MANTIS: System Support For *M*ultimod*A*l *Ne**T**works of *I*n-situ *S*ensors

Hector Abrach, Jim Carlson, Hui Dai, Jeff Rose,

Anmol Sheth, Brian Shucker, and Richard Han

Contact: rhan@cs.colorado.edu

Technical Report CU-CS-950-03

April 2003

University of Colorado

Department of Computer Science

MANTIS: System Support for *M*ultimod*A*l *NeT*works of *I*n-situ *S*ensors

ABSTRACT
Technology trends are extending the flexibility and convenience of today's sensor networks through the introduction of new capabilities such as multi-frequency radio transceivers, GPS, and innovative system management tools. This paper expands upon these motivations to present an integrated general-purpose MultimodAl system for NeTworks of In-situ wireless Sensors (MANTIS). The MANTIS system promotes multimodal flexibility and ease of use through its support for multimodal sensing including GPS-enabled location and time, multi-frequency communication, multitasking sensor nodes, and a new multi-platform operating system called MANTIS OS (MOS). For the novice, MANTIS provides convenient tools such as a simple cross-platform API, a remote shell for debugging and logging into MOS nodes, fine-grained dynamic reprogramming via the radio, and in-board sensor placement. For the expert, MANTIS supports true networked emulation of MOS sensor nodes as X86 processes, as well as seamless bridging between an X86-based network of MOS virtual sensors and an actual sensor network of active physical MOS nodes, called nymphs.

1 INTRODUCTION

The growing popularity of wireless sensor networks has placed increasing demands upon the infrastructure of today's general-purpose hardware/software sensor systems to support improved flexibility, ease of use, and lower cost. Standard sensor systems such as Berkeley's Mote/TinyOS architecture [Hill00] and the MetaCricket [Mart00] have been highly successful in supporting the prototyping, programming, testing, and deployment of sensor networking applications. Such systems offer an integrated combination of a general-purpose hardware platform, an open source embedded operating system, a uniform API and system management tools. These sensor systems have been adapted for deployment in applications as diverse as habitat monitoring of seabirds [Main02] and communicating musical toys [Mart00].

As users expand the list of sensor networking applications to include military, home, office, marine, and meteorological scenarios, sensor systems will be expected to provide even more flexibility and functionality than ever before, e.g. location awareness via Global Positioning System (GPS) and flexible radios with multiple frequencies and transmit powers. In addition, users of such systems will range from novices with little experience in software and/or hardware to technical experts building complex deployments and test suites. This multimodal usage model places a premium on convenience and ease of use.

In order to accommodate a broader range of applications and users, sensor networking systems can either become more specialized to fill specific niches or modes, or can become more generalized to meet the growing set of demands. The first approach is to specialize sensor systems for each application, e.g. customized biomedical sensing systems. Berkeley's tiny Smart Dust Spec is a general-purpose sensor platform and nephew to the Mote, yet is customized for miniaturization, achieving dramatically reduced size at the cost of reduced capability [Hill03]. An alternative approach is to increase the capabilities of existing general-purpose systems in both hardware and software. For example, the MICA2 follows this approach by extending the flexibility and convenience of Mote-based sensor networks through the introduction of new capabilities such as multi-frequency radio transceivers and innovative system management tools [Xbow03].

This paper presents an integrated general-purpose MultimodAl system for NeTworks of In-situ wireless Sensors (MANTIS). The MANTIS system promotes multimodal flexibility and ease of use through its support for multimodal sensing including GPS-enabled location and time, multi-frequency communication, multitasking sensor nodes, and a new multi-platform operating system called MANTIS OS (MOS). The MANTIS system leverages the core Mote hardware architecture but institutes additional multimodal enhancements that include extensions to the sensing via GPS, novel exploitation of the multi-frequency radio, and simplification of the hardware to lower costs and accommodate novice users. These collective enhancements are combined with other modifications described in Section 3, leading to a hardware

platform that we term the "nymph". To demonstrate the versatility and power of these integrated multimodal hardware enhancements, the implementation of a time synchronization application is described in Section 4. GPS-enabled sensor nodes provide a pure clock for assessing the accuracy of various time synchronization algorithms over multi-hop sensor networks linked by multi-frequency radios. The benefit of GPS sensing is that it supports additional sophisticated sensing modes such as time and location that exceed the capabilities of traditional resistive sensors such as temperature sensors and photo cells. The benefit of multi-frequency communication compared to narrowband communication is that collisions and hence variations in packet latency can be reduced, resulting in improved accuracy of time synchronization algorithms. The benefit of integrated system support for multimodal operation is that the combination of multimodal sensing and multimodal communication enables the in-situ deployment and evaluation of accurate time synchronization algorithms in real sensor networks.

While multimodal hardware support required only modest enhancements to existing designs, multimodal software support requires a broader and deeper design that encompasses support for multi-platform operation, true multitasking, and multimodal tools/UI that enable both novices and experts to conveniently utilize this general-purpose system. To support the novice user, MANTIS' design goal is to provide convenient tools such as a simple cross-platform API, a remote shell for debugging and logging into MOS nodes, fine-grained dynamic reprogramming via the radio, and in-board sensor placement. For the expert, MANTIS' design goals are to support multitasking on ATMEGA platforms, true networked emulation of sensor nodes on X86 platforms, and mixed-mode heterogeneous networking that seamlessly bridges between an X86-based network of virtual sensors and an actual sensor network of active sensor nodes. The benefit of mixed-mode networking is to enable phased deployment of algorithms and protocols such that testing begins in a virtual network followed by a seamless transition into a functioning physical network. The breadth and depth of these multimodal software design goals and features motivated the development of a new embedded MANTIS OS (MOS). These multimodal features of MOS are summarized in Figure 1, along with the multimodal nymph hardware platform.

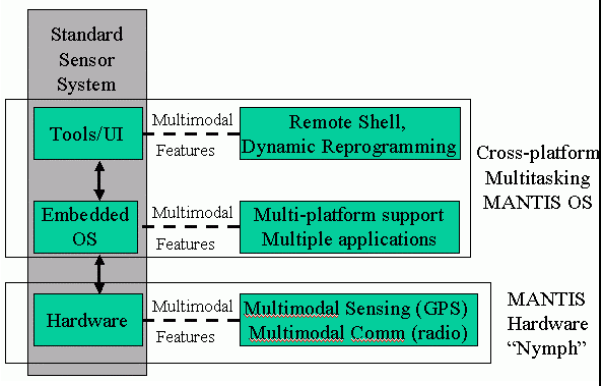

Figure 1. Multimodal features enhance the standard architecture of sensor systems, enabling multi-platform interoperability, multitasking, multimodal sensing including GPS, and multi-frequency communication.

In the remainder of the paper, Section 2 provides a more detailed overview of the MANTIS system architecture. Section 3 describes multimodal network programming via the MOS shell, dynamic reprogramming, and uniform API. Section 4 focuses on multimodal sensing via GPS and multi-frequency communication. Section 5 focuses on multimodal deployment, highlighting the cross-platform support of MANTIS.

2 MULTIMODAL SYSTEM ARCHITECTURE

The objectives of the MANTIS platform called for the design of a new multimodal hardware platform that would support the advanced features of the MANTIS operating system (MOS). Using the Berkeley Mica mote as a starting point, we have developed a new sensor node that combines new power management capabilities with a sophisticated, streamlined software development cycle, yet maintains interoperability with many sensor networks that are in place today. The MANTIS sensor node, or Nymph (a nymph is a baby Mantis), also features all the characteristics that must be expected of a wireless sensor node: small size, a good performance/power consumption ratio, wireless networking, and low cost. This section presents a high-level overview of the Nymph hardware, and discusses the implications of our design decisions.

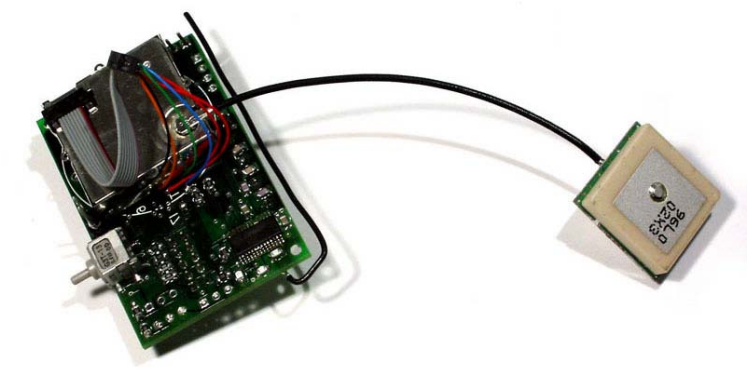

Figure 2. GPS-enabled MANTIS Nymph. GPS antenna extends to the right.

2.1 Hardware architecture

The MANTIS Nymph is a single-board design, incorporating the microcontroller, analog sensor ports, RF communication, EEPROM, and serial ports on one dual-layer 3.5 x 5.5 cm printed circuit board. The Nymph is centered around the Atmel ATmega128(L) microcontroller unit (MCU), which provides low power operation and has very good I/O support, including interfaces for two UARTs, an SPI bus, an I^2C bus, and eight analog-to-digital converter channels. In addition to the 4kB EEPROM included in the ATmega128(L), the Nymph provides an additional 64kB EEPROM external to the MCU. The unit may be powered by either batteries or an AC adapter, and a set of three on-board LEDs are included to aid in the debugging process.

The Nymph includes a built-in JTAG interface, which allows the user to easily download code to the hardware. This addition eliminates the requirement for a separate programming board, simplifying the process of reprogramming the nodes while reducing the cost of the overall system. As an added benefit, the JTAG port allows the user to single-step through code on the MCU, and also supports our remote shell, a capability that is discussed in more detail in section 3.

As mentioned above, the ATmega128(L) has 2 built in USARTs. The Nymph uses one of the USARTs to supply a serial port (RS232) for connection to a PC, while the second is used as an interface to the optional GPS unit, described below. Due to the importance of low power consumption, we chose the MAX3221 RS232 serial chip, which may be set to three different power saving modes: power-down, receive only, and shutdown. MOS is able to take advantage of these modes to wake the chip on an as-needed basis, helping to reduce the overall power consumption of the Nymph.

To handle wireless communication, the Chipcon CC1000 radio was chosen. The CC1000 supports four carrier frequency bands (315, 433, 868, and 915 MHz) and allows for frequency hopping, which is very useful for multi-channel communication. The CC1000 is also one of the lowest power commercial radios currently available, and provides an RSSI output that connects to the ATmega128(L), allowing MOS to optimize the radio to further reduce the power consumption. The radio uses a Manchester encoding scheme that allows up to 76.8 kBaud, and has a range of up to 200 meters in an obstacle free environment.

Since the power efficiency and size of the MANTIS Nymph was of greatest priority we had to choose a voltage regulator that came in a very small package and of maximum efficiency. We decided to use Texas Instruments TPS60204 100mA, 3.3V DC/DC converter due to its 90% efficiency, as well as its voltage input capabilities, allowing the connection any DC power source in the range of 1.8V to 3.6V. This converter also features a low-battery interrupt output pin, which tells the MCU when the power is low so that the Nymph can perform any backup tasks it needs to perform before mandatory system shutdown. The Nymph can be powered by either a battery or an AC adapter; it is designed to hold a 24mm diameter

lithium ion coin cell battery (CR2477), but any other battery in the range of 1.8V to 3.6V can be connected using a special battery plug.

Test Performed	Input Voltage (V)	Load Current (mA)
Just Operating System Running	3.0	15
Scheduled Single Thread While(1)	3.0	16
Scheduler with blinking LED ON	3.0	24
Scheduler with blinking LED OFF	3.0	16
Reading Sensor Data on while(1)	3.0	16
Sensing and Sending over Serial at 19.2kBaud	3.0	16
Sensing and Sending over radio, Transmitting at max power	3.0	75
Sensing and Sending over radio, Transmitting at min power	3.0	42
Sensing and Sending over radio, Receiving at max power	3.0	42
Sensing and Sending over radio, Receiving at min power	3.0	29
Absolute Max power with all LEDs on and radio transmit at max power	3.0	95
Absolute Max power with all LEDs on and radio receive at max power	3.0	62
Single LED power consuption	3.0	9
Everything in sleep mode	3.0	under 1
Ellite Nymph - Operating System and GPS	3.0	101
Ellite Nymph - Everything Running at maximum power including LEDs	3.0	174

Table 1. Power consumption of the Nymph in various modes of operation.

The attention paid to power-conserving measures in the hardware design of the Nymph allows for very low power operation, making the platform well suited for use in environments where external power is not available. Table 1 summarizes the power consumption of the Nymph under a variety of operating conditions.

2.2 Sensor connectivity

The Nymph provides three interfaces to which sensing hardware can be connected. In order to facilitate rapid prototyping in a research environment, the Nymph has solderless plug connections for both analog and digital sensors, which eliminates the need for an external sensor board for many applications. An additional advantage is lower cost - a user need only buy a 10-cent sensor in order to build a prototype, as opposed to an expensive external sensor board. Each connector provides lines for ground, power, and sensor signal, allowing basic sensors such as thermistors and photo sensors to be easily connected, as well as more complex devices such as infrared and ultrasound receivers. The I^2C bus permits additional extensions to the hardware if so desired. The Nymph also has exported serial capabilities via the second UART, allowing the connection of daughterboards that can communicate via a serial connection. Our first such addition is a GPS package that seamlessly outputs both position and time information to the sensor node, described below.

2.3 Development

In order to make programming for the MANTIS platform easy and flexible, we have developed both hardware and software support for multiple types of developer environments. In addition to the onboard JTAG interface, discussed above, we have developed a bootloader mechanism that allows developers to connect the Nymph directly to a PC via the serial port; this avoids the expense of the additional hardware that is required to support reprogramming via the JTAG port. This feature has been extended to the radio interface so that it is even possible to reprogram the system software without being in direct physical contact with a sensor node. This allows both the operating system and user applications to be completely reprogrammed while the sensor nodes are deployed.

A major concern for the design of the MANTIS platform was ease of debugging. Due to the fundamentally limited nature of the feedback mechanisms that sensor nodes have onboard, debugging sensor network applications can be a very difficult process. To address this problem, we have sought to create a set of tools that simplify debugging at every stage in the development cycle.

Foremost among these features is the ability to compile and run one or more instances of MOS on the Linux platform as standard x86 processes. The x86 version of MOS (XMOS) can communicate not only with other XMOS nodes, but it can also communicate over a serial link to MOS running on the AVR (AMOS) Nymph hardware. This system is unique in that each node is actually an instance of the operating system itself rather than a simulation. Using standard Unix socket interfaces it is also possible to communicate with other XMOS nodes over the Internet or a local TCP/IP network. Once an application has been developed and tested in the XMOS environment, it can be easily downloaded to a nymph running AMOS.

In addition to the MOS operating system, we have developed a suite of Linux tools that greatly simplify the problems of cross-platform development. A tool installation script allows the complete development to be set up by executing a single command. Next, we have developed a shell utility that communicates with programs on the nymph platform. This tool is initially used for communicating with the bootloader for downloading code from the host system, but once running, it is possible to connect to a command-server application that gives developers easy access to various operating system diagnostics. This functionality will be extended to include various network analysis and application configuration features as well. Another important capability of the MANTIS platform is that a developer can connect to the nymph and inspect its state while software is running. In difficult debugging situations, the JTAG interface can be used for line by line, in-system debugging using GDB.

2.4 Global Positioning System (GPS)

The MANTIS Nymphs are the first sensor nodes that support GPS, allowing the Nymphs to correlate sensor readings with chronological and geographic data. This feature is implemented as an external sensor board connected to the Nymph via the second UART. We currently use the Lassen SQ GPS board from Trimble, which may draw up to 30mA when running at the highest level of performance. Due to the complexity of GPS, the board may require up to two minutes to compute its initial location on power-up; however, once this initial information is computed only 30 seconds are required for further location readings. The GPS position is accurate to within 10 meters 90% of the time.

2.5 Preview of the MANTIS Operating System

The MANTIS Nymph hardware platform was designed to complement our operating system, the MANTIS OS (MOS). A central goal of MOS is to provide a programming environment that is similar to UNIX, giving developers a familiar set of tools for constructing sensor network applications. MOS is a true multi-threaded operating system that provides an abstraction layer in order to reduce the complexity of multi-threaded programming in a resource-constrained environment. A layered network stack and hardware driver system are also included, simplifying communication in an embedded platform.

3 MULTIMODAL NETWORK PROGRAMMING

The MANTIS system is intended to support users with a broad range of programming experience, from the beginner to advanced systems programmers. To this end, the MANTIS operating system presents a familiar API and programming interface, and provides a range of programming modes to accommodate many users.

The main design goals of the MOS kernel are:

- Small memory footprint
- Fast context switching

- Thread safe device drivers to protect I/O access
- Simple programming (user) interface

3.1 Features of the programming environment

The MANTIS operating system (MOS) itself is coded mostly in C, and it presents a simplified UNIX-like C programming interface. An application developer may write application code in standard ANSI C and compile it with gcc, avoiding the need to learn a specialized language or compiler. MOS implements largely UNIX-like semantics, including a subset of POSIX threads, mutual exclusion semaphores, counting semaphores, and standard POSIX thread management functions. This structure provides several advantages over existing sensor network systems:

1. Many programmers are already familiar with POSIX thread semantics, which flattens the learning curve considerably.
2. MOS can be implemented easily on top of existing operating systems, such as Linux or Mac OS X, increasing the number of devices that can participate in a MANTIS network.
3. A great deal of existing code, including a significant amount of open-source code, can be trivially ported to MOS.

Traditional micro-kernel operating systems [Rash89] are inherently inefficient due to the overhead of communication between the user space components of the OS, making the use of a more extensive but preemptible kernel preferable. This however leads to a more difficult design and implementation of the kernel as now a single address space supports concurrent actions. With multiple concurrent threads, it is possible to design more flexible sensor network applications. Despite the similarities to many flavors of UNIX, MOS is distinct in several important ways. MOS provides flexibility to the end application developer and abstracts away the resource management from the developer. For example, in a typical event-driven operating system for sensor networks tasks must be carefully designed and are required run to completion before other tasks can be scheduled. Since MOS is intended for use on sensor nodes it is optimized for efficiency (especially with respect to data memory), and thus provides a simpler set of system calls. The entire MOS kernel uses only 12 Kbytes of text memory, and approximately 500 bytes of data memory. In addition, MOS makes weaker assumptions about the capabilities of the underlying hardware: For example, it is possible to run MOS on a platform that has no memory protection hardware.

Since MOS accepts applications written in ANSI C, expert developers are given a great deal of power and flexibility. For beginning developers, a higher-level front end language that compiles down to C code is appropriate; the development of such a language specifically tailored to sensor network applications is currently being considered.

MOS also supports dynamic memory allocation for the core kernel modules, enabling runtime thread execution. Thus, multiple sensor network application threads can be loaded onto the flash initially, and commands can be sent at runtime to start up new threads. This enhances the flexibility available to the expert developer.

3.2 Dynamic reprogramming

Once an application has been developed or modified, the sensor network user has the additional challenge of testing and deploying the application in the network. Using existing sensor network systems, this task may be difficult, possibly requiring physical interaction with every node in the network. This is clearly undesirable, since sensor networks may be deployed in inaccessible areas and may scale to thousands of nodes [Tila02], making physical access to each node prohibitively expensive or even impossible. To overcome the difficulty of reprogramming the network, MOS includes two reprogramming modes.

The simpler programming mode is similar to that used in many other systems and involves direct communication with a specific MANTIS node. On a Nymph, this would be accomplished via the serial port: The user simply connects the node to a PC and opens the MANTIS programming tool. Upon reset,

MOS enters a bootstrap routine that checks for communication from the programming tool. At this point, the node will accept a new code image, which is downloaded from the PC over the direct communication line. From the programming tool, the user also has the ability to inspect and modify the node's memory directly (peek and poke), as well as spawn threads and retrieve debugging information—including thread status, stack fill, and other such statistics—from the operating system.

The more advanced programming mode is used when a node is already deployed, and does not require direct access to the node. The spectrum of dynamic reprogramming of in-situ sensor networks ranges from fine grained reprogramming (modifying constants like sampling rate) to complete reprogramming of the sensor nodes. MOS has a provision for reprogramming any portion of the node—up to and including the operating system itself—while the node is deployed in the field. This is accomplished through the MOS dynamic reprogramming interface. The capability to use the dynamic reprogramming interface will be built into the MANTIS programming tool.

Current solutions for dynamic reprogramming [Levi02b] are virtual machine (VM) -based where the VM resides over the underlying sensor operating system and processes the incoming code capsules. A special stack-based instruction set is used to reprogram the sensor nodes, reducing the amount of data that is transmitted over the network. In contrast to the VM based approach, MOS allows binary updates to reprogram a node. The developer does not need to learn a new stack-based instruction set; instead, the existing deployed application only needs to be modified and recompiled, then transmitted to the MANTIS node. One disadvantage with our technique is that the size of the update that is transmitted over the wireless link to reprogram the sensor nodes may be large, resulting in a high communication cost. We use techniques commonly used in software Distributed Shared Memory (DSM) systems [Cart95] to reduce communication due to false sharing. In a DSM system a diff of the modified virtual memory page and the original page is constructed. As an update, only this diff is transmitted, rather than the entire modified virtual memory page.

Similarly, the MANTIS programming tool maintains a copy of each node's currently loaded code image at the user's PC. When the user wishes to update the code image, the tool will first compiles the new code image. It will then produce a patch that will transform the old code image into the new, which is sent to the node over the network in the normal way. Upon receiving the patch, the node reads its existing code image and applies the patch, storing the revised code image into its EEPROM (or whichever nonvolatile storage is available on the given platform). The node then sets the “reprogram on boot” flag and resets itself. This flag is recognized by the MOS bootstrap code, which then reprograms the text memory with the new code image, clears the reprogram flag, and resets the node again. Thus, the entire code image, with the exception of the locked bootstrap section, may be reprogrammed over an arbitrary network while the node is deployed.

3.3 Remote shell

Traditional solutions for network management such as SNMP [Case90] are not applicable to highly dynamic sensor networks. Existing solutions for monitoring sensor networks look at topology extraction [Deb01] and computing summaries of network properties for energy efficient monitoring of sensor networks [Zhao03]. In addition to these mechanisms, the user may wish to manage the nodes in the network in other ways. To provide this ability, MOS includes a remote shell. From any device in the network equipped with a terminal (a PC, for example), the user may invoke the remote shell and “log in” to a node. This node may be either a physical node (e.g. on a Nymph or Mica board) or it may be a virtual node running as a process on a PC. From the remote shell, the user may alter the node's configuration settings, run or kill programs, display the thread table and other operating system data, inspect and modify the node's data memory, and call arbitrary user-defined functions. The shell is a powerful debugging tool, since it allows the user to examine and modify the state of any node, without requiring physical access to the node.

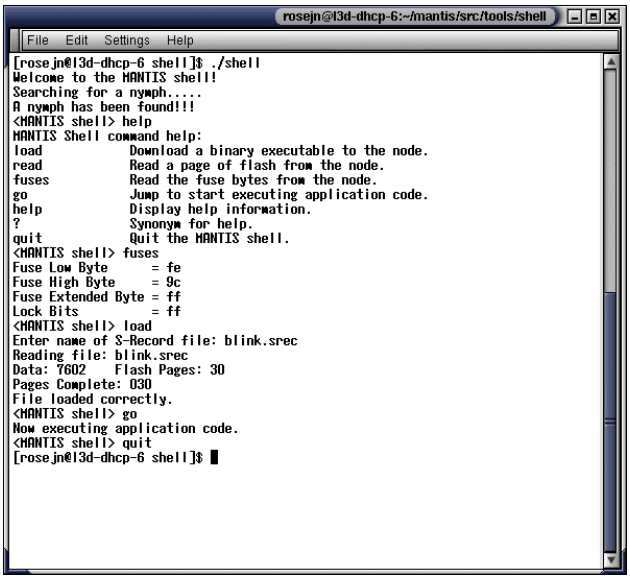

Figure 3. Screenshot of the remote shell.

4 MULTIMODAL SENSING AND COMMUNICATION

To demonstrate the benefits of integrated multimodal sensing and multimodal communication, we construct and analyze GPS-enabled sensor networks in this section. Section 4.1 implements a location mapping application using mobile GPS-enabled sensor nodes. Section 4.2 describes a joint multimodal framework for testing time synchronization algorithms in sensor networks. GPS-enabled sensor nodes provide clocks at each sensor node to assess the accuracy of time synchronization algorithms, including variations of the Network Time Protocol (NTP) [Mills94].

4.1 Location Mapping With GPS-enabled Sensor Nymphs

With the addition of GPS to the MANTIS system, we have greatly expanded the potential of the nymph platform. Instantly we have added location awareness to any outdoor deployment of the system; furthermore, we additionally get an accurate time signal and elevation information. Using the TSIP interface of the Trimble LassenSQ module we can receive periodic messages containing all the GPS information, or it is possible to put the module in query mode which allows us to request specific pieces of information. After the initial startup period where the GPS receiver constructs an almanac of satellite information, it is possible to get continuous location updates even if the whole module is moving at very high speeds. This feature enables vehicle or wildlife tracking to be integrated into a sensor network like never before. For application scenarios where the sensor nodes will be placed in static locations, it is possible to get a single GPS reading and then power down the whole module to minize power usage. This location information can then be used in a multitude of algorithms such as the construction of routing tables or the determination of rf output power levels. As shown in Figure 4, our

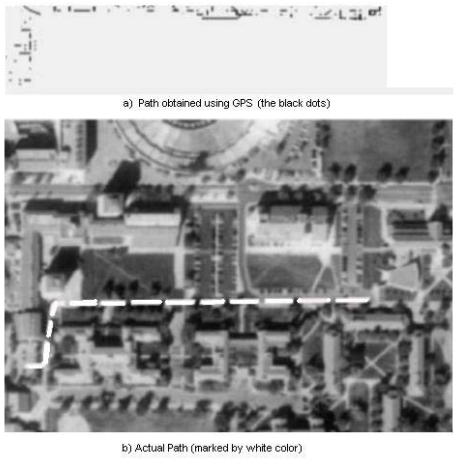

Figure 4. (Top) computed GPS path taken by mobile GPS-enabled sensor node. (bottom) overlaid path on satellite picture of campus.

initial test of the new GPS capability was to log the location information on a single node as it was taken across the university campus at walking speeds. This simple demonstration of the GPS unit tests the mobility of the system as well as our driver implementation for MOS.

4.2 Time Synchronization with GPS Clock and Multi-channel Radio

Time synchronization is an important issue in general sensor networks for a variety of reasons. First, it is often critical to know the time stamp when a sensor reading is taken. If there is no globally synchronized clock on board each sensor node, then the time stamp given for a certain sensor reading may be inaccurate after the local clock has drifted away from the true time. Inaccurate time stamps for sensor readings from different sensor nodes can lead to reordering of events, and possibly even a reversal of actual events. A second motivation behind distributed time synchronization is to synchronize the wake and sleep periods of a sensor network, i.e. employ time synchronization to enhance low power management. An effective technique is to sleep the sensor network with a low duty cycle and periodically wake up it up briefly to query the sensor readings before going back to sleep [Main02]. However, in a sensor network, the sleep times need to be coordinated rather than random. Because sensor networks involve multi-hop routing, random sleep times may very well sleep a neighbor just as an adjacent node wakes, resulting in slow propagation of packets through the network graph filled with sleeping nodes. Rather, the intent is to synchronize the wake and sleep times of the entire network, so sensor data can be quickly routed to the base station while the network is briefly awake.

Figure 5 illustrates a general network topology in which some nodes are GPS-enabled while others either lack GPS or lack line of sight to be able to observe GPS satellites, perhaps because of obstacles such as foliage, mountains, or buildings. Those nodes that lack GPS can employ a time synchronization protocol like NTP to obtain an accurate local clock from the GPS backbone nodes In NTP, a node calculates the local clock by first querying a node with a true clock, namely our GPS-enabled nodes, and then constructs a local estimate of the true time returned in the response packet by factoring in the roundtrip time and processing time at the true-clock node. For example, nodes N1 and N2 are shown querying the nearest GPS-enabled node to obtain their clocks. A variety of other time synchronization protocols have also been evaluated for wired networks [Liao99], ad hoc networks [Rome01], and have been proposed for wireless sensor networks [Gane02, Elson02a], though without the benefit of GPS evaluation in a true deployment. Even in a network that is fully GPS-enabled, some nodes may be obscured, so that a time synchronization protocol is still a requirement.

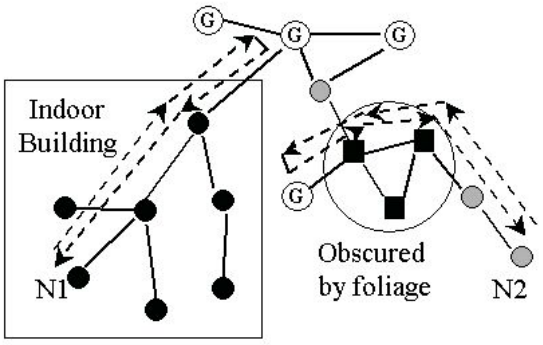

Figure 5. Active GPS sensor nodes (G) provide clock synchronization to N1 (indoors) and N2 (outdoors, lacks GPS) via a protocol like NTP. (dark squares have GPS but are obscured)

The goal of this section is to demonstrate the strength and versatility of multimodal sensing and communication by providing a GPS-enabled framework for evaluating the accuracy of any time synchronization algorithm deployed in-situ within a real sensor network. As shown in Figure 2, we have integrated Trimble's Lassen SQ GPS chip onto our nymph platform via the second USART interface to the ATMEGA 128 microcontroller. Through this simple multimodal extension to the hardware, we have attained the capability to evaluate the accuracy of any number of time synchronization protocols. On a given sensor node, the estimated clock obtained from any arbitrary time synchronization algorithm is compared to the authoritative clock available on the GPS-enabled sensor node right after reception of query response in order to assess the accuracy of the time estimation algorithm.

We selected NTP as an obvious first time-synchronization candidate for evaluation, as pictured in Figure 5. We constructed an experiment consisting of a network topology with five sensor nymphs in a row, with the outermost node requesting time synchronization from the innermost node. This linked routing chain is

similar to the linear route in Figure 5 taken by node N2's NTP request, which propagates to the GPS-enabled node four hops away. Broadcast routing was used to connect each node to its neighbors. The outermost node made one request to the GPS-enabled node three hops away and estimated the clock from that roundtrip time using NTP. This estimate was compared to the authoritative clock from GPS. Sync-requests are initiated by the client every 10 seconds. Each nymph nodes picks a random start point. This experiment was repeated a hundred times to obtain an average estimate of the inaccuracy. We selected a second algorithm to evaluate called Hierarchical NTP (HNTP), similar to [Gane02]. In HNTP, each node only makes an NTP query to its upstream parent. Thus, in HNTP, clocks are distributed outward from an authoritative source, neighbor by neighbor. The advantage of HNTP is that clock updates can be distributed in a scalable manner. Queries are only local to an upstream neighbor, rather than global back to the source. This is helpful in a sensor network where energy efficiency is at a premium. Time synchronization requests no longer propagate throughout the network.

In addition to NTP and HNTP, we chose to evaluate both algorithms over multi-frequency radios, to further demonstrate the utility of joint multimodal operation simultaneously in both the sensing and communication dimensions. The Chipcon CC1000 radio on the nymphs supports spread spectrum multi-channel communication. We observe that the accuracy of NTP's time synchronization's algorithm is highly dependent on the variation in delay experienced during the roundtrip NTP query. Indeed, the research literature goes to great lengths to statistically model this variation [Arvi94,Cris89]. MAC-layer collisions in a wireless sensor network increase the variation in delay experience by NTP queries, and hence reduce the accuracy of NTP time estimation. By employing a separate control channel for time synchronization queries and responses, variations in latency due to collisions are reduced, resulting in more accurate clock estimation. We labeled multi-channel NTP as MNTP, and similarly multi-channel HNTP as M-HNTP. To model modest packet traffic consisting of sensor data, a sixth node neighboring one of the three intermediate nodes injected 150 packets to the network within the 10 minute trial period.

The results of our experiments are shown in Table 2. NTP by itself achieves an average accuracy within 38 μs of the authoritative GPS clock. Hierarchy via HNTP reduces the error further, achieving mean time synchronization within 23 μs of the true clock. Hierarchy via hop-by-hop synchronization significantly reduces the roundtrip deviation. We speculate that hop-by-hop synchronization bounds the delay variance at each hop, removing the cumulative effect of multi-hop latency that increases variation. Multi-channel communication achieves a further reduction in both the mean and variance, MNTP achieving 20.9 μs accuracy, while MHNTP achieves the best accuracy of around 10.44147 μs. As we had surmised, multi-frequency communication reduced collisions hence the variance in delay compared to the simple NTP time synchronization algorithm and thereby improved the time estimation accuracy. This simple experiment demonstrates the flexibility of integrated multimodal sensing and communication. We believe the performance could be improved by further optimization.

Algorithm	**Mean Error**	**StdDev**
NTP	39.81979	43. 2736
HNTP	23.6972	26.438
MNTP	20.9021	22.9463
MHNTP	10.44147	11.5746

Table 2. Time synchronization accuracy in μs.

5 MULTIMODAL DEPLOYMENT

Due to the wide availability and support of the complete programming environment for the AVR microcontroller under Linux and Windows, it is possible to build MOS, with minor modifications, as an application that runs on the X86 platform. We call this user space application running on an X86 platform

XMOS. XMOS provides wrappers around some of the core kernel modules of MOS viz. networking and scheduler, thus enabling XMOS nodes to provide the same uniform interface as that provided by AMOS. This enables MANTIS to support mixed-mode networks, consisting of XMOS nodes and AMOS node seamlessly interacting with each other resulting in simplified development, experimentation, phased deployment and debugging of sensor network applications.

5.1 Bridging to AMOS: phased deployment into physical world

Typically, sensor network applications are initially tested either in simulation or in a small scale test deployment before the actual real deployment. Both these methods fall short of thoroughly testing the sensor network application leading to unforeseen problems. For example, data centric routing protocols like [Inta00] are sensitive to the physical phenomena that are sensed and simulations may not be able to completely verify all the test cases. However, our hybrid XMOS/AMOS multimodal deployment allows phased deployment of the sensor network application into the physical world. Initially only a small number of AMOS nodes are deployed and a set of XMOS nodes are simulated in the virtual environment. As the sensor network application is developed and tested, more AMOS nodes are deployed into the physical environment, leading to a controlled, phased deployment. For example sensor networks are often deployed in extreme habitats like mountains, forests, industrial plants, etc. Our phased deployment scheme allows a small network to be deployed and the XMOS nodes are fed with the actual sensor data of the small scale deployment. The XMOS network is tested thoroughly and gradually more AMOS nodes are deployed, eventually leading to a full scale sensor network deployment. At each stage of the phased deployment, the modified sensor network application is migrated from the XMOS platform to the AMOS platform using the dynamic re-programming scheme.

Figure 6 shows the structure of the network, with the two networks connected to each other via a serial RS232 link. A simple flooding algorithm is deployed on the AMOS nodes as well as the virtual XMOS nodes. Transparent flooding of packets from one network to another via the serial link is facilitated by the uniform API, allowing the two hybrid XMOS/AMOS platforms to co-exist. Thus, a mos_send(…) system call on the AMOS nodes causes the data to be transmitted over the radio. Node *X* which is configured as a bridge node would additionally also send the data over the serial link using the mos_uart_send(..) call. Similarly, a mos_send(…) call on the XMOS nodes causes the data to be transmitted over the IP network with the bridge node *Y* additionally transmitting the data over the serial link using the mos_uart_send(…) system call. Hence, the same application runs transparently over the XMOS and AMOS platforms allowing detailed and fine grained simulation of the sensor network application via seamless bridging of the physical and virtual network.

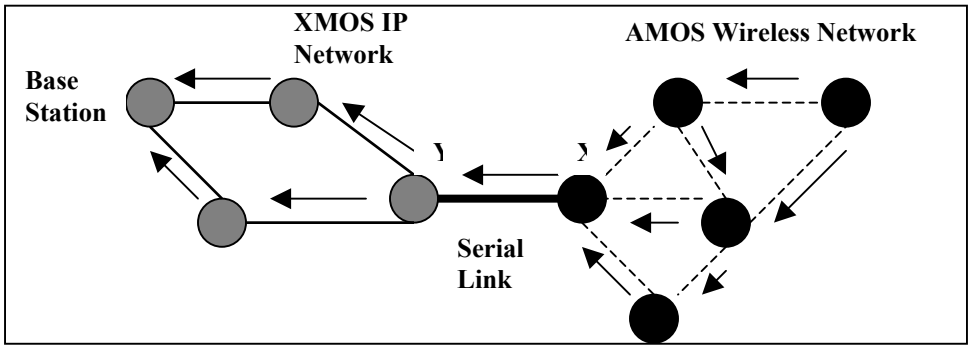

Figure 6. Heterogeneous XMOS/AMOS network.

For large scale sensor network deployments, multiple XMOS nodes can be simulated over a single physical PC. This facilitates testing scalability of a sensor network and provides a centralized control point to simulate the power usage model, different network topologies, the radio communication, different sensor drivers etc. However, a drawback of the simulation is the difficulty of simulating wireless communication channels, especially indoor communication [Hash93]. Simulating the wireless channel using an 802.11 network interface is not sufficient to simulate the nymph wireless channel. We address this problem by

connecting a nymph to the serial interface of a laptop and using the nymph as a wireless interface for the XMOS process running on the laptop.

5.2 Framework for sensor network simulator

[Levi02a] is a simulator for TinyOS, allowing applications built for the PC platform to be debugged and verified before they are deployed on actual sensor nodes. Packet injection into the network is simulated by either injecting packets by statically reading from a pre-configured file or by a java application which can inject packets created by the user. However, behavior of most sensor network applications is sensitive to the real sensor readings and the physical environment in which the applications are deployed are very difficult to simulate on the virtual sensor nodes. By routing the actual physical phenomena into the virtual network, MANTIS allows the developer to observe the true behavior of the application on the virtual nodes. [Park00], an extension to ns2, is a simulation framework which models the sensor nodes and also provides a "hybrid" simulation combining the real and virtual network. However, the sensor network applications are required to be re-implemented for the target platform, resulting in two completely different code bases that must be maintained. The transparent bridging between the AMOS/XMOS networks, MANTIS provides a prototype extensible framework for simulating applications and hardware for sensor networks and also simultaneously expediting the develop-test-deploy cycle for sensor network applications.

5.3 Multiple base station flexibility

Topology of sensor networks is typically asymmetric, with a centralized controller called the base station controlling the underlying sensor network. The base station is responsible for maintaining the global state of the network and the book keeping information required for the management of the sensor network. In such an asymmetric topology the centralized base station is a single point of failure and failure of the base station may lead to the collapse of the entire sensor network. Very often it is required to have multiple base stations in a network which distribute the load and also maintain redundant information about the network to make base station failures transparent. In a typical asymmetric topology the base station is connected to the sensor network via a single serial link. Compromise of this sensor node could also bring the entire sensor network down by not allowing any of the sensor network data routed up to the base station. Our hybrid XMOS/AMOS network topology fosters the experimentation of placement of multiple base stations in a sensor network. Figure 7 shows a configuration of a sensor network with multiple XMOS base stations connected via serial links to the AMOS nodes. Thus failures of multiple XMOS nodes can be gracefully tolerated.

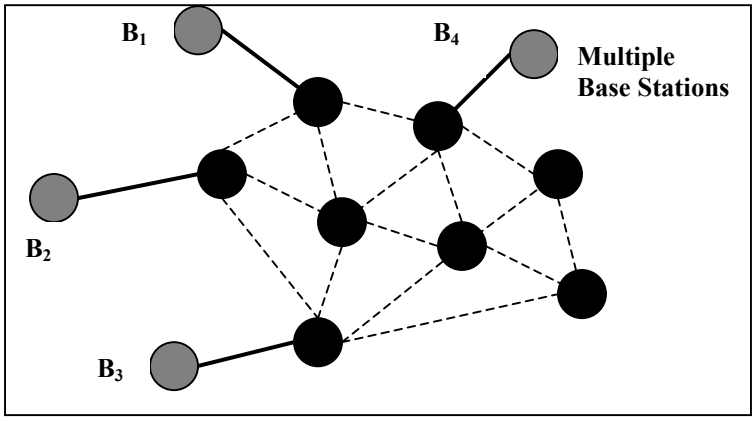

Figure 7. Multiple XMOS base stations interfacing with AMOS network.

Very often in a sensor network deployment it is not possible to have the base station which is directly connected to the sensor network to behave as a gateway to the Internet and ship data to a centralized reliable data collection repository. Often this reliable centralized repository may be many hops away from

the sensor network. By bridging the AMOS and XMOS network, multiple XMOS nodes transparently route the data over multiple hops over the IP network to the centralized repository. Figure 6 shows the base station located 2 hops away from the network of AMOS nymphs. The uniform interface across XMOS and AMOS platforms allows the same routing protocol bridge seamlessly across the two networks.

6 RELEASE PLAN

Our plan is to release MOS online by the late Spring of 2003 at the following Web site, *http://mantis.cs.colorado.edu*. Similarly, the hardware design will be released on this site. All code and schematics will be freely available. As the MANTIS project evolves, updates to MOS and the hardware design will be posted on this Web site. Forums for discussion and continued development by the research community will also be provided, and may be eventually migrated to sourceforge. Our intent is to enable researchers and developers to participate in the design and evolution of the MANTIS nymph and open source OS, thereby accelerating the introduction and integration of new features and improved performance.

7 CONCLUSION

The MANTIS platform has been designed to meet the demands of advanced multimodal sensor network deployments while simplifying the process of application development. The MANTIS Nymph sensor node provides a hardware implementation that is ideally suited to take advantage of the unique power conservation and dynamic reprogramming capabilities of MOS in a small, robust, and inexpensive package. As the core of the MANTIS system, MOS provides developers with a familiar programming environment, and the flexible, platform-independent nature of the kernel allows MOS software to be deployed on a variety of architectures with little or no modification. We have demonstrated the practical significance of these features by a new approach to sensor network simulations that leverages the ability of MOS to run as a UNIX process, and the integration of a GPS unit that allows the synchronization of timing and location information with sensor data.

8 REFERENCES

[**Arvi94**] K. Arvind, "Probabilistic Clock Synchronization in Distributed Systems," IEEE Trans. parallel and Distributed Systems, vol. 5, no. 5, pp. 475-487, May 1994.

[**Atme03**] Atmel AVR 8-bit RISC processor, http://www.atmel.com/products/AVR

[**Cart95**] J. Carter, J. Bennett, W. Zwaenepoel “Techniques for Reducing Consistency-Related Communication in Distributed Shared-Memory Systems” ACM Trans. on Computer Systems 13(3), pp. 205-243, August 1995

[**Case90**] J. D. Case, M. Fedor, M. L. Schostall, and C. Davin. RFC 1157: Simple network management protocol (SNMP). RFC, IETF, May 1990

[**Chip01**]Single chip ultra low power RF transceiver http://www.chipcon.com/files/CC1000_Data_Sheet_2_1.pdf, 2001

[**Cris89**] Flaviu Cristian. Probabilistic clock synchronization. Distributed Computing, 3:146–158, 1989.

[**Deb01**] B. Deb, S. Bhatnagar, B. Nath “A Topology Discovery Algorithm for Sensor Networks with Applications to Network Management”, DCS Technical Report DCS-TR-441, Rutgers University May 2001

[**Elso01**] J. Elson, D. Estrin “Time Synchronization for Wireless Sensor Networks” Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS), Workshop on Parallel and Distributed Computing Issues in Wireless and Mobile Computing, San Francisco, California, USA. April 2001

[**Elso02a**] J. Elson, L. Girod, D. Estrin “Fine-Grained Network Time Synchronization using Reference Broadcasts”, In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, MA. December 2002.

[**Elso02b**] J. Elson, K. Römer, "Wireless Sensor Networks: A New Regime for Time Synchronization", in proceedings of the First Workshop on Hot Topics In Networks (HotNets-I), Princeton, New Jersey. October 28-29 2002

[**Enge99**] P. Enge, P. Misra, "Special Issue on Global Positioning System," Proceedings of the IEEE, Volume 87, No. 1, January, 1999, pp. 3-15.

[Eyes02] Eyes: Energy Efficient Sensor Networks, http://eyes.eu.org, March 2002

[**Gane02**] S. Ganeriwal, R. Kumar, S. Adlakha, M. Srivastava, "Network-wide Time Synchronization in Sensor Networks," Technical report, University of California, Los Angeles, Dept of Electrical Engineering, 2002.

[**Hill00**] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, "System Architecture Directions For Network Sensors", ACM Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) 2000, pp. 93-104.

[**Hill03**] Jason Hill's Smart Dust Spec, www.cs.berkeley.edu/~jhill/spec/index.htm

[**Inta00**] C. Intanagonwiwat, R. Govindan, D. Estrin. "Directed diffusion: A scalable and robust communication paradigm for sensor networks." In Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom) 2000, pp. 56–67.

[**Kahn99**] J. M. Kahn , R. H. Katz , K. S. J. Pister, "Next century challenges: mobile networking for "Smart Dust"", MobiCom 1999, pp. 271-278.

[**Kern84**] B. Kernighan, R. Pike, "The Unix Programming Environment" Prentice Hall, 1984

[**Levi02a**] P. Levis, N. Lee "Nido System Description", http://webs.cs.berkeley.edu/tos/tinyos-1.x/doc/nido.pdf, October 2002.

[**Levi02b**] P. Levis, D. Culler "Mate: a Virtual Machine for Tiny Networked Sensors" ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Oct. 2002.

[**Liao99**] C. Liao, M. Maronosi, D. Clark, "Experience With an Adaptive Globally-Synchronizing Clock Algorithm," In Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), 1999, pp. 106-114.

[**Main02**] A. Mainwaring, J. Polastre, R. Szewczyk D. Culler, J. Anderson,"Wireless Sensor Networks for Habitat Monitoring", First ACM Workshop on Wireless Sensor Networks and Applications (WSNA) 2002, pp. 88-97.

[**Mart00**] F. Martin, B. Mikhak, and B. Silverman, "MetaCricket: A designer's kit for making computational devices," IBM Systems Journal, vol. 39, nos. 3 & 4, 2000.

[**Mill91**] D. Mills, Z. Yang, T. Marsland (Eds.) "Internet Time Synchronization: the Network Time Protocol" Global States and Time in Distributed Systems, IEEE Computer Society Press 1991

[**Mills94**] D. .Mills, Internet Time Synchronization: The Network Time Protocol. In Zhonghua Yang and T. Anthony Marsland, editors, Global States and Time in Distributed Systems. IEEE Computer Society Press, 1994.

[**Park00**] S. Park, A. Savvides, M. B. Srivastava, "SensorSim: A Simulation Framework for Sensor Networks", In the Proceedings of MSWiM 2000, Boston, MA, August 11, 2000.

[**Priy00**] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan, "The Cricket Location-Support System." Proc. 6th ACM MOBICOM, August 2000, pp. 32-43.

[**Rash89**] R. Rashid, R. Baron, et al. "Mach: A Foundation for Open Systems". In Proceedings of the 2nd IEEE Workshop on Workstation Operating Systems. 1989.

[**Rome01**] K. Romer, "Time Synchronization in Ad Hoc Networks", MobiHoc 2001.

[**Tila02**] S. Tilak, N.B. Abu-Ghazaleh, W. Heinzelman, "A taxonomy of wireless micro-sensor network models", ACM SIGMOBILE Mobile Computing and Communications Review, Vol. 6 Ch. 2 pages 28-36. 2002.

[**Xbow03**] Crossbow Mica2 and Mica2dot Motes and Sensors, http://www.xbow.com/Products/Wireless_Sensor_Networks.htm, 2003.

[**Wan02**] C. Wan, A. Campbell, L. Krishnamurthy, "PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks", First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA) 2002.

[**Want92**] R. Want, A. Hopper, V. Falcao, J. Gibbons, "The Active Badge Location System," ACM Transactions on Information Systems, Vol. 10, No. 1, January 1992, pp. 91-102.

[**Zhao03**] J. Zhao, R. Govindan, D. Estrin "Computing Aggregates for Monitoring Wireless Sensor Networks", In proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications Anchorage, AK. May 2003.