2.8 Programming and Platforms (PRO)

CENS systems research strives to advance the state of the art in scalable, distributed observing systems. To this end, our research efforts have focused on two critical areas: the design and evaluation of architectures and programming systems, and the development of practical tools and platforms. Together, these two research directions will eventually (and, in some cases, already) enable the Center to field sophisticated, rapidly reconfigurable, multi-user observing systems that support advanced sensing modalities.

Architecture and Programming Systems

The effort to deploy a long-running reliable ENS observing system is currently perhaps an order of magnitude more than that of writing the application itself. Our ongoing research will change this: by developing visibility into a deployed system, by developing languages and tools that enable robust application development, and by re-architecting ENS software to ensure the development of manageable software, our research will enable nimble and sustainable observing systems.

Visibility: Observing the internal execution of severely resource constrained wireless embedded devices remains a critical block to widespread adoption of embedded wireless sensing systems. Our LowLog system obtains semantically dense execution logs, provides triggered retrieval of these logs, and supports correlation of logs to help diagnose problems. LowLog automates and makes readily available standard logging techniques, such as inserting an arbitrary logging preamble to functions to log. At the same time, it enables exploring non-traditional logging techniques such as caller side logging, capturing compressed runtime control flow decision logs, and the impact of alternate log token encoding mechanisms.

Robust development: The failure of sensing device software can sometimes have life-threatening consequences. Medical monitors, or sensors that monitor the infrastructure, must be certifiably robust. Our Virgil project aims to develop programming language tools and software that can certify the robustness of sensing devices and software. The project is developing a domain-specific language that will encourage design for certifiability and make certification easier, and domain-specific tools for certifying the four key properties of space bounds, soft-real-time response, life time, and meaningful results. The certification tools will increase our confidence in sensing devices and help decrease the scope, duration, and cost of the testing effort.

Software architecture: Sensor network programmers must cope with a variety of concerns, including severe resource and energy constraints, consistency and synchronization among nodes, and node failure. Programmers currently address these concerns by interweaving the code for these concerns together with application logic. Separation of concerns will enable these different constraints to be decoupled from application logic, and will allow applications to more easily cope with changes in the environment, or new functional or performance requirements. Building upon our prior work on the Pleaides programming language, we have been exploring new language mechanisms that enable application programmers to specify these concerns, and new compiler and runtime algorithms that address these concerns in an automated fashion. Our initial focus has been on robustness to, and recovery from, node failure.

Finally, our Tenet project revisits the architectural foundations of the sensor network systems built and deployed by CENS. The project has made considerable progress in the past year, having developed a robust duty-cycling mechanism called AEM that allows duty cycles of 1 to 3%, while maintaining the generality of Tenet. It also conducted a 3-month long Cyclops deployment at the James reserve, enabling biologists to observe avian breeding patterns over a relatively long time period at fine spatiotemporal scales.

Tools and Platforms for Observing Systems

A second systems research thrust is the development of mature tools and platforms for deployments of observing systems. Our efforts in this thrust revolve around three issues: low power, rapid deployment, and time services.

In Low-power Energy Aware Processing (LEAP) we have created platform support for deep, accurate, runtime energy consumption information. Building upon our previous years’ research on fine-grained energy accounting instrumentation in embedded systems, we have now scaled it to desktops as well as datacenter servers with
emphasis on multi-core processors. We also fuse these direct measurements with indirect information obtained from a performance counter based behavioral model to provide unprecedented visibility of per-process CPU and RAM energy consumption information on multi-core systems. Evaluation with carefully designed experiments shows that our system is able to provide per-process energy information with an accuracy of at least 96%.

The GeoNet platform (see 2.03 Seismic; SEI 01) for rapid and distributed geophysical sensing builds upon the LEAP technology. It enables a rapidly installable wirelessly linked seismic network to measure earthquake or volcano sources in the near field to understand the underlying physics, or in buildings to understand earthquake damage. During this second year of the effort we partnered with one of the leading manufacturers of seismic recording systems, Refraction Technology, Dallas Texas, or Reftek, to construct several prototypes. We used the Mexico and Peru networks based on our older platform technology as testbeds for development of GeoNet systems software targeted at the seismological community. The software advances included an improved Disruption Tolerant Shell (DTS), measurement of radio link quality (ETT), network logging, an embedded web interface based on Emstar for deployment and maintenance, network timing, and a new routing protocol that caches the routes across sleep cycles for a fast startup.

Crucial for platforms such as GeoNet is high quality time information, which is the topic of our final activity, Time Synchronization–Take 2. We have explored two new directions. First, we have developed a new method of post facto time synchronization for broadband seismic arrays called data driven time synchronization (DDTS). We are implementing DDTS for seismic networks using microseisms as the underlying characteristic for synchronization and applying it to time offset data from the joint CENS and CalTech Meso American Subduction Experiment (MASE). Second, we have developed a new type of local clock source called Crystal Compensated Timer (XCT) that has frequency stability characteristics similar to timers based on temperature compensated crystal oscillators (TCXO) but is much cheaper and less power hungry because of its digital nature, and thus promises to enable hitherto impossible to achieve duty cycling and time synchronization capabilities. In collaboration with researchers at Berkeley, we are incorporating this technology into the new Quanto Testbed Mote platform with the iCount fine-grained energy monitoring software.
PRO 01 Tenet: Architecture for Tiered Embedded Networks

PRO 01.1 People

- Principal Investigators: Ramesh Govindan, Eddie Kohler, Deborah Estrin
- Faculty: Karen Chandler (UCLA, CENS) Deborah Estrin (Dept. of Computer Science UCLA) Ramesh Govindan (Dept. of Computer Science, USC), Eddie Kohler (Dept. of Computer Science UCLA)
- Researchers: Vinayak Naik (UCLA, CENS)
- Graduate Students: Omprakash Gnawali (Dept. of Computer Science, USC), Ki-Young Jang, (Dept. of Computer Science, USC), Jeongyeup Paek (Dept. of Computer Science, USC), Marcos Vieira (Dept. of Computer Science, USC)

PRO 01.2 Overview

This project revisits the architectural foundations of the sensor network systems built and deployed by CENS. Sensor network researchers originally envisioned large networks of tiny wireless nodes with simple gateways to the Internet. Because of energy and network constraints, the tiny nodes themselves would collaboratively process data in-network in complex, application-specific ways. In reality, though, the sensor networks we deploy mostly perform continuous data acquisition, and incorporate little or no on-mote multi-node data fusion. We believe that two factors can explain this development. First, the introduction of masters: 32-bit CPU-class nodes for which power can be engineered. Masters are an integral part of every network we deploy, and rather than acting as mere Internet gateways, they participate in the functionality of the network. (Yet our current architectural principles take little advantage of them!) Second, the unexpectedly high complexity of mote-based multi-node data fusion makes implementing such functionality a bad tradeoff. By not optimizing for the system as a whole, we are missing the opportunity/have overlooked to build a software architecture that promotes on-board mote processing of its local time series in an adaptive and efficient manner.

Tiered data-collection networks are therefore here to stay. Unfortunately, only the original architectural principles are available to sensor network designers. When designers follow those principles the resulting systems are fragile and overly complex. Even worse, they are difficult to repurpose; and the master nodes on which network health depends remain underutilized.

An architecture is needed to guide the construction of scalable, evolvable, and replicable sensor systems that will serve the vast array of applications currently awaiting deployment. The Tenet project is developing such an architecture.

PRO 01.3 Approach

Many current large-scale sensor network deployments are tiered. The lower-tier, composed of motes, contains sensing and actuation functionality and enables infrastructure-less instrumentation of physical spaces and artifacts. The upper-tier, consisting of 32-bit nodes, masters, is free of energy constraints and provides increased network and computational capacity, enabling large-scale deployments.

The Tenet architecture prescribes a functional separation between motes and masters, with the goal of reducing overall system complexity. The architecture asserts that it is still desirable for the motes, which should be optimized for low power operation, to do local aggregation, compression, and even filtering of its time series data. However, cross node aggregation, filtering, and processing is best done by the master.

Fundamentally, Tenet constrains collaborative multi-node in-network processing to be performed on the master nodes. In-network aggregation and fusion is inherently somewhat centralized in that data from multiple nodes is sent to a common node to be processed. The masters in a tiered architecture are natural fusion and aggregation points from the perspective of capacity (CPU, storage, bandwidth and energy). Constraining aggregation to be performed at masters results in a simpler architecture relative to one that allows aggregation on topologically
convenient, but resource-constrained motes. In Tenet, motes are tasked by applications running on masters, and can implement simple logical elements such as thresholds and compression, but any further computation takes place only on masters. Finally, masters can collaborate with one another to implement distributed applications for tracking, detecting spatio-temporal events, or (as in this proposal) multi-robot coordination (Figure 1).

PRO 01.4 System(s) Description and/or Experiments
The current Tenet system consists of five components: a tasking library which supports the execution of small program fragments called tasks; a routing subsystem which uses a multi-sink version of a standard tree routing protocol for routing data from motes to masters; a task dissemination subsystem which ensures the reliable delivery of tasks from any master to all the motes, a transport subsystem which provides end-to-end reliable transmission of sensor data from motes to masters, and a time synchronization component which ensures that all the motes maintain a globally synchronized time.

Recently we have designed a sixth component: Application-informed Energy Management (AEM) to duty-cycle the radios of sensor nodes running Tenet. AEM performs static analysis of Tenet tasks to infer application workload in manner transparent to the application. AEM then sets up and coordinates network-wide radio duty-cycling tailored to the application workload and the expected traffic pattern within the network, using parameters derived from

Figure 1: A Tiered Sensor Network

Figure 4: AEM and LPL duty-cycles with various workloads.

Figure 5: Data delivery latency with different workloads.
static analysis. The number and duration of data schedules, the times during which the radio is on for transmission and reception of data packets in the network, is scaled in proportion to the application workload. Moreover, the duration of radio on-times is variable and elastic, enabling AEM to easily adapt to load transients and retransmissions. Our comparison of AEM’s performance with that of LPL, the state-of-the-art MAC-based duty-cycling system applicable to Tenet, shows that AEM achieves up to 15% lower duty-cycle than with LPL with the range of workloads we examined.

PRO 01.5 Accomplishments

Software:
We released the second version of Tenet (Tenet 2.0) software in June 2008. This release includes robust version of many experimental features we developed in the past. The instruction to download and use the software can be found at http://enl.usc.edu/software.html.
Tenet 2.0 includes support for Centroute as an alternative routing protocol. It includes support for Cyclops for imaging applications. Tenet 2.0 also includes experimental support for TinyOS 2.1 and the ability to duty-cycle the radio to save energy.

Deployment:
In the past, we have implemented and deployed Pursuit Evasion Game and seismic monitoring network on a suspension bridge.
Most recently, we have extended Pursuit Evasion Game into a multi-agent pursuit Evasion Game in which multiple robotic pursuers collectively determine the location of multiple evaders, and try to corral them.
We also deployed a 19-node image sensing network based on Cyclops platform to monitor bird nests in an area of 0.05 square miles. Our three-month long deployment at the James San Jacinto Mountain Reserve resulted in the collection of over 102000 images. Our biologist users found the on-line, near-real-time access to images to be useful for obtaining data on the nesting behavior of bird species.

PRO 01.6 Future Directions
Our work for next year will be focused on deployment of Tenet as well as evolution of the architecture and its components to allow Tenet to be used in a variety of settings.
We plan to deploy Tenet for seismic sensing of buildings using the imote-2 based platform. We are currently porting Tenet to this platform and designing the software to manage and coordinate the deployments.
To enable energy-efficient deployments of Tenet, we designed AEM to duty-cycle the radio. Although we

Figure 6: System architecture and testing scenario of multi-agent Pursuit Evasion

Figure 7: Two Pioneer robots in pursuit of evaders. The Pioneer robots use Tenet to task the sensor network to obtain information to localize the evaders.
have experimented with AEM in the in-door laboratory setting, we have not yet deployed Tenet with AEM in the field. We plan to take that step and use the lessons learnt to further improve the design of AEM.

The current release of Tenet schedules tasks at a granularity of a single tasklet. We plan to implement Tenet using the threads primitive that has recently become available in TinyOS. Using threads will allow Tenet to be robust even in applications with long running CPU intensive tasks.

Figure 8: Bird nest and image sensing hardware organization.

Figure 9: Bird nest image sensing network topology.

PRO 01.7 Other Invited Presentations
PRO 02 Low Power Energy Aware Processing (LEAP)

PRO 02.1 People
- Principal Investigators: William Kaiser
- Faculty: William Kaiser, Electrical Engineering, UCLA
- Researchers: Thanos Stathopoulos, Electrical Engineering, UCLA
- Graduate Students: Sebi Ryffel, ECE Dept, ETHZ, Dustin McIntire, Electrical Engineering, UCLA

PRO 02.2 Overview
The LEAP project aims to provide deep, accurate, runtime energy information on a range of computing platforms, including embedded systems, desktops as well as datacenter servers. In the past, LEAP has focused mainly on 32-bit embedded systems. This year, LEAP research focused on desktop and server class systems, with emphasis on multi-core systems. The goal of the LEAP project remains the accurate, runtime energy accounting of hardware and software entities of a computing system, for the purposes of energy attribution and optimization.

PRO 02.3 Approach
We investigate the problem of energy attribution and accounting for multi-core systems. We introduce a hardware assisted direct energy measurement system that integrates seamlessly with the host platform and provides detailed energy information of multiple hardware elements at millisecond-scale time resolution. We also introduce a performance counter based behavioral model that provides indirect information on the proportional energy consumption of concurrently executing processes in the system. We fuse the direct and indirect measurement information into a low-overhead kernel-based energy apportion and accounting software system that provides unprecedented visibility of per-process CPU and RAM energy consumption information on multi-core systems.

PRO 02.4 System(s) Description and/or Experiments
Our overall architecture is based on three components: the Runtime Direct Energy Measurement System (RTDEMS), the performance counter-based indirect energy measurement model and the Linux Energy Apportioning and Accounting Platform (LEA2P) that fuses data from RTDEMS and the model-based system to provide per-process energy information for arbitrary processes in multi-core systems.

The Runtime Direct Energy Measurement System (RTDEMS) is the adoption of the embedded low power energy-aware processing (LEAP) project to desktop and server class systems. RTDEMS differs from previous desktop class energy measurement approaches in that it provides both real-time power consumption information and a standard application execution environment on the same platform. As a result, RTDEMS eliminates the need for synchronization between the device under test and an external power measurement unit. Moreover, RTDEMS provides power information of individual subsystems, such as CPU, GPU and RAM, through direct measurement, thereby enabling accurate assessments of software and hardware effects on the power behavior of individual components. Figure 1 presents a hardware diagram of RTDEMS.

Figure 1: Hardware diagram of the Real-Time Direct Energy Measurement System.

2009 Annual Report
Center for Embedded Networked Sensing

2.8 Programming and Platforms
necessary, are by themselves insufficient to resolve the per-process energy attribution problem in multi-core systems; additional, indirect energy information is required. We argue that performance counters are suitable indirect indicators for energy apportion. Performance counters are available on most modern processors and can be accessed without incurring significant overhead. Additionally, performance events can be counted for each core separately and can therefore measure the behavior of each core individually, thus providing the additional visibility that our direct measurement system lacks. Finally, previous work has shown performance events to be good indicators for energy usage. We note that we do not use performance counters to estimate total energy consumption as our RTDEMS measurement system provides us with direct and accurate measurements. Rather, after acquiring the total energy consumption through RTDEMS, we use performance counters to solve the energy apportion problem. We also use performance counters as indicators for SDRAM energy consumption, in addition to CPU energy consumption.

We combine the results of RTDEMS and the performance counter model-based indirect energy measurement approach into a unified energy apportioning and accounting system. The system is able to resolve, at runtime, the energy apportioning of individual concurrently executing processes in a multi-core system, for the two main computational components, CPU and SDRAM. The software architecture of our system is shown in Figure 2.

![Figure 2: The Energy Apportioning And Accounting System Architecture.](image)

**PRO 02.5 Accomplishments**

To test the validity of our results, we designed experiments for which we are able to assert a particular apportion. We then compare the asserted values with the solution found by our online algorithm. We chose a sequential memory access benchmark as our test program, as it allows us to test SDRAM as well as CPU energy apportioning, by controlling the number of memory accesses. We also chose a memory buffer of 512MB in order to minimize the impact of the CPU's cache management, which is beyond our control. By executing two instances of the memory benchmark, A and B concurrently on two different cores and by controlling the number of accesses over the memory buffer, we assert the energy apportioning to be proportional to the number of memory accesses performed by the two processes. For example, if process A accesses the memory buffer once while process B...
accesses it twice, we assert that the correct memory energy attribution would be 33% for process A and 66% for process B. We note that this assertion holds for our benchmark because the type and locality of memory accesses is the same for both processes, as opposed to arbitrary processes and tasks, where neither type nor locality can be known in advance. As Figure 3 shows, our system is able to provide per-process energy information with an accuracy of at least 96%.

**PRO 02.6 Future Directions**

We plan to extend our system to account energy usage of other components such as hard drives and network cards, which requires the design and implementation of suitable energy models. Furthermore, we aim to replace the initial calibration phase necessary for model learning with an online model learning system. We also plan to explore alternative accounting systems such as per-activity accounting. We also intend to build on our resource container implementation and provide a more powerful interface for container manipulation to user-space applications.

![Figure 3: Asserted and measured CPU and SDRAM energy of two tasks A and B.](image)
PRO 03 Time Synchronization - Take 2

PRO 03.1 People
- Principal Investigators: Mani Srivastava
- Faculty: Deborah Estrin, Computer Science, UCLA, Mani B. Srivastava, Electrical Engineering, UCLA.
- Graduate Students: Martin Lukac, Computer Science, UCLA, Thomas Schmid, Electrical Engineering, UCLA.

PRO 03.2 Overview
The quality of time information at a node is affected by many factors: jitter in computational and communication latencies, time interval between resynchronizations, accuracy of time stamping network wireless packets, and quality of local clock source (quantization, frequency tolerance, aging, and drift). Much of the early work on time synchronization in sensor networks (including our own work at CENS) focused on the design of time synchronization protocols and regression mechanisms for coping with noisy time stamping of packets used during synchronization.

Improvements of the local clock source itself were left out as it was taken as a given that anything with better drift characteristics than a simple crystal oscillator would add too much cost to the hardware, and utilize too much energy. This is despite the importance of drift as a source of error. Drift impacts time accuracy because of drift since the last resynchronization, change in drift due to temperature gradients, time taken to resynchronize subsequent to long periods of sleep, and drift during the message exchange used for measuring clock offsets between neighboring nodes. The first three factors are important in any system, while the last factor is important primarily in systems using slower acoustic signals for communication (e.g. underwater) or in systems needing very high accuracy knowledge of time (sub-microseconds).

Other aspects left out of the early work on time synchronization protocols were revealed through experiences deploying research systems such as our seismic and acoustic arrays. The time synchronization protocols worked but when faced with system failures beyond the control of the software system (unresponsive GPS, bad network connectivity, and hardware errors) the temporal integrity of the data suffers: a node's notion of time diverges from reality, leading to inaccurate time stamping of events and measurement of time intervals. Maintaining temporal integrity is critical in these systems to correlate events across the network and obtain scientific results.

PRO 03.3 Approach
In the first part of this project, we developed a new type of local clock source called Crystal Compensated Timer (XCT). The developed solution has frequency stability characteristics similar to timers based on temperature compensated crystal oscillators (TCXO) but has the potential to be much cheaper and less power hungry because of its digital nature. These are the ideal characteristics needed for wireless sensor networks, where energy is scarce, the components have to be cheap, but time still has to be measured with high accuracy. The basic idea behind the XCT is simple. It exploits the manufacturing imprecision in the production of AT-cut crystal oscillators resulting in different angles at which crystals are cut. Alternatively, crystals from different manufacturers are often cut at different angles.

By measuring the imprecision between frequencies two crystals at the same time in a pre-calibration step, a simple algorithm can at run-time compensate for the drift induced by temperature on the crystals. This is a concept we call Differential Drift.

Having this technology, we will now study fundamental limits of time synchronization. We will specifically look at the clock as a source of error, and the requirements on that clock accuracy and power consumption in order to achieve a certain time synchronization accuracy. As a simple example, an early result shows that for two systems, A and B, with different clock technologies, for system B to be more energy efficient than system A, the clock system used in system B must use less power ($P_{eb}$) than the active power consumption of system A ($P_{a}$), multiplied by twice the precision of system A's clock stability ($s_{a}$), or: $P_{eb} < 2*P_{a}*s_{a}$.
With the insights gained from these theoretic investigations, we will develop a new time synchronization protocol, that takes into account the different clock sources existing in a heterogeneous sensor network. We will accomplish this with the help of two tools, a simulator that includes an accurate clock model, and a new testbed that will allow us to verify the simulation results. Both tools are currently under development, and we will go into some more details about them in Section 3.

In the second part of our project, we have developed a new method of post factor time synchronization for broadband seismic arrays called data driven time synchronization (DDTS). DDTS uses underlying characteristics of the data to provide time synchronization. These characteristics must be independent of the characteristics, features, and events used to obtain science results from the data. To apply DDTS there needs to be a model of the underlying characteristic that can provide correlations across the network. Our initial exploration focuses on applying DDTS to seismic data.

**PRO 03.4 System(s) Description and/or Experiments**

We are currently developing two tools that will help us refine the research results and verify our theoretic analysis. The first tool is a simulator called Open Castalia. Open Castalia is based on the sensor network simulator Castalia. The major changes to Castalia are a new clocking subsystem that accurately simulates different behavior of clocks over time, like quantization, frequency drift due to temperature changes, etc. This simulator will play a crucial role in developing the next generation of time synchronization protocols, since it accurately models errors that were so far ignored as being too small. However, if one starts to push the limit of synchronization accuracy, these errors start to become significant, and one can not ignore them anymore.

The second tool is a new testbed based on the Quanto Testbed Mote platform. The Quanto Testbed Mote allows to measure its power consumption using a technology called iCount. Our goal is to develop infrastructure software that can simply extract these power consumptions from a larger deployment of nodes, and then calculates the isolated power consumption of an individual distributed service, like time synchronization. This infrastructure software is called the Managed Node Interface (MNI) for the Quanto Testbed at NESL.

We are implementing DDTS for seismic networks using microseisms as the underlying characteristic for synchronization and applying it to time offset data from the joint CENS and CalTech Meso American Subduction Experiment (MASE). Microseisms are wave energy that is generated by the ocean and travel through the crust across the ocean floor and continents. They are considered background noise and are filtered out of most seismic data. Microseisms can appear in the 0.03 to 0.3 Hz frequency range, requiring the use of broadband seismometers. Microseisms are generated by opposing oceanic surface waves: the interaction of opposing waves creates enough
pressure on the ocean floor to generate seismic waves. The microseism period is dependent on the ocean depth, the surface wave period generated by the wind, and the interference of all the points in the ocean generating the seismic waves. There are a number of frequencies that microseisms typically exhibit due to the interaction of the waves. Figure 2 shows an frequency spectrum of 17 minutes of MASE data (100Hz data filtered and decimated to 1Hz). The approximate 6 second period energy in the signal is clear. Figure 3 also shows the waveform for 60 seconds of 100Hz MASE data, both unfiltered and filtered. The approximate 6 second period microseisms is clearly visible in the filtered data. In the MASE data, the dominant period of the microseism energy traveling north

Figure 2: A spectrogram of 17 minutes of MASE data and 60 seconds of filtered and raw data. The 6 second microseisms are visible in both the time and frequency domains.

Figure 3: Travel times for April and May 2006 for four station pairs. The *'s represent the calculated travel times and the lines are the travel times generated by our Model. The RMSE for each station pair is approximately 0.05 seconds.
through the array is 6 seconds, while the dominant period of the microseism energy traveling south through the array is approximately 20 seconds.

To apply DDTS we have developed a model of microseism propagation through our seismic array. We can compute the travel time of microseisms between any two MASE stations using cross correlation. Our model is based on a new observation: the travel time over a 24 hour period between pairs of stations fluctuates by up to two seconds and these fluctuations are correlated across independent pairs of stations in the network. This suggest that there is a common bias in the arriving energy and it implies the bias in the sources of the microseisms is in the far-field because it is common-mode across the array. Our model can be used to predict these fluctuations in the travel time for a station with offset time and we use this prediction to obtain a time correction for the offset data. Figure 3 shows the travel times for April and May 2006 for four independent pairs of stations; the correlation of the fluctuation of the microseism travel times are clear.

To test and evaluate applying DDTS to seismic data we use real data from the MASE deployment and introduce time offsets and drift. By applying the method to the data to which we introduced faults we can evaluate how well our method can correct the data. We also evaluate how time offset errors affect science results such as local earthquake localization. And finally we apply the method to data that has time offsets and drift from the MASE deployment.

**PRO 03.5 Accomplishments**

In earlier exploratory work we have studied the viability of our dual crystal oscillators in lab-scale experiments using a controlled temperature chamber and a modified TMote with two crystal oscillators. Our initial results are very encouraging, and indicate that we can achieve frequency stability of around 0.31 ppm over -10 degree C to 60 degree C at < 0.5 mW power consumption. Simulations using 3 years of temperature data from MossCam at James Reserve suggest that this approach can reduce accumulated drift by 30X from around 1000 s using uncompensated crystal oscillator to < 3 s using our compensation approach. This is significantly better performance at a lower power and cost (rough estimates) than commercial TCXOs (which cost > $50 for 1 ppm performance).

We have successfully applied DDTS to correct seismic data. Figure 3 show the measured microseism travel times for four stations pairs and the travel time generated by the model. The RMSE for each station pair is approximately 0.05 seconds. Figure 4 shows the correlograms between the two stations CASA and ZACA for each data between 09/01/2005 and 10/31/2005. The peak of the correlogram represents the travel time between the two stations. The station ZACA experienced a time offset and drift towards the end of September as is evident in the

![Figure 4: The daily correlograms for 09/05 and 10/05 between ZACA and CASA. The shift in the peaks of the correlogram indicates data which has a time offset.](image-url)
correlograms being shifted to the left approximately 265 seconds. Figure 5 shows the correlograms after we apply the time correction using DDTS: all the correlograms line up and the travel time is now consistent.

![Figure 5: The same data and time as Figure 4, but after the time corrections using DDTS have been applied.](image)

**PRO 03.6 Future Directions**

Our plan is to develop a new time synchronization protocol based on the insights we gained from our theoretic analysis. The protocol will be inspired by IEEE 1588, the precision time synchronization protocols for wired networks, but will be adapted to the requirements of low-power embedded systems.

We can currently repair the time to about 0.1 seconds. We hope to reduce this further and attempt to remove limitations of our method by exploring other methods of processing the microseism data, including using code of the correlation and the correlation of the code of correlations.

We are also working on correlating the fluctuations in daily microseism travel time with the weather using the data and the WaveWatch III from the National Centers for Environmental Prediction. Correlating the fluctuations in microseisms with the weather can help us identify the sources of microseisms and potentially provide more accurate time corrections.
PRO 04 Virgil: towards certified sensor nodes

PRO 04.1 People

- Principal Investigator Jens Palsberg
- Xiaoli Gong, Graduate Student, Computer Science, UCLA
- Kannan Goundan (funded by CENS), Graduate Student, Computer Science, UCLA
- Jens Palsberg, Professor, Computer Science, UCLA

PRO 04.2 Overview

A medical device should not crash or confuse. A device crash can be anything from inconvenient to life threatening, while confusing device behavior can lead a user to draw an incorrect medical conclusion. We envision a certification tool that can meet challenges related to space bounds, soft-real-time response, life time, and meaningful results. We aim for both fundamental advances in programming language design and static error checking, as well as progress on how to do applications programming for medical monitoring devices. Our goal is to take a major step towards design for certifiability and to bring closer the day when the FDA will use static error checking tools frequently and routinely. A medical monitoring device collects data from the body, carries out local computation, and sends data to an external computer. Together, the device and the external computer form a small sensor network with a few sensors and one base station. We are in the early stages of a NSF-funded collaboration with Majid Sarrafzadeh at UCLA whose group has built software for four monitoring devices that were then tested by doctors and patients in the UCLA Medical School. Majid's devices monitor such things as pressure changes in the upper urinary tract, myotatic stretch reflex, neurological disorder, and diabetic foot ulcer (see Figure 1). The devices are small and the software is typically on the order of a few thousand lines of source code. Eventually we want to be able to certify the software in Majid's four devices.

![Figure 1: A device for monitoring diabetic foot ulcer.](image-url)

PRO 04.3 Approach

Our goal is to develop

- a domain-specific language that will encourage design for certifiability and make certification easier, and
- domain-specific tools for certifying the four key properties of space bounds, soft-real-time response, life time, and meaningful results.

In particular, we want the tools to do static error checking, that is, certify the software without running the software. The certification tools will guarantee that certain problems cannot occur; and rigorous testing can then focus on problems that were left unaddressed by the certification tools. The certification tools will increase our confidence in medical devices and help decrease the scope, duration, and cost of the testing effort.

PRO 04.4 System(s) Description and/or Experiments
Our own language Virgil is our starting point for designing a new domain-specific language. Virgil is a statically-typed, object-oriented language in the tradition of C++, Java, and C#. Virgil is designed specifically for high-level, type-safe systems programming, including programming of device drivers for sensors, radios, timers, analog-to-digital converters, etc., and is targeted to run on tiny devices such as sensor nodes.

PRO 04.5 Accomplishments
We have written drivers in Virgil for all the Mica2 devices. The final driver that we completed was the radio device driver which unsurprisingly turned out to be a major challenge. As a result, we now have the entire base functionality of TinyOS written in Virgil. Software can now control a sensor node using Virgil code alone. We have also studied approaches to extending Virgil from being a language for programming single nodes to become a language for programming an entire network of sensor nodes. We are focusing on X10, an object-oriented language from IBM. The X10 language contains two key constructs for concurrent programming called async and finish that we believe can be valuable for programming an entire sensor network. In collaboration with two IBM researchers, we have published a paper in OOPSLA 2008 on the X10 type system that we hope will be directly applicable to our language design and software certification effort.

PRO 04.6 Future Directions
We will continue our work on porting the software in Majid’s four devices from NesC to Virgil.
We will investigate how to implement the functionality of SOS in an extension of Virgil. We will try to extend Virgil to support programming of an entire sensor network. We will implement tools for certifying key properties of Virgil programs.

PRO 04.7 Other Invited Presentations
PRO 05 Seeing the Forest Through the Trees: Distributed Logging for Distributed Systems

PRO 05.1 People

- Principal Investigator: Mani Srivastava, Todd Millstein
- Todd Millstein, Assistant Computer Science, UCLA, Mani Srivastava, Electrical Engineering, UCLA.
- Roy Shea, Graduate Student, Computer Science, UCLA.

PRO 05.2 Overview
Observing the internal execution of severely resource constrained wireless embedded devices remains a critical block to widespread adoption of embedded wireless sensing systems. Exposing this execution is critical for enabling system developers to debug and understand the systems that they deploy. Past CENS projects have provided initial steps to minimize the occurrence of and diagnose instances of bugs in such systems:

- Kairos's centralized program abstraction enabled writing a single clear and maintainable program for an entire network of devices rather than requiring developers to manually implementing tricky distributed logic.
- Lighthouse's static analysis isolated resource sharing problems before system deployment preventing a class of problems commonly found on embedded systems.
- Harbor's software isolation prevented runtime memory faults on embedded devices without MMUs, which have a long history of runtime memory faults.
- Sympathy's runtime heartbeats carrying node centric diagnostic information helped diagnose many network level faults within the class of embedded wireless sensing systems.

Our current work continues by providing detailed insight into the runtime operations of these bottom tier systems. We are working to provide this insight through three core areas of innovation for this class of systems: improving the ease of obtaining high quality log data, developing novel mechanisms for triggering log retrieval from within a deployed or testbed network, and minimizing the overhead of log correlation.

PRO 05.3 Approach

Our approach focuses on three core tasks: obtaining semantically dense logs, exploiting triggered log retrieval, and building exploring new log correlation techniques.

Inexpensive storage media provides traditional server systems with the luxury of logging first and looking for meaningful data within the logs later. But on resource constrained wireless and embedded sensing system, where both memory and stable storage are limited, there is only room to log the highest quality data. Our research creates high quality logs by combining static program analysis with automated insertion of logging statements. Static analysis helps optimize away redundant logging while ensuring complete log coverage over specified regions of interest. For example, we've developed a domain specific compression technique that captures complete and compact runtime call traces by only logging runtime control flow decisions affecting call dispatch. Automated log insertion removes the burden of, and inevitable mistakes introduced from, manual code instrumentation. We measure the success of this work by comparing the runtime memory usage, transmitted log size, and execution impact of our improved logging mechanisms to the current logging alternates.

Triggered log retrieval strives to collect logs only when they will provided the greatest utility. The key to providing system users with this ability is developing a clear language within which to describe the conditions, perhaps distributed across multiple devices, when a particular type of log is of value. Declarative and macroprogramming approaches are two promising areas that provide foundation work that we plan to apply to the specification of triggers.
Our research targets networks of embedded devices, so it is critical that our resulting infrastructure includes the ability to correlate logs originating from more than one source. To reduce both the bandwidth consumed by log transfer and the delay in identifying events of interest, we are exploring techniques to push log correlation deeper into the network. We hope to reduce correlation overhead to such a degree that it can be used directly as an expressive triggering mechanism for gathering logs.

PRO 05.4 System Description

Over the past year we have focused on creating the infrastructure required for gathering high quality logs from wireless and embedded systems. The key component of our logging implementation is a suite of tools called LowLog that provide a variety of automated instrumentation services. These services instantiate our new ideas for gathering high quality logs. Standard logging techniques, such as inserting an arbitrary logging preamble to functions to log, for example, have been automated and made readily available to developers. At the same time the suite of tools enables exploring non-traditional logging techniques. For example we explore caller side logging, capturing compressed runtime control flow decision logs, and the impact of alternate log token encoding mechanisms.

Our current framework was first developed for the SOS operating system and recently ported to the TinyOS operating system. The current TinyOS architecture is illustrated in the figure above. Our logging system is added to TinyOS applications by extending the main application configuration with the LogTap component to provide safe sharing of the messaging infrastructure between our logging framework and the original application.

Developer specified regions of interest (ROI) drive code analysis and subsequent instrumentation performed by LowLog. Back end utilities are used to present users with a view into the logs that are streamed out of the network from the instrumented applications.

This infrastructure provides a foundation for gathering high quality log data upon which to explore triggered log retrieval and better log correlation.

PRO 05.5 Accomplishments

Using the LowLog suite of tools with our logging infrastructure we have demonstrated that high quality call traces can be collected using only a fraction of the bandwidth required by naive, but commonly used, logging mechanisms. Additionally we have observed that other forms of logs, such as recording run time control flow decisions in subsystems of interest, can be captured using a reasonable bandwidth. Finally, we are beginning to use the LowLog suite in our daily development cycle to diagnose problems within the wireless and embedded sensing systems we work with.
PRO 05.6 Future Directions
From the base provided by LowLog we are now transitioning our work towards triggered log collection and improved log correlation. Our initial work will explore the role of triggering log retrieval based on violation of expected log behavior. An initial version of this can be thought of as online and per-component suppression of repeated or expected logs. At the same time we will begin exploring how distributed triggers can be expressed in macroprogramming type languages.

In the second half of the year we will focus our work on triggering log collection based on log correlation. Initial correlation work will use a centralized server as we explore the types of correlation useful for identifying meaningful events across logs. Subsequent work will migrate this correlation out into the distributed network of embedded sensing devices.
PRO 06 Separation of Concerns for Sensor Network Software

PRO 06.1 People
- Principal Investigators: Todd Millstein
- Faculty: Ramesh Govindan, Computer Science, USC. Todd Millstein, UCLA.

PRO 06.2 Overview
Sensor network applications must cope with a variety of concerns, including severe resource and energy constraints, consistency and synchronization among nodes, and node failures. Current sensor network platforms therefore allow the application programmer to closely monitor and directly control the handling of various resources. While this design enables programmers to obtain acceptable application performance, it forces the code for an application’s functionality to be tangled with the code for handling non-functional concerns like resource constraints. Such tangling makes sensor network programming complicated, tedious, and prone to errors. It also makes sensor network components hard to reuse across applications, since the application logic is inextricably linked with many other concerns.

Similarly, tangled concerns prevent programs from being easily adapted to changes in the environment, new performance requirements, or new functional requirements. Most sensor network applications make tradeoffs among various concerns, in order to obtain acceptable application performance while satisfying resource constraints and maintaining energy efficiency. Since the performance of sensor network applications can be quite sensitive to the hardware platform and environment, the application designer may have to vary the tradeoffs and decisions she makes for various concerns depending on the particular deployment scenario. Doing this while the concerns are entwined with the functionality of the application can be quite difficult, resulting in unintended changes in functionality and other errors. Since sensor networks are known to be notoriously hard to debug (various debugging systems such as Sympathy, Nucleus, Clairvoyant, and Hermes are testament to this), tangled concerns can thus result in a lot of wasted programmer effort and time.

PRO 06.3 Approach
Separating various non-functional concerns from application logic, while still retaining programmer control over their handling, solves a number of the above problems. It promotes code reuse, by allowing generalized solutions to be provided for individual concerns. It also decouples application functionality from various non-functional concerns, improving visibility into the application logic and making it easier to analyze, while exposing tradeoffs among concerns which can be tweaked independently from the application functionality. Thus, a separation of non-functional concerns from application logic is quite desirable.

Some concerns like communication etc., can be easily separated into logical sections, using modular programming techniques and by designing modular architectures for sensor network systems. This insight has been used in the design of TinyOS to allow for separation of simple concerns into modules. Also, researchers have designed modular architectures atop TinyOS, which separate various communication and link-layer related concerns, as well as device-level power management concerns from the rest of the system.

For a number of concerns, however, it is quite difficult to cleanly separate them from the rest of the program, in both design and implementation, using the above-mentioned techniques. The code handling them is scattered throughout the system, and/or tangled with various other concerns. These concerns are known as cross-cutting concerns. Sensor network applications have their share of cross-cutting concerns which are hard to separate from application logic. Some examples of such concerns are fault tolerance, energy management, etc.

Researchers have identified a few high-level/cross-cutting concerns like energy management, heterogeneity, and failure recovery, and worked on individual solutions for separating them from application functionality. For example, language based solutions have been proposed to separate energy management from the application logic. Solutions have also been proposed to handle heterogeneity and role assignment and failure recovery. These
solutions are very specific, applicable only to the particular concern that they try to separate, and hence not generalizable to other concerns.

As the sensor networking hardware and applications are getting more advanced, researchers are aiming for not only energy efficiency, but also improved performance and robustness of their programs. Consequently, a number of new high-level, cross-cutting concerns are rising into prominence. A number of applications now list as one of their requirements concerns such as fault-tolerance, timing constraints, and reliability of communication. Currently, there is no one single solution which separates and handles all these concerns.

**PRO 06.4 System Description**
We propose the use of annotations to separate various high-level concerns from the application logic. The idea is that the application designers can annotate their application logic with various annotations representing how each concern should be handled. These annotations may be parameterized, that is, they permit a degree of control over their function. Also, the set of annotations may be extended by the programmer if they want to address a concern in a fashion not provided by any of the existing annotations.

We are leveraging our earlier programming platform for sensor networks, Pleiades, as a testbed for our ideas on separation of concerns. Pleiades is an example of a “macroprogramming” language for sensor networks. In this style, a sensor network is programmed centrally, with the compiler automatically partitioning the application to run on the individual nodes of the network.

A macroprogramming language is a natural starting point for support of separation of concerns, since it already raises the level of abstraction over traditional node-level programming. Annotations for separation of concerns can leverage this higher level of abstraction for increased expressiveness and usability.

Pleiades currently provides a default handling for all non-functional concerns, as implemented by the Pleiades compiler and runtime system. For example, the Pleiades compiler and runtime system coordinate to determine a strategy for implementing an application’s functionality across a network in a manner that minimizes communication costs. The Pleiades language also automatically ensures serializability for its concurrent for loop, thereby guaranteeing a strict form of consistency across nodes. Our aim is to design annotations that can augment Pleiades to allow programmer flexibility and control in the handling of these and other concerns.

**PRO 06.5 Accomplishments**
This year we have focused on an important non-functional concern for sensor networks: fault tolerance. Previously, the Pleiades implementation was vulnerable to node failures, with even a single node failure able to cause the application as a whole to fail. We have augmented the Pleiades system with a strategy for automatically detecting and recovering from failures. Further, we have implemented a declarative set of program annotations that allows both failure detection and recovery to be controlled by the application programmer, while keeping this concern completely separate from the main application logic.

A node is considered to have failed if a request to the node times out. For example, a failure could be detected when the Pleiades runtime attempts to obtain the value of a variable stored on a particular node by sending the node a message, but the node does not respond. Programmers can configure this failure detection algorithm by supplying a timeout period via an annotation.

Pleiades also provides annotations to allow the user to specify desired recovery actions to take whenever a particular type of failure occurs. For example, variables may be annotated to specify default values to be used in case the variable cannot be accessed due to node failure. In a car-parking application that we have implemented, each node maintains a Boolean variable isfree indicating if the node’s associated parking spot is available. Setting the variable’s default value to false ensures that if a node fails, the execution continues with the assumption that the failed node does not have a free parking spot, and hence gracefully deals with this failure.

Finally, we have incorporated special annotations for handling failures during a Pleiades cfor loop, which concurrently executes each iteration at a different node in the network. If a node executing a cfor iteration fails,
this is detected by the Pleiades runtime system. The Pleiades language allows the user to annotate cfor loops with a condition on how many failed iterations can be tolerated before having to consider the entire cfor as being failed. For example, an annotation could indicate that failures of up to 50% of the nodes executing a cfor iteration can be tolerated, but beyond that threshold the entire loop should be aborted and considered failed.

**PRO 06.6 Future Directions**

Our initial experience with declarative annotations for detecting and recovering from failures has been positive. However, failure is just one of a number of non-functional concerns of importance to sensor network applications. In the next year, we plan to leverage our experience with failures to design an annotation language that is declarative and easy to use while allowing 7.4 for the specification of a variety of different concerns. This is a daunting task with a number of important issues that must be addressed:

**Determining exactly how much control to give to the programmer over the handling of various concerns:** Should we provide certain concern-specific primitives that the programmer can use to construct their own mechanisms, should we only allow for simple tuning of the mechanisms using parameters, or should we give the programmer free reign?

**How extensible should the annotation language be?** We already recognize that extensibility is an important goal, and ideally, our framework should be completely extensible, in that new concerns should be easily incorporated and handled. However, we worry that letting the system be so extensible would make it complicated and unwieldy to use.

**Design of a general mechanism to describe tradeoffs:** We would like to have a simple, and general mechanism by which programmers can define their performance requirements for the application, and also describe the tradeoffs among various concerns.