Trio: Enabling Sustainable and Scalable Outdoor Wireless Sensor Network Deployments

Prabal Dutta[†], Jonathan Hui[†‡], Jaein Jeong[†], Sukun Kim[†],
Cory Sharp[⋆], Jay Taneja[†], Gilman Tolle[†‡], Kamin Whitehouse[†], and David Culler[†‡]

[†]UC Berkeley EECS Dept.
Berkeley, California 94720

[‡]Arched Rock Corporation
San Francisco, California 94105

[⋆]Moteiv Corporation
San Francisco, California 94105

ABSTRACT

We present the philosophy, design, and initial evaluation of the Trio Testbed, a new outdoor sensor network deployment that consists of 557 solar-powered motes, seven gateway nodes, and a root server. The testbed covers an area of approximately 50,000 square meters and was in continuous operation during the last four months of 2005. This new testbed in one of the largest solar-powered outdoor sensor networks ever constructed and it offers a unique platform on which both systems and application software can be tested safely at scale. The testbed is based on Trio, a new mote platform that provides sustainable operation, enables efficient *in situ* interaction, and supports fail-safe programming. The motivation behind this testbed was to evaluate robust multi-target tracking algorithms at scale. However, using the testbed has stressed the system software, networking protocols, and management tools in ways that have exposed subtle but serious weaknesses that were never discovered using indoor testbeds or smaller deployments. We have been iteratively improving our support software, with the eventual aim of creating a stable hardware-software platform for sustainable, scalable, and flexible testbed deployments.

Categories and Subject Descriptors

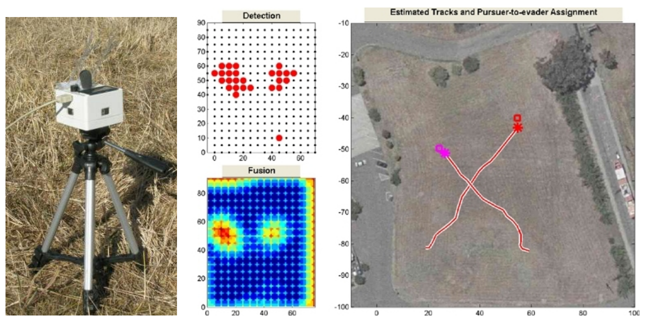
C.2.3 [**Computer-Communication Networks**]: Network Operations; C.3 [**Special-Purpose and Application-Based Systems**]: Real-Time and Embedded Systems

General Terms

Management, Measurement, Performance, Design, Reliability, Experimentation

Keywords

Sensor Networks, Testbed, Large-Scale, Long-Life, Detection, Target Tracking, Surveillance



(a) Trio Mote (b) Multi-Target Tracking Application

Figure 1: The Trio platform and an experimental multi-target tracking application that uses it.

1. INTRODUCTION

Experience shows that developing large-scale, long-lived, outdoor sensor networks is challenging. Current simulators like TOSSIM [12] and PowerTOSSIM [19] fail to capture the complex physical phenomena that appear in real deployments. Indoor testbeds like MoteLab [23] and Mirage [4] use real radio hardware that provides much greater communications realism but does not capture the nuances of outdoor environments. Portable testbeds like EmStar [7] allow realistic outdoor experimentation but their wired backchannels are a double-edged sword: on one hand they provide great visibility but on the other hand, they limit the scale of the deployment. Testing at realistic scales is important because each order of magnitude increase in network size ushers in a new set of unforeseen challenges. Outdoor sensor network deployments like ZebraNet [14], GDI [20], Redwoods [22], VigilNet [9], and ExScal [2] provide unparalleled realism but these networks have achieved either large scale or long life, but usually not both, as described in Section 2.

In this paper, we identify and begin to address the myriad challenges of moving a testbed from the friendly confines of the indoors to the unpredictable world outside. Our principal hypothesis was that an outdoor testbed needed to be *scalable*, *sustainable*, and *fail-safe flexible* to meet the testing needs of system and application developers. To test this hypothesis, we designed the *Trio* mote and deployed a testbed of 557 Trio nodes over an area that covers approximately 50,000 square meters. Our colleagues developed a multi-target tracking (MTT) application to help evaluate the testbed. Figure 1 shows Trio and the MTT application, and Section 3 presents the system architecture.

Trio is a new, open experimental platform designed to better address the requirements of a large-scale, long-lived, outdoor testbed. The Trio nodes reside in the lowest tier (Tier-1) of the architecture. Trio provides support for application-level experimentation through a sensor suite optimized for detection and classification of humans and vehicles as well as support for system-level experimentation through hardware and firmware for fault-tolerant operation. Trio integrates Telos [17], the eXtreme Scale Mote (XSM) [6], and Prometheus [11], and improves upon their designs in several ways. Trio addresses *sustainable* operation through a solar-power based renewable energy supply with supercapacitor and Lithium-Ion battery storage elements, support for efficient *in situ* maintenance and fail-safe operation, and environmentally-hardened package design. Trio addresses *scalability* by reducing the cost of human-in-the-loop operations. Fail-safe *flexibility* is addressed through the use of the Deluge network reprogramming system, watchdog and grenade timers, and one-touch recovery. The details of the Trio sensor node are presented in Section 4.

The 557 Trio nodes in the testbed are organized into multiple routing trees, with each tree rooted at a gateway. Gateways forward traffic between the 802.15.4 Trio network and an 802.11 wireless backbone network. Each Trio node dynamically associates with the closest gateway based on routing cost. Gateways are physically distributed throughout the mote tier and they support network scalability through hierarchy: gateways can be added as the number of nodes increase. Gateways support sustainable operation through solar power and since they simply forward traffic statelessly, flexibility is not required. The gateways sit in Tier-2 of the system architecture along with 802.11 repeaters and an access point which bridges the 802.11 network to an 802.3 Ethernet network. Section 5 presents the details of Tier-2, the gateway tier, in our system architecture.

A single root server sits in Tier-3 and connects to all of the gateways. The server maintains a TCP session with each gateway while a daemon on the server multiplexes these TCP sessions using gather-scatter communications and exposes them as a single TCP session. This approach simplifies client interaction by presenting a unified view of the network and abstracting superfluous implementation details. The server also runs network monitoring and management software. This software allows active querying or passive monitoring of the network and stores the resulting data for online or later offline analysis. The monitoring software supports scalability by aggregating large amounts of information from several sources into simple but informative graphics and tables. The management software supports flexibility by allowing network programming and other control functions. Section 6 discusses the software that runs on the server.

Multi-target tracking (MTT) was the first application to use the Trio Testbed. A simple detection algorithm reported detection events to clients, sitting in Tier-4, using collection routing. Oh et al. ported their MTT algorithms [16] to receive detection events via the root server. Using a 144-node subset of the testbed, they demonstrated the ability to disambiguate multiple crossing tracks that intersect in space and time, as shown in Figure 1(b). The use of the testbed for this application has highlighted problems with our system software and has raised new challenges that we would not have discovered in a small-scale or indoor setting. Section 8 discusses these discoveries.

2. RELATED WORK

In Section 1, we hypothesized that sustainability and scalability were necessary for achieving coverage over large extents of space and time, and flexibility was essential for a testbed to be useful beyond a narrow application window. This section reviews the scale, lifetime, and flexibility of earlier systems but our review is *neither exhaustive nor exact* since several deployments remain unpublished and published work does not always provide exact statistics.

2.1 Deployments

While several deployments have realized aspects of the large-scale, long-lived, sensor network vision, no single system has integrated all of the pieces into a cohesive whole. Figure 2 illustrates this point: sensor network deployments have achieved either scale or lifetime, but usually not both.

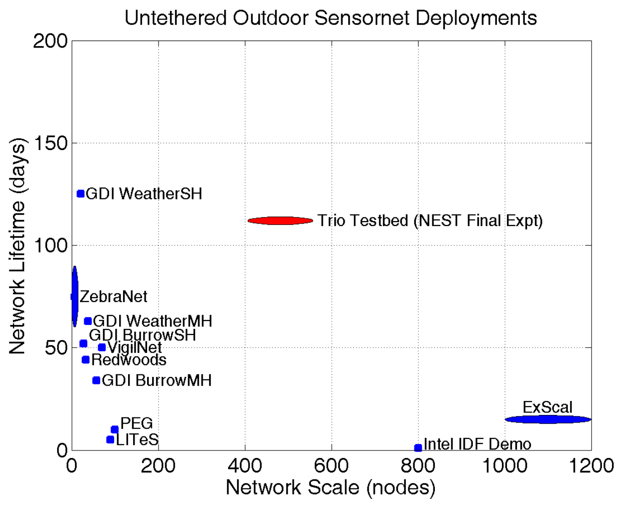


Figure 2: Scale and lifetime of several earlier sensor network deployments and the Trio testbed. An ellipse indicates an uncertain estimate or variance.

ZebraNet is sustainable because of its renewable energy supply and environmentally-hardened enclosure. However, it uses an unscalable TDMA MAC with time-synchronized and statically assigned timeslots. Since "ZebraNet only expects tens of nodes...this non-scalable solution is acceptable." [14] ZebraNet is also limited in its flexibility since occasionally-connected, mobile nodes cannot be easily reprogrammed. Other than ZebraNet, none of the networks presented in Figure 2 used a renewable energy supply, so their lifetimes were limited to weeks or months.

Non-sustainable outdoor sensor network deployments have resorted to one of three approaches to budgeting energy: "fill the bank", "count your pennies", or a hybrid of the two. PEG [18], LITeS [1], and ExScal [2, 3] primarily followed a "fill the bank" approach in which some power management existed but periodic battery replacement was an element of the operational strategy. Unfortunately, this approach is unscalable since the level of effort required to manually replace or recharge batteries grows with the number of nodes in the network. VigilNet [9], GDI [20], and Redwoods [22] used a "count your pennies" approach which requires meticulous analysis of energy usage and a fine-grained allocation of the available energy. This approach, primarily used for carefully controlled, medium-term, outdoor deployments, does not afford flexibility or experimentation: once the application requirements are specified, the energy budget determined, and the battery selected, it is difficult to increase sampling, reporting, or communication rates without materially affecting lifetime. All of the deployments shown in Figure 2 are *application-specific* to varying degrees. The main issue with application specificity is that it is difficult to leverage these systems for other purposes, as would be necessary for a testbed. Except for ExScal and the Intel Developer's Forum demo, these earlier networks are small to medium scale, consisting of 100 or fewer nodes. At these scales, maintenance and management overhead does not dwarf other activities. Larger scale deployments, however, must pay careful attention to the maintenance and management issues that affect scalability and their platforms must be designed accordingly [2, 6].

2.2 Platforms

Table 1 summarizes a number of sensor network platforms. A sustainable platform requires a renewable energy supply because without it, frequent manual intervention is necessary to replace batteries. A flexible platform allows programming at both the application *and* system levels. Flexibility is a fundamental requirement for testbeds. Fail-safe flexibility requires support for recovering from buggy and possibly Byzantine programs, and is important when it is difficult or impossible to manually recover and reprogram nodes, as is the case for a large-scale, untethered testbed.

Table 1: Details of several sensor network platforms (top) and the three Trio building blocks (bottom).

Platform	Sustainable	Flexible	Fail-safe
ZebraNet [14]	Yes	No	No
Redwoods [22]	No	No	No
PEG [18]	No	Yes	No
GDI [20]	No	Yes	No
VigilNet [9]	No	Yes	No
LITeS [1]	No	Yes	No
Telos [17]	No	Yes	No
XSM [6]	No	Yes	Yes
Prometheus [11]	Yes	Yes	No

None of the platforms presented in Table 1 support both sustainable and fail-safe flexible operation. Therefore, considered individually, none of these platforms meet the needs of large-scale, long-lived, outdoor testbed. However, by incorporating features of several different platforms, a single platform can be constructed that provides all of the features that we desire for an outdoor testbed node. Indeed, the Trio platform takes exactly this evolutionary approach to platform design rather than the more risky one of simultaneously innovating in the areas of energy harvesting, processor/radio design, sensing, and fault-tolerance.

The Trio platform is composed of the Telos [17], XSM [6], and Prometheus [11] platforms. Trio directly integrates Telos into its design. From the XSM, Trio borrows and improves upon the hardware grenade timer for fail-safe flexibility and sensing circuitry for acoustic, magnetic, and passive infrared. Prometheus [11] provides the basis of the photovoltaic energy scavenging system in Trio but the Trio implementation improves upon the original design by adding support for fail-safe flexibility.

3. SYSTEM ARCHITECTURE

Our system is based on a four-tier hardware architecture that consists of a mote tier (Tier-1), a gateway tier (Tier-2), a server tier (Tier-3), and a client tier (Tier-4). In parallel with this hardware architecture exists the software architecture that consists of system software on the mote tier, and middleware services and application software that is distributed across the mote, server, and client tiers. Figure 3 illustrates the testbed hardware architecture and Figure 4 illustrates the testbed software architecture.

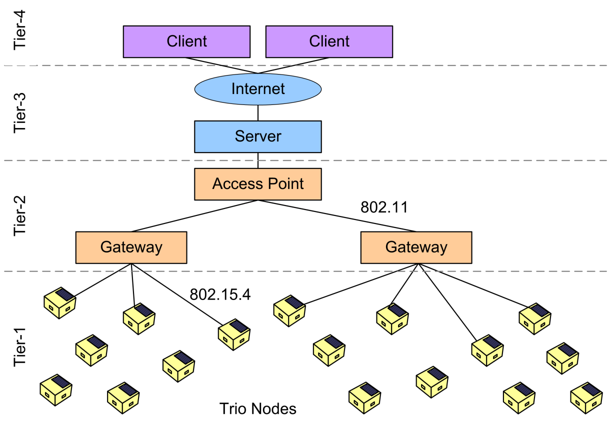


Figure 3: The system hardware architecture consists of four tiers: mote, gateway, server, and client.

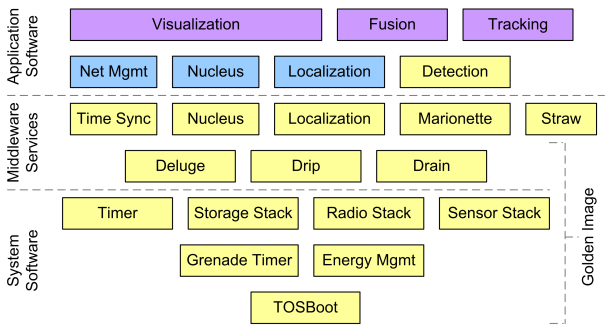


Figure 4: The software architecture consists of system software on the mote tier, and middleware services and application software that is distributed across the mote, server, and client tiers. The shading of the boxes correspond to the tier in which the software runs.

The four hardware tiers serve different functions:

- **Mote Tier.** The mote tier consists exclusively of *Trio* nodes that run embedded applications. This tier is responsible for sensing, local processing, and communications using an 802.15.4 wireless network.
- **Gateway Tier.** The gateway tier consists of *gateways*, *repeaters*, and *access points*. Gateways are physically distributed throughout the mote tier and they forward traffic between the 802.15.4 mote network and the 802.11 backbone network. Repeaters simply rebroadcast traffic and serve to extend the range of the

802.11 backbone network. The access point bridges the 802.11 backbone network with an 802.3 Ethernet network that is connected to a single root server.

- **Server Tier.** The server tier consists of a *root server* which runs network monitoring processes, gathers statistics on network behavior, multiplexes traffic from multiple gateways, and provides information on the health of the network to system users.

- **Client Tier.** The client tier consists of one of more *desktop computers* that run client-side applications. These applications access the network via the server tier which forwards traffic to and from the gateway tier.

The software architecture tiers serve several functions:

- **System Software.** The system software includes a *bootloader*, low-level *device drivers*, and *operating system services*. We use a customized version of *TOS-Boot*, the standard TinyOS bootloader, to initialize the system hardware, check for abnormal termination, and revert the system to a trusted code image called the *Golden Image*. The device drivers include software for managing the system peripherals, controlling the energy suppy, and intelligently managing energy transfer between the primary and secondary supplies. Finally, the operating system services include timers, storage, networking, and sensor stacks.

- **Middleware Services.** The middleware services include *network programming*, *routing*, *time synchronization*, *remote procedure calls*, *network management*, and *reliable data collection*.

- **Application Software.** The application software includes *localization* and *detection* software which run on the mote tier; network *management and visualization* software that runs on the server tier; and *localization*, *detection*, *fusion*, *tracking*, and *visualization* software that run on the client tier.

4. TIER-1: THE TRIO NODE

The Trio node, shown in Figure 5, is designed for long-lived operation with minimal physical maintenance. Trio itself is based on Telos [17], which supports low-power operation and remote reprogrammability, a necessity for flexible, long-lived applications. The Trio grenade timer and sensor suite are based on the XSM [6] and include passive infrared motion sensors, a magnetometer, and a microphone. The Prometheus solar charging system [11] provided the basis for a renewable energy supply.

4.1 Sustainable Operation

Sustainable operation is supported in two ways: through a renewable energy supply and by environmentally hardening the Trio enclosure.

4.1.1 *Renewable Energy*

Trio circumvents the typical lifetime limitation resulting from a non-rechargeable battery by including a renewable energy supply based on the Prometheus solar charging system [11] for maintenance-free self-charging. We modified the original Prometheus design to improve its performance and ensure fail-safe operation. A Trio node with a depleted capacitor or battery starts to wake up after solar energy charges the supercapacitor enough to produce a supply voltage of 1.8 V, the minimum operating voltage for the MSP430 processor and CC2420 radio. However, initializing sensor modules and writing to flash requires a higher supply voltage. A component of our Prometheus driver enforces hysteresis and waits to wake up the rest of the system until the supply voltage has risen past 2.75 V, which is enough to power the sensors and write to the flash memory. The application program is only started once the system voltage exceeds 2.75 V.

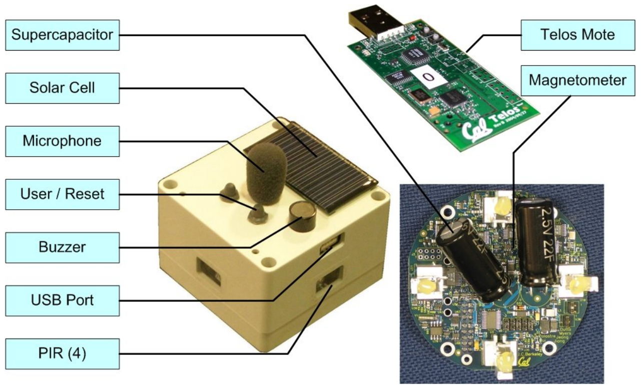


Figure 5: A Trio node and its key components.

4.1.2 *Environmental Hardening*

One of the key design challenges was to harden the Trio for an outdoor environment without hampering sensor performance or node maintainability. Several components of Trio (e.g. solar cell, passive infrared sensor, microphone, buzzer, and user/reset switches) are exposed to the environment for sensing, solar energy harvesting, and user input. We made these components weather-resistant so that Trio nodes could operate under varying weather conditions. To facilitate node programming, the USB port is also left exposed, though it must be sealed for long-term weather resistance. We had originally planned to mold a custom elastomer plug to seal the USB port but ended up using cellophane tape which we eventually discovered was a poor substitute. We used camera tripods to elevate the nodes, allowing Trio to avoid ground moisture, better detect objects using its passive infrared sensors, and obviate the need to mow the grass under the nodes. However, placing the node on a tripod led to an unexpected issue of bird droppings accruing on the nodes.

When the droppings cover part of the solar module on the Trio node, the amount of power that the solar module can provide diminishes. Solar cell *modules* are constructed from multiple solar *cells* typically connected in series. If any cell in the module is partially covered, then the output of the *entire module* is reduced by the proportion of the individual *cell* that is covered. Since each cell is very thin (3.4 cm by 0.4 cm), even a small bird dropping can reduce the output power of the solar cell by half or more. To dissuade birds from using Trios as perches, we mounted clear plastic spikes on top of the nodes. Since shadows from the plastic spikes can cause a similar loss of power, we also oriented the nodes so that the solar cells face south.

4.2 Efficient Physical Interaction

The Trio node supports scalable operation through efficient physical interaction. Trio exposes user and reset buttons, which allows efficient interaction with the node. The reset button provides two simple functions: reset and recovery. If the reset button is depressed once, the node resets. If, however, the reset button is quickly pressed and released three times in a row, then the active program reverts to the trusted Golden Image. The node also provides audible feedback about certain states and state transitions. When a node's capacitor voltage drops below a safe operating voltage, the node chirps every few seconds. If the node collects enough energy to operate, it chirps three times in quick succession as the application is started. These audible cues allow operators to passively gauge system status.

4.3 Fail-safe Flexibility

Since Trio can be programmed wirelessly using the Deluge network programming system, it is possible to program Trio with a buggy or even Byzantine program. Deluge is included in the Trio platform software, so network programming is automatically compiled into every application that uses the Trio libraries. The external flash can be used to store up to seven programs and simple Deluge commands can be issued to switch between the programs.

Several mechanisms are used to support fail-safe operation and recover from buggy or Byzantine programs. A watchdog timer ensures that software is making progress, tasks are executing, and interrupts are being handled. A grenade timer ensures that a node can recover from Byzantine applications by periodically transferring control to a trusted kernel. A USB override allows even the trusted code to be reprogrammed if necessary. A hardware override on the power system ensures that the system always reverts to the solar power supply in the event the battery dies during operation.

4.3.1 *Watchdog Timer*

A watchdog timer on the Telos microcontroller must be cleared periodically by the application to keep the microcontroller from resetting itself. The watchdog timer is reset in an interrupt context to ensure that timer interrupts are still firing. Sitting above the hardware watchdog timer is a software watchdog which checks if the scheduler is still executing: the hardware watchdog is only reset if the software watchdog tasks are executed to completion at some rate.

4.3.2 *Grenade Timer*

A grenade timer can be started by the bootloader and be used to periodically reset the node and force trusted code to run. If the grenade timer fires unexpectedly, the TOSBoot bootloader will detect this condition and automatically load the Golden Image, reverting the node to a known good state from which new code can be downloaded. The hardware grenade timer can be configured and started by TOSBoot to recover from Byzantine programs, but we turned this feature off for our experimentation because it results in periodic node resets.

4.3.3 *Power Switch Override*

An analog SPDT switch is used to switch between the supercapacitor and battery power supplies. In the original Prometheus design, the input pin that controls the power switch floats until the node wakes up and sets it. In the event the pin is set to use the battery, and the battery becomes depleted, the node will become inoperable and only manual intervention can recover it. Trio fixes this problem with a small hardware change: a pull-down resistor attached to the pin ensures that the switch is set to use the capacitor after each reset or power cycle. This change ensures that the Trio node will eventually run if there is enough sunlight. We also diode-OR connected the supercapacitor and battery to the analog switch power line to ensure that the switch has power if *either* supply has power.

4.3.4 *USB Override*

Trio exposes a USB port which can be used to program or power the node. The USB port allows the node, including the trusted bootloader, to be reprogrammed without disassembly. When the Trio is plugged into USB, the platform drivers automatically charge the system's supercapacitor and Lithium battery. Hence, even if a node has depleted its energy buffers, simply plugging the node into USB will replenish the supercapacitor and battery.

5. TIER-2: A NETWORK OF GATEWAYS

In a large-scale deployment, a gateway tier with high-bandwidth, wireless backbone spread throughout the network can serve several purposes: it can partition the traffic to lessen the overall network utilization; it can provide points for traffic observation; and it can support scalability through hierarchy. Hierarchy allows a large sensor network to be partitioned into multiple smaller networks that operate in parallel.

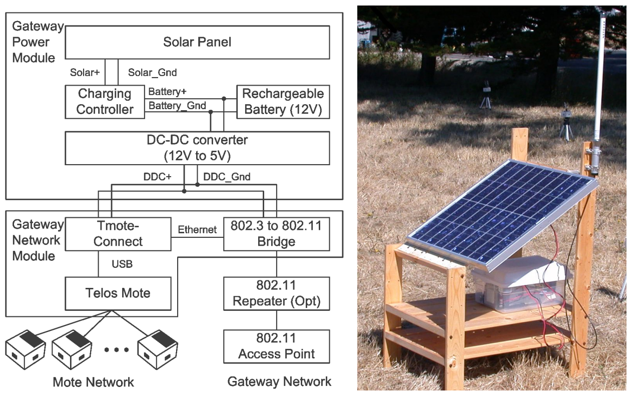


Figure 6: Gateway node and network architecture.

Figure 6 shows a gateway node and the backbone network architecture. The backbone network consists of a gateway node that forwards mote traffic to and from the 802.11 backbone network, optional 802.11 repeaters, and an 802.11 access point that connects this network to the root server.

A gateway node includes three major components: a Telos mote, a Telos-to-Ethernet gateway that forwards messages from the attached Telos mote to the Ethernet interface (Moteiv Tmote Connect), and an 802.3-to-802.11 bridge that forwards messages from the Tmote Connect to the 802.11 network. A 9 dBi omnidirectional antenna extends the gateway radio range. We tested the link throughput at 200 m and measured a packet yield exceeding 90%.

5.1 Sustainable Operation

With long-term sustainability in mind, we designed the gateway nodes to operate on solar power. The power supply for a gateway node consists of a solar panel, a charging controller, a gel cell battery, and a DC-DC converter. We chose a large, 17 Ahr gel cell battery to store solar energy because it simplified charging and provided the capacity needed to last through cloudy days. The gateway power system only required basic over-charge protection, provided by an off-the-shelf battery charging controller (Sunguard-4 by Morningstar). To determine the required output of the solar panel, we estimated the energy supply and usage for a gateway node using the following heuristic: the energy a solar panel provides on the surface of the Earth for one day is approximately five times the maximum power output, P_{max}, observed from the solar panel [8]. Suppose that a gateway node has an average power draw of P_{avg}. Since the energy supply should be greater than the energy usage for one day, the following inequality holds: $P_{max} \times 5$ hours $\geq$ $P_{avg} \times 24$ hours.

Each of the two major components of a gateway node has an advertised maximum power draw of 5 W at 5 V. To support the total power draw P_{avg} of 10 W, P_{max} should be greater than or equal to 48 W. We chose a 50 W solar panel from Kyosera to supply power for the gateway node.

5.2 Supporting Scale Through Hierarchy

We deployed seven gateway nodes to support 557 Trio sensor nodes. Because the diameter of the network was larger than the 200 m range of the gateway nodes, 802.11 repeaters with higher-gain antennas were placed at strategic locations in the field. The backbone network required basic IP routing, and management was performed through the web consoles of both the Tmote Connect and the 802.3-to-802.11 bridge. Each gateway node was assigned an IP address on the same subnet as the access point. Any computer in the same subnet can receive data packets from a gateway node by running the TinyOS SerialForwarder tool. Packet streams from all of the gateway nodes are combined using a multiplexing version of SerialForwarder running on the root server. An application that needs to communicate with multiple gateways connects directly to the server's SerialForwarder, saving the application from maintaining multiple packet forwarding connections. This unified stream was also made available to processes running on the server itself, which we used to run the network management toolchain described in the next section.

6. TIER-3: THE ROOT SERVER

The Trio testbed consists of a total of 557 Trio nodes distributed over an area of approximately 50,000 square meters. The testbeds large scale and remote location makes it difficult to monitor the nodes directly and raises the need for remotely-accessible tools to manage the network. To this end, the Golden Image and the management framework that runs alongside testbed applications on the mote includes the Nucleus network management system, a second-generation version of SNMS [21].

6.1 Network Health Monitoring

In its most basic usage scenario, the Nucleus query system enables a testbed user to determine which nodes are running at any particular time. The Nucleus query server that runs on the root server provides an XML-RPC interface, and that interface is used by a monitoring daemon that periodically injects queries into the network, collects responses, and records statistics. The monitoring daemon tracks which nodes are running, which nodes had been running but have stopped responding, and which nodes have never run. The monitoring daemon also marks a node as awake if a gateway overhears a protocol message containing a source address, like Drain and Deluge. Passive health monitoring provides frequent data on the nodes near the gateways in between infrequent periodic queries. The daemon then provides this collected health information to a PHP-based web application which fuses this data with previously measured GPS coordinates for each node and produces real-time network health maps that can be accessed remotely. Our team found these maps to be invaluable during our experimentation because it allowed us to visually identify network problems. A typical network health map is shown in Figure 7.

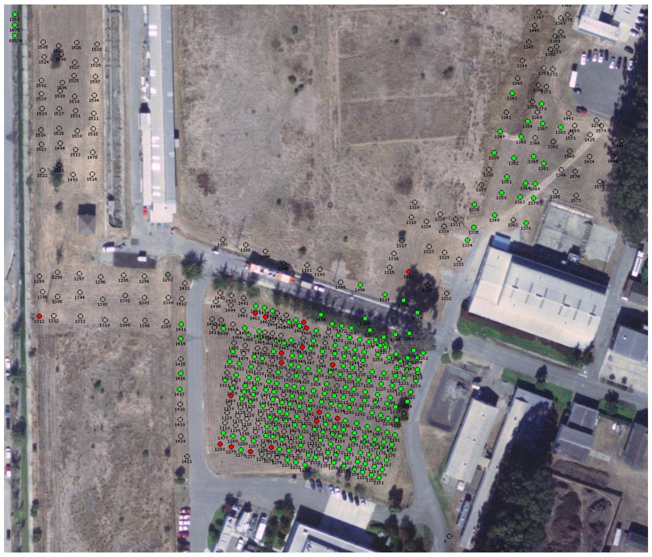


Figure 7: Nucleus Network Health Map. This map shows the health of the Trio nodes deployed at Richmond Field Station.

6.2 Power Monitoring

In addition to ensuring that the network is running, a user of the testbed should be able to verify that it is running sustainably. The monitoring daemon uses Nucleus to query the Prometheus logic running on each node, periodically collecting measured battery and capacitor voltage, along with flags indicating whether the node is charging its battery or running on it. This information is also displayed on the map provided by the web management console for online viewing, and is logged on the server for offline charting and analysis. Nucleus enabled us to study the behavior of Prometheus at scale, and Deluge enabled us to respond by installing fixes to the logic itself.

6.3 Monitoring Network Programming

The monitoring daemon also collects information from Deluge, enabling us to track the progress of an image through the network and visually identify nodes that are unable to acquire the image. Low battery voltages can prevent Deluge from writing data to the flash storage, which leads to low-voltage nodes requesting new data but never saving it. This Deluge "tension" can create hotspots of traffic within the network that impede the flow of application and management data. The health maps enabled us to identify the tense nodes and reboot them or simply shut them off. This tension also exposed a previously unknown failure mode for the epidemic protocols in use.

6.4 Monitoring and Control of Applications

The Nucleus management framework, or an alternate visibility and debugging system called PyTOS [24], can provide remote monitoring and control of an application running on the testbed. Even though executing a new application stops the Golden Image from running, maintaining the ability to query nodes and build health maps is highly desirable. The event detection application that we studied on this testbed included PyTOS, which the team working on that application found useful for exporting data and exposing functions for remote control. Though management traffic can conflict with an application being tested, perpetually available management is fundamental to the successful operation of our long-lived outdoor testbed.

7. TIER-4: CLIENT APPLICATIONS

We created this testbed for a large-scale study of multi-target tracking algorithms developed by our colleagues at UC Berkeley [16]. During the weeks leading up to the demonstration of these algorithms, our colleagues exercised the testbeds network reprogramming functionality by installing a new version of their application nearly every day, adhering to a familiar "deploy first, develop later" mentality that would usually require a wired testbed. The network management component was invaluable in ensuring that every node in their portion of the network was running prior to starting each experiment. On August 30th, Oh et al. demonstrated, in front of a large audience, real-time tracking of three people crossing paths through the center of the testbed field. The success of this application, and the subsequent four months of testbed operation, demonstrates that our system is functional and usable. This short section is included for completeness but for additional details about the application, we refer the interested reader to Oh et al. [15] and Chen et al. [5].

8. DISCUSSION

In this section, we highlight some of the insights and challenges that emerged during the development, deployment, operation, management, and maintenance of the Trio sensor network testbed.

8.1 Experiences with Renewable Energy

Renewable energy, in the form of a solar power supply, has been both the benediction and bane of this experience. Our most pleasant discovery was how renewable energy fundamentally simplifies system operation, management, and maintenance, enabling the familiar "deploy first, develop later" approach used with wired testbeds. Unfortunately, the dynamics of solar power and the logistics of node initialization raised many new concerns and exposed several previously unknown weaknesses in our network protocols and management strategies.

8.2 Limited Availability

Our testbed can be operated at 100% duty-cycle during only a few hours in the middle of day when direct sunlight is present. However, a duty-cycle ranging from 20% to 40%, depending on the time of year, allows continuous operation. This availability limitation stems from several factors: our own limited appreciation for subtleties of solar energy harvesting, the high power draw and lack of a low-power TinyOS MAC layer for the CC2420 radio, and several oversights in the design of the Prometheus solar energy harvesting system. We did not fully consider the many factors which affect solar energy harvesting. These factors include seasonal and daily variation in solar power, the angle of inclination of the solar cell, the effect of dirt and bird droppings on the solar cell, the importance of maximizing power transfer from the solar cell, and the policy surrounding energy transfer between the primary and secondary energy stores.

8.3 Emergency Battery Daemon

A consequence of limited availability – that is, operating with a power deficit – is that we cannot rely on the battery to supply enough power at all times. This realization led us adopt a policy that prevents Prometheus from switching to the battery automatically in times of low energy availability. Because Prometheus no longer switches automatically from capacitor to battery, we added a component called the Battery Daemon that enabled us to manually manage this switchover. The Battery Daemon uses Drip to disseminate a command that directs each node to acquire a short lease. While holding the lease, a node can switch to battery when either the capacitor voltage runs low or, with a different command, until the lease expires. Figure 8 shows a day in which the Battery Daemon was running.

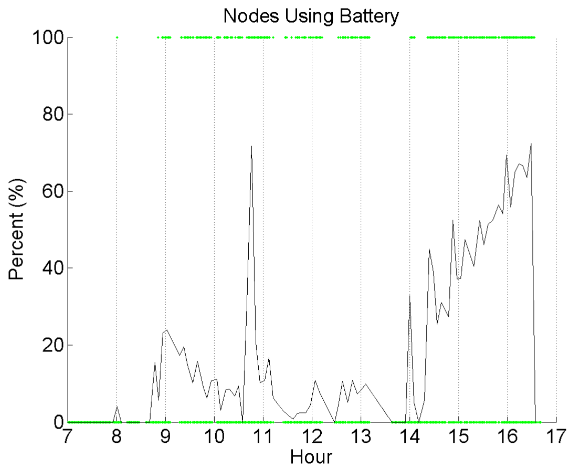


Figure 8: The proportion of nodes using the battery over the course of a day in which the Battery Daemon was running.

Note that the percentage of nodes using the battery rises in the afternoon from 0% at 13:30 hours to just under 70% at 16:30 hours, which corresponds with the setting sun. The Battery Daemon is turned off at 16:30 hours. This leasing mechanism allows us to extend the lifetime of the network while reducing our chances of accidentally draining the batteries, as would be the case if the charging logic automatically switched to the battery source anytime the capacitor voltage became depressed.

When the battery daemon is not active, partially-occluded sunlight causes the nodes to quickly exhaust their capacitors and go to sleep. These nodes then remain asleep for a longer time until their capacitors have been charged again. When running the epidemic algorithms in Deluge and Drip on top of these frequently rebooting nodes, we discovered new protocol failure modes.

8.4 Epidemic Protocol Failures

The Golden Image includes Deluge [10] and Drip [21], both of which use the Trickle algorithm [13]. In these protocols, one node can send an advertisement message that contains out-of-date metadata, which causes neighboring nodes to generate traffic in order to update the advertising node. When exercising the protocols at scale, we noticed that our unstable solar power supply led to nodes powering down, losing their saved metadata, and sending out-of-date advertisements when power was restored. During times of low or occluded sunlight, these reboots occured frequently enough that the excess update traffic created network hotspots. This excess traffic noticeably slowed down network programming time and disrupted network monitoring and management operations. The power instability present in this testbed has exposed several problems in our network and transport protocols that we would not have, and indeed had not, discovered on a stable indoor testbed.

8.5 Variability at Scale

Figure 8 illustrates significant variance across the nodes in their solar energy harvesting and an almost linear growth in the percentage of nodes using the battery in the afternoon from 0% at 13:30 hours to just under 70% at 16:30 hours. These results are surprising because all nodes in our testbed run the same software and are oriented in the same way with the solar cells facing south. Of course, some nodes forward more traffic than others but in these experiments, all nodes that were running had their radios turned on as well. The reason for the wide variance in the times at which nodes switched from capacitor to battery is unknown but we plan to further analyze the data to see if there are spatio-temporal patterns underlying these trends.

Several months after the initial deployment, and several days after letting the network remain unused, we tracked the battery and capacitor voltages over the course of one day. Figure 9 shows the mean and spread in battery voltages, and Figure 10 shows the mean and spread in capacitor voltages, recorded by the management software during this experiment. The cluster of solid lines before 08:00 hours at 3.7 V come from four powered nodes. The vertical gaps are the result of occasional failures of the packet forwarding daemons. Battery voltages below 2.5 V are anomalous readings, as are the arcing traces of capacitor voltages above 3.7 V, which we are unable to explain.

9. CONCLUSIONS

This paper presents our early experiences with a large-scale, long-lived, outdoor sensor network testbed. We deployed a network of 557 motes, seven gateways, two repeaters, and a root server over an area of approximately 50,000 square meters. This network was operational for the last four months of 2005. In the course of using and managing the testbed, we have discovered many subtle but serious flaws with the network management, power management, and networking software. While these flaws have severely limited the hour-to-hour availability of the testbed, they have not eliminated it altogether because the testbed itself supports fail-safe reprogramming of the support software.

Despite the many challenges with using a renewable energy source, we have found the experience liberating in many ways: even though the system can only operate at a 20% to 40% duty-cycle, we no longer worry about energy replenishment at a node level. The predictability of daily operation, cloudy winter days notwithstanding, coupled with the flexible yet fail-safe properties of the platform, have enabled us to use the untethered, outdoor testbed in ways we did not anticipate when we started this project. Several researchers have used the testbed to evaluate various aspects of sensor network operation. At the system programming level, the testbed was used to develop and evaluate a suite of programming tools. At the management level, we have used the deployment to evaluate network management tools. At

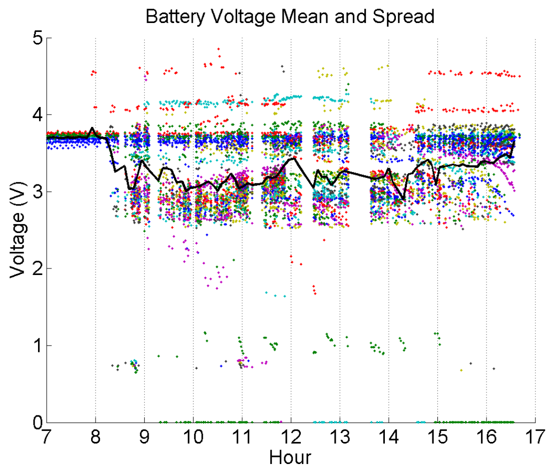


Figure 9: Battery voltages. Each dot represents a distinct sample.

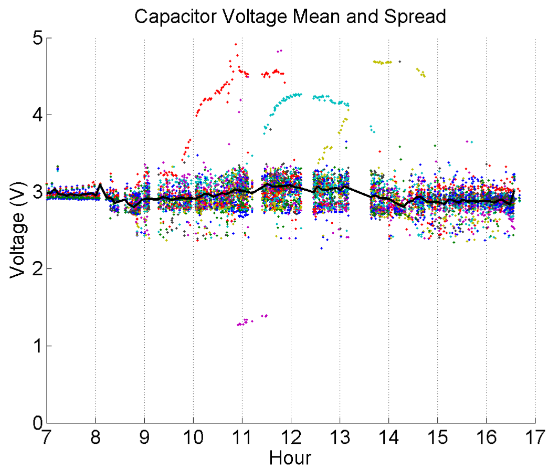


Figure 10: Capacitor voltages. Each dot represents a distinct sample.

the application level, intrusion detection, distributed target tracking, and multi-target tracking algorithms have been evaluated. At the middleware services level, the testbed has been used to evaluate collection, dissemination, and network programming algorithms. At the operating system and device driver level, we have explored several battery charging algorithms and many questions surrounding the firmware, kernel, and application boundaries.

Although we succeeded in executing the target-tracking application described earlier, we must now turn to improving the stability of the testbeds own management software. A wireless outdoor testbed depends on support software in a way that an indoor testbed with wired backchannels does not, and the demands on this software actually turn out to be quite complex. The Golden Image, normally considered to be a small "trusted" piece of code, includes environmentally-responsive energy management, an epidemic command protocol, an epidemic bulk dissemination protocol, an adaptive collection routing layer, a query system, and a storage manager for binary images – a full operating system. We have discovered unexpected problems with this large software stack, but the point remains: we can still program and manage the network, which means that these problems should be repairable without tearing it down and starting over.

We have been improving both the performance of the support software and the availability of the testbed itself through an iterative process with the eventual goal of enabling a stable hardware-software platform for sustainable and scalable operation. To this end, the future work is clear: the system software, and in particular, the TinyOS 802.15.4 radio stack, must be adapted to support low-power operation, the middleware services must be improved to better handle intermittent connectivity and fluctuating power, and the power management algorithms must work across a broad set of scenarios which arise in practice. A wireless outdoor testbed depends on system software and middleware services in a way that an indoor testbed with wired backchannels does not. These new dependencies have exposed weaknesses in existing software and have outlined promising new research directions.

10. ACKNOWLEDGMENTS

We would like to acknowledge the contributions of Warner Carlisle, Phoebus Chen, Eric Fraser, Mike Howard, Xiaofan Jiang, August Joki, Philip Levis, Mike Manzo, David Molnar, Gary Myers, Songhwai Oh, Joseph Polastre, Travis Pynn, Peter Ray, Tanya Roosta, Shankar Sastry, Bruno Sinopoli, Shawn Schaffert, Robert Szewczyk, Ray Woodfin, and Bonnie Zhu. This work was supported by DARPA under grant F33615-01-C1895 ("NEST"), California MICRO, and the National Science Foundation Graduate Research Fellowship Program.

11. REFERENCES

[1] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks*, 46(5):605–634, December 2004.

[2] A. Arora, R. Ramnath, E. Ertin, P. Sinha, S. Bapat, V. Naik, V. Kulathumani, H. Zhang, H. Cao, M. Sridharan, S. Kumar, N. Seddon, C. Anderson, N. Trivedi, C. Zhang, M. Nesterenko, R. Shah, S. Kulkarni, M. Aramugam, L. Wang, M. Gouda, Y. Choi, D. Culler, P. Dutta, C. Sharp, G. Tolle, M. Grimmer, B. Ferriera, and K. Parker. Exscal: Elements of an extreme scale wireless sensor network. *IEEE RTCSA*, 2005.

[3] S. Bapat, V. Kulathumani, and A. Arora. Analyzing the yield of exscal, a large-scale wireless sensor network experiment. In *Thirteenth IEEE International Conference on Network Protocols (ICNP 2005)*, 2005.

[4] P. Buonadonna, B. Chun, A. AuYoung, C. Ng, D. Parkes, J. Shneidman, A. C. Snoeren, and A. Vahdat. Mirage: A microeconomic resource allocation system for sensornet testbeds. *IEEE EmNetS-II*, 2005.

[5] P. Chen, S. Oh, M. Manzo, B. Sinopoli, C. Sharp, K. Whitehouse, G. Tolle, J. Jeong, P. Dutta, J. Hui, S. Shaffert, S. Kim, J. Taneja, B. Zhu, T. Roosta, M. Howard, D. Culler, , and S. Sastry. Experiments in instrumenting wireless sensor networks for real-time surveillance. In *International Conference on Robotics and Automation (video)*, 2006.

[6] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler. Design of a wireless sensor network platform for detecting rare, random and ephemeral events. *IEEE IPSN*, 2005.

[7] J. Elson, S. Bien, V. Bychkovskiy, A. Cerpa, D. Ganesan, L. Girod, B. Greenstein, T. Schoellhammer, T. Stathopoulos, and D. Estrin. Emstar: An environment for developing wireless embedded systems software. *UCLA CENS Technical Report No. 9*, 2003.

[8] D. Fradella. Building-integral on-site solar and wind power systems. `http://home.earthlink.net/~fradella/green.htm`.

[9] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Transactions on Sensor Networks*, 2005.

[10] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. *ACM Sensys*, 2004.

[11] X. Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. *IEEE SPOTS*, 2005.

[12] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. *In Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, 2003.

[13] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In *NSDI*, 2004.

[14] T. Liu, C. Sadler, P. Zhang, and M. Martonosi. Implementing software on resource-constrained mobile sensors: Experiences with Impala and ZebraNet. *ACM MobiSYS*, 2004.

[15] S. Oh, P. Chen, M. Manzo, and S. Sastry. Instrumenting wireless sensor networks for real-time surveillance. *In Proc. of the International Conference on Robotics and Automation*, 2006.

[16] S. Oh, L. Schenato, and S. Sastry. A hierarchical multiple-target tracking algorithm for sensor networks. *IEEE ICRA*, 2005.

[17] J. Polastre, R. Szewczyk, and D. Culler. Enabling ultra-low power wireless research. *IEEE SPOTS*, 2005.

[18] C. Sharp, S. Schaffert, A. Woo, N. Sastry, C. Karlof, S. Sastry, and D. Culler. Design and implementation of a sensor network system for vehicle tracking and autonomous interception. *IEEE EWSN*, 2005.

[19] V. Shnayder, M. Hempstead, B. rong Chen, G. Werner-Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. *In Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, 2004.

[20] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. *ACM Sensys*, 2004.

[21] G. Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. *IEEE EWSN*, 2005.

[22] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. *ACM Sensys*, 2005.

[23] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: A wireless sensor network testbed. *IEEE SPOTS*, 2005.

[24] K. Whitehouse, G. Tolle, J. Taneja, C. Sharp, S. Kim, J. Jeong, J. Hui, P. Dutta, and D. Culler. Marionette: Using rpc for interactive development and debugging of wireless, embedded networks. In *IPSN/SPOTS 2006*, 2006.