Design and Implementation of a Sensor Network System for Vehicle Tracking and Autonomous Interception

Cory Sharp Shawn Schaffert Alec Woo Naveen Sastry Chris Karlof
Shankar Sastry David Culler
University of California, Berkeley

Abstract

We describe the design and implementation of PEG, a networked system of distributed sensor nodes that detects an uncooperative agent called the *evader* and assists an autonomous robot called the *pursuer* in capturing the evader. PEG requires services such as leader election, routing, network aggregation, and closed loop control. Instead of using general purpose distributed system solutions for these services, we employ whole-system analysis and rely on spatial and physical properties to create simple and efficient mechanisms. We believe this approach advances sensor network design, yielding pragmatic solutions that leverage physical properties to simplify design of embedded distributed systems.

We deployed PEG on a 400 square meter field using 100 sensor nodes, and successfully intercepted the evader in all runs. While implementing PEG, we confronted practical issues such as node breakage, packaging decisions, in situ debugging, network reprogramming, and system reconfiguration. We discuss the approaches we took to cope with these issues and share our experiences in deploying a large sensor network system.

I. Introduction

The problem of vehicle tracking with autonomous interception provides a concrete setting in which to advance sensor network and control system design. In our case, wireless sensor nodes containing magnetometers are distributed throughout a large physical area to form a diffuse sensing field. An uncooperative agent, the *evader*, enters and moves within this area, where it is detected by the magnetometers. Unlike environmental monitoring, it is not enough to obtain measurements of the physical disturbance caused by evader; we want to process the readings within the network and take action in a timely matter. The *pursuer*, a cooperative mobile agent, enters the field and attempts to intercept the evader using information obtained from the sensor network and its own autonomous control capabilities.

Local signal processing can be performed at each node to distill higher-level events given magnetic-field measurements due to motions of multiple vehicles. Clusters of nodes that sense sufficiently strong events can collectively compute an estimate of the position of the vehicle causing the disturbance. These potentially noisy estimations from multiple objects must be disambiguated and used to make continual pursuer course corrections.

The autonomous interception problem concretely manifests many of the capabilities envisioned for sensor networks [1], [2], including several levels of in-network processing, routing to mobile agents, distributed coordination, and closed-loop control. We address these issues in terms of the whole system design, rather than as isolated subproblems. Indeed, this whole-system view yields pragmatic solutions that are more simple than what is generally found in the literature for individual subproblems.

We built and demonstrated a working purser/evader system comprising a field of 100 motes spread over a $400m^2$ area in July 2003. The evader was a four-wheeled robot driven by a person using remote control. The pursuer was an identical robot with laptop-class resources. This paper describes the design, implementation, and experience with PEG. Section II discusses our overall design philosophy with respect to related work. Section III describes the application, the hardware platform, and the overall software system design. Section IV describes the in-network processing components for local detection and aggregation into a position estimate. Section V focuses on effective mobile-to-mobile routing, including effective tree formation and efficient landmark routing. Section VI describes the design of the navigation and control of the pursuer. Section VII evaluates our system design, and Section VIII outlines several of the experiences that have an impact on the overall system design and implementation.

Our main contributions are:

- We describe the design and implementation of PEG, a networked system of distributed sensor nodes that detects an evader and aids a pursuer in capturing the evader.
- We employ whole-system analysis and utilize spatial and physical properties to design efficient and simple distributed algorithms. We believe this approach is applicable to a variety of applications.
- We demonstrate one of the first operating large-scale tracking and pursing system that uses computationally and bandwidth limited sensor nodes.
- We share practical advice for deploying large sensor network applications, including package design, debugging techniques, and high-level network management services.

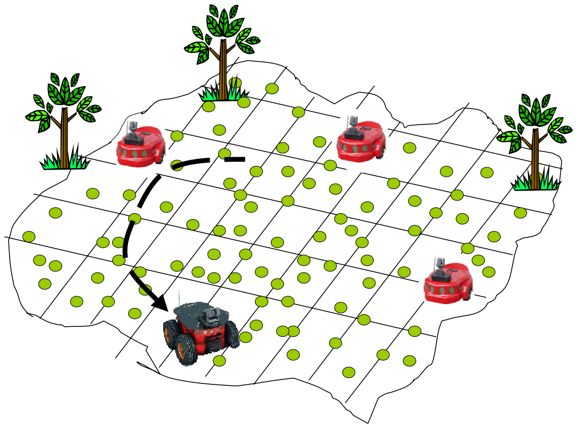

Fig. 1. Illustration of an intruder interception system using sensor networks to detect the intruder (arrow) and convey such information to the pursuing robots. Each pursuer matches the sensor data to its local map for path estimation and interception of the intruder.

II. Approach

Variants of the pursuer-evader problems have been well studied from a theoretical point of view [3], [4] and have been used for distributed systems research [5]. Sophisticated algorithms [6], [7] have been developed to associate readings with logical tracks of multiple objects. Elaborate data structures are used to deal with dynamic track creation and elimination of potential tracks caused by input noise. In addition, there is work [8] on closed loop control for the autonomous pursuer starting with various assumptions about what information is provided to the robot.

The early work on wireless sensor nets observed that distributing intelligence throughout the sensor array dramatically simplified the tracking problem [2]. When dense sensing is employed, each patch of sensors only has to deal with a few objects in a limited spatial region. Signal processing is greatly simplified because the sensors are close to the source, so the SNR is high. Physical constraints, such as speed of movement, allow for low-level filtering of false positives.

Others have studied a decentralized form of the problem where object tracking, classification, and path estimation are performed by a network of wireless sensors [9], [10]. In this formulation, sensing and detection are performed by local collaborative groups, each responsible for tracking a single target. The solution is cast in traditional distributed system terms with an explicit representation of the group associated with each object. Movement of the object involves nodes joining and leaving the group. Leader election is performed so that a particular node represents the object at any point in time and typically as the root of the collaborative signal processing. Recent work optimizes a single objective function that combines the cost of signal processing and tracking with the benefits of obtaining the given data [11]. Sophisticated programming environments have been proposed [12] to maintain the distributed data structure representing each logical entity and the set of nodes associated with its track. Unsurprisingly, these studies suggest that quite powerful nodes are required to perform distributed tracking. In addition, Welsh and Mainland have proposed a higher-level programming environment that uses abstract regions to simplify development of sensor applications [13]. Recently, other researchers have begun to focus on the use of very simple outputs from dense networks of sensors. For example, Aslam et al. [14] explore tracking where each sensor reports only a single bit of information of whether the disturbance is getting closer.

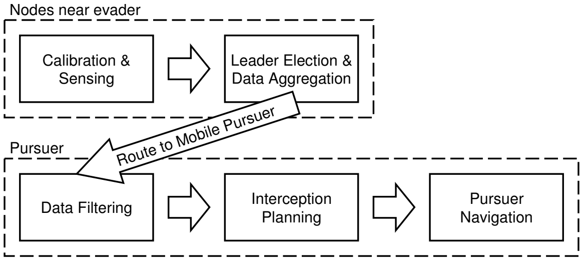

Fig. 2. Logical flow of information in PEG. After the nodes calibrate their sensors, they listen for events in the network. When events are sensed near several nodes, a leader is elected to aggregate the data into one packet. This packet is routed to the moving pursuer via multi-hop routing. After data filtering and interception planning, the pursuer chases the evader.

We adopt a light-weight approach that stems from two key observations. First, the autonomous interception problem admits a natural hierarchy. The lowest-tier of nodes, which are the most numerous and most resource constrained, are responsible for simple detection functions and for providing distilled information to a higher-tier. The higher-tier is capable of doing more substantial processing and initiating actions based on the information. In a basic tracking problem, the higher tier might include computer-controlled cameras, whereas in the interception problem it is a mobile pursuer. Elements in the lower tier generally do not need to know much about the track or the identity of the object, as their behavior does not change based on that information. The robots are power intensive and require substantial local processing, hence they are a natural point of concentrated processing.

Second, in-network processing at the lowest tier is essential to conserve bandwidth, thereby reducing contention and keeping notification latency low. The processed results need not be perfect as they will be further analyzed by the higher tier. For example, an inconsistent leader election may cause two closely related position estimations for an object at nearly the same time. This is easily addressed in the higher level processing. Inconsistency is far more benign here than in the settings where distributed consensus is typically used, for example to avoid multiple financial transactions [15], [16].

III. System Architecture

To provide pursuers with accurate detection events quickly and often, we developed services for detection, routing, data processing, and pursuit. We provide a sense of the overall information flow and describe the constituent system services. Additional issues of power management and authentication and encryption are beyond the scope of this paper.

A. Software Services

Figure 2 illustrates the information flow from the lower-tier sensing field to the higher-tier processing unit at the

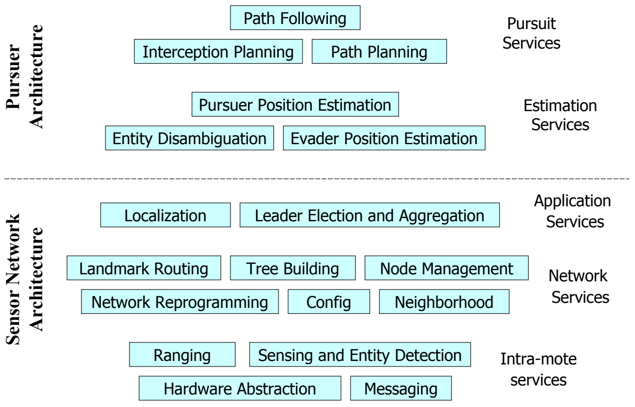

Fig. 3. Hardware and functional division of services. The dotted line separates services running on the pursuer from services running in the sensor network.

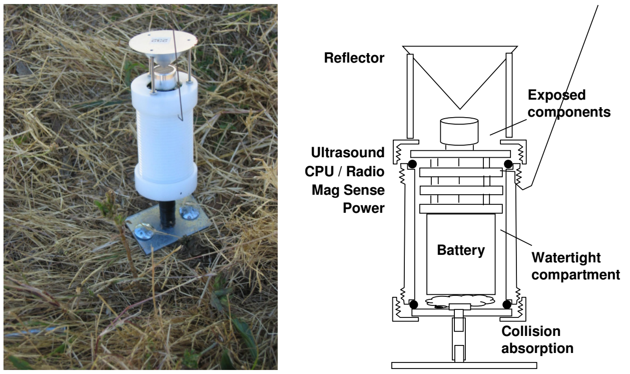

Fig. 4. Photograph of a PEG sensor deployed in the field on the day of the demonstration (left), and a schematic of its basic elements (right). The height between the plastic end caps is precisely 3.0 inches (7.6 cm), and the height from the bottom spring base and top of the ultrasonic cone is slight less than twice that distance.

pursuer. The sensor network detects the evader and routes this information to the pursuer, and the pursuer acts on this data to intercept the evader. Figure 3 shows the overall system architecture of the services required to implement PEG. The sensor tier basically performs two high-level services: self-localization and vehicle detection. The first core service, localization, is used to build a coordinate system of the entire network upon which the pursuer can map the collected detection events to meaningful physical locations. Ad-hoc self-localization is achieved using time-of-flight ultrasonic ranging technology with anchor-based localization algorithms. The system architecture supports self-localization, but it was not used in the live demonstration due to sporadic errors; this is redressed in later work [17].

When a vehicle is present, the sensing and detection component (Section IV-B) of nearby nodes will trigger detection events and invoke the leader election algorithm (Section IV-D) for data aggregation. The process of leader election is realized over a tuple-space neighborhood service (Section IV-C). The elected leader will propagate the aggregated data to the pursuers using landmark routing, which operates over a simple tree building mechanism (Section V).

When sensor readings reach the pursuer, the pursuer uses an entity disambiguation service to determine the cause of the event: the evader, the pursuer, or noise. Sensor readings that are determined to correspond to the evader are sent to the evader position estimation service. The pursuer position estimation service uses data from the GPS unit to determine an estimate of the pursuer position. Estimates of the position of the pursuer and evader are sent to the interception service, which generates a interception destination for the pursuer. This destination is processed by the path planning service to generate a feasible route. Finally, the route is submitted to the path following service that tightly controls the pursuer along this route. These mechanisms are further developed in Section VI.

Beside the core functionally required for PEG, new system services are also implemented to ease the difficulty in managing and configuring the network at the time of deployment. The Config component allows run-time configuration of system parameters that are useful for system tunings. The node management component is used for node identification, debugging, and network-wide power cycle management. Finally, network reprogramming allows rapid reprogramming the entire system over the wireless medium, which is valuable for rapid update of code image. We discuss these deployment issues in Section VIII.

B. Sensor Tier Platform

The sensor tier of our system consists of Berkeley Mica2Dot motes [18], a quarter-sized unit with an 8-bit 4 MHz Atmel ATMEGA128L CPU with 128 kB of instruction memory and 4 kB of RAM. Its radio is a low power Chipcon CC1000 radio that delivers about 2 kB/s application bandwidth with a maximum communication range of around thirty meters for our particular antenna and environment. Each node uses a magnetometer to detect changes in a magnetic field, presumably caused by a nearby moving vehicle. An ultrasound transceiver at 25kHz is used for time-of-flight ranging. A reflector cone is situated above the transceiver to diffuse the ultrasonic waves for omni-directional ranging which significantly reduces the ranging radius to about 2 meters.

Figure 4 shows the complete packaging of a sensor node. At the bottom of the node is a base that secures the node to the ground and extends it a few inches above the ground. The battery, voltage conversion board, magnetic sensor, and the Mica2Dot are all protected by the plastic enclosure. The side of the enclosure has a hole that allows a quarter-wavelength piano wire antenna to be connected to the Mica2Dot. The only sensor exposed is the ultrasound transceiver at the top, with the cone securely mounted above it. The complete packaging is robust against impact from vehicles, and the spring at the base keeps the node upright even after collisions to elevate the node a few inches above the ground plane for effective radio communication.

All nodes at the sensor tier run TinyOS [18], an event-driven operating system for networked applications in wireless embedded systems. The implementation of all the core services shown in Figure 3 consumes about 60 kB of program memory and about 3 kB of RAM.

C. Higher Tier Platform

Our ground robots are essentially mobile off-road laptops equipped with GPS. Each robot runs Linux on a 266 MHz Pentium2 CPU with 128 MB of RAM, 802.11 wireless radio, a 20 GB hard drive, all-terrain off-road tires, a motor-controller subsystem, and high-precision differential GPS. This platform is sufficient to execute the simple higher-tier services shown in Figure 3. The GPS typically provides estimates every 0.1 seconds with an accuracy of about 0.02 meters. The top speed of the robot is about 0.5m/s, with independent velocity control for each wheel. In our deployment, we used one pursuer and one evader, each the same model robot.

IV. Vehicle Detection

Detecting a vehicle in the network begins with a node gathering and processing data leading up to the formation of a position estimate report. In this section, we show how a bandwidth analysis of the overall system drives the design of our phases of vehicle detection.

A. Bandwidth-Driven Design

We design our sensor network to provide full, redundant sensor coverage – for sensors placed in a grid, a vehicle excites at least four and up to nine sensors. From this coverage requirement, we design the rest of the detection system with an understanding of the impact of low-level decisions on regional bandwidth limits.

Presuming a local aggregate bandwidth of 40 packets per second, a single node can provide up to four reports per second before a region of nine nodes saturates the shared channel. If each node sends these detection events, the local channel will be saturated leaving no bandwidth for other communications such as routing these readings to the pursuer. Additionally, as more vehicles are added to the system, routing the data will increasingly tax the bandwidth of the system. Clearly, we must use some techniques to conserve bandwidth.

We use local aggregation to reduce many detection events into one position report, We allocate half the total bandwidth for exchanging local detection reports and the remaining bandwidth for system wide behaviors such as routing position estimates to pursuers. Even though sharing local detections uses a significant portion of the local bandwidth, the pursuer still receives frequent position updates. We decompose this overall process into three distinct phases: calibration and sensing, local detection reports, and leader election and position estimation.

B. Calibration and Sensing

The magnetometer measures the entire magnetic environment. This includes static structures such as the Earth's magnetic field (these magnetometers are often used for digital compassing) as well as, for instance, underground metal pipes, the metal in a desk chair, or rebar in the concrete of a parking garage structure. To detect changes in the magnetic field caused by a moving vehicle, this static environment must be accounted for in each node's measurements. Each node subtracts the output of an moving average from each reading. This sets a minimum detectable speed on a vehicle, because a sufficiently slowly moving object will be indistinguishable from the static environment.

One interaction that we did not expect is a relationship between the radio communication and the magnetometer. Because of the proximity of the radio chip and the magnetometer chip, which is in part a result of the small package design but also exacerbated by our hardware design, radio transmissions excite significant readings from the magnetometer. As a workaround, we invalidate magnetometer readings for a short period whenever a radio packet is sent or received at the node.

C. Local Detection Reports

To decide if a node should share its calibrated reading with its neighbors, the node compares the 1-norm of its magnetic reading against a preset threshold value. If the detected value exceeds this threshold, the node sends a message including the magnetic magnitude and its own (x, y)-position as 8.8 fixed point values in meters. To limit a node's local detection report rate to 2 packets per second, each node is subject to a reading timeout of 0.5 seconds during which it is not allowed to share a new reading.

To share data among a neighborhood of nodes, we evolved a new programming primitive called Hood [19]. A neighborhood in Hood defines a set of criteria for choosing neighbors and a set of variables to be shared. The neighborhood membership, data sharing, data caching, and messaging is managed by the Hood abstraction, allowing the developer to focus on the properties of a neighborhood instead of its mechanics. Hood exploits the cheap broadcast mechanism of a sensor network to allow asymmetric membership – a node broadcasts changes to its neighborhood values and doesn't know or care what other nodes consider it a member, which is different from the group collaboration work found in [9], [10]. Overall, our design is well suited for unreliable communication channels such as those in sensor networks, and defers concerns of reliability and consistency to the application level.

For PEG, we created a `MagHood` that manages the messaging and caching of local detection reports and prescribes the neighborhood membership criteria. Because the magnetometer neighborhood represents a local physical relationship, and because radio connectivity doesn't have a clean relationship with respect to physical distance, membership in the neighborhood is restricted to only those nodes within 3 meters. And, similar to the report timeout, readings are invalidated after timeout of 0.5 seconds, which sets a time window on the validity of a neighbor's reading.

D. Leader Election and Position Estimation

We cast leader election as primarily a bandwidth reduction technique and relax the usual requirement of correlating a single leader with a single entity, unlike [9], [10] where vehicle classification is done on the sensor node. High level processing on the pursuers imposes model constraints to correlate position estimates with individual entities. This decomposition allows us to construct a significantly more simple and robust leader election protocol.

Using `MagHood`, each node gains a view of the recent detection reports from nodes in its neighborhood. At this point, the leader election protocol requires no additional communication – a node elects itself leader if it has the maximum magnetometer magnitude among the nodes in its neighborhood. This lightweight mechanism embodies the idea of loose consistency: in the worst case, every node that detects the location of a vehicle reports it.

The pursuer receives reports about all position estimates in the network – those caused by the evader, by itself, or by noise. Even with this policy, redundant leaders for a single object are the exception not the rule, because leadership changes smoothly over time given the physical interactions. Additionally, this design implicitly supports multi-object tracking by providing all the data necessary for the pursuer to do centralized filtering and correspondence.

The position estimate report contains an 8.8 fixed-point (x, y)-position calculated as a center of mass, the total number of nodes contributing to the report, and the sum of the detection magnitudes. Similar to previous timeouts, a node is only allowed to become leader and send a position estimate report at most every 0.5 seconds. This estimated position is again a loosely consistent value – instead of using a time synchronization service to guarantee that all readings happen within a strict epoch, the cache timeout establishes a notion of a weak epoch.

Within an epoch, a node elects itself leader the instant it determines it has the largest detection magnitude. As an alternative, if a node would have waited a period of time for additional readings, there exist certain vehicle paths that prevent any node from becoming a leader, which would produce no position estimates for the pursuer. Furthermore, this protocol ensures that a node can become a leader if its detection exceeds the threshold, meaning that the maximum speed of the vehicle is only a function of the properties of the sensor and the allocated bandwidth for position reports.

V. Routing

The primary routing requirement in PEG is to deliver the evader detection events, as sensed by the network, to the mobile pursuers. That is, we must route packets from many sources to a few mobile destinations. This differs from the typical many-to-one data collection traffic model found in other sensor network applications [20], [21], [22]. However, it resembles some of the work found in the mobile computing literature, which provides different approaches to support this mobile routing service. In this section, we first explore these approaches and then discuss a simple and efficient landmark routing approach to arrive with a solution, which is potentially applicable to systems other than PEG.

A. Design Approaches

One design approach is to treat the entire network and the mobile pursuers as one ad-hoc mobile system, and deploy well-known mobile routing algorithms such as DSR, AODV, and TORA [23], [24], [25] to provide an any-to-any routing service. These protocols are designed to support any pairs of independent traffic flows while the traffic in PEG are correlated and directed only to a few moving end points (the pursuers). Nonetheless, the resulting routing paths with this approach would be efficient as these algorithms optimize routes based on the shortest path metric.

Another approach is to decouple the network from the mobile pursuers and exploit the static network topology to decrease the communication complexity for routing. This resembles the home-agent work found in mobile computing [26], where every pursuer is assigned a home-agent for data forwarding purposes. For example, recent work to support group communication among a set of moving agents over a sensor network in a bounding box has been proposed [27]. It assumes that any-to-any routing comes free by using geographical routing and maintains a horizontal backbone across the bounding box. Through these home agents on the backbone, communication between the moving agents and the network is achieved. Mobile agents need to register with the backbone to discover new home agents as they move; these migrating home agents allow more efficient routing paths to be established. The communication complexity thus depends on the overhead of backbone maintenance and home agent migration frequency.

For efficiency and simplicity, the approach we use also exploits the static network topology, but we do not assume any geographical routing support. Furthermore, we minimize protocol communication overhead by distributing soft state across the network and slightly sacrificing routing efficiency. We use *landmark routing* [28] to split the many-to-few routing problem into two subproblems: many-to-landmark and landmark-to-few. Landmark routing is a simple mechanism that uses a known rendezvous point to route packets from many sources to a few destinations. For a node in the spanning tree to route a detection event to a pursuer, it first sends a message up a spanning tree to the root node, the landmark. Then the landmark forwards the message to the pursuer. The original landmark paper discusses the scalability of this approach using a hierarchy of landmarks. In this work, we only consider a single landmark. This approach results in longer routing paths as traffic must be relayed through the landmark, which hurts latency, but requires less control bandwidth to maintain routes to the moving target than other protocols such as AODV.

For the many-to-landmark routing, we first considered using a simple grid-based routing such as [29] since location information of each node is known and the network layout is a grid. However, we did not pursue this direction since it does not address link reliability issues, which is essential for creating reliable routing over unreliable links. Our approach to the many-to-landmark routing is based on a simple flooding mechanism that can rapidly build spanning

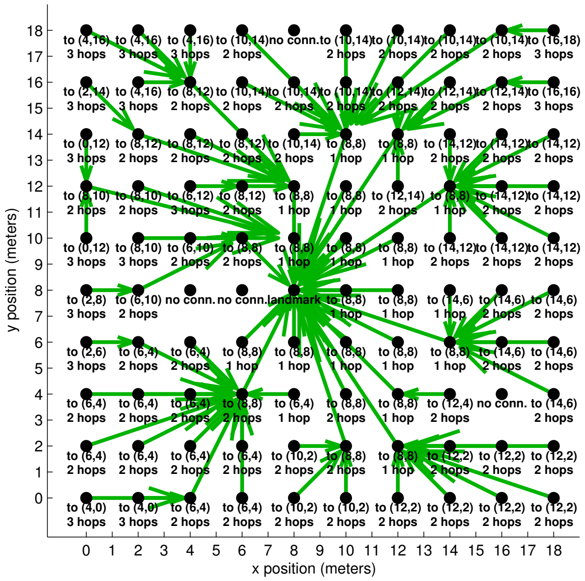

Fig. 5. Spanning tree generated by PEG using 100 mica2dot nodes. PEG uses a basic flooding algorithm that adapts to different node densities and parent selection algorithms for better trees.

trees with efficient routes and reasonable communication reliability. This approach of rapid tree-building provides a quick fail over to cope with the flakiness found in sensor networks, and maintains the soft-state design principle.

B. Building Good Trees

For many-to-landmark routing, we rely on a spanning tree rooted at the landmark. All packets received or generated by a node are forwarded to its parent until they reach the landmark.

A common approach to building a spanning tree is to flood the network with a beacon, and each node marks its parent in the tree as the first node from which it receives the beacon, and then rebroadcasts the beacon. This approach of flooding the network and routing using the reversed paths is used in ad-hoc routing algorithms such as AODV [24] and DSR [23] to build a topology quickly and trade off optimality for handling mobility.

In such a topology formation process using flooding, two potential problems must be addressed: quality route selection and the broadcast storm problem [30]. The routing protocol must avoid selecting bad links for routing; in particular, asymmetric connectivity should be avoided if routing paths are formed by reversing the path discovered by flooding. Route selection solely using hop count cannot address these issues. The second issue is related to the broadcast storm, which occurs when many nodes receive a beacon simultaneously and attempt to rebroadcast the beacon immediately. As a result, a storm of packet collisions is created and significant message losses would occur, which leads to an ill-formed topology containing many *back edges*. Back edges occur when nodes miss a beacon message because of collisions, but later overhear it from nodes further down the tree. Back edges create unnecessarily long routes.

In exploring the different routing algorithms that use flooding for route discovery, not all of these issues are addressed by the protocols together. Empirical data in real sensor networks have shown that if these issues are not addressed, the resulting topology will be ill-formed and the routing paths are likely to be composed of long unreliable links not suitable for multihop communication [31]. Following our simplicity guideline, we devised two simple mechanisms that interact with the routing layer and the link layer to address these two issues together.

Our first challenge in building spanning trees is route selection. The goal is to ensure a high end-to-end packet transmission rate while minimizing the tree's depth. A node must consider both the quality of the link to its parent and its parent's tree depth. Without any rapid link estimation mechanism, we rely on the received signal strength indicator (RSSI). Recent studies for both sensor networks [32] and 802.11 networks [33] showed that RSSI is not a good predictor of link quality. However, we can exploit spatial information to our advantage to rely on RSSI values. With all nodes on roughly a grid configured to transmit at the same power and the fact that signal strength decays at least $1/d^2$ when close to the ground, it is possible to use RSSI threshold filter to scope the neighborhood relative to the physical distance. By empirically measuring the relationship between link reliability and RSSI values among all the nodes at different grid distances beforehand, we determine a high confidence RSSI value for the entire network that maximizes communication distance while reliably preserving bidirectional link reachability. This is essential for our landmark approach since it utilizes the reverse paths on the self-discovered spanning tree. When receiving messages, a node only accepts it if it is greter than the RSSI threshold value, even if it was able to properly decode the message. The routing layer can be a simple algorithm that selects the shortest hop-count parent that passed the RSSI filter. Section VII presents measurements of the end-to-end reliability of the routing paths discovered by this approach.

The second challenge is to alleviate the broadcast storm problem. We used a time-delayed back off that adapts to the observed cell density. Broadcast storms occur because several nodes simultaneously attempt to rebroadcast the beacon. Suppose, instead, each node waits a random time before rebroadcasting the beacon, then network congestion decreases. With random back-off, the number of nodes in a radio cell should be proportional to the number of potential wait times: as node density increases, nodes must wait longer periods of time. Choosing the maximum wait time, then, requires knowledge of the density; by choosing a sufficiently large interval, we can guarantee that each node has sufficient time to broadcast its announcement without preventing other nodes from doing so.

An alternative extends the prior idea to result in a more adaptive technique. Upon the reception of a broadcast message, each receiver starts a timer to fire in a random amount of time less than R. Every time the node receives a broadcast message before its timer expires, it resets the timer to fire in a new random time. Thus, the R interval from which nodes wait is significantly smaller than in the naive protocol since the total wait time is now adaptive and inversely proportional to the radio cell density, which is the number of times the same broadcast message is heard. In both sparse and dense networks, then, propagation within local cells finishes in $R \cdot n/2$ time, where n is the cell density and R is chosen uniformly.

One typical spanning tree is in Figure 5. The data was collected from 100 mica2dot nodes with 2m spacing in either direction. The landmark is located near the center of the field, at position (8,8). The tree has depth three, considerably less than if grid routing were used. Four nodes did not join the spanning tree because they are broken; Section VIII addresses breakage. Additionally, most nodes' parents are physically closer to the landmark. In those cases where this is not true, such as at (0,10), the physically closer parent is not any closer by the hop-count metric.

C. *Efficient Landmark Routing*

With the spanning tree built using the previous mechanism, any nodes in the network can send messages to the landmark, which must then be able to forward the messages to the moving pursuers. To accomplish this, the pursuer periodically informs the network by picking a node in its proximity to route a special message to the landmark, thereby laying a "crumb trail." Instead of maintaining all the routing states at the landmark, this message deposits a "crumb" with each intermediate router on the spanning tree so that messages destined to the pursuers at the landmark can reverse the path that the crumb message took. Since each node along the crumb trail knows its next hop, communication overhead is smaller as it is not necessary to include the entire reverse path in each packet.

We support multiple concurrent crumb trails, allowing for many mobile destinations. Each crumb trail is identified by the pursuer's ID when it deposits its crumb.

The pursuer increments a sequence to give a time dimension to these crumb trails such that these paths can dynamically track pursuer's position. All such routing states are soft in that they become stale over time, and thus, stale crumb trails prune themselves automatically.

Our approach to solving the many-to-few routing problem is efficient. The tree-building overhead is an $O(N)$ operation. As discussed earlier, ad-hoc protocols require $O(N)$ overhead to route to a mobile destination. In our landmark scheme, the overhead in maintaining mobility is solely the crumb messages, which has a communication complexity of $O(d)$, where d is the network diameter. This means that we can route to a pursuer with significantly less overhead.

Note that there is no explicit coupling between the landmark and the moving pursuers as in the case of the home agent approach. In fact, the pursuers do not even know the address of the landmark. This is a nice property, allowing the landmark node to move over to another node if the first fails.

A shortcoming of this approach is that the landmark is a central point of failure. However, there are many techniques to eliminate this vulnerability. Since the crumb trails and the spanning tree can be built rapidly, it is easy to switch over to another landmark if the original fails. Additionally, it is simple to maintain two separate instances of landmark routing with independent crumb trails and landmarks. This quick switch over capability is important to cope with the flakiness inherent in sensor networks.

VI. Navigation and Control

The pursuer must decide how to assimilate an aggregated sensor packet to minimize evader capture time. In this section, we will describe the difficulties in designing this control system, the techniques used to overcome these difficulties, and the final architecture. Although the control architecture we present is not new, the modality of the sensor network data is significantly different than those of traditional control systems. We will discuss how these concerns are addressed with our implementation.

A. *Design Issues*

Classical feedback control design [34] typically assumes that periodic sensor readings occur and arrive at their destination in zero time, that the computation of the control law is instantaneous, and that control signals are applied to the actuators immediately. These requirements are typically necessary to analyze a controller's stability and performance. Several techniques have been studied to relax these assumptions [35], and some researchers have suggested that new techniques need to be developed [36]. However, in practice, these constraints are typically approximated by using a sufficiently high frequency of sensor readings, by minimizing timing jitter and latency, and by reducing computation time of the control law. Typical implementations of control systems achieve these approximations through the use of locally resident sensors, actuators, and powerful computational hardware. However, due to the distributed, low-power nature of sensor networks, many of these assumptions are violated.

In PEG, we can approximate instantaneous control computation by assuming that the pursuer is a powerful node that performs all the control computation. However, the application of classical control techniques is frustrated by the tendency of the sensor network data to be noisy, arrive late, lack time-stamps, and arrive without periodicity. High speed controllers, such as a path follower, further highlight the difficulty of control using only sensor network data. In our case, a feedback implementation using only sensor network data would require artificially slowing down the dynamics of the system.

Furthermore, nodes will fail at times due in part to faulty hardware and collisions with robots. This presents another characteristic of sensor networks that differ from a typical control setting. When operating in a sensor network, a controller must additionally compensate for sporadic sensor readings due to badly behaving nodes. For such problems, it is not always possible for a controller to maintain a constant level of performance. We seek to provide high performance of the controller while allowing for a graceful degradation in performance as the data qualitatively deteriorates, while ensuring safety properties such as not leaving the field.

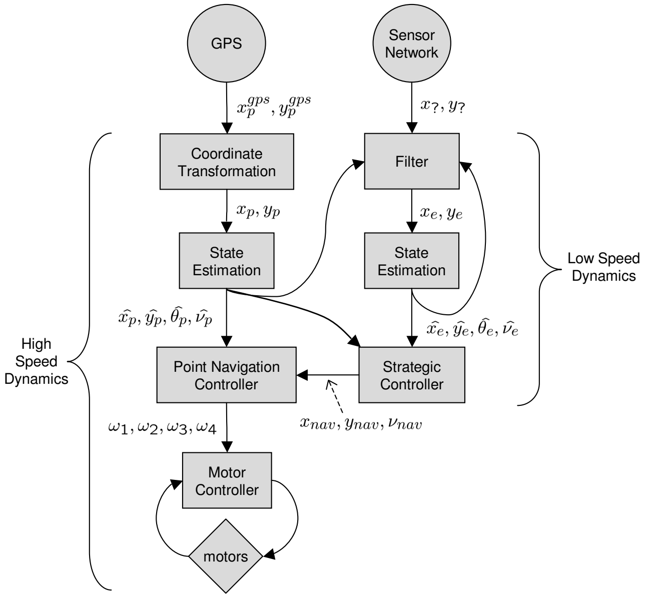
GPS
Sensor Network
x_p^{gps}, y_p^{gps}
$x_?, y_?$
Coordinate Transformation
Filter
x_p, y_p
x_e, y_e
State Estimation
State Estimation
Low Speed Dynamics
High Speed Dynamics
$\widehat{x_p}, \widehat{y_p}, \widehat{\theta_p}, \widehat{\nu_p}$
$\widehat{x_e}, \widehat{y_e}, \widehat{\theta_e}, \widehat{\nu_e}$
Point Navigation Controller
Strategic Controller
$x_{nav}, y_{nav}, \nu_{nav}$
$\omega_1, \omega_2, \omega_3, \omega_4$
Motor Controller
motors

Fig. 6. Block diagram view of the hierarchical multi-rate pursuer controller. All variables except those with a GPS superscript represent values relative to the mote coordinate system. Furthermore, the subscripts p and e, indicate pursuer and evader respectively. Finally, w_k represents the set-point for velocity of the k^{th} wheel

B. Design Choices

To overcome the aforementioned difficulties, we make several design choices for the pursuer control. First, we cleanly separate the control system from the sensor network as much as possible. To this end, the network provides sensor readings to the pursuer, but all processing and control computation occurs on the pursuer. Second, we apply more traditional control techniques to the pursuit algorithm, changing the design where necessary for sensor network data.

A pursuit control system ultimately consists of a system for estimating the position of the evader, for strategically deciding where the pursuer should go next, for planning a path to the next destination, and for following that path. To achieve the best estimation of the evader's position, we prefer a model for the evader's dynamics with unknown control input. However, this is an unnecessary burden considering the specification of our system. For instance, the robots can quickly change the velocity of each of their wheels independently (within about 0.2 seconds), which, as far as a sensor network that reports every 1-3 seconds is concerned, allows the robot to virtually change its speed and direction instantly.

In PEG, the pursuer only needs data every few seconds from the sensor network, but requires much more timely location information for navigation. We use GPS for navigation which provides updates about every 0.1 seconds with an accuracy of about 2cm.

C. Implementation Overview

In this section, we outline our final controller design which is illustrated in Figure 6. First, two different coordinate systems must be addressed: the GPS coordinate system and the coordinate system relative to the mote network. To work within a single coordinate system as soon as possible, we immediately convert GPS measurements into the mote coordinate system. GPS provides estimates of the pursuer's position in GPS coordinates, $[x_p^{gps}, y_p^{gps}]^T$. Using a *fixed, known* homogeneous coordinate transformation Φ, we compute the pursuer's estimated position in the mote coordinate system as $[x_p, y_p, 1]^T = \Phi * [x_p^{gps}, y_p^{gps}, 1]^T$. Using a trace of these values, $(\ldots, [x_p^k, y_p^k]^T, [x_p^{k+1}, y_p^{k+1}]^T, \ldots)$, we can compute a full state estimation for the pursuer, which includes estimations of the position, the velocity, and the orientation.

Many techniques exist for state estimation [37], [38], in our case, it is enough to use simple techniques. For the estimated position $[\hat{x}_p, \hat{y}_p]^T$, we simply use an average of the two most recent GPS positions. These estimated positions then form another trace $\Lambda = (\ldots, [\hat{x}_p^k, \hat{y}_p^k]^T, [\hat{x}_p^{k+1}, \hat{y}_p^{k+1}]^T, \ldots)$. For the orientation estimate $\hat{\theta}_p$, we use an average of the angle between pairwise combinations of the last four estimated positions; i.e., the four most recent entries in Λ. The magnitude and direction of the velocity $\hat{\nu}_p$ is estimated by using the two most recent entries in Λ.

Turning to the evader's state estimation, we first receive an estimate of an unknown object's position in the network, $(x_?, y_?)$. Using a previous estimation of the evader's position $(\hat{x}_e, \hat{y}_e)$ and a current estimation of the pursuer's position $(\hat{x}_p, \hat{y}_p)$, a filter determines if the reading corresponds to the evader, the pursuer, or noise in the system. A simple strategy for doing this is a probabilistic confidence system. If we let α be the average error of the sensor network due to true positives, i.e., not including error due to false positives. Then, we can safely disreguard sensor reports within α of the pursuer's estimated location, since these reports must correspond to the pursuer or a captured evader (assuming that our capture radius is greater than α). Sensor reports within $2 * \alpha + |\nu_{max}| * t$ of the previous estimate of the evader's position are assumed to be the evader, where t is the elapsed time since the previous estimate and ν_{max} is the maximum velocity of the evader.

If the new sensor reading is determined to be the evader, this value is used to update the state estimate of the evader using techniques similar to those for the pursuer. In this case, our estimate of the velocity and orientation will be of much lower quality. However, because the strategic controller only updates every time it receives a new evader state estimate (at a much lower rate than the actual velocity and heading of the evader can change) it is unnecessary to exploit knowledge of the evader's orientation and velocity.

Using position estimates of the pursuer and evader, the strategic controller chooses the pursuer's next destination $[x_{nav}, y_{nav}]^T$ and interception speed ν_{nav}. In making this choice, the strategic controller attempts to minimize capture time. Again, we use a simple strategy: the pursuer moves to the estimated position of the evader. Finally, the point navigation controller will compute a path to the new destination. This path is realized by continually issuing new set points $(\omega_1, \omega_2, \omega_3, \omega_4)$ for the velocity of each robot wheel, such that the robot moves forward enacting turns as needed to

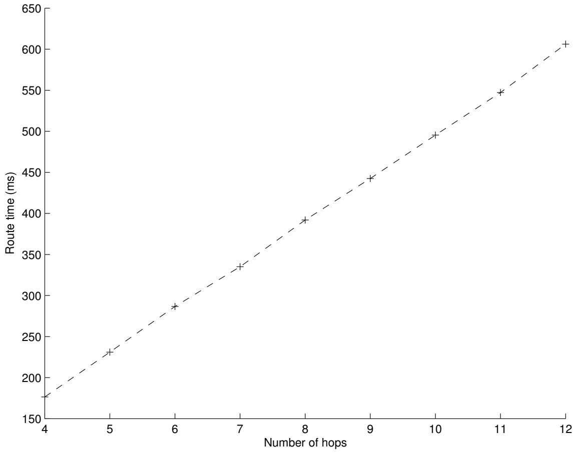

Fig. 7. Latency of packets routed through PEG's landmark routing algorithm. Each data point represents the average time to route 200 packets through the given number of hops on a 36 node indoor Mica 2.

reach it's destination.

In conjunction with the aforementioned processes, the controller maintains safety specifications by applying hard constraints to the controller at various points. To ensure that the pursuer never leaves the network, the point navigation controller always compares the pursuer's estimated state with the fixed, known values of the network boundary. If the pursuer is leaving the network, the point navigation controller directs the pursuer to the center of the network until further notice. Additionally, if the strategic controller notices that the pursuer's estimated state is within the capture radius of the evader's estimated state, it has the point navigation controller stop the pursuer. The pursuer remains there until a new evader update farther away appears; at which time the control system reinitiates pursuit of the evader.

VII. Evaluation

A. Routing Service

One of the most important metrics for evaluating the multihop routing service is end-to-end reliability, especially when the topology is built over many unreliable links during a network-wide flooding. We created a set of micro-benchmark experiments to measure end-to-end success rate of packet delivery of any random pair of nodes in the network using our landmark routing. In all these measurements, we do not use link retransmissions. For latency tests, there is no contention on the channel because only one packet is being sent at a time. The end result is very promising. For paths that have lengths varying from 4 to 6 hops, the average of end-to-end success rate consistently falls in the range of 95% to 98%. This implies that our topology formation can build trees that are reliable for bi-directional communication.

Another metric that is important to PEG is the end-to-end latency in delivering the detection events to the moving pursuers. As discussed before, our landmark routing approach trades off route efficiency for simplicity and low protocol overhead. The simplicity of the landmark routing scheme produces routes that may be longer than necessary since the message must pass through the landmark. For example, for a neighbor to route a message to an adjacent node, it must traverse through the landmark which could be far away. There could be delay from many sources: the time it takes a packet to travel, the processing time, MAC back off time, and routing decision time. By observing the packet size and extra synchronization overhead coupled with the radio bandwidth, we can conclude that a packet occupies the channel for 26.2 ms. The MAC waits a uniformly random time between 0.4 ms and 13.0 ms before sending a packet, averaging in a 6.7 ms delay. We measured the latency that it takes for our algorithm to route packets in Figure 7 on a field of 36 sensor nodes. For a least squares minimum fit on the data in the figure, we find the slope of the line is 53 ms/hop, so we conclude processing time is consistently around 20 ms. Even if the landmark route is 6 hops while the optimal path is a single hop, the landmark routing will take 225 ms longer than necessary. In this time, the evader can only travel 13 cm, an insignificant distance compared to the precision of the measurements. Thus, even though landmark routing may choose longer routes, the extra routing time is within our requirements.

B. Tracking and Interception

To evaluate the system in a large scale demonstration on July 14, 2003, we deployed a field of 100 nodes and performed a half-dozen runs[1]. The evader was controlled by a driver not affiliated with PEG. The evader can leave the sensor grid area, though the pursuer cannot – the maximum speed for either robot is about 2.25 mph or around 1 meter/second. The pursuer was able to successfully capture the evader in all cases; we define success when the pursuer arrives within a grid square of the evader. Figure 8 displays one such interception. Initially, the pursuer is in a different orientation from the evader. It first orients itself towards the evader before capturing the evader. The sequence spans 26 seconds and ends when the pursuer touches the evader.

In order to display more quantitative data, we would like to analyze network traces from an actual run from our July demo. Unfortunately, our demonstration was not sufficiently instrumented to collect data, and we have subsequently instrumented and re-deployed PEG. Figure 9 demonstrates our efforts on a 7x7 field of sensor motes with a 2m spacing. The grid displays the actual track of the evader in a solid line demarcated with 10s intervals in squares, as determined by GPS. Each star shows the leader node that sent a packet to the moving pursuer after aggregating the detection data. We draw a dashed line from the leader to the corresponding point on the evader's path when it makes the detection report. When the dashed line is short, it indicates a successful, low error detection reading. There are no reports when the evader leaves the playing field around time 100s.

The results also indicate a few noisy nodes. The pursuer must filter out this noise in estimating the evader's position. For example, node (12,12) detects a spurious reading at

[1] A movie of all runs is available at `http://webs.cs.berkeley.edu/nestdemo.mpg`

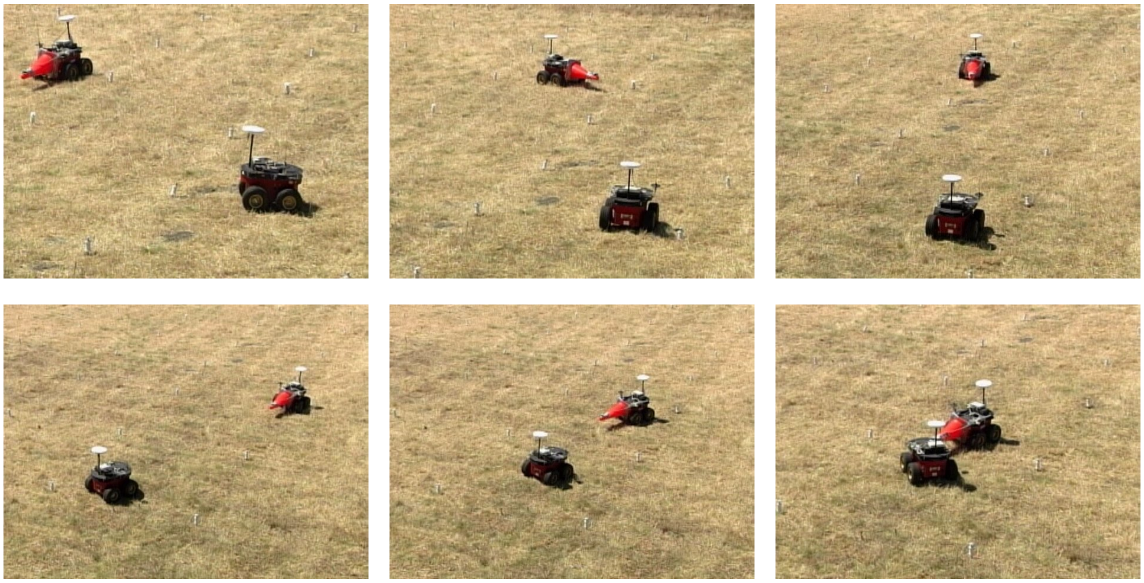

Fig. 8. This sequence taken from a video of the live July demonstration shows a successful capture of the evader (foreground) by the pursuer (background).

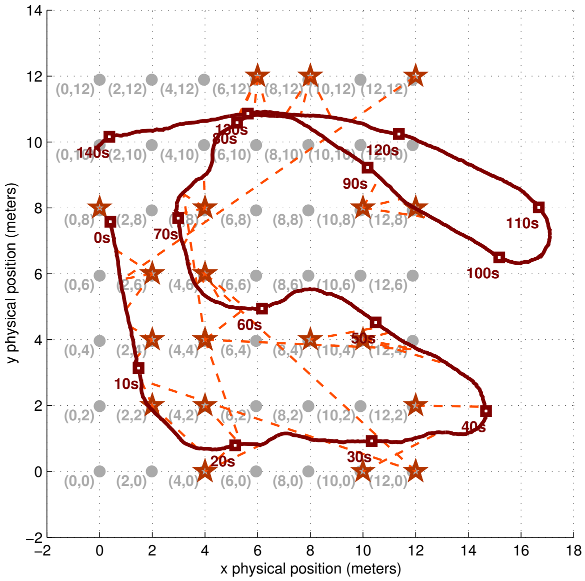

Fig. 9. Intruder tracking using PEG. Evader GPS position is shown as a solid line. Detection event leaders are shown as stars. Using dotted lines, leaders are linked to the evader's position at the time of detection.

around 4 seconds. In analyzing this plot, we found 4-5 spurious readings. Additionally, we manually squelched the output of nodes (4,10) and (4,12). Their magnetometers were not properly calibrated and would generate a false reading every few seconds. Just as in our original run, we found that a few nodes in every deployment would not act properly when deployed; in such situations, we needed to suppress a handful of nodes from reporting. For larger or longer deployments, we foresee an automatic health monitoring service that, in its simplest form, reboots or powers down a node when it behaves outside specified tolerances. As we discuss in Section VIII, accurate debugging and network analysis tools are a necessity for large sensor network deployments.

VIII. Deployment Experiences

Through the course of designing and implementing PEG, we faced various system issues, including system breakage, packaging, in situ debugging, network programming, and system reconfiguration. In this section, we discuss the approach we took to cope with each of these issues. These implementation experiences apply to many kinds of large sensor network applications.

A. Breakage

In the course of deploying and operating PEG, we noticed a moderate rate of breakage in terms of node failure, similar to the experience reported in other sensor networking deployments [22]. Some of this is due to our inexperience as packaging engineers. However, in the course of disassembling the packaging, reprogramming, charging the battery, reassembling, and re-deploying, we noticed a trend of a few percent of the nodes failing at run time. Out of this experience came the maxim that "*Every touch breaks*." This reinforces our design philosophy of maintaining soft state, loose consistency for inter-nodal coordinations, and rapid fail over in network topology formation. Furthermore, the system services for in situ testing and development, as shown in Figure 3, are therefore sought to eliminate *any* need to physically handle nodes. We believe that these system services are useful even when future sensor nodes become more robust.

B. Packaging

A real-world sensor deployment must carefully consider node packaging, and we discovered that that packaging requirements for deployment are different from those for development. For development, the packaging should expose access for convenient debugging, reprogramming, and battery recharging. However, we did not properly anticipate such need, and during development, we would frequently need to disassemble the packaging in order to fix broken components, reprogram the nodes, or recharge the batteries. If we had better foresight in our design, we would have designed the packaging to support reprogramming and recharging without full package disassembly.

After deployment we discovered that the packaging was interfering with the magnetometer. The piano wire antenna, battery, and metallic spring base all align the magnetic field in the proximity of the magnetometer, significantly reducing its sensitivity and overall range of detection. The design process should accommodate a series of revisions, because defects may only become apparent when the complete design is implemented and deployed in the sensing environment.

C. *Debugging*

Debugging large sensor network applications at deployment time is a challenging experience. Pre-deployment testing using simulations and controlled experiments over testbeds are extremely useful as they allow us to extract information about the external and internal states of each node. However, in a real deployment, collecting state information can be difficult, especially when the packaging is designed for deployment. For example, even if the EEPROM fully logs the transient internal states of each node, correlating them in a network-wide temporal order can be difficult, especially without time synchronization. In our deployment, we did not have adequate time to explore this option.

Instead, we exploit a large antenna to snoop on network traffic. This non-intrusive approach allows the collection of as much external states of the network as possible, does not affect the application, and enables a direct communication with each of the node in the network.

A set of services under the node management category in Figure 3 are implemented to address in situ debugging. Additionally, we place a version control number into each binary to ensure code compatibility across all the nodes in the network. We use a basic “ping” like service to verify that a node is up. The ping reply also reports the version control number of its code binary, allowing us to detect incompatible binaries. In addition, some of the basic primitives for node management such as node reset, sleep, and active mode control are also supported over wireless control.

The big antenna allows us to remotely control and debug each node in the network. We implement a set of management scripts on a PC computer to invoke the sensor node management services to administrate the system through the antenna. Packet traces are archived for off-line debugging and visualization of the entire system to understand the global behavior, which is extremely useful in system tunings. Nodes can send packets with an ASCII text payload to act as a “printf” to signal the occurrence of some critical debugging events in a human readable form.

For larger, real world deployments, we have since developing a multi-hop system management architecture [39] to subsume the role of the big antenna. This lower layer can perform system health monitoring, remote control, and data logging; and, it integrates seamlessly with a dispersed, higher power second tier to optimize data gathering. We look forward to reporting on the success of this architecture for real deployments in future work.

D. *Hierarchy of Programming and Reconfiguration*

In sensor networks, the need for a form of *in situ* programming presents a new kind of requirement for remote configuration tools. Besides the common need for wireless network-wide reprogramming, there is also a need to perform in situ protocol parameter tuning since analytical analysis is often insufficient to accommodate environmental effects. For example, there are configuration options of the code that need to be decided at the time of deployment, including the application’s sensing policy, sensor calibrations, and communications parameters that rely on the cell density. Furthermore, some of these configurations may need to be set on varying granularities, ranging from individual nodes, a select few subset of nodes, to the entire set of nodes en masse. We have implemented both the network reprogramming and config services as shown in Figure 3 to address these needs.

Our design supports wireless network programming, which is an alternative solution to installing new binaries over many nodes by hand. For a team of five people working with one hundred nodes, manual programming takes two hours with an additional two hours to re-deploy the nodes in the field. This approach is clearly not amenable to a rapid debug and test cycle.

Using network programming, nodes receive the binary image over the radio. By exploiting the shared wireless medium, many nodes can be reprogrammed simultaneously and selectively. We anticipated using network reprogramming for our deployment, but we could not develop a sufficiently reliable network reprogramming mechanism for our purposes[2]. Given the problems we encountered at the time, the entire process would have taken longer than individually reprogramming each node.

Interestingly, with a network service we call Config, the limitations of manually programming each node and our inability to use network reprogramming did not pose a great hindrance in our deployment. We spend the majority of our time tuning the algorithms to work properly at scale in the environment. The Config service addresses this issue efficiently and allows run-time adjustments of the internal states of each node. For example, Config allows us to selectively enable sections of the code, adjust parameters, modify calibration values, and adjust variables at run time.

Config is a smart configuration system that takes the place of a traditional approach to using a local configuration file per node. Configuration values are declared in the code with a specific configuration identifier, as shown in this example:

```
//!! Config 31 {uint16_t RFThreshold = 200;}
```

In this case, the RFThreshold parameter, with a default value of 200, is preprocessed with compilation tools to convert it to be a member in a global Config data structure.

[2] Subsequent work has improved upon our initial foray. [40]

Config is tightly integrated with the scripting environment in Matlab, allowing the large antenna to be used for debugging. Therefore, it is easy to change configuration values for a subset of the nodes or all the nodes from a PC in run time.

When a user changes a node's configuration value, the change is automatically reflected in that node's global Config data structure. And, the application is notified through an asynchronous event of the change to the data value. Config also supports queries of the current set of configuration values on each node. With a rich configuration capability in place and a bit of creative programming to utilize it, the resulting application is quite malleable, saving us a lot of time from installing new code images.

IX. Conclusion

Designing and implementing PEG enables us to establish relevant system design principles that are useful to other sensor networking systems. Our whole-system design analysis provides a clean process of problem decomposition. It allows complexity to be placed at the appropriate levels of the system to achieve overall simplicity in system implementation. Simplicity is further achieved by exploiting environmental and physical characteristics of the application at deployment time. Protocols should exploit soft state, loose consistency, and rapid fail over when appropriate to cope with the lossy wireless channel and the somewhat unreliable sensor network hardware platform. The system management and debugging infrastructure should be well designed to anticipate the need of system reconfigurations at deployment time.

Our system decomposition allows each of the subsystems to be reusable by a wide variety of sensor network applications. The neighborhood abstraction and leader election mechanisms apply to any monitoring system requiring local data aggregation. The density adaptive flooding mechanism avoids the broadcast storm problem for other data dissemination protocols. The landmark routing subsystem is useful for any application with moving entities. The network management and debugging services are useful for deploying other sensor networks. The data filter and robustness of the control system design are applicable to other sensor network applications with embedded actuators.

We demonstrate a working system that not only monitors sensory data but also tracks and controls a higher tier system to accomplish a cooperative task in real time. The system assumes very little processing and communication requirements on the sensor tier. Furthermore, throughout our design we exploit the physical properties of PEG to achieve a functional, simple design that is robust to failures. We believe the same design philosophy should be followed in building future sensing and actuating systems.

In the near future, we will deploy an order magnitude larger network to achieve many of the same goals as this work. We will leverage the lessons from this work to establish a platform well suited for long lifetime and large scale remote reprogramming, reparameterization, and system management. The overall goal of this redeployment is to focus on data gathering and methodical system study. In this new deployment, we will be able to introduce and measure greater variation: robot speed, node spacing, node topology, GPS resolution, sensing fidelity, sensing period, and so on. This initial effort described in this work has been invaluable for the experience, and we hope to extend that with a breadth of experiments that describe in detail the behavior of the many facets of this kind of system and application.

Acknowledgments

We'd like to thank everyone who worked on PEG, including: Phoebus Chen, Fred Jiang, Jaein Jong, Sukun Kim, Phil Levis, Neil Patel, Joe Polastre, Robert Szewczyk, Terrence Tong, Rob von Behren, and Kamin Whitehouse. This work is funded in part by the DARPA NEST contract F33615-01-C-1895 and Intel Research.

References

[1] D. Estrin, L. Girod, G. Pottie, and M. Srivastava, "Instrumenting the world with wireless sensor networks," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001)*, 2001.

[2] G. Pottie and W. Kaiser, "Wireless integrated network sensors," in *Communications of the ACM*, May 2000.

[3] T. Parsons, "Pursuit-evasion in a graph," in *Theory and Application of Graphs (Y. Alani and D.R. Lick, eds.* Springer-Verlag, 1976, pp. 426–441.

[4] I. Suzuki and M. Yamashita, "Searching for a mobile intruder in a polygonal region," in *SIAM J. Comput.*, vol. 21, October 1992, pp. 863–888.

[5] R. Vidal and S. Sastry, "Vision-based detection of autonomous vehicles for pursuit-evasion games," in *IFAC World Congress on Automatic Control*, 2002.

[6] H. Pasula, S. Russel, M. Ostland, and Y. Ritov, "Tracking many objects with many sensors," in *In Proceedings of IJCAI-99*, 1999.

[7] D. Reid, "An algorithm for tracking multiple targets," in *IEEE Transactions on Automatic Control*, vol. 24:6, 1979.

[8] J. Hespanha and M. Prandini, "Optimal pursuit under partial information," in *In Proceedings of the 10th Mediterranean Conference on Control and Automation*, July 2002.

[9] R. R. Brooks and P. Ramanthan, "Distributed target classification and tracking in sensor networks," in *In Proceedings of the IEEE*, August 2003, pp. 1163–1171.

[10] J. Liu, J. Liu, J. Reich, P. Cheung, and F. Zhao, "Distributed group management for track initiation and maintenance in target localization applications," in *In Proceedings of the 2nd Workshop on Information Processing in Sensor Networks (IPSN '03)*, April 2003.

[11] F. Zhao, J. Liu, J. Liu, L. Guibas, and J. Reich, "Collaborative signal and information processing: An information directed approach," in *Proceedings of the IEEE*, 2003, pp. 91(8):1999–1209.

[12] B. Blum, P. Nagaraddi, A. Wood, T. Abdelzaher, S. Son, and J. Stankovic, "An entity maintenance and connection service for sensor networks," 2003. [Online]. Available: citeseer.ist.psu.edu/blum03entity.html

[13] M. Welsh and G. Mainland, "Programming sensor networks using abstract regions," in *In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI' 04)*, March 2004.

[14] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus, "Tracking a moving object with a binary sensor network," in *Proceedings of the first international conference on Embedded networked sensor systems*. ACM Press, 2003, pp. 150–161.

[15] K. Birman, "The process group approach to reliable distributed computing," in *Communiation of the ACM*, vol. 36(12), 1993, pp. 37–53.

[16] D. Cheriton, "Understanding the limitations of causally and totally ordered communication," in *SOSP*. ACM, December 1993.

[17] "Anonymous xxx fixme xxx," in *Citation omitted for blind reviewing; we will make this available to any of the program chairs at the reviewer's request*.

[18] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *Proceedings of ACM ASPLOS IX*, November 2000, pp. 93–104.

[19] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler, “Hood: a neighborhood abstraction for sensor networks,” in *In Proceedings of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys ’04)*. ACM Press, June 2004.

[20] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao, “Habitat monitoring: Application driver for wireless communications technology,” in *ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, April 2001.

[21] S. Madden, “The design and evaluation of a query processing architecture for sensor networks,” Ph.D. dissertation, UC Berkeley, 2003.

[22] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, “Lessons from a sensor netowkr expedition,” in *In the 1st European Workshop on Wireless Sensor Networks (EWSN 04)*, January 2004.

[23] D. Johnson and D. Maltz, “Dynamic source routing in ad hoc wireless networks,” in *Mobile Computing*. Kluwer Academic Publishers, 1996, pp. 153–181.

[24] C. E. Perkins and E. M. Royer, “Ad hoc on-demand distance-vector (aodv) routing,” in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[25] V. Park and M. Corson, “A highly adaptive distributed routing algorithm for mobile wireless networks,” in *In Proceedings of the IEEE INFOCOM ’97*, April 1997.

[26] C. E. Perkins, B. Woolf, and S. R. Alpert, “Mobile ip design principles and practices,” January 1998.

[27] Q. Fang, J. Li, L. Guiba, and F. Zha, “Roamhba: maintaining group connectivity in sensor networks,” in *Proceedings of the third international symposium on Information processing in sensor networks*. ACM Press, 2004, pp. 151–160.

[28] P. Tsuchiya, “The landmark hierarchy, a new hierarchy for routing in very large networks,” in *Special Interest Group on Data Communication (SIGCOMM)*, 1988, pp. 36–42.

[29] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Gouda, Y. Choi, T. Herman, S. Kularni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita, “Line in the sand: A wireless sensor network for target detection, classification, and tracking,” in *OSU-CISRC-12/03-TR71*, 2003.

[30] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu, “The broadcast storm problem in a mobile ad hoc network,” *Wirel. Netw.*, vol. 8, no. 2/3, pp. 153–167, 2002.

[31] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker, “Complex behavior at scale: An experimental study of low-power wireless sensor networks,” in *Technical Report UCLA/CSD-TR 02-0013*, February 2002.

[32] J. Zhao and R. Govindan, “Understanding Packet Delivery Performance in Dense Wireless Sensor Networks,” in *Proceedings of the first international conference on Embedded networked sensor systems*. ACM Press, 2003, pp. 1–13.

[33] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, “A high-throughput path metric for multi-hop wireless routing,” in *Proceedings of the 9th annual international conference on Mobile computing and networking*. ACM Press, 2003, pp. 134–146.

[34] W. J. Rugh, *Linear System Theory*, 2nd ed., ser. Information and System Sciences Series. Upper Saddle River, New Jersey 07458: Prentice Hall, 1996.

[35] P. Marti; G. Fohler, K. Ramamritham, , and J. M. Fuertes, “Jitter compensation for real-time control systems,” in *22nd IEEE Real-Time Systems Symposium*, London, December 2001.

[36] P. Marti; R. Villa; J. M. Fuertes, and G. Fohler, “Stability of on-line compensated real-time scheduled control tasks.” in *IFAC Conference on New Technologies for Computer Control*, Hong Kong, November 2001.

[37] B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House, February 2004.

[38] P. Varaiya and P. Kumar, *Stochastic Systems: Estimation, Identification, and Adaptive Control*, ser. Information and System Sciences Series. Upper Saddle River, New Jersey 07458: Prentice Hall, 1986.

[39] G. Tolle and D. Culler, “Snms – xxx fixme xxx,” in *This work has been submitted and is currently under review; we will make this available to any of the program chairs at the reviewer’s request.*

[40] N. Reijers and K. Langendoen, “Efficient code distribution in wireless sensor networks,” in *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*. ACM Press, 2003, pp. 60–67.