Evaluation

To evaluate the performance of the TinyOS system model we return to our set of key software requirements: Small physical size, concurrency intensive, efficient modularity, limited physical parallelism, and robust.

Small physical size: Application code size is our primary metric of performance for small physical size. TinyOS enables incredibly small applications. In particular, our scheduler only occupies 178 bytes and our complete network sensor application requires only about 3KB of instruction memory. Furthermore, the data size of our scheduler is only 16 bytes, which utilizes only 3% of the available data memory on the baseline Rene platform. An entire multi-hop data collection application comes in at 226 bytes of data, still under 50% of the 512 bytes available on the Rene platform.

Concurrency-intensive operations: It is essential that the software architecture be able to provide efficient concurrency; network sensors need to handle multiple flows of information simultaneously. In this context, an important baseline characteristic of a network sensor is its context switch speed. In the TinyOS event based system, the cost of propagating an event is roughly equivalent to that of copying one byte of data. This low overhead is essential for achieving modular efficiency. Posting a thread and switching context costs about as much as moving 6 bytes of memory. Our most expensive operation involves the low-level aspects of interrupt handling. Though the hardware operations for handling interrupts are fast, the software operations that save and restore registers in memory impose a significant overhead. Several techniques can be

used to reduce that overhead such as partitioning the register set [32] or use of register windows [38].

Efficient modularity: One of the key characteristics of our systems is that events and commands can propagate through components quickly. Projects such as paths, in Scout [39], and stackable systems [40] have had similar goals in other regimes.

The static compile-time optimization contained in the NESC compiler effectively eliminates the overhead associated with the TinyOS component model. At compile time, the component boundaries are automatically eliminated. Full program analysis is used to create eliminate system partitions and balance code size with execution time. In most cases all of the overhead associated with the component model is optimized out of the final compiled binary.

Limited physical parallelism and controller hierarchy: We have successfully demonstrated a system managing multiple flows of data through a single microcontroller. Figure 3-7 shows the work and energy distribution among each of our software components while engaged in active data transmission. It shows that 66.48% of the work of receiving packets is done in the RFM bit-level component which utilizes 30.08% of the CPU time during the entire period of receiving a packet. This consumes 451.17nJ per bit. Even during this highly active period, the processor is idle approximately 50% of the time. The remaining time can be used to access other sensors, like the photo sensor, or the I2C temperature controller. In Chapter 5 we investigate the tradeoffs associated with physical versus virtual parallelism in depth.

Diversity in usage and robust operation: Finally, we have been able to test the versatility of this architecture by creating sample applications that exploit the modular structure of our system. These include source based multi-hop routing applications, active badge-like [41] location detection applications and sensor network monitoring applications. Additionally by developing our system in C, we have the ability to target multiple CPU architectures in future systems.

3.7 TinyOS Summary

TinyOS is an operating system specifically designed to address the needs of wireless sensor networks. It is based on an event driven execution engine that simultaneously provides efficiency and fine-grained concurrency. TinyOS's event based

Componets	Packet reception work breakdown	Percent CPU utilization	Energy (nJ/bit)
AM	0.05%	0.02%	0.33
Packet	1.12%	0.51%	4.58
Radio Handler	26.87%	12.16%	182.38
Radio decode thread	5.48%	2.48%	37.2
RFM	66.48%	30.08%	451.17
Radio Reception	-	-	1350
Idle	-	54.75%	-
Toal	100.00%	100.00%	2028.66

Componets	Packet transmission work breakdown	Percent CPU utilization	Energy (nJ/bit)
AM	0.03%	0.01%	0.18
Packet	3.33%	1.59%	23.89
Radio Handler	35.32%	16.90%	253.55
Radio decode thread	4.53%	2.17%	32.52
RFM	56.80%	27.18%	407.17
Radio Transmission	-	-	1800
Idle	-	52.14%	-
Toal	100.00%	100.00%	4317.89

Figure 3-7: Detailed breakdown of work distribution and energy consumption across each layer on the Rene node.

communication is encapsulated in a component model that allows state-machine based components to be efficiently composed together into application-specific configurations.

The TinyOS communication system is based on the Active Messages communication paradigm taken from high-performance computing environments. When data is received from the network, message-specific handlers are automatically invoked. TinyOS includes a special-purpose memory model which allows a fixed set of message buffers to be pre-allocated and shared amongst several applications.

Now that we have an understanding of the software that will be running on these devices, we can begin to dive into the key issues associated with their networking protocols. In the next chapter we discuss a family of system-level tradeoffs and their impact on node performance. We use this analysis to develop a generalized node architecture that is the cornerstone of this thesis.

Chapter 4: Generalized Wireless Sensor Node Architecture

A key step in exploring the challenges of building a low-power wireless communication platform is in understanding the system requirements for the communication mechanisms necessary to construct the multi-hop networks that are envisioned. The focus of this chapter is to develop a general architecture that addresses the needs of wireless sensor networks. The architecture is not tied to any particular radio or processing technology but rather details how the computation and communication should brought together. It is designed to have communication subsystem that allows for flexible, application specific optimization of communication protocols while simultaneously obtaining high bandwidth and efficiency. To set the stage for the design, we first describe the basic operations associated with communication across a radio link. This exposes several of the core design challenges including the concurrency intensive nature of wireless sensor networks. To address the set of core challenges presented we develop a generalized architecture for a wireless sensor node. In the following chapters we analyze specific implementations of this general architecture.

4.1 Wireless communication requirements

To communicate over a wireless link, protocols must be built up from the raw electro-magnetic signaling primitives. A transmitter must carefully modulate the RF carrier while receiver performs demodulation and signal analysis. Figure 4-1 illustrates the key phases of a packet-based wireless communication protocol. It is important to

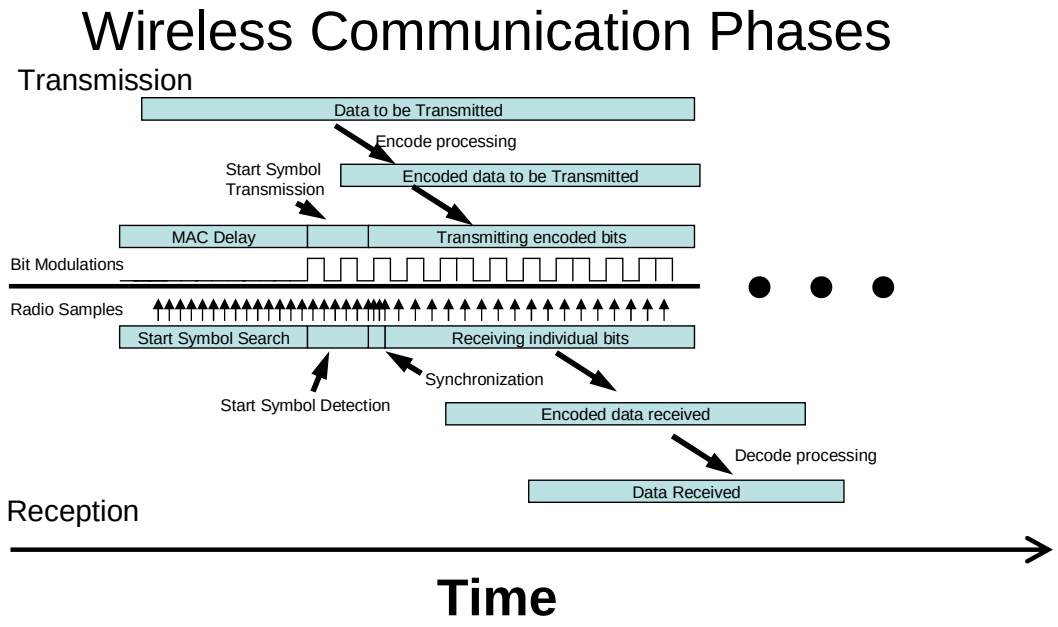

Figure 4-1: Phases of wireless communication for transmission and reception.

note that many of the operations must be performed in parallel with each other. This can be seen in the distinct layers that overlap in time.

The first step in the communication process is to encode the data for transmission. The coding schemes are designed to increase the probability of a successful transmission by preventing and correcting slight errors. For efficiency reasons the encoding process is pipelined with the actual transmission process. Once the first byte is encoded, transmission may begin. The remaining bytes can be encoded as preceding bytes are transmitted.

Coding schemes can range from simple DC-balancing schemes, such as 4b-6b or Manchester encoding, to complex CDMA schemes. In either, a collection of one or more data bits, called a *symbol*, are coded into a collection of radio transmission bits called *chips*. Manchester encoding has two chips per symbol which represents 1 bit of

data. Direct sequence spread spectrum and CDMA schemes often have 15 to 50 chips per symbol with each symbol containing 1 to 4 data bits.

The actual transmission begins with the execution of a media access control protocol (MAC). MAC protocols are designed to allow multiple transmitters to share a single communication channel. One of the simplest MAC protocols is carrier sense media access (CSMA) where each transmitter first checks for an idle channel prior to each transmission. If the channel is busy, it waits for a short, random, delay after which it reinitiates the transmission.

The first piece of data to be actually transmitted over the radio link is a synchronization symbol or start symbol. The start symbol signals to the receiver that a packet is coming and is used by the receiver to determine the timing of the arriving transmission. The start symbol is immediately followed by the encoded data transmitted as a serial stream. As the transmission proceeds, the transmitter must precisely control the timing of each bit transition so that the receiver can maintain synchronization. Skewed bit transitions can cause the sender and receiver to get out of synch, resulting in an unsuccessful transmission or corrupted data.

For a receiver, the first part of data reception is to detect that a transmission has begun. The channel is monitored continually in order to filter out background noise and detect the special start symbol. The length and format of the start symbol can be optimized for the expected noise levels. In order to properly detect the start symbol, the receiver must sample the channel at least twice the radio chip rate. Otherwise, the relative phase of the sampling and the transmission may result in the receiver missing the start symbol.

Once detected, the receiver must then synchronize itself to the exact phase of the incoming transmission. This synchronization step is critical in allowing the receiver to determine the start and end of each bit window being used by the transmitter. Synchronization requires the incoming transmission to be sampled higher than twice the bit rate so the timing of the bit transitions can be determined.

Once synchronized, the receiver then samples the value of the incoming signal at the center of each bit. Precise care must be taken to minimize skew in the sampling rate and timing. As the individual bits are extracted from the radio, they are assembled into blocks that are the encoded version of actual data messages. Finally, the blocks are decoded back into the original data and assembled into a packet. The decoding process can often correct bit errors in the received signal and reproduce the original data.

4.2 Key issues architecture must address

There are four key issues that must be addressed by the system architecture. First, from our basic description of a wireless communication protocol we can see that several operations must occur in parallel. The channel must be continually monitored, data must be encoded, and bits must be transferred to the radio. The ability to deal with fine-grained concurrency is required in order to perform these operations in parallel. Secondly, the system must be flexible to meet the wide range of target application scenarios. Third, the architecture must provide precise control over radio transmission timing. This requirement is driven by the need for ultra-low power communication in the data collection application scenario. Finally, the system must be able to decouple the data path speed and the radio transmission rate. We show how a direct coupling between

processing speed and communication bit rates can lead to sub-optimal energy performance.

4.2.1 Concurrency

The architecture must provide an efficient mechanism to handle fine-grained concurrency. In both the transmission and reception case, the computations associated with wireless communication must occur in parallel with application-level processing and potentially with intermediate level packet processing. Sensor events and data calculations must continue to proceed while communication is in progress. This is particularly true when considering overlapping start-symbol detection and application processing.

For example, start symbol detection must be performed continually when waiting for incoming communication. If any other processing is to be performed in must be done in parallel with the start symbol search. In addition to the parallel operation of protocol processing and application-level processing, we it is also required that several different steps of protocol processing must be performed in parallel. In Figure 4-1 shows how data encoding and data transmission are pipelined together in order to reduce transmission latency.

The system architecture must provide a fine-grained concurrency mechanism that allows for protocol processing and application-level processing to proceed concurrently.

4.2.2 Flexibility

The architecture must provide the flexibility to support a wide range of application scenarios. For deeply embedded systems, it is important that a system is

flexible enough to support a wide rage of application-specific protocols. Unlike cell phones, wireless local area networks, or Bluetooth devices, wireless senor networks do not have a fixed set of communication protocols that they must adhere to. Deeply embedded wireless networks can exploit tradeoffs between bandwidth, latency, and in-network processing to reduce power consumption by orders of magnitude.

For example, if sensor data is sampled only once per minute, it might be acceptable to delay transferring the data for several seconds, allowing the network to coordinate many such flows efficiently while operating at a low duty cycle. Such` optimizations can extend battery life from weeks to years.

In addition to supporting flexible communication protocols, it is also important to support flexible interfaces between protocols and applications. A rich interface between application processing and communication processing allows programmers to create arbitrary protocol decompositions. Internal protocol state can be exposed up to applications and applications can have fine-grained control over the protocols.

A simple example of the need for flexible protocols that allow applications to view the underlying channel characteristics can be seen in [42]. They discovered that many 802.11 cards abstract away signal quality from application level processing which eliminates their ability to support RF localization applications. Had the protocols for interfacing between applications and the communications hardware been flexible, that information could have been easily exposed.

4.2.3 Synchronization

The architecture must provide the ability to achieve precise node-to-node synchronization and give the application direct control of radio timing. This plays a

critical role in many wireless sensor network applications. As described in Chapter 2, our data collection application scenario relies on time synchronization to schedule the child-to-parent communication windows and the temporal accuracy of data collection. The short duration and frequency of the transmission windows makes it important for the application to have precise temporal control over the radio. This is in contrast to the communication patterns of a cell phone where communication occurs infrequently (a few calls per day) and for long durations (for minutes per call). A 10 ms delay before a cell phone call starts or stops has a negligible impact on the user experience. However if a packet is just 2 ms late in our data collection application scenario, it will be lost.

To allow applications to achieve precise time synchronization, radio communication protocols must provide exact timing information as to when data packets are received. Additionally, they must allow the application to precisely control when radio communication begins as well as when the radio is switched on or off for reception.

4.2.4 Decoupling between RF and processing speed

Finally, the system architecture must allow a decoupling between RF transmission rates and processing speed. A radio is the most efficient when data transmissions occur at its maximum transmission rate. When transmitting with a fixed power, reducing the transmission time reduces the energy used. However, modern studies in low power processor design and dynamic voltage scaling have shown that processors are the most efficient when they spread computation out in time as much as possible so that they can run at the lowest possible voltage [43, 44]. Ideally, for power consumption reasons, we would want the controller to perform all calculations as slowly as possible and just as the computation is complete, the radio would burst out the data as quickly as possible.

Coupling the speed of the microcontroller to the data transmission rate forces both pieces of the system to operate at non-optimal points. This coupling can be seen in the design of a wireless sensor node where a balance must be formed between the speed of the controller and the speed of the radio.

To dig into this requirement further we can analyze a performance shortcoming of the Rene platform. On the Rene platform, only 10% of the radio's 115Kbps capability is used. Increasing the clock speed of the microcontroller could easily increase the radio transmission speed because the speed of Programmed I/O is directly proportional to the data path speed. If the clock speed of the microcontroller were doubled, the resulting transmission rate would be 20Kbps and the energy consumed per bit by the radio would decrease by 50%. However, the speed increase from 4MHz to 8MHz increases the power consumption of the microcontroller from 15mW to 50 mW according to the AT90LS8535 data sheet [27]. Additionally, the idle mode power consumption would increase from 9mW to 28 mW. The savings in radio transmission power is quickly

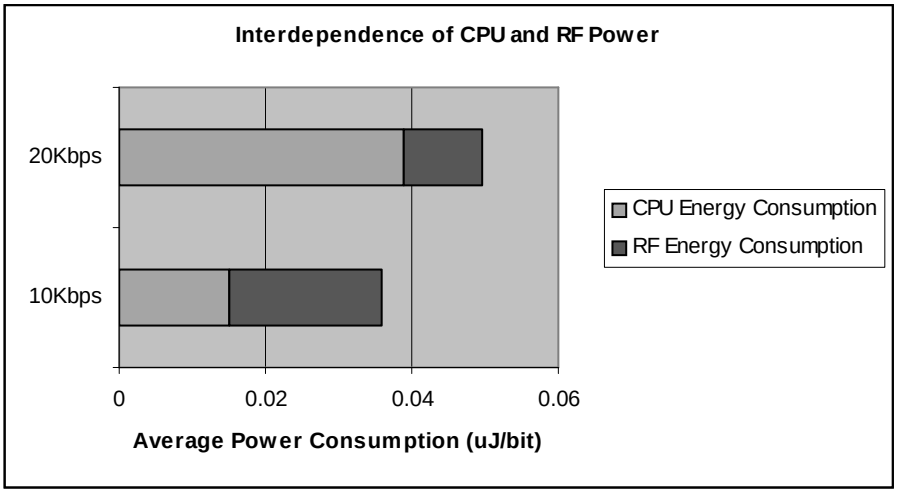

Figure 4-2: Energy cost per bit when transmitting at 10Kbps and 50Kbps broken down into controller and RF energy.

offset by the increase in the controller's power consumption.

To make this concrete, we consider an application where a device is transmitting 100% of the time, doubling the transmission rate would allow the device to spend 50% of its time in the idle mode. With a radio transmission consuming 21mW, the average power consumption of the improved system would be approximately .5*21mW+.5*50mW+.5*28mW or 49.5 mW. On the other hand, our original system only consumed an average of 36 mW. While expecting a decrease in power consumption, we see a net increase.

To address this issue, the system architecture must provide a mechanism for decoupling the transmission rate from the datapath speed. This observation provides an indication that there needs to be dedicated special-purpose resources for interacting with the radio. Special purpose hardware can operate at frequencies that are optimal for the radio, while the general-purpose controller is tuned to the optimal rate for computation. Through buffering the special-purpose hardware can perform a rate matching between the general-purpose data path and the radio. The amount of buffering sets the level of decoupling. If the entire packet were buffered the processor speed could be scaled to just meet the packet processing requirements set by the application scenario.

At first the need to decouple the transmission speed from the processing speed seems at odds with the goal of allowing applications precise, direct and flexible control over the radio communications. However, a processor supported by specialized hardware accelerators has the ability to do both.

4.3 Traditional Wireless Design

Traditional wireless designs such as cell phones, 802.11 wireless cards and Bluetooth enabled devices all choose to address the concurrency and decoupling issues by including a dedicated protocol processor. Application and protocol-level processing can proceed in parallel because they operate on separate physical hardware.

The protocol processor handles the real-time requirements of modulating and demodulating the radio channel, encoding the data for transmission, and power cycling the radio. It refines the raw data stream coming from the radio into a formatted packet interface that the application processor can easily handle. For example, the host channel interface (HCI) of Bluetooth provides a high-level packet interface over a UART. The intricacies of packet synchronization, channel encoding and media access control (MAC) protocols are all hidden from the application. The speed of the protocol processor is then set to meet the requirements of the communication protocols.

While simple, there are several disadvantages to including a dedicated protocol processor. The inclusion of the additional processor does not reduce the amount of computation that must be performed; it simply confines it to a secondary processor. It only provides a concurrency mechanism that allows application and protocol processing to occur simultaneously. Unfortunately, the packet processor is a system resource permanently dedicated to radio processing. This strict partitioning of resources leads to non-optimal resource utilization. Additionally, inefficient chip-to-chip communication mechanisms must be used to transfer data between the protocol processor and the application processor. In the end, the use of dedicated protocol processors leads to inefficient resource utilization.

An additional drawback to using a dedicated protocol processor, is that the application interface to the communication protocols must occur over a narrow chip-to-chip interface. A fixed set of control messages that travel between the application and protocol processor must be defined. This physical barrier between the application and protocol processing makes it difficult to perform cross-layer optimizations. We show how cross-layer optimizations that are prevented by this artificial barrier can lead to order of magnitude performance increases. Traditional wireless designs sacrifice flexibility and create strict system partitions when they include dedicated protocol processors.

An alternative to the protocol processor based approach is represented by the Rene design discussed in Chapter 2. Instead of using a dedicated packet processor, we time-share a single execution engine across application and protocol processing. The concurrency requirements of the system are met virtually by fine-grained interleaving of event processing in TinyOS instead of physically. This allows the computational resources of the system to be dynamically partitioned between application and protocol processing and avoids the overhead of shuttling packets between the processors, as well as the need to define a fixed interface. We show that the benefit of dynamic resource allocation can outweigh the overhead of using virtual parallelism. Additionally, in a virtually partitioned system, the interface between protocol and application processing is purely in software. This allows for rich interfaces to be constructed that can give applications tight control over the communication protocols. A rich interface between applications and their protocols can enable a host of system level optimizations.

4.4 Generalized architecture for a wireless sensor node

We have developed a generalized architecture for wireless sensor nodes that builds from the single controller design used in Rene. The architecture is based on the premise that shared pools of resources should be used when possible to exploit the benefits of dynamic allocation, that buffering needs be used to decouple the general-purpose data path and the radio, and that protocol flexibility is essential. It addresses performance and efficiency issues by including special-purpose hardware accelerators for handling the real-time, high-speed requirements of the radio. Accelerators provide general building blocks, not complete solutions.

Figure 4-3 depicts our generalized architecture. The core of the architecture is a central computational engine that is timeshared across application and protocol processing. Only a single computation engine is included because it allows the allocation of all processing resources to a single task when necessary. We show that this allows for an efficient use of processing capabilities.

Ideally, this single processor includes extra hardware support for the fine-grained concurrency that it must provide. As this processor is intended to be allocated to multiple concurrent operations, it must be designed so that context switching is as efficient as possible. A traditional mechanism for decreasing context switch overhead is through the inclusion of register windows. Multiple register sets can be included in the CPU so that each context switch does not require the registers to be written out to memory. Instead, the operating system simply switches to a free register set.

As is typical in microcontroller designs, the data path is connected to the rest of the system components through a shared interconnect. Memory, I/O ports, analog-to-

digital converters, system timers and hardware accelerators are attached to this interconnect. By utilizing a high-speed, low latency interconnect, data can be moved easily between the processor, memory and peripheral devices.

In addition to allowing the CPU to interact with its peripheral devices, this shared interconnect also allows the individual peripheral devices to interact with each other. A peripheral placed on this bus has the ability to pull data directly out of the memory subsystem or to push data into a UART peripheral. This creates a highly flexible system where a data encoding peripheral can pull data directly from memory and push it into a data transmission accelerator, such as modulation of an RF communication channel. In

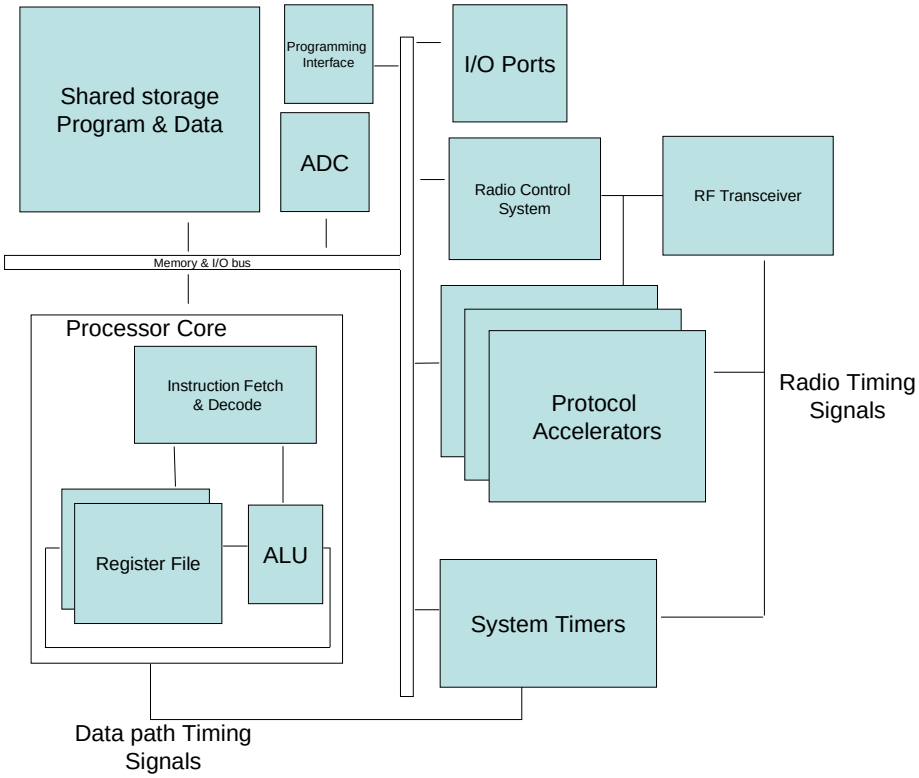

Figure 4-3: Generalized Architecture for embedded wireless device.

such a system, the CPU is simply orchestrating the data transmission; it does not have to directly handle the data.

All devices on this shared interconnect operate through a shared memory interface. Each device has control structures that are mapped into a shared address space. This allows components that were not originally intended to function together to be combined in new and interesting ways. A data encoder designed to read from memory, transform data, and write to memory many not even know that it is actually pulling data from a radio receiver block's memory interface and pushing it into a UART's portion of the communication memory. In this architecture, the size of the shared address space dedicated to each operation can be set dynamically to meet application requirements.

The true power of this system is in the special-purpose hardware accelerators that it enables. These accelerators provide efficient implementations of low-level operations that are inefficient on a general-purpose data path. Each accelerator is designed to provide support for operations that are critical to sensor network communication. By increasing the efficiency of these operations, the overall power consumption of the system can be greatly reduced. It is important that these accelerators are communication primitives instead of complete protocol implementations so that the system can support a wide range of communication protocols simultaneously simply through software reconfiguration.

The hardware accelerators also support operations that need to be performed as fast as possible to optimize the radio power consumption. This includes support for start symbol detection as well as the low-level bit modulation. The goal is to include the

minimum hardware functionality that is necessary to efficiently support the needs of applications and decouple communication rates from processing rates.

While the hardware accelerators are designed to provide primitives that can be used to construct communication protocols, these primitives are not necessarily simple. For example, a hardware accelerator for encryption would be considered a primitive. This component would take the data it is given and encrypt or decrypt it as necessary. As a hardware accelerator it could be used for encrypted communications, data authentication, or to ensure that data stored in an off-chip flash is kept secure. The hardware accelerator would be implemented so that the core cryptographic transformation was exposed to the CPU and other peripherals. This is in contrast to a system where an entire secure communication subsystem would be encapsulated without exposing any of the internal primitives. Once again this design choice allows for flexibility without sacrificing efficiency.

4.5 Architectural Benefits

To fully understand the tradeoffs and benefits of our generalized architecture we can return to the set of issues presented earlier in this chapter. While the hardware accelerators directly provide the ability to decouple the processing and communication rates, we must explore the concurrency and flexibility mechanisms of this architecture.

4.5.1 Concurrency

To evaluate the performance of the concurrency mechanisms, we analyze the performance of the most concurrency intensive aspect of communication, start symbol detection, with respect to two system architectures. We compare our shared processor

architecture to one that has a pair of processors, one dedicated to protocol processing and one dedicated to application processing. We break the CPU work down into three categories: search time, which is the cycles dedicated to performing the start symbol search, application time, which is cycles that the application can decide how to use, and overhead. In the case of our shared CPU, system overhead is the cost of performing context switches. In the dedicated protocol processor case, system overhead is the cost of the chip-to-chip communication and the over-provisioning required to handle worst-case deadlines.

To compare these two design points we fix the start symbol transmission rate at 10 Kbps and analyze the CPU usage of both systems assuming the same microcontroller is used. As a baseline for our shared CPU system, we use a single 4 MHz microcontroller. In the dedicated protocol processor based system, we size the protocol processor to just meet the worst-case timing requirements of the start symbol search and then size the application processor so that the system has the total equivalent processing of the 4 MHz system.

The start symbol search used in TinyOS [45] requires 49 cycles per sample on average, but has a worst-case overhead of 62 cycles. To detect a start symbol arriving at 10Kbps, a sample must be taken every 50 us. Assuming optimistically that the cost of the chip-to-chip interface is zero, the protocol processor must be able to perform the 62 cycle worst case sample every 50 us, so it operates at 1.25 MHz. Therefore, the application processor would be left with the remaining 2.75 MHz (line 2 of Figure 4-4). In actuality, we expect significant additional overhead caused by communication between the

application and protocol processor. To give the benefit of the doubt to the protocol processor approach, we ignore this for now.

On the shared CPU system, the context switch overhead must be included for each sample. Without any optimizations, 116 cycles are used per sample on average. This leaves just 42% of the 4 MHz processor (1.68 MHz) available for the application as shown in the top line of Figure 4-4. 33.5% of the CPU resources are spent on context switch overhead. In contrast, 6.5% of the CPU resources of the dedicated processor system are spent on the over-provisioning overhead. Figure 4-4 shows that there is more processing available to the application in the dedicated protocol processor case (line 2 and 3) than in the case of the non-optimized shared controller (line 1). With a brute force, naive, implementation, our concurrency model is significantly less efficient than the dedicated processor case.

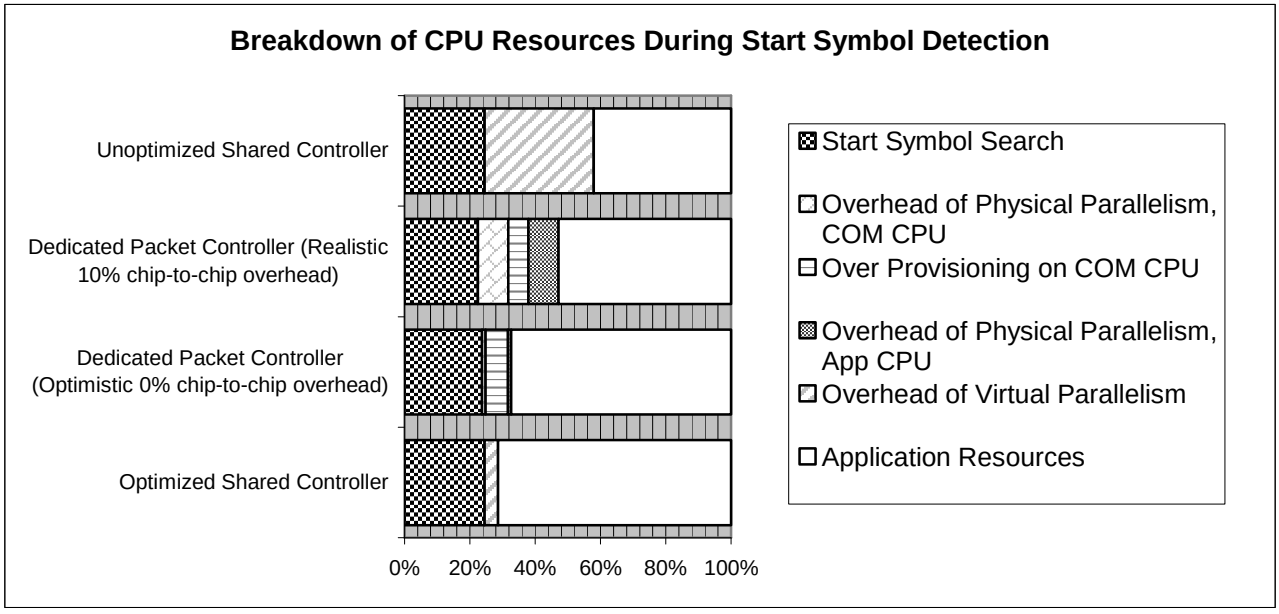

Figure 4-4: Breakdown of CPU usage for start symbol detection overhead broken down into search time, overhead, and CPU time for applications. For the dedicated controller case the breakdown is included with a realistic 10% chip-to-chip overhead estimate and with a best-case 0% overhead estimate.

However, while the over-provisioning is a fundamental part of the partitioned system, context switch overhead is something we can optimize. We can exploit dynamic resource allocation to reduce the average context switch overhead in the shared case. So far, our analysis is based on having both implementations process every sample in real time. While it is essential that a design be able to processes the important bits in real-time, not all bits have to be processed in that fashion. The first half of the start symbol can be detected lazily and then switch to reading each sample in real-time. If just 8 samples could be processed in batch, the average context switch overhead would be reduced by a factor of 8. This would result in a 4.19% context switch overhead, as shown at the bottom of Figure 4-4 (line 4), which is less that the 6.5% over provisioning overhead required in the case of the dedicated controller. Because processing the samples in batch does not change the worst-case requirements, the dedicated controller does not benefit from this optimization.

This analysis suggests that the widespread use of dedicated protocol processors is difficult to justify on performance or utilization grounds, especially when the natural dynamic partitioning of resources in the shared case is used to improve overall system performance. While the fixed overhead of context switching can be significant, techniques can be used to properly amortize it. The intermittent nature of the real-time requirements of wireless protocols makes it beneficial to have dynamic resources partitioning.

4.5.2 Protocol Flexibility and Timing Accuracy

We also must confirm that our architecture allows for protocol flexibility and direct control over communication timing. A key strength of this general architecture over traditional wireless systems is that it allows application developers direct access to each step of the communication process. Instead of abstracting the timing of bit sampling behind several system partitions, an application can directly calculate the arrival time of each radio chip. In turn, in can also control the exact transmission time. This creates the basic primitives for extremely precise communication scheduling. This tight coupling between applications and their communication protocols can lead to ultra precise time synchronization.

One of the key phases in wireless protocols is a media access control (MAC) phase. Arbitrating for the channel introduces random delay in packet transmission times. An application cannot know precisely when a packet will be transmitted when it initiates the send operation. When performing time synchronization over the radio link, this random delay degrades the accuracy of the synchronization algorithms. However, if there is a tight coupling between the application and the protocol, the MAC layer can inform the application what delay a packet experiences before it is transmitted. Additionally, this information can be used to modify the packet once the transmission begins so that it properly reflects the time that it was sent. This can lead to microsecond accurate timestamps on packet transmission.

4.6 Summary

In this chapter we have set the background for, and explored the underlying challenges of wireless node design. We then presented a generalized architecture that

addresses these issues. Now, we must evaluate this architecture on real-world hardware implementations. In the next chapter we present a first approximation of this generalized architecture in the form of the Mica node. This design serves to validate the architecture principles we have chosen. With it we will demonstrate the flexibility and efficiency of this architecture. It is the first step towards the final single-chip implementation presented in Chapter 6.

Chapter 5: Approximation of General Architecture: Mica

The Rene node served as a starting point for our study by allowing us to develop the core TinyOS concepts. Only after a complete system was developed around Rene, was it possible to analyze the hardware tradeoffs of the platform. The mica platform builds on Rene by adding key hardware accelerators in order to validate the generalized architecture.

Mica is the first test of our generalized architecture. Like Rene, it continues to have a central microcontroller that performs all computation but it supplements the controller with hardware accelerators, as dictated by our general architecture. The accelerators we have chosen to include support to increase the transmission bit rates and timing accuracy.

Unfortunately, Mica is only an approximation of our generalized architecture. Existing interfaces available on commercial microcontrollers severely limit how fully the architecture can be implemented, but nonetheless substantial performance and efficiency gains are possible. The design provides the baseline for more aggressive implementations and is useful in bringing to light the interactions between the various types of system components and their impact on performance. It is built using a single CPU that provides multithreading in software using the TinyOS concurrency mechanisms [45].

5.1 Mica design

Mica combines an Atmega103 processor with a RFM TR1000 radio, external storage and communication acceleration. A direct connection between the application controller and transceiver provides the flexibility to meet application designs. The hardware accelerators optionally assist to increase the performance of key phases of the wireless communication.

The form factor (1.25× 2.25 inch) is a similar size as a pair of AA batteries, although, we have compressed a variant of the design to about the size of a 2.5-centimeter coin .5 cm thick. The standby current for Mica's components is just a few micro amps, enabling applications that last for years on a single set of batteries. Mica also improves experimental flexibility by including an expansion bus that connects to a wide array of sensor boards. Currently sensor boards include support for monitoring thermal temperature, barometric pressure, magnetic fields, light, passive infrared, acceleration, vibration, and acoustics.

5.1.1 Block diagram overview

Figure 5-1 shows the Mica architecture, consisting of five major modules: processing, radio frequency (RF) communication, power management, I/O expansion, and secondary storage. A quick survey of the major modules provides a general overview for the system as a whole, hence, a detailed bill of materials, device schematic and datasheet for all components mentioned can be found at http://www.tinyos.net.

The main microcontroller is an Atmel ATMEGA103L or ATMEGA128 running at 4 MHz and delivering about four million instructions per second (MIPS) [46]. This 8-bit microcontroller has: 128-Kbyte flash program memory, 4-Kbyte static RAM, internal

8-channel 10-bit analog-to-digital converter, three hardware timers, 48 general-purpose I/O lines, one external universal asynchronous receiver transmitter (UART), and one serial peripheral interface (SPI) port.

Normally, programming of these embedded microcontrollers occurs during manufacture with a firmware upload or during field maintenance. In our design, however, the embedded network can be dynamically reprogrammed during routine use. The coprocessor handling wireless reprogramming is the same controller used on Rene. Additionally, to provide each node with a unique identification, we include a Maxim DS2401 silicon serial number—a low-cost ROM device with a minimal electronic interface and no power requirements [47]. The Mica radio module consists of an RF Monolithics TR1000 transceiver. It is configured in a manner similar to that of the Rene node.

A 4-Mbit Atmel AT45DB041B serial flash chip provides persistent data storage [48]. We chose this chip because of its interface and small footprint, 8-pin small-outline integrated circuit. It stores sensor data logs and temporarily holds program images that

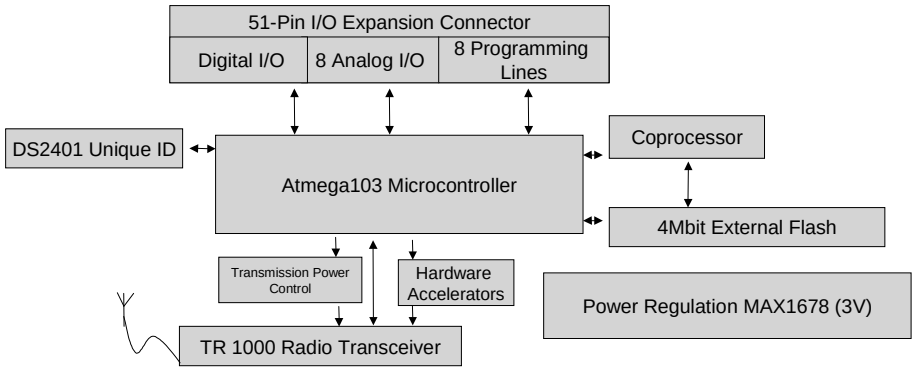

Figure 5-1: Block diagram of Mica architecture. The direct connection between application controller and transceiver allows the Mica node to be highly flexible to application demands. Hardware accelerations optionally assist in communication protocols.

received over the network interface. To hold a complete program the flash must be larger than the 128-Kbyte program memory. This prevented the project from considering use of the lower power, electronically-erasable-programmable-ROM-based solutions used on Rene because they are generally smaller than 32 Kbytes.

We designed the power subsystem to regulate the system's supply voltage; a Maxim1678 DC-DC converter provides a constant 3.3-V supply. Mica operates with inexpensive alkaline batteries and boosts their output voltage. We chose the Maxim chip because of its small form factor and high efficiency. The converter takes input voltage as low as 1.1 V. Input voltage significantly affects the TR1000's transmission strength and receive sensitivity. For ultra low-power sleep mode, disabling the power system lets the system run directly off the unregulated input voltage, reducing power consumption by the boost converter and the microcontroller. The radio will not operate, however, without the boost converter enabled. Figure 5-2 summarizes the Mica node's component power consumption.

The I/O subsystem interface consists of a 51-pin expansion connector that we designed to interface with a variety of sensing and programming boards. We divided the connector into the following sections: eight analog lines, eight power control lines, three

Mica Node Component Power Consumption

	Active	Idle
CPU	16.5 mW	30 uW
Radio	21 mW (TX), 15 mW (RX)	0 uW
Silicon ID	.015 mW	0 uW
External Flash	45 mW	30 uW
LED's	10 mW	0 uW

Figure 5-2: Breakdown of active and idle power consumption for Mica hardware components at 3V.

pulse width modulated lines, two analog compare lines, four external interrupt lines, an I^2C-bus from Philips Semiconductor, an SPI bus, a serial port, and, a collection of lines dedicated to programming the microcontrollers.

The expansion connector can also be used to program the device and to communicate with other devices, such as a PC serving as a gateway node. Additionally, it contains a standard UART interface to control or provide data to any RS-232 protocol based device. Dozens of sensor boards with a variety of sensors have been developed that use this expansion connector. It has even been used to allow the Mica node to control a handful of inch-sized micro-robotic platforms.

5.1.2 Raw radio interface

The radio subsystem is a prime example of the rich simple interface approach. From this subsystem's low-level interface, designers can build customized arbitrary signaling protocols to meet application requirements. The radio dictates certain parameters, that is, signaling should occur over a half-duplex bit-serial link using amplitude shift keying with a minimum physical bit time of 10 microseconds (μs), and the need for a rough direct current (DC) balance (for example, there should not be more than four consecutive high bits or four low bits). Beyond that, system software components determine all aspects of the communication protocol.

The TR1000 radio used on the Mica allows the controller direct access to the signal strength of the incoming RF transmission as well as a sampling the level of background noise during periods when there is no active transmission. Using this information in multi-hop networking applications can dramatically improve efficiency by allowing applications to use links with good signal to noise ratios. Additionally,

interactions between transmitter and receiver are very predictable, as is the delay through the radio interface. Because the radio can be powered on and off from software quickly and predictably, low duty cycle operation can occur without global coordination or complex time slotting. This direct, low-level interface to the radio provides flexibility for application developers. Researchers from the University of California, Los Angeles have exploited this flexibility to create energy aware MAC protocols [50].

5.2 Communication accelerators

The drawback of a low-level interface to the radio is that it places a significant overhead on the main controller. The frequency of programmed I/O operations and the inefficiency of conventional instruction sets for expressing these operations make it difficult to achieve high bit rates. It is inefficient to handle the bit-serial sliding window correlation operation required for start symbol detection on a general-purpose data path. However, this operation takes just a few gates to implement in hardware. To compensate for this, the Mica architecture includes hardware accelerators for the most demanding primitives used in protocol construction and allows these primitives to be composed in software, making rich interfaces available. On the Mica platform, we built the critical hardware accelerators using conventional serializers in unconventional ways. While they provide a significant efficiency boost, they represent just the beginning of what is possible as we progress toward custom devices that fully realize our general architecture.

The first accelerator we include provides a simple shared memory buffer between the bit-parallel data path and the bit-serial radio channel. It lets the processor deal with communication data in efficient chunks and overlap data manipulation with the low-level spooling of bits to or from the radio. The mechanism does not dictate channel coding or

signaling and can be bypassed to let the microcontroller directly interact with the radio using programmed I/O. This option meets real-time requirements when necessary, for example, during start symbol detection, but does not force real-time processing of all transfer options. This accelerator lets the Atmel controller drive the radio at 50 kbps using a small fraction of the processor, whereas programmed I/O peaks at 20 kbps using the entire processor. (Other microcontrollers with a double-buffered SPI transmit port can achieve the full 115 kbps of the radio.) The byte streaming performance of more sophisticated versions of this accelerator would approach conventional direct memory access channel performance, yet with precise control over timing.

The Mica node also incorporates a synchronization accelerator that captures the exact timing of an incoming packet to within one clock cycle (250 nanoseconds) at the start of packet reception. Shared memory stores this packet time stamp for the data path to access. This hardware accelerator is critical to our design because at a high bit rate, accurately determining transmission timing is difficult. The synchronization accelerator forwards this time stamp to the buffering accelerator so that automatic channel sampling can be performed by the buffering accelerator at the center of each bit transmission.

We built both of these hardware accelerators out of standard microcontroller functional units. SPI is a synchronous chip-to-chip interconnect with a serial data line and a separate clock signal that an external SPI master will provide. We drive the asynchronous radio by combining the functionality of input capture registers, timer controlled output pins, and the SPI communication hardware. The input capture register automatically captures a timing pulse contained in the packet. This value is used to configure the timing register that controls an output pin. One control option lets

hardware automatically toggle the output pin each time the counter expires; this output pin becomes a clocking signal that times each arriving bit. Finally, SPI hardware captures the value of the radio signal each time the counter triggers the clock line. We accomplish this by connecting the counter controlled output pin to the synchronous clock line of the SPI port.

During reception, we combine the radio's incoming receive data with an artificially generated clock signal to create an SPI master. The clock signal is the output of the timing register fed back into the controller's SPI port. This results in the SPI port automatically latching and buffering the incoming transmission with precise timing, as long as the internal timing register is configured correctly at the start of reception.

As in the general architecture presented in Chapter 4, these hardware accelerators are primitive building blocks that can be flexibly used to create communication protocols. They do not abstract away the low-level information. Rather they attempt to expose as must information as possible up to the microcontroller's data path so that flexible new compositions are possible.

5.3 Evaluation

To evaluate the Mica node we will return to the set of requirements we used to develop our general architecture. In particular we focus on the Mica node's ability to support the concurrency requirements of sensor networks, the ability to decouple communication and data path speed, the ability to provide precise temporal control over the radio and the ability to support a flexible set of communication protocols.

5.3.1 Concurrency Management

To analyze the efficiency of our concurrency mechanism, we return to our start symbol detection analysis. To optimally exploit the dynamic CPU allocation, we implemented a two-phased start symbol search mechanism. In the first stage, samples are buffered into 8-sample blocks and analyzed in batch. If the first half of the start symbol is detected, the next set of samples is analyzed in real-time. If the remainder of the symbol is found, then the packet reception is started. If the symbol is not found, then the CPU returns to batch processing mode.

This exploits the buffering accelerator to amortize the context switch cost across multiple bits of data. It reduces the average context switch overhead from 33.5% down to just 4.2%. This reduces the average CPU usage of the start symbol search and the context switch overhead to 22.1% for the 10Kbps start symbol search. Not only did we successfully amortize the context switch costs, but the batch mode processing of the search itself is more efficient. The special purpose hardware has increased the CPU resources available to the application processing by 85% and outperforms our optimally

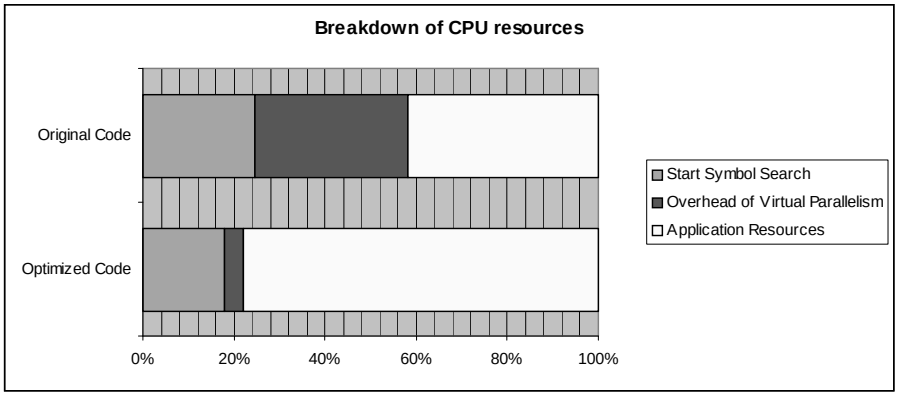

Figure 5-3: CPU utilization of start symbol detection. Optimizations enabled by the Mica node significantly increase CPU efficiency.

partitioned dedicated packet controller based system.

With this new start symbol search efficiency we can either spend more time on application processing, slow down the CPU, or increase the start symbol transmission rate. Depending on application specific demands all three of these options could lead to a more efficient design point system. Figure 5-3 outlines these results.

5.3.2 Interplay between RF and Data path speed

In addition to using the hardware accelerators to decrease overhead, we have also used it to decouple the data path from the transmission rate. During transmission, the real-time, low-level bit sampling is now being performed in hardware. Instead of using programmed I/O for each bit, we interface the data path at the byte level. Ideally, we would have included more buffering than a single byte, but it was not possible when using commodity Atmel microcontrollers. However, including this limited amount of decoupling allows us to significantly increase transmission rates.

The implementation demonstrated on Rene used programmed I/O exclusively and recorded a maximum bandwidth of 10Kbps. This was limited by the CPU overhead associated with taking an interrupt with each bit. With our exploitation of the SPI based hardware accelerator, we have been able to reach speeds of up to 50 Kbps on the same processor. Our 5x speedup is limited by the tiny amount of buffering present on our device. We have been able to demonstrate 115 Kbps transmissions on a TI MSP430 [24] microcontroller that that also runs at 4 MHz, but has two bytes of buffering in the SPI port. In addition to increasing the bandwidth, we also have decreased the CPU overhead by dealing with data in byte-sized chunks. Eliminating the bit manipulation operations from the data path significantly reduced the computational overhead on the system.

5.3.3 Interface flexibility

The third architectural issue is the efficient support of flexible protocol interfaces. The increase in performance that we have shown demonstrates our designs ability to be efficient. To demonstrate its flexibility we have implemented a pair of protocols that exploit application specific optimizations. They are a high accuracy time synchronization mechanism and an ultra-low power RF sleep protocol.

5.3.3.1 Cross-layer optimizations

Many deeply embedded applications spend the vast majority of the time in a very low-power state, slowly draining their available energy supply by processing sensor readings to confirm that no particular action need be taken. Infrequently, these applications communicate monitoring results and, rarely, portions of the network become extremely active upon detection of an important event. Performance improvements in an application's active work, such as processing and messaging, enhance the application's overall effectiveness and let it return to a low-power state more quickly. However, the larger impact of the richly integrated approach to wireless design is that enables optimization of the rest of the application. Often these optimizations reach across traditional layers of abstraction, using low-level information to achieve a high-level goal.

1) Cost of checking = (radio on time) * (radio power consumption)

2) Power consumption = (checking frequency) * (cost of checking)

3) Average wakeup time = ½ (checking period) = 1/(2 * checking frequency)

Figure 5-4: Equations for determining the power consumption of a sleeping network.

In addition to delivering high bandwidth data communication using standard protocols, we have also been able to demonstrate how the flexibility we provided plays a critical role in enabling systems-level optimizations. We show this by implementing four application specific protocols that improve overall system performance for their intended application. The first of these is a protocol that explores the tradeoff between transmitter and receiver overhead.

5.3.3.1.1.1 RF Wakeup

The raw interface to the radio can be exploited to implement an ultra low power radio-based network wakeup signal. As seen in our alarm system application, it important to be able to initiate unscheduled communication while the network is in a low-power state. The first step in this process is to wake-up a sleeping network.

For any RF based wake-up protocol, each node must periodically turn on the radio and check for wakeup signal. Figure 5-4 contains the equations necessary to determine the power consumption of a sleeping network. Each time a node checks for the wakeup signal, it will consume energy equal to the power consumption of the radio times the time the radios is on (1). The power consumed by the sleeping node will be the energy used each time it checks for the signal times the frequency of the check (2).

Minimizing the time a radio must be turned on each time a node checks for the wakeup signal and minimizing the check frequency are the two mechanisms for reducing the energy consumption of the system. However, the frequency that nodes check for a wake-up signal determines the amount of time that it takes for the network to wakeup. If each node were to check for a wake-up signal every minute, the average expected wake-up time for a single node would be 30 seconds (3). This means higher frequency checking yields faster wake-up times and better application performance. Because of this, we focus on minimizing the time it takes to check for a wakeup signal.

Using a packet-based radio protocol, each node has to turn on the radio for at least two packet times[1]. In our system, a packet transmission time is approximately 50ms, so each node would have to be awake for at 100ms each time it checks for a wake-up message. If a node needs to wake-up every minute, this yields a best-case radio duty cycle of .2%. This time window gets even larger when considering the effects of a multi-hop network and the contention associated with having a large number of nodes retransmit the wake-up signal simultaneously.

Instead of interacting with the radio over a high-level packet interface, our low power sleep mode implementation interacts directly with the analog base-band output. The wake-up signal is nothing more than a long RF pulse. Each time a node checks for the wake-up signal, it can determine that the wake-up signal is not present in 50us. The 50us signal detection time is a 2000x improvement over a packet-based detection time. In our implementation, we choose to sample every 4 seconds which results in a .00125 %

[1] If a node were sending a continual stream of wake-up packets, listening for two packet times would ensure that a complete packet was transmitted while the node was awake.

radio duty cycle, a 160x improvement over the packet-based protocol that was sampling once per minute. This ultra-low duty cycle implementation has been used to consistently wake-up multi-hop sensor network of more than 800 nodes. This radically different signaling scheme would not have been possible if a protocol processor was constraining applications use of the radio.

The system-level protocol for managing the transition between the ultra-low power sleep mode and the active mode is quite simple. Assuming that nodes start in the sleep mode, each node checks for the wake-up signal. Once a wake-up signal is detected, it listens for a secondary message that contains information about how long it is to wake-up and if it should wake-up its neighbors. If neighbors are to be woken up, then the node will retransmit the pair of wake-up messages. Supplementing the ultra-low power wake-up message with a control message prevents unwanted network wake-up. The control message can exploit error checking and message authentication techniques that are not possible with the low-power wake-up message. If a secondary message is not received that the node can quickly return to the low-power state.

Once a network is awake, it can be placed back into a low-power state by flooding out a command message. The only requirement is that the nodes delay for a short period of time after receiving the sleep command so that they ensure all of their neighbors have heard the message also.

5.3.3.1.1.2 Low Power Listening

In sensor networks, multi-hop routing topologies are built by having intermediate nodes act as relays for remote nodes. These routing nodes must listen for communication and propagate messages towards their destination. While listening, the

radio consumes almost as much energy as when transmitting. Even when no communication is taking place, considerable amounts of energy is spent searching for the next packet. In the data collection network the energy spent while waiting for a transmission can represent more than 50% of a node's total energy budget. Embedded sensor researchers have claimed that high-level protocols must be used to reduce energy consumed of nodes when not actively transmitting [2]. As discussed in Chapter 2, one solution is to have windows of communication periods and windows of sleep periods[51]. However, it has been shown that in some situations poor interaction between high-level power saving techniques and low-level communication protocols can actually lead to an increase in energy consumption when using windowing mechanisms [52].

We have been able to show that alternative is to modify the lowest levels of the communication stack. Unlike windowing, which layers itself on top of standard low-level protocols, we change the underlying communication protocol to optimize for the receiver power consumption. Instead of having the sender simply transmit a start symbol followed by a packet, we can require that the sender first transmit a preamble to get the attention of any possible receiver. It can then follow with the start symbol and packet. The receiver then only has to listen often enough to pick up any portion of the attention signal. Precise control over the power state of the radio allows the protocol to turn off the radio between each sample. The duty cycle of the receiver becomes proportional to the length of this preamble. Once detected, the preamble will cause the receiver to search for the pending start symbol.

This protocol optimization trades power consumption on the sender for power consumption on the receiver. The sender must transmit longer, but the receiver can

sample the radio channel less frequently. The optimal ratio is dependent on the communication patterns of the application. Specifically, it depends on the transmission rate, duty cycle and the maximum allowable delay.

We have implemented a version of this scheme where the receiver has a 10% duty cycle and the sender must transmit a preamble of 5 bits. The 10% receiver duty cycle not only results in a 90% reduction of radio power consumption, it also produced a 33% reduction in the CPU overhead associated with monitoring the radio. On the other hand, the sender only incurs a slight increase in overhead – less than 1% overhead on a 30-byte packet. This slight overhead increase is because the sample rate of the receiver is 3000 times per second. Fine-grained control is used to power-on, sample, and power-off the radio in 30us. Application level windowing protocols generally have window sizes of seconds or more.

Depending on application specific goals, the receiver overhead can be reduced arbitrarily at the expense of bandwidth, latency, and transmission overhead. Moreover, an application can change the protocol at runtime based on network activity. This simple optimization demonstrates the benefits that are enabled by our architecture for flexible communication protocols by exploiting the ability to tailor protocols to application specific criteria.

5.3.3.1.1.3 Time synchronization

Our data collection application requires time-correlated sensor readings and precise communication scheduling. The accuracy of distributed synchronization protocols is bounded by the unpredictable jitter on communication times [53]. Aside from variations in the actual network latency, the variation in delay going through

sophisticated protocol processors can be hundreds of milliseconds due to buffering, MAC protocols, and back off. Unlike wide-area time synchronization protocols—such as network time protocol[8]—we can determine all sources of communication delay [54]. Our rich interfaces allow us to expose the sources of delay to the application, reducing the unknown jitter. Additionally, by exploiting shared system timers, we can assign precise timestamps to incoming and outgoing packets beneath the sources of variation.

We designed the Mica platform to use an internal 16-bit high-frequency counter to act as the lower 16 bits of a 32-bit continually running system clock. This highly accurate system clock is directly linked to the synchronization accelerator used to capture the exact timing of an incoming packet. To synchronize a pair of nodes, a packet can be time-stamped with a sender's clock value as it is transmitted and after all MAC delays have occurred (prior to transmission, a node might have to wait for the radio channel to be clear). The receiver's synchronization accelerator can then tag the packet with the receiver's clock value. The application can use the difference between the two time stamps to determine how to adjust its system clock.

Our implementation synchronizes a pair of nodes to within 2 μs. We can directly attribute the skew of +/- 2 μs to several sources of jitter. When sending, there is a jitter of +/- 1μs in the transmission propagation due to the internal circuit dynamics of the radio. Hardware then captures the arriving pulse with an accuracy of +/- .25μs. Finally, to synchronize the clock based on the captured value introduces an additional +/- .625 μs of jitter. This implementation is only possible because of the shared access to a high-accuracy system timer between the bottom of the network stack and the top of the

application. Partitioned wireless devices—such as Bluetooth chipsets—hide the exact timing information from applications.

Generally the receive packet path has less jitter than the transmit path, so an alternative approach uses the broadcast nature of the radio channel to synchronize multiple receivers, even if they are poorly synchronized with the transmitter. Mica's time synchronization primitives provide highly accurate pair wise synchronization between any two nodes. This primitive can aid construction of network-wide time synchronization to within tens of microseconds.

5.3.3.1.1.4 Localization

Node localization is a key part of our node tracking application scenario. Direct access to the base-band signal being output by the TR1000's receiver chain can be exploited to assist with localization. Several groups have attempted to perform localization in a sensor network by using RF signal strength[42, 55, 56]. The radio is used as an analog sensor to detect the strength of an incoming signal. RF propagation models are then applied to infer distance from a collection of strength readings.

By providing the processor direct access to the raw base-band signal, our platform gives the application developer as much information as possible about the incoming signal. The central controller can look at the signal strength of each individual bit as well as the level of the background noise. Additionally, because the sender has direct control over the base band signal being transmitted, it can intentionally transmit long duration pulses of variable strength to help the receiver determine the reception strength more accurately.

An alternative to RF localization is to use acoustic localization[57]. The propagation delay of acoustic pulses is measured and used to infer distance. For accurate results, it is essential to use the radio to achieve precise time synchronization between two nodes in order to correlate their sensor readings in time. For accurate results, the transmission time of the acoustic pulse must be accurately communicated to the receiver and correlated with the receive time. The precise communication synchronization provided by the radio layer allows this to be done incredible accurately.

5.4 Blue: a follow-on to Mica

The design point represented by Mica has been taken one step further by Dust Inc. The Dust Inc. Blue mote replaces the individual system components while maintaining the same basic architecture. The Blue mote continues to run TinyOS, yet it exploits a new generation of microcontroller and radio.

5.4.1 CPU

The first system component to be changed is the central microcontroller. Blue abandons the AVR based microcontroller in favor of a 16-bit Texas Instruments MSP430 controller [24]. The TI controller has been optimized for low-power, battery operated systems. As noted in Chapter 2, a key microcontroller characteristic is the ability to quickly enter and exit sleep modes. The MPS430 has the ability to enter and exit a 2 uA sleep state in under 100 us. Additionally, the core has been designed to let subsystems operate while the system clock is disabled. ADC operations, UART communication and radio data transfers can proceed while the main CPU core is dormant. These features greatly reduce the duty cycle and in turn energy consumption of the CPU core.

The TI CPU has additional benefits. The overall power consumption of the CPU in active mode is reduced from the 5 mA of the AVR core to just 1 mA when operating at 3 V. Additionally, the TI CPU is capable of operating over a voltage range down to 1.8 volts. This allows it to operate directly off of alkaline batteries without requiring a costly boost converter.

Additionally, the ADC resolution is increased from 10-bits to 12-bits and the ADC sample rate is increased to 100 Ksps. Finally, the part cost of the TI CPU is significantly less that that of the Atmel version. Overall the blue mote is expected to yield a 50% cost reduction and a 4-6x life improvement.

5.4.2 Radio

The second significant upgrade on the Blue node is the use of an FM radio instead of an AM radio. The Blue node employs a 900 MHz FSK radio – the Chipcon CC1000

Figure 5-5: Follow-on to the Mica node, blue increase range and reduces power consumption. Blue nodes have a 4-6x longer battery life over Mica on alkaline batteries.

that we outlined in Chapter 2[23]. While consuming more power than the RFM TR1000 radio used on the Mica node, the CC1000 has increased sensitivity and increased transmission power. The use of FM modulation also allows the CC1000 to be more tolerant to external interference.

Additionally, the CC1000 radio is frequency agile. The carrier frequency of the blue mote can be tuned anywhere in the 902-928 frequency range. By default, the Blue node provides 25 evenly spaced channels. These channels can be used to avoid external interference or to divide network bandwidth across the individual channels. On blue, the CC1000 provides 76.8 Kbps of bandwidth on each channel, up from the 50Kbps possible on the Mica node single channel.

5.5 Summary

The Mica node is the first approximation of our generalized architecture. Built from off-the-shelf components, it has proven invaluable in verifying the validity of our general architecture. However, it is constrained by the available interfaces and internal behavior of commercially available components. Despite its limitations, it has been used by over a hundred research groups as a platform for experimenting in the area of wireless sensor networks.

Mica is a careful balance of efficiency and flexibility. Efficiency is essential for all sensor network systems while flexibility allows the Mica platform to address a wide range of application scenarios. The success of the Mica platform enables us to take the next evolutionary step – single chip integration.

The Mica node has demonstrated how the generalize architecture presented in Chapter 4 has the potential to produce highly efficient and flexible nodes. When

compared to the Rene platform, the mica hardware accelerators significantly increase communication bit rates and timing accuracy while reducing CPU overhead.

The Blue platform follow-on to the Mica node maintains the same basic architecture of the Mica node but selects hardware components that significantly improve node lifetime and communication range. As described in Chapter 2 the ability to tolerate a wide range of input voltages significantly increases node lifetime when operating off of alkaline batteries.

In the next chapter we make the leap from constructing system out of off-the-shelf components to creating custom silicon. This allows us to fully realize the generalize architecture presented in Chapter 4. We show the inclusion of key hardware accelerators can lead to dramatic improvements in performance without sacrificing efficiency. The design process results in a single-chip CMOS design that integrates a radio, computation, and storage into a single millimeter-sized node.

Chapter 6: Integrated Architecture for Wireless Sensor Nodes – Spec

The next leap towards demonstrating the power of our generalized node architecture is to move beyond what is possible with off the shelf components and design a custom integrated solution. While approximating our general architecture, the Mica and Blue nodes were constrained by existing inter-chip interfaces. Development of a custom ASIC allows us to tear down the artificial constraints imposed by commercial components. Through the use of custom silicon we are able to achieve orders-of-magnitude efficiency improvements on key communication primitives. Additionally, we demonstrate that our generalized architecture allows us to achieve these performance improvements without sacrificing flexibility.

Figure 6-1: The single chip Spec node pictured next to a ballpoint pen.

In addition to the architectural benefits, single-chip integration also has significant size and efficiency improvements. Our latest design is an ASIC based node called Spec. Despite including a microcontroller, SRAM, communication accelerators and a 900 MHz multi-channel transmitter, the Spec node measures just 2.5 mm on a side in a .25 um CMOS process. Its architecture blends the efficiency of a dedicated hardware implementation with the flexibility of software by providing low-level primitive accelerators that are dynamically composed together. The integration results in a device which only requires a handful of low-cost external components, including a crystal, battery, inductor and an antenna to be a complete wireless sensor network node.

6.1.1 High Level Overview

In order to support a wide range of application scenarios and the ability to perform application specific optimizations, the Spec node is designed to be a collection of basic primitives that can be flexibility composed. This is in contrast to systems that are built as a hierarchy of specialized sub-systems. Instead of having protocol engines and dedicated radio controllers, Spec is built out of low-level pattern matchers, timing extractors and serializers. The simple primitives can be composed together by the central controller to implement a wide array of application specific communication protocols.

The true benefits of our generalized architecture are only realizable when combined with high speed on-chip interconnects. With Spec we are able to design specialized accelerators targeting specific communication needs and connect them to a

shared, high-speed interconnect. This high speed, low-latency interconnect is essential in providing an efficient way of composing and configuring the discrete primitives.

The accelerators in the Spec node were selected so that the CPU intensive portions of RF communication could be offloaded. As we considered each hardware accelerator, we compared the potential in energy saved against the loss of flexibility. If we only wanted to implement a single protocol, we could implement it directly in hardware. However, such a node could only support a handful of applications. By carefully selecting the correct set of primitives, we can implement a large set of protocols with a minimal impact on efficiency.

The Spec node consists of SRAM, a RISC CPU microcontroller core, a 900 MHz RF transmitter [58], a collection of communication hardware accelerators, and an analog-to-digital converter [59] all packed together. In addition to providing primitives to assist in implementing RF communication, the Spec node also has support for the highly efficient interrupt handling. Profiling of TinyOS revealed the significance of efficient interrupt handling. Sensor networks must be continually monitoring their environment. Each sample triggers several interrupts that perform small amounts of computation. Interrupt overhead accounts for a significant amount of the processor's energy consumption; this makes it an ideal candidate for optimization.

6.1.2 General Block Diagram

Figure 6-2 depicts the general block diagram of the Spec node. The CPU core is a basic 8-bit RISC core with 16-bit instructions. To remain compatible with the Mica instruction set, Spec has a Harvard architecture with separate address spaces for data and

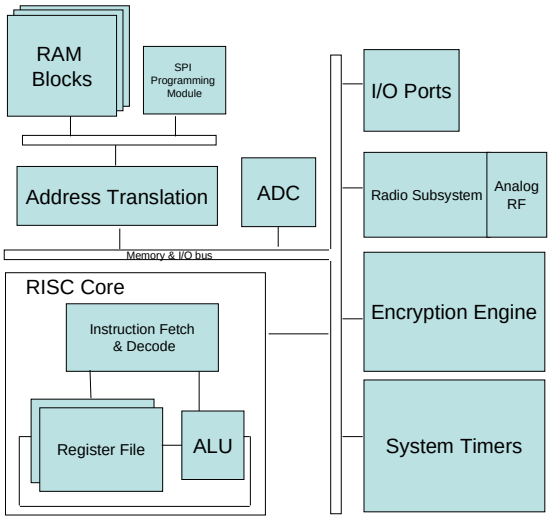

Figure 6-2: Block Diagram of Spec, the single chip wireless mote.

program memory. The CPU can execute ALU operations in a single cycle but memory operations take two cycles. The instruction set architecture contains 32 general purpose registers some of which can be combined together to form 16-bit addresses for use in load and store operations. Additionally there is a dedicated stack pointer that is automatically manipulated by jump and return instructions. On Spec, architected registers are supplemented by an additional set of 32 general purpose registers to allow for optimization of interrupt handling routines.

The CPU core is connected to a bank of 6 memory blocks each containing 512 bytes of data. These blocks can be individually mapped into either the data or program address space of the CPU. Only 3K of memory is included in our current design. Clearly the total amount of ram could be increased at the expense of die area. It is important that the memory be divided into banks so that it can be migrated between

instruction and data memory. This mapping is handled by an address translation unit that is interposed between the memory and the processing core. The memory mapping unit contains space for 16 physical frames. The Spec chip uses 7 of them – 6 for the memory banks and one for the interface to the radio subsystem.

In addition to the memory controller, the CPU is also connected to an ultra-low power analog to digital converter, an encryption accelerator, general purpose I/O ports, system timers, a chip programming module, and a RF sub-system. For illustrative purposes we have grouped the communication accelerators into a subsystem. However, it is actually composed of a collection of accelerators that we describe in detail. Each of the communication accelerators connect directly to the microcontroller's data bus. The accelerators can be linked together or used in isolation.

The communication subsystem contains primitives that: extract and generate bit timing, perform pattern matching for start symbol detection, stream received and transmitted data into and out of memory, encrypt and decrypt data automatically and provide memory-mapped control interfaces to the radio circuitry.

6.1.2.1 CPU Core

The standard 8-bit RISC microcontroller on the Spec node is a traditional single cycle core. It has a 16-bit instruction bus and an 8-bit data bus. The only form of pipelining is a pre-fetch of the next instruction while executing the current instruction. Decode, execution, and write back are performed in a single cycle. In addition to traditional RISC instructions, Spec also includes a set of instructions for interacting with

64 specialized I/O registers. I/O operations to these registers can be performed in a single cycle. However, memory operations require two cycles.

The actual instruction set architecture is not of particular importance. Key constraints are that the instruction representation must be compact (as is the case with most microcontrollers), and that I/O operations can be performed efficiently.

More important than the actual ISA is the ability to efficiently handle interrupts. The event based programming of TinyOS relies heavily on the ability to process interrupts quickly. We include specialized support for interrupt handling in the form of register windows. The CPU core includes two set of registers. One set is used for general processing and one for use in the context of interrupts. Background processing such as TinyOS tasks can easily share one set of registers. When an interrupt arrives, the interrupt handler can quickly switch register windows by executing a single specialized instruction. It can then proceed to execute the handler code and return. The interrupt handler can only use the secondary register set if it does not re-enable interrupts during its execution.

To access the secondary register set, a specialized “switch registers” instruction has been included in the ISA. This single instruction replaces the multiple memory accesses that are generally required to write the saved registers out to memory.

6.1.2.2 Paged RAM

In addition to a specialized data path, the Spec chip has a specially designed memory system. Traditional embedded systems have a strict partition between program and data memory. This is done, in part, because embedded systems are generally designed with a ‘write once, run forever’ model in mind. Embedded software is usually

written with a single application in mind and then burned into a device where it will remain for the life of the device. The dynamic nature of sensor networks makes it impractical to maintain that programming model.

While a strictly partitioned memory system is practical in traditional application scenarios, wireless sensor networks demand more flexibility. The wide range of target applications makes it desirable for nodes to be field programmable. Users want to be able to download new programs over the radio and into the application space.

To support this style of operation, the Spec chip includes a specialized paged memory system where physical page frames can be dynamically mapped into any part of program or data memory. With this mechanism applications can be loaded over the radio and into data memory. Once the application is complete and verified, the memory frames used to store the program can be re-mapped into the instruction memory of the CPU. At the same time, the original application is automatically mapped out of the instruction memory. In just a few cycles, the entire program memory can be swapped out. In the event that the new program is determined to be faulty, the original program can be quickly restored.

The Spec memory mapped system contains 16 page frames each containing 512 bytes of memory. The page frame mapping is controlled by writing to special addresses located at 0xffc0. During operation, the low 9 bits of the memory address are passed unchanged to the memory system. The remaining 7 bits are passed through a remapping table that dynamically performs the translation. Both the instruction and data memory addresses are translated simultaneously by separate control blocks.

When a physical page is mapped into a new location that is already filled, the old page is removed from the address space all together. The old page must be mapped elsewhere in the address space before it can be accessed again. When mapped into data memory, each page is byte addressed over an 8-bit data bus. When mapped into data memory, the ram provides a 16-bit, word addressed, read-only memory.

6.1.2.3 Communication Acceleration

One of the primary concepts contained in the Spec design is the inclusion of hardware acceleration primitives designed to reduce the cost of radio communication without sacrificing flexibility.

The communication accelerators on the Spec node include support for synchronization, timing extraction, encryption and data serialization. During communication, the first of the accelerators to come into play is the synchronization accelerator.

6.1.2.3.1 Synchronization

The synchronization accelerator is designed to process the incoming bit stream and detect the presence of a pre-determined start sequence. First, a configuration register is used to specify the value and length of the target sequence. Then the incoming data stream is automatically analyzed in search of the sequence. When detected, the accelerator signals the CPU via an interrupt mechanism. It also is interconnected to a data serialization accelerator with automatically records the incoming transmission.

6.1.2.3.2 Memory Mapped I/O & Data serialization

Data serialization acceleration has been included to automatically stream data between the radio and memory system. It is a simplified DMA engine. During radio transmissions, each bit must be recorded or transmitted serially. The accelerator is enabled by specifying a memory address for the start of the packet and a maximum packet length. In TX mode you set the packet length to a non-zero number to initiate a transmission. The transmission pointer is automatically incremented with each byte transmitted. Accordingly, the remaining packet length is decremented to represent the number of bytes left in the transmission. Both of these fields can be accessed by the processor to monitor progress.

During reception, either a signal from the CPU or a signal from the synchronization accelerators causes the incoming transmission to be recorded to the address of the receive memory pointer. Data is recorded based on an receive data length field.

Memory operations for the I/O system are automatically interleaved with the memory operations of the CPU. Most CPU instructions do not access the data memory, only instruction memory. Spec has been designed so the program and data memory access can occur in parallel. The RF system can claim access to the memory bus when it is not in use by the CPU. The RF system has enough internal buffering to handle potential delays. The memory addresses used by the RF system are virtual memory address prior to translation by the memory map logic.

6.1.2.3.3 Transmission Timing

The serialization and synchronization accelerator are both supported by a timing accelerator that provides RF transmission timing information. This accelerator is

connected to the high frequency oscillator used by the CPU and can be programmed to generate timing pulses at integer multiples of the clock period.

The transmission bit rate is programmed by specifying the clock divider corresponding to 5x the transmission rate. This allows the timing accelerator to generate both the individual bit timing pulses and pulses for 5x over sampling. The 5x over sampling is used during radio reception to detect edge transition in the data stream. This allows the transmitter and receiver to remain synchronized during communication.

6.1.2.3.4 Encryption

Many sensor applications require that data be both confidential and authentic. Our security application scenario demands that all alarm conditions are genuine. However, encryption is a costly operation on a general purpose CPU. Spec provides a hardware accelerator for encrypted communication.

When enabled, all communication over the radio link is encrypted prior to transmission and decrypted upon reception. An application must begin each communication in the clear so that receivers can lock-on to the incomming signal and select appropriate encryption settings. At that point, the encryption accelerator can be engaged to perform the data convolution.

The encryption mechanism on Spec is a LFSR based stream cipher similar to that used in Bluetooth. Prior to engaging the encryption accelerator the application must generate a unique key sequence. Both the transmitter and the receiver must share the same initialization key.

This encryption accelerator provides data security but does not provide data authenticity. It is possible for an attacker to modify transmitted messages. Data integrity must still be provided by the application.

6.1.3 Radio Back End & ADC Sub blocks

The only analog circuitry contained on the Spec design is inside the radio [58] and ADC (analog-to-digital converter)[59] sub-componnts. Both the radio and the ADC are included as pre-designed blocks. The ADC was designed to be as low power as possible. It performs an 8-bit conversion with just 27 pJ of energy. Its ultra-low power consumption is ideal for use in wireless sensor networks. It is capable of operating at over 10,000 samples per second while consuming less that a microwatt.

The transmitter was also designed with power consumption in mind. In isolation it consumes less that 1 mW while transmitting at -7 dBm. It is a frequency shift keyed (FSK) radio that is capable of transmitting at a wide range of frequency in the 900 MHz band. It can communicate to the Chipcon CC1000 radio used on the Blue node at up to 40 feet in a lab environment.

The transmitter uses a 32.768 KHz oscillator as its primary frequency reference. It only requires support form a couple of off-chip components. In addition to the 32 KHz oscillator, it requires an inductor and an antenna. Advances in CMOS technology may eventually allow the inductor to be constructed on-chip. The radio is configured through a set of registers that allow the CPU to control transmission strength, transmission frequency and frequency separation. The radio is capable of transmitting up to approximately 100 Kbps.

On the current implementation, only a transmitter is included on-chip. The Spec design can also use an off-chip 900 MHz transceiver built by RF Monolithics. This is the same radio used on the Mica node. A low-power on-chip receiver is currently under development and has been simulated to consume just 300 uA at 3V.

6.1.4 Digital frequency lock registers

The analog RF block contains the primitive functions necessary for RF transmission. However, it is not capable of self-tuning. A digital frequency control block is included for frequency tuning. The frequency control block implements a frequency lock loop between the 32.768 KHz crystal oscillator and the RF transmission frequency.

The frequency lock is performed by counting the number of oscillations of the RF transmission frequency that occur between each oscillation of the 32.768 KHz clock. The 900 MHz carrier signal is fed into a specialized block of low-power dynamic logic that divides the signal by a factor of 8. Once the signal is in the range of 100 MHz, a standard digital counter is used as a cycle counter.

The frequency lock block is configured by specifying a target count that represents the desired frequency. The block then adjusts the control lines of the register to achieve the programmed frequency. If a 902.5 MHz signal is desired, then 27542 can be fed into the frequency control register. (27542 * 32.768KHz = 902.5 MHz). This results in the ability to adjust the carrier frequency in 32 KHz steps. To increase the fidelity of the adjustment, the 32 KHz reference can be divided down by up to a factor of 32 in order to produce a minimum frequency step of 1 KHz resulting in an adjustment accuracy of 1 KHz.

The frequency control block uses a successive approximation algorithm to quickly narrow in on the desired frequency. Once locked, the control block continually tracks the desired frequency by incrementing and decrementing the radio control words as necessary. In addition to providing frequency tuning, the frequency control block also controls the frequency separation of the FSK modulation. Frequency separation can range between 10 KHz and 10 MHz. For debugging purposes the frequency control block can be bypassed to directly access the RF control lines.

The interface to the analog RF blocks consist of 16 frequency control lines, 24 power, bias current, and amplitude control lines.

6.1.5 Physical Characteristics

The Spec design was fabricated by National Semiconductor using a .25 um CMOS process. Figure 6-3 gives a general feel for the area required by the various sub-

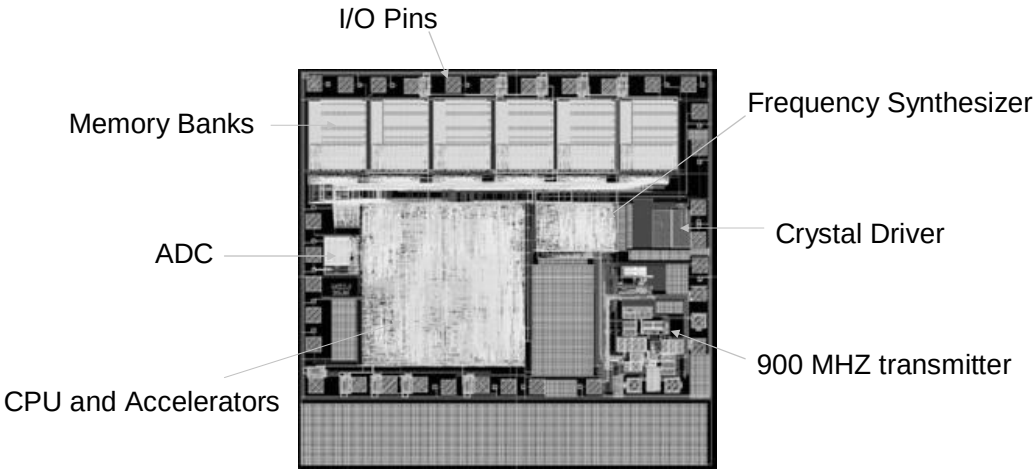

Figure 6-3: Design layout for single chip mote. Large central block contains CPU, timers and hardware accelerators. Across the top are 6 memory banks. The ADC is center left. The radio is in the bottom right corner.

components of the design. A large fraction of the area is consumed by the ram blocks – approx 20 – 30%. Additionally, the analog RF circuitry requires approximately the same area as 1 kilobyte of SRAM.

The digital logic contained in the design was synthesized from behavioral VHDL. After simulation, the high-level VHDL code was mapped into standard cells provided by National Semiconductor using Ambit Build Gates and layout was performed with Silicon Ensemble – both tools from Cadence Design Systems. In addition to simulation, the functionality of the digital core was also verified by downloading it onto a Xilinx FPGA.

The actual properties of the National Semiconductor cell libraries are confidential however, for analysis purpose we assume that the registers that we are using have a delay of approx 1 ns and consume approximately 1 pJ per transition. These approximations

Logic Core Area Breakdown

Component	(mm^2)	%
CPU Core	0.3806	47%
RF System Interface	0.1886	23%
Central Memory Controller	0.0769	10%
UART	0.0407	5%
Timer	0.0406	5%
SPI Programming	0.0283	4%
Service Module	0.0172	2%
Output Ports	0.0120	1%
ALU	0.0114	1%
ADC Interface	0.0100	1%
External Mux	0.0033	0%
RAM Access Controller	0.0030	0%
Input Port	0.0007	0%
Complete Logic Core	**0.8065**	**100%**

Figure 6-4: Area breakdown of digital logic modules included on Spec. Total area of wiring and gates using National Semiconductor standard cells in .25 um CMOS.

are well below the actual performance and therefore represent a upper bound on the energy consumption of our basic blocks. Even with these sub-optimal approximations, the central core of our device can operate in excess of 50 MHz – well beyond the target of 4Mhz.

Figure 6-4 gives a breakdown of the area used by various system blocks. This information can be used to evaluate the area/power tradeoff associated with each system. The RISC CPU core requires just .381 mm^2. The RF subsystem is approximately half that size at .188 mm^2. It is important to note that the areas reported are the total area of the logic blocks. It is not always possible to achieve logic 100% density. The .806 mm^2 of logic is placed inside the 1mm^2 block shown in Figure 6-3.

6.2 Performance

In our evaluation, we compare the Spec design against the MICA platform. To recap, the key characteristics of the Mica platform are that the CPU consumes 15 mW when active at 4 MHz and the radio consumes 21 mW while transmitting and 15 mW when receiving. The maximum transmission speed is 50 Kbps. Our goal is to focus on the architectural improvements of the Spec design over the Mica design. For a fair comparison we assume that our core consumes the same 15 mW when active as the Mica core. While in actuality we consume only approx 1.5 mW, the reduction in power not due to an architectural difference, but simply the CMOS process used. Our comparison focuses on the duty cycle of the CPU.

6.2.1 Start Symbol Detection

In analyzing the power consumption of the wireless networking protocols implemented on the MICA platform, the power consumption of start symbol detection was the single largest component of total CPU power consumption. On Mica, prior to receiving a packet, the channel is sampled at 20 Kbps in order to search for the arrival of a 10 Kbps start symbol. The CPU contains a buffer of the past 30 samples and performs a pattern matching operation on the arriving stream. Each sample requires individual processing from the CPU. Additionally, because the data portion of the packet immediately follows the start symbol, each sample must be processed in real time so that the symbol is detected before the data arrives.

This software based approach to start symbol detection takes approximately 100 cycles per sample. At 20 K samples per second, this results in 2 M cycles per second. On the Mica node this requires 7.5 mW of power (50 % of the CPU). Not only does the software start symbol detection consume a significant amount of energy but the available CPU resources, it also limits the maximum sample rate. 50 Kbps starts symbol detection would be impossible at any energy budget.

As discussed, the Spec node solves this problem by including a hardware pattern matching primitive. This primitive block is configured with a pattern up to 24 bits long and signals when it is matched. It works in conjunction with a timing accelerator that automatically sets the sampling rate. While sampling the channel, the pattern matching primitive actually samples at 5 times the bit rate. This allows it to average several samples to reduce the impact of noise.

The hardware implementation of this primitive requires .1 mm2 of chip area. A majority of that goes to the 120 register bits required to hold the 24 bits over sampled by 5x (24*5 = 120). Using the 1 pJ/transition estimate of the energy consumption of a register described above, the total power consumption of the accelerator when searching for the same 10 Kbps start symbol is just 5 uW. Despite the 5x over sampling, the power consumption is reduced by a factor of 1,500.

6.2.2 Interrupt Handling Overhead

Another frequent and expensive operation on the Mica node is the saving and restoration of registers when handling an interrupt. The real-time nature of wireless sensor networks makes handling interrupts a critical and frequent operation. The standard interrupt routine used in TinyOS requires that 14 of the 32 registers be saved prior to the execution of the interrupt handler. With a 2 cycle memory latency, this saving and restoring of the registers requires 56 cycles per operation. When sampling an audio waveform at 5K samples per second to detect an acoustic pulse, this would represent an overhead of 280,000 cycles per second. When broken down into energy, the saving and restoring of registers for interrupt handling requires 210 nJ/interrupt.

The Spec node adds specialized hardware support for accelerating the execution of interrupts. Instead of having only a single register set that must be saved for each interrupt, the Spec node introduces an additional register set and an instruction that can be used to switch between them. During operation, one register set is used for general-purpose execution and the second is used for short, non-preemptable, interrupts. Thus, when executing a non-preemptable interrupt, the system executes a single instruction to

swap register sets. In this design, the stack pointer is preserved when register sets are switched. This allows the interrupt routine to use continue to use the system stack after switching register sets.

The inclusion of this additional register set reduces the cost of interrupt overhead to just 7.5 nJ/interrupt – 2 instructions. The overall cost of saving and restoring the registers is reduced from 210 nJ to just 7.5 nJ, a factor of 28. We also considered including additional register sets for other system functions. However, the area of each additional register set – .113 mm^2 – made this prohibitively expensive. Each register set represents 30 % of the CPU core's area.

6.2.3 Program Memory Management

A key requirement for wireless sensor networks is that the nodes need to be reprogrammed in the field. As nodes are deployed in large numbers it becomes impractical to collect them for reprogramming. It is important that they be able to accept a program over the radio and then start executing it. Additionally, as the cost of the individual nodes drop it will become impractical to interface directly with each node produced. The nodes will be exclusively programmed over the radio.

With the Mica architecture, a device must be equipped with enough storage to hold 3 distinct program images. During operation, the node has a currently executing image stored in its program memory. To reprogram itself it downloads a second code image and stores it in external memory. Once complete the downloaded image is copied over the running image and booted. However, in case the new image is damaged or non-

functional, a backup image must also be stored. In total, three copies are required, one running and two in memory.

In order to reduce this overhead, the Spec node has the ability to dynamically map pages into the program and data memories of the microcontroller. This allows the device to download its new code image into ram and then remap the ram pages into the instruction memory. With this method the code images can be swapped in place. This results in a 33% reduction in memory requirements.

This is implemented by performing an address translation on all addresses leaving the data path. The top 6 bits of both the instruction and data address are passed through a translation table that maps the address to one of 16 physical page frames. The Spec node currently had 6 of these frames filled with memory. With this setup, arbitrary allocations of data and instructions memories can be achieved. Additionally the small 512 byte page size keeps fragmentation to a minimum. The translation tables can be modified by accessing special memory address locations.

6.2.4 Timing Extraction

During transmission precise synchronization between transmitter and receiver must be maintained. Slight skew in timing mechanisms may cause bits to be lost if periodic re-synchronization is not performed. While it is not strictly necessary for short transmissions, it becomes required as transmission length increases.

Currently, the Mica platform does not support long packet lengths. For it to support long transmission, the receiver would have to periodically measure edge

transitions in order to re-synchronize. With the Mica's CPU already heavily loaded during transmissions, this is a difficult operation.

In order to facilitate long transmissions as well as to improve the reliability of short transmissions, the Spec node includes a timing extractor primitive that automatically signals the center of each bit transition. The timing extractor samples the channel at 5x the bit rate and searches for perfect edge transitions. If a clean transition occurs within 1/5 of a bit time of the expected transition, it updates the expectations. A clean transition is when 4 consecutive samples of '0' are followed by four consecutive samples of '1' or vice-versa. This allows transmitter/receiver pairs to successfully communication even if their time references differ by up to 10%. Had the Mica node had this ability it could have used a low-cost 4 MHz resonator instead of the high accuracy 4 MHz crystal.

6.2.5 Encryption Support

In addition to providing primitives for low-level operations, the Spec node also includes primitives to support complex operations such as encryption. The Spec node includes an encryption accelerator that can be used to automatically encrypt and decrypt messages for transmission. The primitive includes four separate 40-bit LFSRs that are xor-ed together generate a single random sequence. This parallels the encryption methods used in the Bluetooth wireless standard. This collection of 4 LFSR offloads a majority of the overhead associated with encryption. Unlike in Bluetooth, the processing for seeding the LFSRs is done in software.

In hardware, the LFSR is simply a shift register augmented with feedback through a handful of xor gates. With the 4 LFSRs containing 160 registers, the power consumption to perform the encryption is just 160 pJ/bit. In contrast, a software implementation of the same algorithm on the MICA CPU core would require 34 (15 load/stores, 19 ALU) instructions per bit for each shift register. This would result in a total of 196 cycles per bit for all 4 LFSRs, or 735 nJ per bit – a 4500x increase. Additionally, at 196 cycles per bit, the maximum encryption/decryption speed would be just over 20 Kbps – well below the current transmission rate of 50 Kbps. The area of the encryption accelerator is just .06 mm2, roughly the same size as the memory address translator.

6.2.6 Memory I/O and serialization

While the Mica node uses clever optimizations to allow the CPU to interact with the radio in byte chucks, CPU involvement still accounts for a significant amount of overhead. Periodic per byte interactions with the radio also prevents the CPU from entering its lowest power sleep state. On Mica node, the byte level interaction combined with the overhead of handling interrupts actually sets the maximum transmission rate it is able to achieve.

The Mica node is only able to achieve a maximum bit rate 50 Kbps despite having a radio capable of 115 Kbps. Additionally, their CPU must be tasked to service the radio every 160 us when transmitting at 50 Kbps. With each byte transmitted, the controller has to execute for approximately 100 cycles. This results in the CPU consuming 375 nJ per byte.

To improve this, the Spec node incorporates a memory-mapped data serializer to reduce the energy consumption while sending and receiving data. Upon detection of start symbol or initiation of transmission, base and length registers are used to automatically read/write data directly to/from ram. After initiating the operation, the CPU doesn't need to be involved until the operation signals its completion. Instead of continually passing each byte of data to the radio every 160 us, the CPU can shutdown and remain in its lowest power state until the transmission is complete. With this primitive, the core energy required per byte transferred is reduced to the cost of a single memory operation or just 3.75 nJ/byte – and improvement of 100x.

6.2.7 Primitives not included

In addition to the primitives that we included, we also evaluated several others that we choose not to include. In general, the energy savings did not outweigh the loss of flexibility and complexity that they would introduce. The first of these primitives is a forward error correction primitive.

The Mica node uses SEC/DED error correction. Each data byte that is transmitted is supplemented by a set of parity bits that can be used to detect up to two errors and fix up to one error. It has been shown that even this primitive form of error correct can significantly improve communication performance [10]. However, the benefit of error correction is dependent on application specific requirements. If channel quality is sufficiently high, simple error detection and retransmission may result in lower energy expenditure.

Spec Hardware Accelerators

Acceleration Type	Performance without accelerator	Performance with accelerator	Improvement Factor
Register Windows	210 nJ/Interrupt	7.5 nJ/interrupt	28
Start Symbol Detection	7.5 mW	5 uW	1500
Encryption Acceleration	735 nJ/Bit	160 pJ/Bit	4594
Data Serilization	375 nJ/Byte	3.75 nJ/Byte	100
Memory Mapping	3 copies required	2 copies required	1.5

Figure 6-5: Overview of the performance improvements achieved by the Spec node's hardware accelerators.

The cost of performing error correction used by the Mica node in software is just 55 instructions per byte encoded. Their encoding process is a rate 3 code with each data byte translated into 3 data bytes. In total the encoding costs just 8.6 nJ per bit. When compared to the 735 nJ required for encryption this is quite small. Additionally it can be performed in software at a maximum rate of 218 Kbps.

CRC calculation hardware was also considered for inclusion. On the Mica node, a 16 bit CRC is attached to each 30-byte packet transmitted. The overhead of that calculation result in 29 instructions per byte or just 13.6 nJ per bit. Because it costs just a few nJ per bit, leaving it in software will not have a significant impact on system performance. Figure 6-5 gives an overview of the performance improvements achieved by the hardware accelerators that were included on the Spec node.

6.3 Cost of flexibility

Instead of providing the various building blocks described above, we could have simply provided a hardware implementation of the protocol stack implemented by the Mica node. Had we done so, the interconnections between the primitives we have listed would have been performed in hardware instead of software.

In addition to the channel encoding and CRC checking, the TinyOS messaging stack also performs address and group ID checking on each message and then parses the message header for dynamic dispatch. These operations are quite inexpensive to perform in software when compared to a hardware implementation. However, the dispatch and header parsing only requires approximately 50 instructions. This translates into just .260 nJ per bit transmitted because the cost can be amortized over an entire packet. While this could be reduced by using dedicated hardware, the flexibility gained by allowing for variable packet formats and application-specific addressing schemes is more significant than a nJ per bit.

6.4 Summary

The Spec chip demonstrates both the value and feasibility of single chip integration of a wireless sensor network platform. 1000x performance improvements were achieved on system functions that were dominating CPU energy consumption on the Mica platform. Additionally, by providing a set of communication primitives instead of a hardware protocol implementation, spec is able to support a highly flexible set of protocols. These components included support for start symbol detection, data serialization, data encryption and timing extraction. Spec also includes architectural support for the TinyOS operating system by including a paged memory system and register windows. Its architecture is focused improving performance without sacrificing flexibility.

Through integration, the Spec chip also drastically reduces device cost allowing for a wide range of application scenarios. The Mica generation of wireless sensor nodes has a manufacturing cost around $60. The spec mote requires approximately $0.25 of

silicon to be combined with a $0.16 battery, a $0.02 inductor and a $0.10 watch crystal. It proves that sub-dollar cost for a fully functional wireless sensor node is possible.

Chapter 7: Demonstration Applications and Performance

The ultimate goal of wireless sensor network research is to enable novel applications that change the way we interact with the world around us. These applications have requirements that differ drastically from those of traditional wireless scenarios. In our exploration of these new scenarios, two distinct classes of applications have emerged. The first is categorized by a low duty-cycle, low data-rate, long latency, static topology and a long expected lifetime. For these networks, lifetime is the main evaluation criterion. A second class of applications is that of highly dynamic sense-and-control networks with higher data rates, latency limits, and highly mobile nodes. Instead of passively monitoring a relatively static environment, these networks attempt to control the environment in real time. This class places strict latency and throughput requirements on the network, which are the main measures of system performance for this class of device. We have evaluated or architecture with respect to both application classes.

7.1 Environmental Data Monitoring

As presented in Chapter 2, environmental data monitoring is a key application area for wireless sensor networks. These scenarios include heating, ventilation, and air condition monitoring of large buildings or climactic monitoring of outdoor habitats. The architectural optimization opportunities come together to determine the application's capability and lifetime at a given energy budget. We use this scenario to track the

performance improvements of the four node architectures we have presented (Rene, Mica, Blue, Spec).

For illustrative purposes we consider a scenario where the network forms itself into a tree and each node samples environmental data every 4 seconds, aggregates a statistical summary over 5 minutes, and transmits the summary to its parent node. Each parent combines the readings from its children and sends a single summary to its parent. In the event of a drastic environmental change, any node can send an emergency notification to its parent at any time. This network is simultaneously acting as a data collection network and an alarm monitoring network. In alarm mode, the parent checks for emergency messages every 4 seconds. Figure 7-1 summarizes the theoretical performance of each of the platforms we presented in this thesis. Computing rates are fixed; the key performance metric is energy efficiency, which we measure as mj/day to accomplish the task. The performance improvements of the mica node over the Rene node can largely be attributed to the hardware accelerators. Overall, MICA's hardware accelerators and exploitation of application specific protocols results in a 10x increase in overall application performance over the baseline Rene node. This comparison is analyzed in more detail below.

7.2 Empirical analysis of performance improvements

To estimate the energy consumption for each of the architectures we have presented, we sum the energy usage of each of the application components: Data Transmission, Rendezvous Overhead, Alarm Checking, Data Sensing, and sleep energy. The energy cost of data transmission assumes that the worst-case node is responsible for five children.

Platform performance on Environmental Data Collection Application

Implementation	Data communication (mJ/day)	Rendevous (mJ/day)	Alarm check (mJ/day)	Sensing (mJ/day)	Total active (mJ/day)	Sleep (mJ/day)	Total (mJ/day)	Expected Lifetime on Idealized AA (years)
Rene	2,289	691.2	52,462	16.2	54,768	4,230	58,998	0.35
Mica	458	34.56	78	16.2	552	4,319	4,871	4.27
Blue	209	72.576	178	1.701	388	259	647	> 10 yr
Spec	2	0.864	6	5.832E-07	9	259	268	> 10 yr
Improvement factor Blue vs Rene	11	10	295	10	141	16	91	> 40

Figure 7-1: Theoretical node energy consumption when performing environmental data monitoring where data is collected every 4 seconds, averaged and transmitted once every 5 minutes. Each node has a maximum of 5 children nodes and one parent node. The table shows results for Rene, Mica, Blue, Spec. The ratio between Rene power consumption and blue power consumption is also shown. The alarm check cost of blue is larger than that of Mica due to the CC1000 having a longer turn-on time. The Sensing cost of blue is reduced by a low power ADC in the microcontroller.

In this application scenario, a node must receive a summary from each of the children, combine them, and transmit a new summary to its parent. The cost of each operation is dependent on the power consumption of the radio system and the bit rate that data can be communicated at. The Blue node shows an 11x improvement over the original Rene and outperforms the Mica node. This is due to an increase in transmission rate enabled by the synchronization and buffering accelerator – blue node has twice the buffering in its serialization accelerator.

As outlined in Chapter 2, because communication is infrequent and the radio consumes energy whenever it is powered on, nodes must leave their radios powered off whenever possible. In the data collection application scenario, nodes agree to communicate at a future time and turn off their radios until that time. In these protocols, synchronization precision translates into lower power consumption. If a scheduled application period of communication is only accurate to approximately 100 ms, then the receiver must wake up 100 ms early to ensure that it is not late; this could result in the

receiver waiting 200 ms before the transmission actually begins. For the short transmissions common in sensor networks, this can result in a 4x to 5x increase in cost over basic packet transmission and reception. The use of bit-level time synchronization on the mica and blue nodes reduces the energy consumed while waiting for the scheduled communication by a factor of 10. You will also note that the Mica node outperforms the Blue node on this operation. This is due to the increased receive power consumption of the CC1000 radio.

In order to meet the strict alarm propagation requirements, each node must also be continually ready to forward alarm messages. This means that they must be frequently checking the RF channel for activity. The relatively high frequency of the wake-up check makes it the single largest consumer of energy in the Rene design. However, exploiting the low power wake-up functionality on Mica and Blue reduces this cost by as much as a factor of 600 on the Blue node. Once again the Mica node performs better than the Blue node due to the characteristics of the CC1000. Spec performs significantly better because of its fast radio turn-on time.

Taken together, the three optimizations (RF wake up, time synchronization, and high-speed communication) reduce energy consumption of the active part of the application by two orders of magnitude. In Figure 7-1 we also show the impact of these optimizations on the overall impact on system lifetime. In the Rene base case, the power drain in sleep state was a reasonable 10% of the overall budget, but it dominates the optimized application. When looking at the application as a whole, battery life is improved by a factor of more that 90, resulting in an expected lifetime of greater than 10

years for a pair of AA batteries. We do not predict a longer lifetime because the shelf life of an alkaline battery is less than 10 years.

It is interesting to note that the added performance improvements of the spec node do not have a significant impact on the application lifetime. For analysis purposes we have assumed that the sleep mode power consumption of the spec node is the same as the blue node. Because the sleep mode power consumption dominates this application scenario once the other factors have been addressed, the improvements in radio power consumption are insignificant. This, however, will not be the case in a node tracking scenario.

7.2.1 Hardware Trial

To verify the empirical analysis performed, we demonstrated this application scenario on the blue platform. Once again, a node collects data periodically from a local sensor, averages its readings and transmits data up once per second. The hardware trial was performed to confirm the theoretical results.

The data collection application used precise time synchronization to maintain communication schedules. The hardware trial showed that transmitter and receiver pairs are indeed able to maintain synchronization of better than 2 ms over the course of the 5 minute communication intervals. At the end of each interval, nodes exchange pair-wise time synchronization information. This information is used to maintain the <2ms synchronization indefinitely. The resulting duty cycle of active communication is just .02% or 60 ms every 5 minutes.

Additionally, when measured experimentally, the blue mote's periodic wake-up checks require 8.8 uJ per check or 190 mJ per day. The experimental sampling cost is

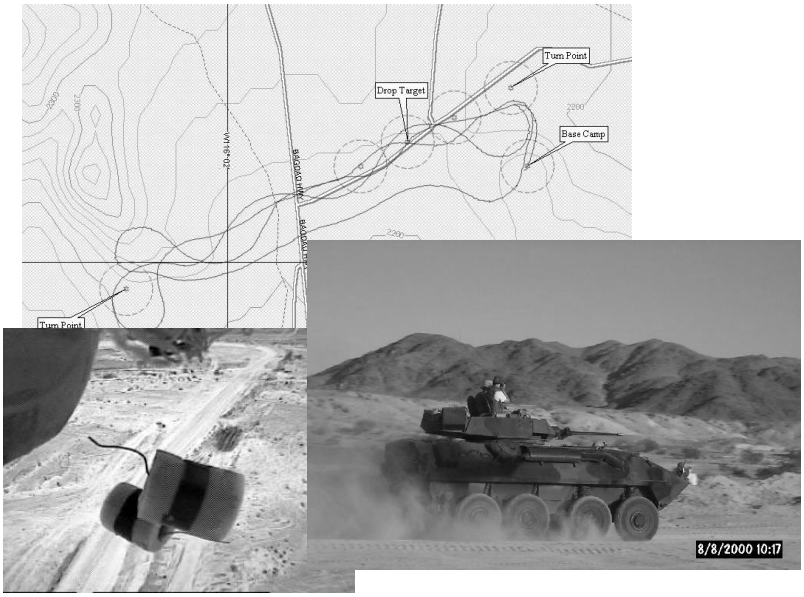

Figure 7-2: 29 palms application. Motes dropped out of an airplane self-assemble onto an ad-hoc network in order to monitor for vehicle activity at a remote desert location.

just 42 nJ per sample or .907 mJ per day. The net result of the hardware trial is that the Blue implementation performs roughly equivalent to the theoretical predictions.

7.3 29 Palms

Another application scenario that has been implemented on the Berkeley motes is that of an isolated network designed to monitor activity at a remote location. The "29 Palms" application used the Rene mote platform in conjunction with magnetometers to detect vehicle activity on a remote desert road.

7.3.1 Application Description

As part of an experiment with the US marines, the motes were deployed to detect vehicle activity at an isolated intersection in the desert near Palm Springs, California. A collection of nodes were dropped in a line along the side of a road. They were dropped

from a small UAV that was flying autonomously based on a GPS flight plan. The plane was pre-configured with a multi-point flight plan which included a low-altitude "bombing run". The motes were released from approximately 100 feet along a track parallel to the road.

Once deployed, the nodes configured themselves into a multi-hop network and synchronized internal clocks so discrete sensor readings could be correlated across multiple nodes. As vehicles passed the network, individual nodes used magnetometers to detect deviations in the magnetic field caused by metal contained in the vehicles. Each node determined the closest point of approach for the vehicle and then assigned a time stamp for the vehicle event. Events from each sensor were then communicated to neighboring nodes. Once a node collected 5 readings, it performed regression analysis to determine the velocity and direction of the vehicle.

The high-level vehicle track information was then logged in persistent storage. Over time, each node collected a set of vehicle track logs in its memory. These logs could then be downloaded by a querying node. In the case of the demonstration application, the vehicle tracks were downloaded back to the UAV that originally released the nodes. After deployment, the UAV departed the area and returned later to retrieve vehicle detection events. Upon request, each node uploaded its data to a node contained in the plane. The plane then returned to base so that the data could be downloaded to a PC for display. The plane itself acted as a physical data shuttle.

7.3.2 Key sub components/application blocks

The 29 Palms application was divided into several distinct sub-pieces: sensor calibration, data analysis, time synchronization, data communication, data regression, position estimation and data logging.

7.3.2.1 Sensor Calibration

Upon deployment, each node must re-calibrate its highly sensitive magnetometer. Any change in node orientation would cause a relative change in the earth's magnetic field. This change must be calibrated out of a sensor once the node has reached its final resting point. This was performed by having a control loop that observed the readings coming from the magnetometer and detecting when the plane flight ended and the node was stationary on the ground.

Once the node determined that it reached its final resting point, the node would begin to tune its physical hardware to compensate out the earth's magnetic field. Essentially they attempted to remove the DC component coming from the magnetic sensor. This was done through the use of digitally controlled potentiometers [60].

7.3.2.2 Data Analysis

Once calibrated, each node continually analyzed the magnetometer sensor readings in an attempt to detect a vehicle. The analog sensor was sampled at 32 Hz and then passed through a pair of IIR filters. The filters were carefully constructed to perform a band-pass operation on the incoming data stream. Each node searched for a transient signal occurring at .2-2 Hz. When a signal arrived in this frequency band, the node assigned a time stamp to the peak of the arriving signal. Figure 7-3 provides a

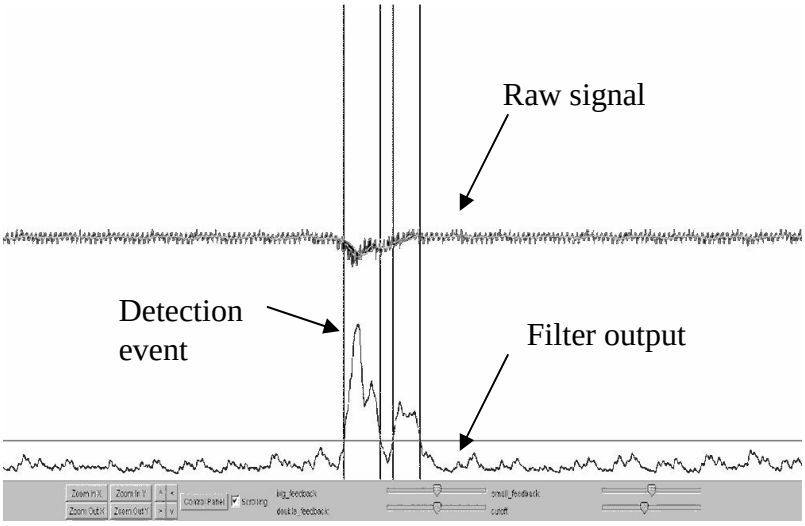

Figure 7-3: Data filtering performed locally on modes to amplify and extract vehicle signal. Time stamp assigned to the highest peak of the red activity line.

graph of the incoming signal from a passing car. The red line represents the filtered signal.

7.3.2.3 Time Synchronization

In order to calculate vehicle velocity, each node must record when a vehicle passes with respect to a global time base. This allows events from multiple nodes to be compared to each other. For this application we targeted time synchronization accuracy of +/- 30 ms. This level of synchronization is sufficient to successfully determine the closest point of approach of a vehicle traveling 50 mph to approx 2 feet.

Each node maintained synchronization using a clock advancing algorithm. Each node would periodically broadcast out its current clock reading. Upon reception of such a broadcast, the node would compare the reading to its local clock and set its clock value to the larger of the two readings. This allows the entire network to automatically lock to the node with the fastest running clock.

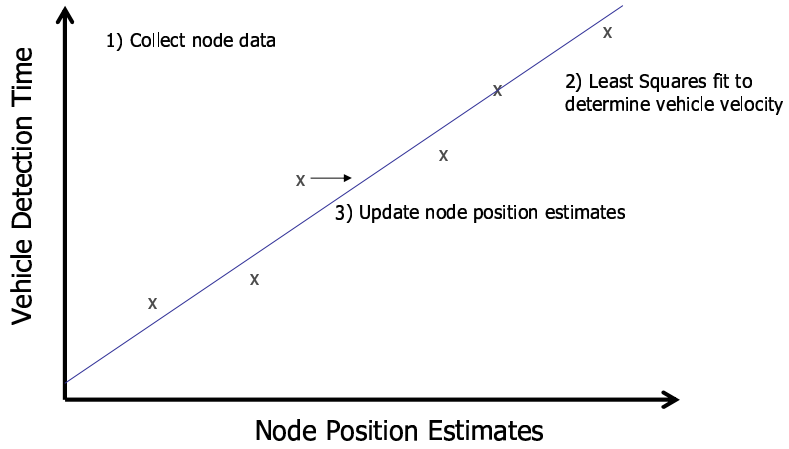

Figure 7-4: Position estimates of each of the sensing node are updated in response to the result of the linear regression that estimates the passing vehicle's velocity.

7.3.2.4 Data regression and position estimation

Nodes used linear regression to determine the vehicle velocity form a collection of readings. Once a node had collected at least 4 vehicle detection events, it would wait for one second – to collect any remaining readings – and then perform a linear regression. The result of the linear regression was an estimation of the vehicle velocity and an estimate of the time it entered the network.

Any error between the estimated track and the actual sensor readings was attributed to the inaccuracy of the sensor readings and to the inaccuracy of the position estimation of each node. The result of the regression was then fed back to help refine the position estimation for each node.

After the initial deployment, each node only had a rough estimate of its location. The nodes were released approximately 10 feet apart. However, after falling for 100 feet and bouncing off the ground, the position of each node will be slightly skewed. Over time, each vehicle track is used to update the position of the sensor nodes through

feedback. The network first calculates the vehicle speed and then asks itself, “If the vehicle was actually going 15 mph, where are the sensors?”

7.3.3 Importance of in-network processing

The 29 palms application scenario demonstrates the importance of in network processing. Each individual sensor node processes over 400KB of data per hour. Collecting the data from hundreds of sensor nodes at a central analysis point would quickly overwhelm the communication capabilities of the network. Additionally, the power consumption would be unacceptable.

By including the data processing at the individual sensor, the communication requirements are drastically reduced. Additionally, the power consumption of the network is tied to the number of vehicles that are detected, not to the amount of time that has elapsed. This is ideal for a network that needs to remain in place for several months in order to confirm that there is no activity in the area. When there are vehicles to detect, the power consumption increases locally to share the vehicle events with neighbors and then declines once the vehicles have passed. Only important readings consume the valuable communication time and energy. Central data analysis requires that all data be communicated regardless of the information value.

7.4 Z-Car Tracking

A more controlled follow-on application to the 29 Palms vehicle tracking demo has been done called Z-car tracking. In this applications, a regular grid of sensor nodes form a mesh of sensing that can automatically direct “high-level” resources towards interesting events.

Figure 7-5: Picture of z-car tracking application deployment. Remote control car tracked as it passed through the field of sensors.

7.4.1 Application Description

The Z-car tracking application has a network of nodes designed to track a vehicle once it enters the sensing grid. Unlike the 29 Palms application, the vehicle detection events are immediately communicated to a high-level resource for prompt action. The small-scale demo involves tracking a remote-controlled car with a video camera through a network of 25 nodes. The vehicle detection algorithm used on each node is the same as that of the 29-Palms tracking demo.

When the vehicle is detected inside the network, local nodes exchange sensor readings with their neighbors and synthesize a single report message that contains an estimated position for the vehicle. This message is then handed to a multi-hop communication mechanism that delivers the report to a camera located at the end of the network. For demonstration purposes, the network is only 20 x 20 feet and the vehicle

being tracked is a small remote controlled toy. However, this demo could be easily scaled to detect larger vehicles by using larger grid spacing.

The multi-hop data communication is based on geographic routing where the packet is addressed to nodes that are physically closer to the data's destination. The data eventually arrives and a special mote that is in control of the video camera. The readings are then used to calculate the most likely position of the vehicle and the video camera is pointed accordingly.

7.4.1.1 Bi-modal operation

The Z-car tracking application has a Bi-modal activity profile. During a real deployment, days or weeks may pass before any activity is detected. During this time, the nodes must remain in a dormant, low-power state. However, they cannot be completely turned off. At any time, they could either detect a vehicle or be called upon to be router node in the multi-hop network.

When activity is required, the network snaps into a high-duty cycle state. Messages are transmitted from the sensing cluster to the camera once per second. All nodes along the routing path from the vehicle to the camera must also remain active because they may be called upon at any time to take over as a primary routing node.

The Z-car application demonstrates the ability of TinyOS and the Mote architecture to flexibility meet a wide array of protocol requirements.

7.4.1.2 Leader Election

While tracking a vehicle, sensor readings from several nodes are synthesized into a single report packet in order to conserve network bandwidth. A vehicle may trigger several sensor nodes as it passes through a network. The information coming from

nodes that are further from the vehicle is less valuable that the data coming from sensor nodes that are closer to the vehicle.

In order to conserve network bandwidth, a local leader is elected to review the data coming from several sensors and to select the most significant data. The filtered data is then handed to the multi-hop layer for the expensive trip across the network. In actuality there are two levels of filtering being performed simultaneously. Each node is continually filtering the raw sensor readings and transmitting its neighbors only what may be significant. Then, the leader then takes the data and determines which of data readings are the most significant.

The leader is elected by being the node with the strongest sensor reading in a given time epoch. Each node collects the reports coming from all of its surrounding neighbors. It then compares the reports to its own local report. The node that has the strongest sensor reading is the leader and selects data from the next three best nodes. This election mechanism was selected because of its low overhead and its tolerance to lost messages.

In the event that a message is lost, our leader election algorithm guarantees that the best data will still be reported. Regardless of what packets are received by the true leader, by definition he will still have the highest quality data. If the leader's data is not properly received by other nodes, then a second leader will be accidentally selected and two packets will be transmitted to the base. In either case the most valuable data is always transmitted.

7.4.1.3 Reliable Message Delivery

Once the data from several sensor nodes has been aggregated together, it becomes essential that the data is delivered reliably across the network. In many sensor applications, lost data is acceptable because the data transmitted is highly redundant. However, we are intentionally transmitting only highly valuable data.

To implement a reliable delivery mechanism, we augmented our communication primitives to include link-level acknowledgements. Each time the application initiates a message transmission, it automatically waits to receive an acknowledgement. When the TinyOS system returns the SendDone event to the application, it includes information about whether an acknowledgement was received or not. A positive acknowledgement signifies that the messages was received by the node that it was addressed to and that the receiving node verified the integrity of the packet using a CRC mechanism.

In the event that an acknowledgement was not received, our multi-hop routing layer automatically initiates a retransmission. This is repeated up to three times before it is assumed that the communication link has been broken. In contrast, the initial sensor report messages are not retransmitted at all. This is because they are simply broadcast messages to surrounding neighbors. There is not a single intended recipient.

7.4.1.4 Geographic Based Routing

The multi-hop routing mechanism used was based on geographic information. The physical location of each node was used to determine a routing structure. Each node was told its location on in the <X, Y> plane. A node would route data through any node that is physically closer to the destination.

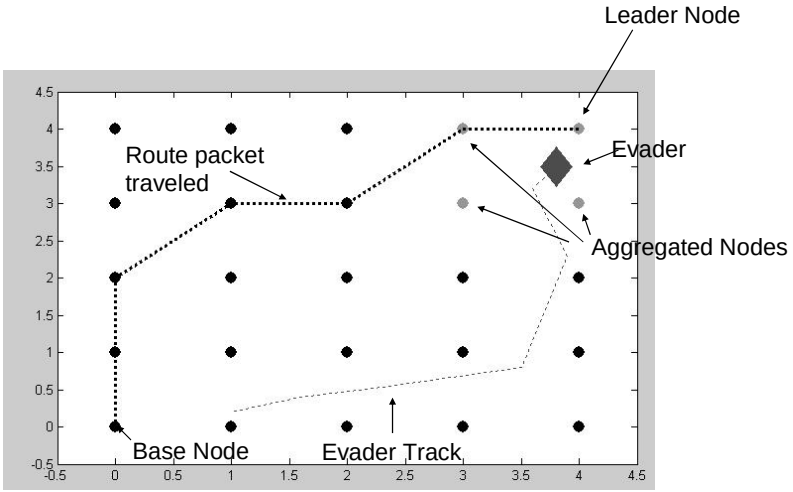

Figure 7-6: Multi-hip network routing used in z-car tracking application. Nodes detecting the event (in green) transmit data to a single leader that transits a single report packet. The report packet travels across the network (dotted line) until it reaches the base node. The packet contains the estimate of the evader's (red) position.

With a grid based network layout, it is easy for each node to calculate the location of nodes closer to the base station. Each node attempts to transmit to any node that is up to 2 grid units away from itself. Greedily, each node first tries to contact the nodes that are both one step in the X direction and one step in they Y direction towards the base station. If a transmission to that node is not successful after three attempts, another node is selected and the process is repeated. Figure 7-6 displays the network behavior while tracking a node.

Once a successful path has been established, the nodes continue to use the same routing path. Only after a local retransmission has failed three times does the network select a new path.

One of the problems faced by geographic routing is that local minima can be formed from which data cannot escape. Sometimes you have to send data away from the destination in order to avoid an obstacle. Unfortunately, we did not address this issue.

Our grid based deployment minimizes the presence of local minima. Multiple node failures are required to create such an obstacle in our grid. Future work would be to include algorithms to avoid such phenomenon.

7.5 Conclusion

Hundreds of applications have been constructed that demonstrate the efficiency and capability of the Mica, Blue and Spec platforms. We have presented three key applications that demonstrate several of the capabilities of the overall system. Low-duty cycle networks built with the Blue node have demonstrated the ability for this architecture to support precise synchronization which leads to very low network duty cycle and multi-year lifetime for the environmental data monitoring scenario. Additionally, complex vehicle tracking deployments have shown TinyOS's ability to meet application specific needs for data communication, data aggregation, sensor analysis and multi-hop routing.

Chapter 8: Related Work

In looking at related work for wireless sensor networks, there are three key areas of active research. The success and widespread adoption of TinyOS has lead to a body of work that is directly built on-top of the TinyOS/Mica platform. Secondly, there is a body of work related to other wireless platforms that are applicable to wireless sensor networks, and finally there is a body of work related to embedded operating systems that may be used in wireless sensor networks.

8.1 TinyOS Supported Work

In this section, we will quickly provide an overview of work that has been done on top of the TinyOS platform. The Mica node has been distributed to over 250 research groups around the world as a platform to support wireless sensor network research. Additionally, it has been the primary research platform for the NEST research program [61]. The Network Embedded Software Technology (NEST) research program is a DARPA funded program with dozens of researchers at more that ten locations around the nation that are investigating the software technologies that must be developed to support distributed, embedded systems including those of sensor networks.

8.1.1 Wireless Robots

One of the research areas inside of the NEST program is that of wirelessly controlled robots. Traditionally research into autonomous robotics has been performed on robotics platforms that cost tens of thousands of dollars. These devices include and embedded PCs running windows or Linux and communicate with 802.11. As an

alternative, several research groups have developed low-cost robots that are controlled by a Mica node running TinyOS.

The COTS bots project at UC Berkeley has built their robots out of low-cost remote control cars [62]. By replacing the standard control system with a wireless sensor/control node, a child's toy is transformed into a research platform. The speed, direction and turning radius are controlled by the wireless sensor node. The COTS bots project created a robotic layer of abstraction out of TinyOS components that refines the basic motor control primitives into TinyOS commands and events

The COTS-bots project is also investigating the integration of sensors to help with navigation. Obvious sensors to include are acceleration sensors, rate gyros, and tilt sensors. Early results from analysis of acceleration data from an ADXL202 suggests that acceleration data alone is not sufficient to perform inertial navigation even when the system is constrained to a single dimension [63]. Low frequency drift in sensor readings leads to unbounded error. However, the accelerometer has been successfully used to detect impacts with obstacles. Additionally, it looks promising to use multiple sensors together to perform inertial navigation.

One of the goals of the COTS-bots project is to develop algorithms for coordination between multiple robots. Instead of deploying a single high-cost robot, they want to deploy a fleet of interconnected robots. This is enabled by using the multi-hop communication algorithms and in-network processing being developed in wireless sensor networks.

8.1.2 Media Access Control and Routing

Media Access Control and transmission scheduling is a key area of wireless sensor network research. Media Access Control, the mechanism for determining who can transmit and when, must be used to optimize for device power consumption. Researchers at UCLA have demonstrated the benefit of exploiting application specific protocols by creating customized MAC layers adapted to sensor networks. Their customized sensor network communication protocols attempt to reduce energy consumption and have shown 2-6x energy improvement when compared to standard 801.11-like wireless networking protocols [51]. Their work demonstrates the advantage of using a flexible, modular system that facilitates customization.

Researchers at UCLA have also generated optimized routing protocols designed to conserve energy in deployed networks[2, 11]. In data collection networks, schemes must be used to evenly distribute load across potential routing nodes. As described in Chapter 2, nodes that are forwarding data for other nodes will consume more energy than other nodes. Over time, their batteries will be depleted and they will be unable to continue forwarding data. This may result in degraded network performance. Evenly dividing the load across all possible routing nodes ensures the maximum possible network lifetime.

8.1.3 Time Synchronization

Time synchronization is a classic distributed systems problem that has been re-addressed in the context of wireless sensor networks. Researchers at UCLA have pushed the limits of distributed time synchronization over wireless sensor networks. The driving

application behind their time synchronization research is to achieve precise localization through acoustic ranging.

The UCLA approach is to use the broadcast nature of wireless communication to send timing signals to multiple nodes [50]. Any node in the network can send out a timing beacon. Any node that hears the beacon records a precise, local time stamp for the signal. It then exchanges its time stamp with all nodes that have also heard the same timing beacon. This mechanism allows each node to determine the relative difference between its local clock and a neighbor clock that also heard the beacon. Each node stores the relative differences between its clock and all neighbors. Additionally it stores an estimate rate of the relative drift. UCLA has been able to demonstrate microsecond accurate time synchronization using this mechanism on the Mica platform.

When a sensing event is detected inside the network, all nodes apply a local time stamp to the event. The nodes then transmit the time stamped detection event through the wireless network. Each time the data is transmitted the receiving node updated the time stamp to account for the difference between its local time base and the time base of the transmitter. The time reading is dynamically updated to be relative to the local clock as it is transmitted across the network. As the data propagates through the network, the time stamp will be updated multiple times. All data that reaches a central collection point, will have time stamps that are relative to the same local time base.

8.1.4 Multi-Hop Routing optimization

One of the most important areas of research being supported by the TinyOS platform is that of multi-hop routing optimizations. The wide range of application scenarios being considered for wireless sensor networks leads to a wide range of potential

multi-hop routing protocols. Tradeoffs in latency, bandwidth and power can be made in each application scenario.

Researchers at UC Berkeley have implemented several generations of multi-hop data collection algorithms. Reference [64] presents a data collection routing protocol designed to construct an optimum spanning tree topology. Instead of searching for a minimum depth spanning tree, reference [10]searches for a tree that maximizes the probability that each node's data will reach the base station on the first try. Each node is assigned a success probability metric that starts from the base station and propagates out. The base station has 100% probability of success and each of its neighbors has a success probability that is proportional to the link probability to the base station. Nodes that can barely communicate with the base station will choose to send data over an extra hop in order to increase the success probability.

8.1.5 TinyDB

One of the most exciting systems to be constructed on top of the TinyOS platform is one call TinyDB [65]. TinyDB is designed to transform a wireless sensor network into streaming database. SQL queries are entered into a user interface and propagated onto the sensor network. The network executes the queries over incoming sensor readings and returns the results. Just like in standard SQL, these queries can include aggregation operations to refine the data into averages, max, or min values [1]. The TinyDB query optimizer automatically distributes the aggregation operations into the network.

TinyDB provides a mechanism for reasoning about how to access data coming from wireless sensor networks. It provides functionality beyond that of standard SQL in

that it supports both long-running queries and event triggers. Unlike in traditional data bases, the data set being queried in wireless sensor networks is an infinite stream of data extending into the future. Every second or perhaps millisecond, each sensor has the potential to generate a new sensor reading. Long-running queries can be issued to the network requesting data to be reported if it exceeds a threshold or fits a special criteria. In TinyDB the parameters of the query will be propagated to each individual node allowing for local data filtering. In addition to automatically transmitting data that meets special criteria, TinyDB also allows for triggers to be included that cause other local actions to occur when a specific sensing condition occurs.

8.2 Wireless Platforms

In addition to the collection of TinyOS related platforms discussed earlier, there are a collection of other wireless networking platforms that are relevant.

8.2.1 Smart Dust

The Smart Dust[66, 67] project at UC Berkeley represents the inspiration for much of the work contained in this thesis. The project has continued to develop miniature scale hardware structures that are advancing the state-of-the-art in wireless sensor network technology. In targeting extreme miniaturization and low-power consumption they are developing many ultra-low power primitives including the radio and ADC components contained on the Spec node.

Additionally, the Smart dust project has pioneered new optical communication technologies through their use of MEMS mirror-based optical communication. By modulating reflected light, their corner cube reflector communication structure is able to

transmit data across mile long links while consuming microwatts [68]. In addition to supporting the development of Spec, they have developed showcase devices that include sensing, computation, solar power sources, and optical communication. They have demonstrated autonomous sensor node that are less that 10 mm^3.

8.2.2 Bluetooth

Bluetooth is a wireless system designed to be the industry standard for low-power, mobile devices [69]. Highly integrated chipsets are being developed that provide RF circuitry and protocol processing on a single chip. While these designs will provide highly efficient implementations of the Bluetooth protocol, they will not provide the flexibility demanded by wireless sensor networks.

Bluetooth has been designed as a wire replacement and includes support for very low-latency communication. To support low-latency and high-throughput operation, Bluetooth uses a channel hopping period of just 600 us. This requires all devices to remain synchronized to within a few microseconds. In wireless sensor networks devices may only send data once per hour. It would be highly inefficient to force these nodes to remain synchronized for the entire hour. However, in Bluetooth it is expensive to enter and leave a network, it would be impractical for low duty cycle nodes to continually enter and leave a Bluetooth piconet. In a standard configuration, it takes over 2.4 seconds for connection establishment. During this time, the master node must be in a high-power scanning mode – typical Bluetooth radios consume hundreds of milliwatts wile monitoring the channel.

Another incompatibility between wireless sensor networks and Bluetooth is that the master-slave topology of Bluetooth requires each slave to report to the master every few seconds. Again, this is not efficient for wireless sensor networks. While the Bluetooth chipsets are highly efficient and well integrated, their inability to provide low-power operation modes makes it inefficient for wireless sensor networks.

8.2.3 Zigbee (802.15.4)

Zigbee is an industrial consortium designed to build a standard data link communication layer for used in ultra-low power wireless application [70]. The Zigbee alliance was formed because its members felt that existing standard technologies were not applicable to ultra-low power application scenarios.

The Zigbee data link layer is designed to operate on top of the IEEE 802.15.4 physical layer [71]. IEEE 802.15.4 is a direct sequence spread spectrum physical layer including transmission bands at 868 MHz, 902-928 MHz and 2.4 GHz. First generation chipsets will be available in Q4 of 2003 [72]. Direct Sequence Spread Spectrum has a distinct advantage over channel hopping mechanisms because hop-sequence synchronization does not have to occur prior to initiating communication. This provides a large power advantage for low-duty cycle devices and addresses a major shortcoming of Bluetooth.

Once completed, the Zigbee standard will specify a communication mechanism that is comparable to the AM communication layer provided in TinyOS. Primitive data framing mechanisms, error detection, and addressing will be standardized to allow compatibility. Zigbee is primarily focused on star topology networks where low-wireless devices communicated to powered collection point. A canonical Zigbee application is a

wireless light switch. This would involve a low-cost battery operated switch communicating to a powered light fixture.

8.2.4 Pico Radio

There are several other groups looking into the architecture of wireless embedded devices. The Berkeley Wireless Research Center's PicoRadio project has identified the importance of application specific protocols, and have built a flexible platform by exploiting reconfigurable hardware [73]. The design incorporates reconfigurable building blocks that connect to their PicoRadio protocol processor, to give it the flexibility to implement a large number of underlying protocols. However, their overall node architecture does include a dedicated protocol processor and maintains a partition between protocol and application processing. The flexibility in protocol structure will still be limited by the protocol processors external interface.

8.2.5 Chipcon CC1010

Several radio manufactures have identified the benefits of integrating general purpose microcontroller onto the same CMOS die with the radio. One commercial part that has just become available is the Chipcon CC1010. It is the integration of a CC1000 transceiver and an Intel 8051 microprocessor. While they were able to bring the configuration registers of the radio onto the I/O bus of the CPU, they did not take the step of providing any additional integration to aid in the communication process. The data interface to the radio remains largely unchanged from what was possible when the microcontroller and radio are two separate chips.

We have demonstrated how the integration of CPU and radio on to a single chip creates new opportunities for interfaces between the radio and the CPU. However the simple form integration done by Chipcon in their CC1010 has simply reduced physical size and cost. The inclusion of communication accelerators are required to allow integration to yield significant performance improvements. The CC1010 does not realize the full potential that is allowed by our generalized architecture.

8.3 Embedded Operating Systems

Traditional real time embedded operating systems include VxWorks [75], WinCE [76], PalmOS [77], and QNX [77, 78] and many others [79-81]. Figure 8-1 shows the characteristics for a handful of these systems. Many are based on microkernels that allow for capabilities to be added or removed based on system needs. They provide an execution environment that is similar to traditional desktop systems. The POSIX [82] compatible thread packages allow system programmers to reuse existing code and multiprogramming techniques.

The largest Real time operating systems provide memory protection given the appropriate hardware support. This becomes increasingly important as the size of the

Name	Preemption	Protection	ROM Size	Configurable	Targets
pOSEK	Tasks	No	2K	Static	Microcontrollers
pSOSystem	POSIX	Optional		Dynamic	PII → ARM Thumb
VxWorks	POSIX	Yes	≈ 286K	Dynamic	Pentium → Strong ARM
QNX Neutrino	POSIX	Yes	$> 100K$	Dynamic	Pentium II → NEC chips
QNX Realtime	POSIX	Yes	100K	Dynamic	Pentium II → 386's
OS-9	Process	Yes		Dynamic	Pentium → SH4
Chorus OS	POSIX	Optional	10K	Dynamic	Pentium → Strong ARM
Ariel	Tasks	No	19K	Static	SH2, ARM Thumb
CREEM	data-flow	No	560 bytes	Static	ATMEL 8051

Figure 8-1: Characteristics and requirements of commercial embedded real-time operating systems.

embedded applications grow. In addition to providing fault isolation, memory protection prevents corrupt pointers from causing seemingly unrelated errors in other parts of the program allowing for easier software development. These systems are a popular choice for PDAs, cell phones and set-top-boxes. However, they do not come close to meeting the requirements of wireless sensor networks; they are more suited to the world of embedded PCs. For example, a QNX context switch requires over 2400 cycles on a 33MHz 386EX processor. Additionally, the memory footprint of VxWorks is in the hundreds of kilobytes. Both of these statistics are more than an order of magnitude beyond the capabilities of the Rene platform.

There is also a collection of smaller real time executives including Creem [83], pOSEK [84], and Ariel [85], which are minimal operating systems designed for deeply embedded systems, such as motor controllers or microwave ovens. While providing support for preemptive tasks, they have severely constrained execution and storage models. pOSEK, for example, provides a task-based execution model that is statically configured to meet the requirements of a specific application. Generally, these systems approach meet space requirements of Rene and represent designs closest to ours. However, they tend to be control centric (controlling access to hardware resources) and do not provide the necessary levels of concurrency. Even pOSEK, which meets our memory requirements, exceeds the limitations we have on context switch times. It is unfortunate to note that while there is a large amount of information on code size of embedded OSes, there are very few hard performance numbers published.

Chapter 9: Conclusions

This thesis has presented a system architecture for wireless sensor nodes that is capable of addressing the strict requirements of wireless sensor networks. By utilizing a single shared controller that is augmented by a collection of specialized hardware accelerators, the architecture is able to support flexible, application-specific communications protocols without sacrificing efficiency. This architecture has been validated through the development of three hardware platforms and a software operating system.

We have developed the TinyOS operating system which provides the fine-grained concurrency mechanisms required to implement wireless sensor network protocols and applications. TinyOS leverages an event based execution model to efficiently share a single processor across multiple independent functional operations. Additionally, TinyOS provides a highly-efficient component model that has almost no runtime overhead yet allows application developers to partition applications into easy-to-manage modules. This allows for the component modules to be verified independently before composing them together into a complete application.

We have presented a generalized architecture that addresses key issues that arise when building a wireless sensor network device that must meet strict power consumption and size requirements. They include: flexibility, fine-grained concurrency, precise synchronization, and decoupling between RF and data path speed. We argue that these properties must be present in the system architecture in order to support wireless sensor network applications. The platform must be flexible enough to meet the wide range of

application requirements that sensor networks are addressing. We have identified core application scenarios that range from environmental data collection, to security networks, to node tracking networks. Each scenario has substantially different communication patters and protocols that must be supported by a single hardware architecture.

To validate our general architecture we first presented the Mica node. Composed from off-the- shelf components, it only approximated our general architecture. It included specialized hardware accelerators that help decouple the RF and data path speed and increase the synchronization accuracy of communication protocols. In evaluation of this platform we demonstrated how these simple accelerators result in significant performance improvements without loosing flexibility. The Mica platform has proven itself both in theory and through deployment in long-term, battery operated application scenarios.

We have made the leap beyond the capabilities of off-the-shelf hardware by fully realizing our general architecture in the form a single-chip CMOS device called Spec. Unconstrained by the capabilities of commercially available components, we have integrated a suite of hardware accelerators and a custom radio in order to realize order-of-magnitude improvements on key evaluation metrics. These include crucial metrics such as communication rates, processing overhead for start symbol detection and timing accuracy. Spec is representative of the future of wireless sensor network devises.

In addition to micro-benchmarks and theoretical analysis, we have also presented real-world application deployments. We have grounded our study by going as far as taking our nodes to the desert to track military vehicle movement during a live-fire training exercise. We have also tracked scale size vehicles in mock-up scenarios, and

deployed countless-other experimental networks. The Mica platform combined with TinyOS has been delivered to over 250 organizations to be the foundation of a nationwide effort into wireless sensor network research and development.

For several years, people have been imagining possible application scenarios for wireless sensor networks. The system architecture presented in this thesis is ready to make those visions a reality. The first commercial usage of wireless sensor networks will likely be in non-mission critical application scenarios. Systems designed to help improve efficiency or provide insight into how a business or ecosystem operates can be deployed without risk of impeding existing activities. Additionally, military applications for wireless sensor networks will become a reality in the near future. Wireless sensor networks have the ability to provide a radically new perspective on the state of the battlefield. Military commanders welcome technologies that integrate their information gathering capabilities and can incorporate the data coming from senor networks without abandoning existing information sources.

The last stage of adoption for wireless sensor networks will be when mission critical information is trusted to them. Industrial process control and security systems demand extremely high levels of reliability and predictability. These systems must be tested and validated for years before they can be trusted for deployment. Failures in an industrial process control system can lead to millions of dollars in lost productivity, injury to personnel, and damage to equipment. While the adoption rate will be slow in these areas, the technological advantages of wireless sensor networks will outweigh the risks.

While the platforms presented here are ready to meet the demands of real-world commercial applications, the technology enabling wireless sensor networks will continue to evolve. As advances in CMOS processes and RF radio technology are incorporated into next-generation wireless senor nodes, the power consumption and lifetime will continually improve. Currently, technology allows for multi-year operation off of a single pair of AA batteries. The upcoming technological advances will most likely be applied to decreasing the power consumption of the device. In turn, this will enable a reduction of physical size of the energy storage required for any given application. As for tighter levels of integration, the cost/size point represented by the Spec platform has reached the point of diminishing returns. Further reduction in the physical size of the radio, processing, and storage is no longer necessary. Only a select few applications have the need for a device that is smaller that 2.5 mm x 2.5 mm. However, all application scenarios can benefit from reduced power consumption which is translated into longer network lifetime and/or increased sample rates.

Fifteen years from now, wireless sensors will be a “behind-the-scenes” technology that has grown to impact every aspect of our lives. All factory and machine command and control systems will have switched over to relying on wireless sensing and control points. The millions of miles of cumbersome wiring you hear about in building control and automation systems will be replaced by an invisible wireless mesh. The devices themselves will become as common place as light switches and thermostats are today. They will be tiny, cheap, commodity pieces of silicon that interact with the physical world. Today we give little thought to the modern electrical grid and what our

lives would be like without it. Tomorrow we will give little thought to wireless sensor network technology and the systems that have grown to impact every aspect of our lives.

Bibliography

1. Madden, S., et al., TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. 2002: OSDI.
2. Intanagonwiwat, C., R. Govindan, and D. Estrin, Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. 2000: Mobile Computing and Networking.
3. Perkins, C., Ad-hoc on-demand distance vector routing. MILCOM, 1997.
4. Berkeley, University of California, 800 node self-organized wireless sensor network. 2001: http://today.cs.berkeley.edu/800demo/.
5. Doherty, L., Algorithms for Position and Data Recovery in Wireless Sensor Networks. UC Berkeley EECS Masters Report, 2000.
6. Mclurkin, J., Algorithms for distributed sensor networks. 1999: Masters Thesis for Electrical Engineering at the University of California, Berkeley.
7. McMahan, M.L., Evolving Cellular Handset Architectures but a Continuing, Insatiable Desire for DSP MIPS. 2000, Texas Instruments Incorporated.
8. Cerpa, A., et al., Habitat monitoring: Application driver for wireless communications technology. ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean, 2001.
9. Mainwaring, A., et al., Wireless Sensor Networks for Habitat Monitoring, in ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02). 2002.
10. Woo, A. and D. Culler, Evaluation of Efficient Link Reliability Estimators for Low-Power Wireless Networks. 2002: Technical Report, UC Berkeley.
11. Xu, Y., J. Heidemann, and D. Estrin, Geography-informed energy conservation for Ad Hoc routing. 2001, ACM Press: SIGMOBILE : ACM Special Interest Group on Mobility of Systems, Users, Data and Computing. p. 70 - 84.
12. Yarvis, M.D., et al., Real-World Experiences with an Interactive Ad Hoc Sensor Network. 2002: International Conference on Parallel Processing Workshops.
13. Whitehouse, K., The design of calamari: an ad-hoc localization system for sensor networks. 2003: Masters Report, University of California at Berkeley.
14. Kymissis, J., et al., Parasitic Power Harvesting in Shoes. ISWC, 1998.
15. Roundy, S., P. Wright, and J. Rabaey, A Study of Low Level Vibrations as a Power Source for Wireless Sensor Nodes. 2002: Computer Communications.
16. Perrig, A., et al., SPINS: Security protocols for sensor networks. Proceedings of MOBICOM, 2001, 2002.
17. Rivest, R., The RC5 Encryption Algorithm. 1994: Fast Software Encryption. p. 86-96.
18. Energizer Battery Company, Energizer AA, engineering datasheet: http://data.energizer.com/datasheets/library/primary/alkaline/energizer/consumer_oem/e91.pdf.

19. Energizer Battery Company, Energizer CR2032, engineering datasheet: http://data.energizer.com/datasheets/library/primary/lithium_coin/cr2032.pdf.
20. Power Stream, NiMH Battery Charging Basics: http://www.powerstream.com/NiMH.htm.
21. McLarnon, B., VHF/UHF/Microwave Radio Propagation: A Primer for Digital Experimenters: http://www.tapr.org/tapr/html/ve3jf.dcc97/ve3jf.dcc97.html.
22. RF Monolithics Inc, TR1000 Data Sheet. 1999: http://www.rfm.com/products/data/tr1000.pdf.
23. Chipcon, CC1000 Single Chip Very Low Power RF Transceiver. 2002: http://www.chipcon.com/files/CC1000_Data_Sheet_2_1.pdf.
24. Texas Instruments, MSP430X13x, MSP430x14x Mixed Signal Microcontroller User Guide. 2003: http://www-s.ti.com/sc/ds/msp430f149.pdf.
25. AMD, AM49DL640BG Stacked Multi-Chip Package (MCP) Flash Memory and SRAM. 2003: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/26090a.pdf.
26. Intel Corp, Intel Press Release: Intel Builds World's First One Square Micron SRAM Cell. 2002: http://www.intel.com/pressroom/archive/releases/20020312tech.htm.
27. Atmel Corporation, AT90S/LS8535 Datasheet. 2001: http://www.atmel.com/atmel/acrobat/doc1041.pdf.
28. Philips Semiconductor, The I2C-Bus Specification. 2000: http://www.semiconductors.philips.com/acrobat/various/I2C_BUS_SPECIFICATION_3.pdf.
29. Atmel Corporation, AT90S2323/LS2323/S2343/LS2343 Datasheet. 2001: http://www.atmel.com/atmel/acrobat/doc1004.pdf.
30. Culler, D.E., J. Singh, and A. Gupta, Parallel Computer architecture a hardware/software approach. 1999.
31. Esser, R. and R. Knecht, Intel Paragon XP/S - architecture and software environment. 1993: Technical Report KFA-ZAM-IB-9305.
32. Culler, D.E., et al. Fine-grain parallelism with minimal hardware support: a compiler-controlled threaded abstract machine. 1991.
33. Blumofe, R., et al., Cilk: An Efficient Multithreaded Runtime System. Proceedings of the 5th Symposium on Principles and Practice of Parallel Programming, 1995.
34. Hu, J., I. Pyarali, and D. Schmidt, Measuring the Impact of Event Dispatching and Concurrency Models on Web Server Performance Over High-speed Networks. Proceedings of the 2 nd Global Internet Conference, IEEE, 1997.
35. von Eicken, T., et al. Active messages: a mechanism for integrated communication and computation. in 19th Annual International Symposium on Computer Architecture. 1992.
36. Gay, D., et al., The nesC Language: A Holistic Approach to Networked Embedded Systems. 2003: Programming Language Design and Implementation (PLDI).
37. Renesse, R.V., et al., A framework for protocol composition in horus. 1995: Proceedings of the ACM Symposium on Principles of Distributed Computing.

38. Agarwal, A., et al., The MIT alewife machine: A large-scale distributed-memory multi-processor. 1991: Proceedings of Workshop on Scalable Shared Memory Multiprocessors.
39. Montz, A.B., et al., Scout: A communications-oriented operating system. 1995: HOT OS.
40. Hutchinson, N.C. and L.L. Peterson, The x-kernel: An architecture for implementing network protocols. 1991: IEEE Transactions on Software Engineering. p. 17(1):64-76.
41. Want, R., et al., The Active Badge Location System. ORL, 24a Trumpington Street, Cambridge CB2 1QA, 1992.
42. Bahl, P. and V. Padmanabhan, RADAR: An in-building RF-based user location and tracking system. IEEE Infocom, 2000. 2: p. 775-784.
43. Chandrakasan, A.P., S. Sheng, and R.W. Brodersen, Low Power CMOS Digital Design. 1992, University of California Berkeley.
44. Pering, T., T. Burd, and R. Brodersen, The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms. 1998.
45. Hill, J., et al. System architecture directions for networked sensors. in ASPLOS 2000. 2000.
46. Atmel Corporation, Atmega103(L) Datasheet. 2001, Atmel Corporation: http://www.atmel.com/atmel/acrobat/doc0945.pdf.
47. Dallas Semiconductor, DS2401 Silicon Serial Number: http://pdfserv.maxim-ic.com/arpdf/DS2401.pdf.
48. Atmel Corporation, AT45DB041B Datasheet. 2001: http://www.atmel.com/atmel/acrobat/doc1938.pdf.
49. Maxim, Maxim 1-Cell to 2-Cell, Low-Noise, High-Efficiency, Step-Up DC-DC Converter, MAXIM1678. 1998: http://pdfserv.maxim-ic.com/arpdf/MAX1678.pdf.
50. Elson, J. and D. Estrin, Time Synchronization for Wireless Sensor Networks: http://www.circlemud.org/~jelson/writings/timesync/timesync.html.
51. Ye, W., J. Heidemenn, and D. Estrin, An Energy-Efficient MAC Protocol for Wireless Sensor Networks. 2001: Submitted for review, July 2001.
52. Stemm, M., et al. Reducing power consumption of network interfaces in hand-held devices. in International Workshop on Mobile Multimedia Communications (MoMuc-3). 1996. Princeton, NJ.
53. Lamport, L., Time, clocks, and the ordering of events in a distributed system. Comm., 1978. ACM 21(7): p. 558-565.
54. Mills, D.L., Internet time synchronization: the Network Time Protocol. IEEE Trans. Communications, 1991. COM-39(10): p. 1482-1493.
55. Doherty, L., K.S.J. Pister, and L.E. Ghaoui. Convex position estimation in wireless sensor networks. in IEEE Infocom. 2001: IEEE Computer Society Press.
56. Borriello, J.H.a.G., Location Systems of Ubiuitous Computing. Computer, 2001. 34(8): p. 57-66.
57. Priyantha, N.B., A. Chakraborty, and H. Balakrishnan. The Circket Location-Support System. in MobiCom 2000. 2000. Boston, Massachusetts.
58. Molnar, A., Personal Communication. 2004: To be submitted to IEEE International Solid-State Circuits Conference 2004.

59. Scott, M.D., An Ultra-Low Power ADC for Distributed Sensor Networks. 2002: Master of Science Theses, UC Berkeley.
60. Dallas Semiconductor, DS1804 NV Trimmer Potentiometer: http://pdfserv.maxim-ic.com/arpdf/DS1804.pdf.
61. DARPA, Network Embedded Software Technology: http://www.darpa.mil/ipto/Solicitations/CBD_02-12.html.
62. Bergbreiter, S. and K. Pister, COTS-BOTS: Specifications. 2002: http://www-bsac.eecs.berkeley.edu/~sbergbre/CotsBots/specs.pdf.
63. Analog Devices, ADXL202/ADXL210: Low Cost ±2 g/±10 g Dual Axis iMEMS® Accelerometers with Digital Output Data: http://www.analog.com/UploadedFiles/Datasheets/70885338ADXL202_10_b.pdf .
64. Hill, J., System Architecture for Wireless Sensor Networks. 2000: Masters Report, University of California Berkeley.
65. Madden, S., et al., TinyDB web page. 2002: http://www.telegraph.cs.berkeley.edu/tinydb.
66. Atwood, B., B. Waraneke, and K.S.J. Pister. Preliminary circuits for smart dust. in Southwest Symposioum on Mixed-Signal Design. 2000. San Diego, Ca.
67. Pister, K.S.J., J.M. Kahn, and B.E. Boser, Smart Dust: Wireless Networks of Millimeter-Scale Sensor Nodes. Electronics Research Laboratory Research Summary, 1999.
68. Chu, P.B., et al., Optical communication link using micro-macined corner cube reflectors. 1997: Proceeding of the SPIE vol. 3008-20.
69. The Official Bluetooth Website: http://www.bluetooth.com/.
70. ZigBee Alliance, The official website of the Zigbee Alliance: http://www.zigbee.org.
71. IEEE, IEEE 802.15 WPAN™ Task Group 4: http://ieee802.org/15/pub/TG4.html.
72. Semiconductor, A., AMI Semiconductor First to Demonstrate Working ZigBee Wireless Device: http://www.amis.com/news/030414_zigbee.cfm.
73. J.L. Da Silva Jr., J.S., M. J. Ammer, C. Guo, S. Li, R. Shah, T. Tuan, M. Sheets, J.M. Ragaey, B. Nikolic, A Sangiovanni-Vincentelli, P. Wright., Design Methodology for Pico Radio Networks. 2001, Berkeley Wireless Research Center.
74. Pottie, G. and W. Kaiser, Wireless Integrated Network Sensors (WINS): Principles and Approach. Communications of the ACM, 2000. 43.
75. Windriver Systems, VxWorks 5.4 Datasheet: http://www.windriver.com/products/html/vxwks55_ds.html.
76. Microsoft Corp., Microsoft Windows CE: http://www.microsoft.com/windowsce/embedded.
77. PalmOS Software 3.5 Overview: http://www.palm.com/devzone/docs/palmos35.html.
78. Hildebrand, D., An architectural Overview of QNX: http://www.qnx.com/literature/whitepapers/archoverview.html.
79. Windriver Systems, pSOSystem Datasheet: http://www.windriver.com/products/html/psosystem_ds.html.

80. QNX Software Systems Ltd., QNX Neutrino Realtime OS: http://www.qnx.com/products/os/neutrino.html.
81. Microware, Microware OS-9: http://www.microware.com/ProductsServices/Technologies/os-91.html.
82. IEEE Std. 1003.0-1995: IEEE Guide to the POSIX® Open System Environment. 1995, IEEE: http://standards.ieee.org/catalog/olis/arch_posix.html.
83. Kauler, B., CREEM: Concurrent Realtime Embedded Executive for Microcontrollers: http://members.dodo.net.au/~void/old/creem.htm.
84. pOSEK: A super-small scalable real-time operating system for high-volume, deeply embedded applications.: http://www.isi.com/products/posek/index.htm.
85. Microware, Microware Ariel Technical Overview: http://www.microware.com/ProductsServices/Technologies/ariel_technology_brief.html.
86. Liu, L.T. and D.E. Culler, Measurements of active messages perfromance on the CM-5. 94: Technical Report UCP//CSD-94-087, University of California at Berkeley, Department of Computer Science.
87. Druschel, P., M. Aron, and W. Zwaenepoel, A scalable and explicit event delivery mechanism of UNIX. 1999: Proceeding of the USENIX 1999 Annual Technical Conference.
88. Homewood, M. and M. McLaren. Meiko CS-2 interconnect. Elan-Elite design. 1993.
89. Muller, M., Zero-Copy TCP/IP with Gigabit Ethernet. 1999: PhD thesis, Institute for Computer Systems, ETH Zurich.
90. Cooper, G.H., Tiny TCP/IP: http://www.unusualresearch.com/tinytcp/tinytcp.htm.
91. Seiko Instruments, S-7600A Series iChip TCP/IP Protocol: http://www.seiko-usa-ecd.com/intcir/products/rtc_assp/s7600a.html.