A Networked Robot System for Wireless Network Emulation

Tzi-cker Chiueh Rupa Krishnan Pradipta De Jui-Hao Chiang

Computer Science Department
Stony Brook University

***Abstract*—A major barrier to advancing modern wireless networking research is the lack of an effective wireless network simulation platform that simultaneously offers high fidelity, scalability, reproducibility and ease of use. MiNT [8], [7] is an innovative wireless network emulation platform that is specifically designed to satisfy all these desirable properties. To support reconfigurable network topology and wireless node mobility, MiNT is built on a networked robot system that carries wireless networking equipments and is designed to be completely tetherless, capable of supporting 24X7 operation, and low-cost. This paper describes the design, implementation and evaluation of this networked robot system. Each robot node in MiNT is an iRobot's Roomba, which is modified to house an embedded PC equipped with multiple wireless networking interfaces and to re-charge the embedded PC through Roomba's built-in self-charging mechanism. For robot navigation and movement, MiNT's networked robot system supports a computer vision-based robot positioning mechanism and a collision avoidance-driven trajectory planning component. Finally, MiNT provides an interactive control interface and visualization interface to give users real-time visibility into and full control over the MiNT testbed.**

I. Introduction

An ideal wireless network simulation system should provide high fidelity in the sense that results of simulation runs are as accurate as measurements taken on real wireless networks, should be able to scale to large wireless networks in terms of the number of network nodes and network flows, should produce the same results for repeated runs of the same experiments, and finally should be easy to use in terms of simulation run set-up and execution. The most prevalent approach to wireless network simulation in the research community today is software-based simulation based on simulator such as ns-2, Glomosim, etc. Software-based wireless network simulators in many cases fail to faithfully capture many real-world radio signal propagation effects such as non-uniform path loss, interference, and multi-path fading [11]. Because of this fidelity limitation, some researchers turn to physical wireless network testbed as an alternative. Although several on-going efforts aim to construct a high-fidelity wireless network testbed for wireless protocol experimentation, they share some of the following weaknesses. First, these testbeds provide limited flexibility in terms of specifying initial wireless network topology with specific signal-to-noise ratios and node mobility patterns at run time. Second, because the operation of these testbeds requires non-trivial human intervention, for example, run-time node mobility, they cannot serve as an autonomic shared infrastructure that supports 24x7 availability, like a standard supercomputer center. Finally, the management and control interfaces of most existing wireless network testbeds are put together in an ad hoc way and often rather primitive.

MiNT [8], [7] is a multi-hop mobile wireless network testbed that (1) offers the flexibility of experiment-specific network topology reconfigurability, (2) supports untethered node mobility during simulation runs, (3) sustains 24x7 autonomous operation, (4) provides comprehensive monitoring and control of the testbed activity, (5) enables hybrid ns-2 simulations and (6) reduces significantly the testbed's physical space requirement using radio signal attenuation. The enabling technology behind MiNT is a networked robot system that carries wireless network nodes and supports network reconfigurability and node mobility. Though conceptually simple, there are several technical challenges in designing and implementing this networked robot system. First, the networked robot system must be battery-operated and self-rechargeable so that it could be completely tetherless and yet support 24x7 continuous operation. Second, to set up a given wireless network topology or to enforce a particular wireless node movement pattern, an accurate robot positioning mechanism is required to track and control the position of each robot. Finally, to grow a mobile wireless network testbed to a large size, larger than 100 nodes, the cost of each robot must be low, and the design of various testbed control functions, such as node movement and position tracking, must be scalable.

Each MiNT node is built on a low-cost commodity robotic vacuum cleaner from iRobot Corporation called Roomba [19], which supports a flexible application programming interface (described later) for external control, is able to carry a large payload, and comes with an effective auto-recharging capability. Mounted on each Roomba is a wireless network node supporting four 802.11 interfaces, each of which is attached to an antenna through a radio signal attenuator to reduce its signal coverage and therefore th testbed's physical space requirement. Roomba's auto-recharging circuitry is modified to power up both a Roomba and the wireless network node it carries. Moreover, a *recharge scheduling algorithm* is developed to determine the next recharge time for each MiNT node. Finally, MiNT incorporates a *computer vision-based* positioning system to track the position of each Roomba robot. This positioning system accurately tracks the robots with zero false positive, and requires only commercial off-the-shelf webcams. The resulting position information is used in both node monitoring and trajectory planning for collision-free robot movement.

The rest of this paper is organized as follows. Section 2 reviews previous works in wireless network testbeds and networked robot systems. Section 3 gives an overview of MiNT's

ROBOCOMM 2007, 14th – 16th Oct 2007, Athens, Greece.

ISBN 978-963-9799-08-0
DOI 10.4108/ICST.ROBOCOMM2007.2167

networked robot system including its hardware components and the systems software base on which it is built. Section 4 describes the computer vision-based positioning system used in MiNT. Section 5 presents the navigation and control system for robot movement in MiNT. Section 6 concludes this paper with a summary of main research contributions and a brief outline of future work.

II. Related Work

Large physical space requirement not only makes management of a wireless testbed difficult, but also adds to its operational cost significantly. This is true for most full-scale wireless network testbeds, like CMU-DSR [15], APE [14], RoofNet [4] and TAP [13]. The set-up cost of some wireless network testbeds, such as Netbed [24], WHYNET [22] and ORBIT [18] is very high, although their cost may be justified if they are used as heavily as their designers hoped for. In this work, we emphasize the flexibility of network topology reconfiguration and node mobility support. Testbeds like Roofnet used a fixed topology and therefore are less flexible. A shared testbed must be rich in its applicability to diverse scenarios, like live experimentation, emulation or simulation. WHYNET, ORBIT and Netbed propose to address a truly diverse set of scenarios compared with others.

Ideally, movements of mobile nodes should be remotely controlled in a programmatic way. Mobile Emulab [9] uses 4 Acroname Garcia robots for mobility. These robotic platforms cost over $1000 a piece, as opposed to the Roomba robotic vacuum cleaners ($249 a piece) [19] that MiNT uses. Moreover, Netbed's robots must be manually taken to their charging bases every 2-3 hours for recharge. Roomba comes with an auto-charging feature, which makes our mobile testbed truly autonomous. The MiNT testbed also pioneers the use of radio signal attenuators to miniaturize the testbed [8]. This greatly reduces the arena of operation thus requiring a smaller number of overhead cameras to track the nodes.

A key design issue in supporting node mobility is to ensure collision-free movement of the robots through careful path and motion planning. Existing literatures [5] have explored robot motion planning for various complex scenarios. Since our testbed offers a much controlled environment, we explore a heuristic that is lightweight and computationally efficient. In contrast to the motion planning algorithm used in Mobile Emulab, because the Roombas do not have object sensor, MiNT integrates obstacle detection into the object tracking subsystem.

Associated with mobility feature is the use of a tracking system for accurately determining the position and orientation of each node. Several previous systems also used a vision-based tracking method to track mobile nodes. Graham and Kumar [10] use ceiling-mounted cameras and colored patterns on toy cars to track them. They use 8 colors and an error-correcting 3x2 colored pattern to track the cars' position in real time. Their system is designed to track up to 22 mobile nodes and is able to uniquely identify nodes as well as provide their position and orientation. MiNT's robot position tracking system used a similar color-based mapping technique, with additional optimizations. Cremean et al. [6] use ceiling-mounted monochromatic camera and binary (black/white) patterns to compute the position and orientation of the nodes. Concurrent to this work, Johnson et al. [24] have also implemented a centralized object tracking system that uses ceiling mounted camera and color patterns to determine the position and orientation of the mobile nodes in their wireless testbed. Their tracking system does not uniquely identify each tracked node, instead locality and motion pattern information is used to determine the identity of nodes.

III. System Architecture

A. Hardware Components

A MiNT testbed consists of a collection of wireless mobile robots managed remotely by a central controller. In addition, there is a position tracking server that determines the exact location of each wireless robot. The overall interaction of the hardware and software components of a MiNT testbed is illustrated in Figure 1.

Mounted on each mobile robot is an embedded computer, an RB-230 RouterBoard, which is a low-power battery-operated small form-factor board equipped with 4 mini-PCI IEEE 802.11 a/b/g wireless LAN (WLAN) cards. Inserted between each WLAN card and its antenna is a radio signal attenuator that decreases the strength of transmitted/received signals and thereby the testbed's physical space requirement. The mobile robot is a commodity robotic vacuum cleaner, Roomba from iRobot. Several modifications were made to the Roomba to allow the central controller to control each Roomba's movement and to leverage Roomba's self-charging capability to charge the the wireless embedded computer's battery. The central controller is a PC that controls the movement of each wireless robot and collects network simulation results in real time. The positioning server is responsible for keeping track of the position and orientation of each mobile robot in the testbed by applying computer vision algorithms on video signals captured from an array of video cameras hanging over the testbed. Figure 2 shows a MiNT prototype that consists of 12 wireless Roomba robots, with the charging stations at the top left corner of the image.

B. Software Components

The key software components in MiNT are: (a) the control daemon running on the central controller, (b) the robot daemon residing on each testbed node, and (c) the interactive monitor and control interface called MOVIE (MiNT cOntrol and Visualization InterfacE). The control daemon collects position updates of testbed nodes from the position tracking server, as well as simulation event traces from the testbed nodes, which it uses to update the visualization interface. It also communicates user-issued control commands, in the form of robot movements or configuration changes, to target robot daemons. Based on the received commands a robot daemon instructs its underlying robot to move accordingly. Because all simulation events from the testbed nodes are routed through the central controller, the control daemon maintains a complete log of the testbed activities during a simulation run. Robot daemons communicate with the central control daemon over an IEEE 802.11g channel that is chosen at start-up time. Other programs running on testbed nodes, for example, the ns-2 network simulator, the TCP load generator, and the RF monitoring agent, go through the robot daemon to communicate

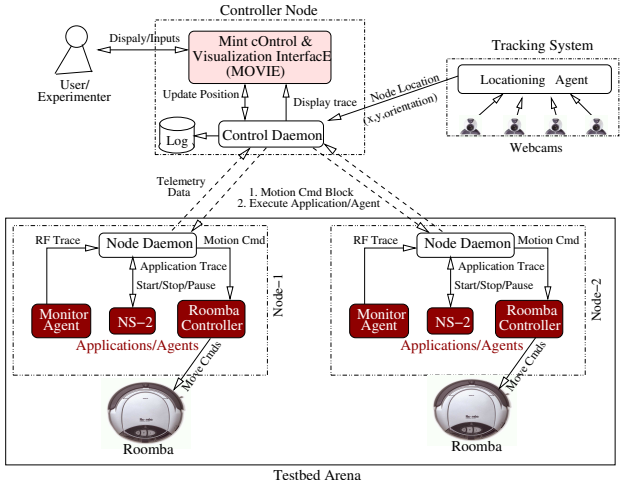

Fig. 1. *In a MiNT testbed, the control daemon running on the control server collects inputs from the robot tracking server and the user, and controls the movement of individual MiNT nodes. It also includes the MOVIE interface for the user to monitor and control the testbed. Each testbed node is a wireless embedded computer mounting on a Roomba robot and runs a node daemon program that communicates with the control daemon over a dedicated wireless control channel. Testbed nodes communicate with peers using wireless NICs that are connected to low-gain antennas through radio signal attenuators. The vision-based tracking server periodically captures images of the testbed area, and processes them to derive the position/orientation of each testbed node.*

Fig. 2. *The current MiNT prototype consists of 12 Roomba robots, each with a wireless computer mounted, and several charging stations for charging their batteries.*

with the central controller. MOVIE provides a comprehensive monitor and control interface that offers real-time visibility of the testbed activity and supports full interactive control over testbed configuration and hybrid simulation runs. MOVIE is derived from Network Animator (NAM), a well-known off-line visualization tool for ns-2 traces. Several enhancements were made to NAM to support real-time monitoring and controlling simulation runs and for interactive debugging of simulation results, such as protocol-specific breakpoints and simulation state rollback.

IV. MiNT Node Design

A. Miniaturization

Setting up a multi-hop wireless network for experimentation is a grueling exercise mainly because the physical space requirement entails a painful logistical effort. For example, the communication distance between a pair of wireless nodes operating in the 2.4 GHz frequency range and using off-the-shelf IEEE 802.11b wireless NICs is around 300 feet; any non-trivial multi-hop wireless network built on this radio technology can easily span a geographical area with a 1000-foot diameter. To set up such a multi-hop wireless network with a specific signal-to-noise ratio between each pair of nodes is not only painstaking and time-consuming, but also error-prone. The logistic effort involved no doubt will have an adverse impact on the number of experiments that researchers can afford to run on such testbeds. Worse yet, simulation of node mobility is also problematic in these testbeds, as they require manual efforts, for example, putting a laptop on a car and driving it around the campus, as in CMU-DSR [15], or asking volunteers to carry mobile devices in an orchestrated manner, as in APE at Uppsala University [14].

The simplest way to limit radio signal propagation distance is to reduce the transmit power on the wireless network interface. One can use a laptop or a PDA with a commercially available wireless interface card that allows setting the transmit power to different values, like 100mW, 50mW, 10mW, 5mW and 1mW. However, experiments on commercial WLAN cards such as Cisco Aironet 350 series show that even at 1mW, the radio coverage, about two mid-sized rooms, is still too large to fit the space budget of MiNT. Therefore, we switched our attention to *radio signal attenuators*, which are designed to decrease the power level of transmitted and received radio signals. Radio signal attenuators are available in two types, *viz.* fixed signal attenuators and programmable attenuators. The fixed signal attenuators are priced in tens of dollars, whereas the programmable attenuators usually cost $1000 a piece. Therefore, we chose to use fixed signal attenuator in the MiNT node design to reduce its per-unit cost. The extent of attenuation (dB rating) is determined based on the desired range of radio signal propagation.

Use of attenuator, which is external to a wireless interface card, requires the use of an external antenna. The attenuator is connected between the wireless interface card and the external antenna using an RF cable with suitable connectors. The problem with this approach is that most commercially available wireless interface cards come with an internal antenna, which does get fully disabled even when the card is attached to an external antenna and radiates enough RF energy to defeat the goal of restricting each wireless interface's radio signal coverage. There are two ways to overcome this problem. One option is to physically cut the internal antenna from the radio signal path, which is reported to be used in the wireless Emulab testbed project [12]. The other option is to use a wireless interface card that does not have an internal antenna. However, such cards are only available in miniPCI and PCI format. Because the RouterBoard platform we chose, RB-230, provides only 1 PCI slot, we opted for miniPCI wireless interface cards and attached multiple of them to a miniPCI to PCI converter, which in turn is plugged into RB-230. Multiple wireless interface cards provide us the flexibility to carry out experiments requiring multiple wireless interfaces per node [17]. To further attenuate radio strength, we used low-gain external antennas, whose power gain is 2dBi.

B. Support for Node Mobility

To support network topology reconfiguration and node mobility requires each MiNT node to be mounted on a mobile robot. The key criterion used in choosing MiNT's mobile robot platform are: (a) low cost, (b) minimal assembly effort, and (c) remote controllability. Hobby robots provide an inexpensive option, but demand substantial assembly efforts. Robots typically used in many commercial and robotics applications, like AmigoBots [2], PatrolBots or Acroname Garcia [1] robots, are very expensive, often going into thousands of dollars per unit. Eventually, instead of choosing standard experimental robotic platforms, we choose the *Roomba robotic vacuum cleaner* from iRobot as the mobility platform for MiNT nodes, for the following three reasons. First, Roomba is a consumer-grade product that carries an inexpensive price tag: its retail price of $200 at the time of our purchase. Second, Roomba can carry a payload of up to 30 pounds, an essential capability to host MiNT's wireless computing platform. Third, Roomba supports a auto-charging capability, which is essential to support one of MiNT's design goals: 24x7 autonomic operation.

Designed primarily to be a vacuum cleaner, Roomba originally did not provide any open API for controlling its movements. We overcame this limitation through a clever use of its IR-based remote control facility. More specifically, we achieve arbitrary Roomba movement using two primitives: (1) move the mobile robot forward, and (2) turn the robot clockwise or counter-clockwise. A Roomba can be instructed to perform these primitives through an out-of-the-box infrared-based remote controller that comes with the Roomba. We learned Roomba's remote control codes using a programmable remote controller called Spitfire [21]. To move a Roomba, MiNT's central control server sends a movement command to the RouterBoard on the Roomba, which relays a corresponding command to a Spitfire controller over a serial port, and eventually the Spitfire controller issues the associated infrared code to instruct the Roomba to move accordingly.

Since 2006, iRobot rolled out a new version of Roomba that supports an electronic and software interface for remotely controlling and modifying Roomba's behavior and monitoring its sensors. This interface is known as the Roomba Serial Command Interface or Roomba SCI [20]. The SCI provides commands to control all of Roomba's actuators, such as its motors, as well as retrieve internal sensor data such as residual battery capacity. Roomba SCI simplifies several aspects in the design of MiNT, with an associated reduction in the overall cost. With SCI, a RouterBOARD is directly connected via a serial cable to a Roomba's serial port, which is a 7-pin mini-din connector. Because this connector does not work on the voltage levels of a standard RS232 connection, a RS232 level shifter is needed to convert RS232 voltage levels to those accepted by Roomba.

The software interface in the SCI expose various internal functionalities in Roomba, including move forward, move backward, rotate a certain angle, dock to a charging station, undock, report residual battery charge of the Roomba and a host of other useful features. The following are the most important primitives provided by SCI:

void command(char*, int): This function takes a string, which specifies the command to execute, and an integer, which is unused, as input. The command string could be one of the following:

- **forward:** move forward by a single step or 20 cm
- **right:** turn by a angle of 10 degrees clockwise
- **left:** turn by a angle of 10 degrees counter-clockwise
- **back:** move backward by a single step or 20 cm
- **power:** turn on the power
- **pause:** stop Roomba from moving
- **dock:** seek the nearest docking station and dock
- **undock:** leave the current docking station and wait for further commands

void getpacket3(): This function reads the third sensor data packet that Roomba sends out. It contains information about Roomba's charging state, battery capacity, voltage and currents of Roomba's battery. This function sets the boolean variable `chargestate` to true if Roomba is found to be charging and sets it to false otherwise. The two integer variables `charge` and `capacity` are populated with current battery charge and total battery capacity, which is usually a constant 2700 for all batteries and the current battery charge varies between 0 and 2700.

short int getdistance() This function returns the distance traveled by a Roomba since the wheel sensors were last polled using **markpoint()** function, which essentially serves as a reset. Because wheel sensor values are only 2-byte long, Roomba can measure a maximum distance of -32,768 mm to +32,767 mm.

SCI also contains a serial port module which is used to establish a serial port communication channel with Roomba, including the set-up of initial communication baud rate.

C. A Complete MiNT Node

The complete design of a MiNT node involves setting up an embedded computer that serves as a wireless communication node, and mounting it on a Roomba robot. Taking into account cost, form factor, power consumption and I/O extensibility, we chose RouterBoard's RB-230 board as the embedded computing platform for the MiNT node. RB-230 is a small-form-factor PC with a 266-MHz processor and runs on an external laptop battery. It also comes with a PCI extension board (RB-14), which allows one to attach 4 Atheors-based 802.11 a/b/g mini-PCI cards, each of which is connected to a 2 dBi external antenna through a 22 dB radio signal attenuator. This adds a total of 44 dB attenuation on the signal path from transmitter to receiver and thus makes it possible to deploy a 12-node MiNT prototype within a space of 132.75 inches X 168.75 inches, as shown in Figure 2. In addition to the fixed signal attenuators, MiNT also allow users to modify the transmit power on the mini-PCI WLAN cards to provide additional flexibility in tuning inter-node signal-to-noise ratio.

Figure 3 shows the current MiNT node prototype. There are two shelves mounted on the Roomba. The laptop battery and the Spitfire universal remote controller sit on the lower tier, while the RouterBoard based wireless networking module sits on the top tier. The four external antennas are mounted on poles located at four corners.

V. Vision-based Robot Positioning

A. Overview

The robot positioning system in MiNT is necessary to maintain the location and orientation information of each

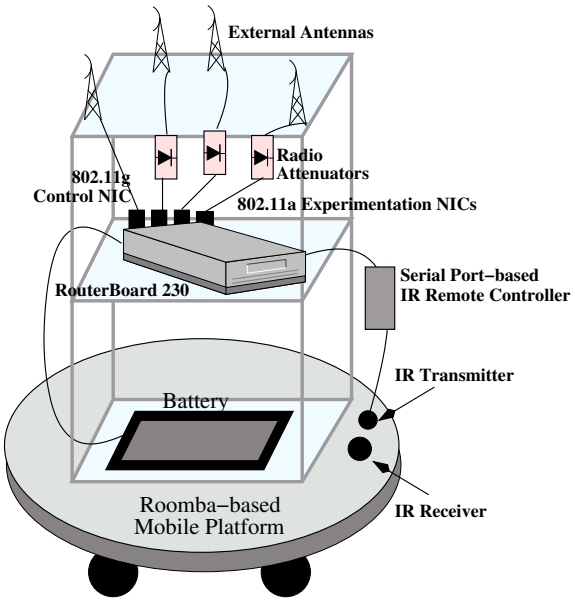

Fig. 3. *A MiNT node comprises of a RouterBoard (RB-230) powered by an external laptop battery, and a Roomba robotic vacuum cleaner whose movement is controlled by a Spitfire Universal Remote Controller. The RouterBoard is equipped with 4 wireless NICs each connected to a separate omni-directional external antenna via a radio signal attenuator.*

MiNT node in real time, and to display it through the control GUI. One simple way to track the position of a mobile robot is to use its odometry data, which tells the distance that a robot travels from a starting point, and the angle rotated with respect to a fixed direction. However, even the new SCI from Roomba does not provide such information. In theory, it is possible to maintain the odometry information based on the movement commands sent to a mobile robot. In practice, factors such as mechanical inaccuracies and difference in floor friction can lead to noticeable errors. Therefore, MiNT needs a more accurate robot positioning system. One option is to use RF/ultrasound-based indoor local positioning systems such as Cricket [16]. However, this option increases the per-node cost, and introduces additional RF interference. Therefore, we choose an optical or computer vision-based position/orientation tracking system that only requires off-the-shelf webcams. The design goal of this tracking system is to uniquely identify each MiNT node, and pinpoint its (X, Y) position and orientation (θ).

Compared with general object tracking, the complexity of MiNT's object tracking system is inherently less demanding because it can afford several simplifying assumptions. First, the lighting condition in the room housing the MiNT testbed does not change much. Consequently, it is not necessary to dynamically account for fluctuation in lighting condition once the color profiles have been calibrated for the initial lighting condition. Secondly, color patterns used to identify individual MiNT nodes can be chosen such that they are different from the background color, in this case the floor's color. Thirdly, placement of webcams that periodically take snapshots of the testbed nodes does not change once they are mounted.

MiNT assigns to each testbed node a unique ID, which consists of multiple colors. Through a simple color recognition algorithm, MiNT could reliably identify 8 distinct colors and

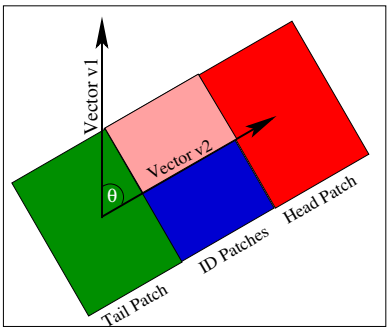

Fig. 4. *The color patch on every node has the same head and tail patch. The vector from the centroid of the tail patch to the centroid of the head patch is used to determine a MiNT node's direction and thus its orientation in the testbed arena. The node location and identification are done using the center patches.*

therefore could potentially support up to hundreds of testbed nodes. Reliable and fast identification of unique colors is at the heart of MiNT's object tracking algorithm. We use the HSV (Hue, Saturation, Value) space to represent colors because the distribution of colors is more uniform in this space, and the Hue and Saturation components are orthogonal to the Value (or brightness) component. To determine the set of colors used in node ID, MiNT first computes the HSV profiles for a number of colors, and eventually arrives at 8 colors that can be clearly distinguished based on at least one of the H, S, or V components. Because the HSV profile for the same color may change substantially from one camera to another owing to subtle differences among CCD sensors in these cameras, we profile each camera separately to arrive its corresponding profiles for these eight colors.

It is possible to use high-resolution cameras with wide-angle lens that support a larger viewport, but come at a cost on the order of thousands of dollars. Similarly, to support high frame rates, it is possible to use hardware decoders on the tracking server. In the end, we decide for our application it is sufficient to use commodity webcams and settle for Logitech Quickcam Pro 4000, which costs $76 and supports a resolution of 320x240 at up to 15 frames/sec.

B. Object Tracking

MiNT associates a four-color pattern with each testbed node, as shown in Figure 4. The head and tail color patches are the same for all testbed nodes. Only the center patch, which consists of two colors, is used in node identification. In particular, the *location* of a testbed node is the centroid of the ID or center patch. The *orientation* of a testbed node is determined based on its direction, which in turn corresponds to the vector connecting the centroid of its tail patch to that of its head patch. Using the same colors for the head and tail patches introduces redundancies that can guard against noises and simplifies the determination of robot orientation. There are two main steps in the identification and positioning of a node: (a) finding the presence of a color patch in the snapshot image, (b) reducing the parsing overhead of each image by using inter-frame coherence in the images captured.

The color recognition algorithm used in MiNT is extremely simple and thus scalable, and does not require any complicated image processing techniques, such as edge detection. Once an image is captured, its pixels are scanned one by one in the scan

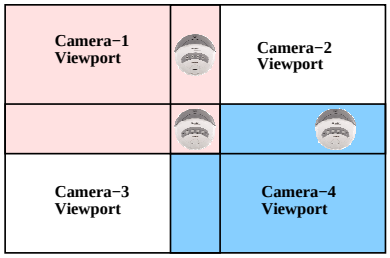

Fig. 5. *The schematic shows the placement of multiple cameras such that viewport overlaps. The overlapped areas are such that at any point in time a node must be fully covered by a camera; none of the camera captures a partial image of the node.*

line order. If a pixel of a known color is detected, then the pixels in its immediate 1-pixel neighborhood are checked for similarity and the pixel block is further expanded. It is possible to detect multiple blobs of the same color in an image captured by a webcam. This can happen if multiple nodes fall within the coverage of one webcam, which could be easily identified by multiple pairs of head and tail patches. It can also arise because of optical noises, which in turn can result because some portions of a color patch may fade, leading to disconnected blobs within a single patch. MiNT uses a merging technique to discern the noise case. For all the blobs of same color, MiNT merges them into a larger blob if their centroids are within a distance less than a pre-defined *merge threshold*, which is chosen so as to prevent merging same-color patches from different nodes, while being able to accommodate blobs that belong to the same patch.

MiNT exploits frame-to-frame coherence to further improve the efficiency of its object tracking algorithm. Given the knowledge of a node's position/orientation at the time when the last image is captured and the temporal distance between the last and the current captured images, MiNT computes a bounding box for that node and scans only pixels in that bounding box to determine the node's current position/orientation. This optimization improves the tracking efficiency because it eliminates the need to scan irrelevant pixels, as well as the tracking accuracy, especially when multiple nodes are close to each other, because it provides additional hints to disambiguate uncertainties.

Because a single webcam cannot cover the entire testbed arena, MiNT uses multiple webcams to fully cover the entire testbed arena. As a node moves from the viewport of one camera to that of another camera, the object tracking algorithm needs to perform necessary "handoff" to continue the node's tracking. Instead of using such handoff approach, MiNT solves this problem by mounting the cameras such that the view ports of adjacent cameras overlap and the overlapped area is large enough to cover a node completely. Because each MiNT node is always fully covered by at least one camera under this camera arrangement, there is no need to coordinate adjacent cameras, as shown in Figure 5.

C. Implementation Issues

In MiNT's robot position system, whose software structure is shown in Figure 6, there is a webcam server that processes images captured by each webcam, and an integrator server that merges node positioning results from webcam servers to generate the corresponding coordinates in the 2D space

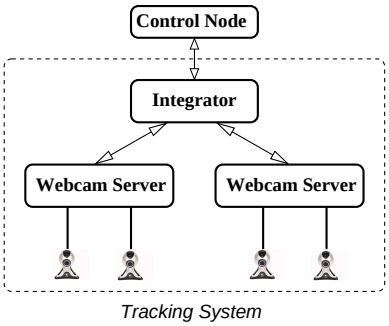

Fig. 6. *MiNT's robot position tracking system consists of a set of webcam servers, which collect images captured by webcams, and an integrator server, which translates node positions reported by webcam servers to global coordinates and sends them to MiNT's control server for display in the control GUI.*

representing the testbed arena and and sends them to MiNT's control server Multiple webcam server instances can run on a single machine along with the integrator server. As the number of webcams increases, the webcam server instances can run on a cluster of PCs instead.

There are several frame grabbing APIs for acquiring images captured by webcams. The frame grabbing APIs on the Windows platform suffer from poor scaling, and give a very low frame rate even when only 2 webcams are plugged into a webcam server machine. VideoForWindows API is unable to support more than one webcam on a single machine, whereas DirectShow API gives a low frame rate of 1.5 frames per second with only 2 webcams. Eventually we chose Camstream [3] to capture webcam images, which uses Video4Linux [23] as the image grabbing API. The driver for the Logitech Quickcam Pro 4000 webcam is Philips USB Webcam (PWC) driver. As a result, the current MiNT prototype is capable of capturing 320x240 frames at 15 frames per second even when 6 webcams are connected to a single machine with a 2.8-GHz X86 processor.

When the integrator server starts, it first listens for a connection request from the control server. Once it connects to the control server, it establishes a connection to the webcam servers, and starts collecting node positioning data periodically. It merges these node positioning data to compute their global coordinates and sends them to the control server. When a robot moves from one webcam's coverage area to another's, the server communicates this transition to the webcam servers associated with these two webcams. This communication allows the webcam that sees the robot for the first time to immediately leverage the bounding box information associated with the robot.

D. Evaluation

In the current MiNT prototype, 6 webcams are mounted at a height of about 9.1 feet with each covering a floor space of 87 inches X 66 inches, with the total testbed arena of 132.75 inches X 168.75 inches. Each webcam has a resolution of 320 x 240, thereby each pixel corresponds to 0.075 square inch area. Factors that affect the accuracy of MiNT's color-based position/orientation tracking algorithm are the size of each color patch, the number of distinct colors used, the stability of lighting condition, optical noise in the patch boundaries, which might distort centroid computation. Given the patch size and

the camera resolution used in the current MiNT prototype, a 1-pixel recognition error could potentially translate to 0.27 inch in location error, and 2.2 degrees in orientation. The positioning accuracy of MiNT's object tracking system is measured in terms of the difference between the location and orientation of a MiNT node as reported by the tracking system and its true coordinates. The measured mean error in position is 0.95 inches with a standard deviation of 1.17 inches, and the measured mean error in orientation is 3.36 deg. with a standard deviation of 2.77 deg.

Another important factor is the scalability of MiNTs object tracking algorithm with respect to the number of robots being tracked. This is measured by the end-to-end time required for positioning each robot, including frame grabbing, node identification, node positioning , and merging of position data from multiple webcam servers. With 12 robots, MiNT's tracking system is capable of producing one position update for every node once every 300 msec when it runs on a single machine.

The current room hosting the MiNT testbed has a limited height. If the mounting points of the webcams are moved higher, each webcam could cover a larger area and the tracking system can scale to a larger testbed size, although each robot's image is of lower resolution. To demonstrate that MiNT's tracking system can indeed work even with lower-resolution images, we scaled down the size of the images captured from the webcams, and ran the tracking algorithm on them. Even when each webcam image is shrunken to 1/16th of its original size (equivalent to placing the camera at 4 times the current height), the tracking system works well. In particular, the tracking systems positioning error just increased from 0.95 inches to 2.32 inches, while the error in orientation increased from 3.36 deg to 4.11 deg.

VI. Navigation and Control

To support node movement, MiNT's control server needs to compute the path for each robot, and sends the corresponding movement commands to the robots at the proper moment. In addition, MiNT constantly monitors each robot's position and orientation, and preemptively avoids collision. More concretely, MiNTuses a static path planning algorithm to compute each moving node's path assuming the rest of the world is static, and then resorts to a dynamic collision avoidance algorithm to detect and avoid collision by fine-tuning statically computed paths.

Given the current position and the target destination of a testbed node (TN), the control server takes a snapshot of the positions of other testbed nodes and treats them as obstacles during the calculation of TN's path. The static path planning algorithm first checks if there is a direct path between TN's current position and its destination. If such paths do not exist, the algorithm identifies the obstacle closest to the source position, and finds a set of intermediate points that lie on the line that passes through the obstacle and is perpendicular to the line adjoining the source and destination, and have a direct path to both the source and destination. If no such intermediate points exist, the algorithm finds a random intermediate point that is δ steps away from the obstacle closest to the source and is directly connected to the source, and repeats the algorithm from this new intermediate point as if it is a new source.

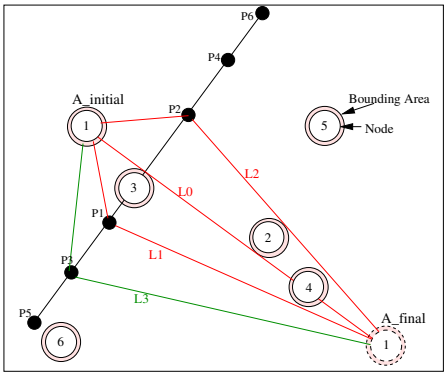

Fig. 7. *Finding the path from N1's current position $A_{initial}$ to A_{final}. As nodes N2, N3 and N4 block the direct path, the algorithm tries to identify an alternate 2-hop path to move N1 from its current source to its destination.*

Figure 7 illustrates how the static path computation algorithm works. Node N_1 is set to move from $A_{initial}$ to A_{final}. However, N_2, N_3 and N_4 block the direct path between $A_{initial}$ and A_{final}. The path planning algorithm first figures out that N_3 is the obstacle closest to $A_{initial}$, and then computes the intermediate points $P_1, P_2, ..., P_6$ to search for 2-hop paths to A_{final}. Because the paths L_1 and L_2 are partially blocked, the algorithm eventually chooses path L_3, which passes through the intermediate point P_3.

In addition to static path planning, MiNT also requires a dynamic collision avoidance algorithm because other MiNT nodes could also be moving and the robot movement is not perfect. Given a snapshot of the testbed at 15 frames per second in the current prototype, MiNT performs a proximity check for each testbed node. If any two nodes are closer than a threshold distance, including when they already collide with each other , the control server stops both of them, computes a new path for each of them if necessary, and moves them on their new path *one after the other*.

The main reason why dynamic collision avoidance is still needed is because the control server does not have a perfect control over Roomba's movement and there are tracking errors in MiNT's positioning system. As a result, the trajectory of a MiNT node may deviate from its statically computed path, and collision between nodes could still arise from time to time even with perfect collision-free path planning.

VII. 24x7 Autonomous Operation

A major challenge in the design of MiNT is how to reduce its administration and maintenance cost while supporting uninterrupted 24x7 operation. Ideally each MiNT node should be self-administered with respect to routine operations, such as recharging the batteries; and self-healing in the face of software/hardware failures.

Because each MiNT node is battery-powered, the batteries must be recharged periodically. Typically charging a node's battery is a manual process that requires the administrator to physically take the node to a charging station [12]. In contrast, MiNT supports automatic recharging of node batteries and thus completely does away with manual efforts in this process. MiNT leverages Roomba's ability to go to a docking station to charge its battery when its battery runs low. A Roomba docking station emits an IR beacon signal, which could be received by any Roomba within a distance of 5 feet. When a

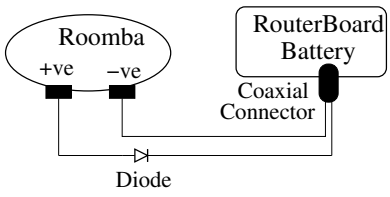

Fig. 8. *The auto-charging circuit for charging a MiNT node Roomba and RouterBoard battery when the mobile node docks itself into a docking station for recharging.*

Roomba's battery capacity drops below a threshold, it starts looking for any beacon signal emitted by a docking station and uses the signal to home into the docking station and recharge its battery.

Unfortunately, Roomba's built-in battery cannot be used to directly power the RouterBoard computer, which is powered by a separate universal laptop battery. To recharge the RouterBoard battery along with the Roomba battery, we connect the RouterBoard battery to the charging tip of the Roomba battery as shown in Figure 8. This allows both batteries to be charged simultaneously from the same docking station. A diode connected between the Roomba battery and the RouterBoard battery ensures that each of these batteries cannot be drained by the other.

To initiate battery re-charging autonomically requires the knowledge of the current residual battery capacity. Although the new SCI interface provides the residual capacity of each Roomba's battery, it offers no information on the residual capacity of the RouterBoard's battery, which we estimated using the following formula:

$$R_{board} = I_{board} - T_{board} * U_{board} - N_{disk} * U_{disk} - N_{packet} * U_{packet}$$

where R_{board} and I_{board} are the residual and the initial charge on the RouterBoard's battery. T_{board} is the amount of time RouterBoard has been on, and U_{board} is the measured power consumption rate during idle time. N_{disk} and U_{disk} are the number of hard disk operations performed by the RouterBoard and the energy consumed per disk operation respectively. Finally, N_{packet} and U_{packet} are the number of packets sent/received by the RouterBoard and the energy consumed per network operation respectively. The energy consumed by IR transmission is negligible.

VIII. Conclusion

This paper describes the design and implementation of a networked robot system called MiNT that is designed to support high-fidelity wireless network emulation. The communications among these robots are through IEEE802.11 wireless LAN interfaces and mainly used to emulate wireless communications required in wireless protocol testing, experimentation and evaluation. MiNT supports several unique features that are rarely seen in other networked robot systems:

- Each MiNT node is completely tetherless and yet is capable of sustaining for a long period of time without human intervention because of its self-charging capability.
- A computer vision-based robot positioning system that is both efficient and scalable, because of it uses a color-based ID scheme and it exploits frame-to-frame coherence.
- Combination of static path computation and dynamic collision avoidance strikes a good balance between implementation complexity and effectiveness in supporting node mobility.
- A self-monitoring mechanism exploiting watchdog timers detects node crash/hang, and preemptively restarts failed nodes without manual efforts.

References

[1] Acroname Garcia Robot. http://www.acroname.com/garcia/tutorials/teademo /teademo.html.

[2] AmigoBot from ActiveMedia Robotics. http://www.activrobots.com/ROBOTS/amigobot.html.

[3] Camstream Webcam Application for Linux. http://www.smcc.demon.nl/camstream/.

[4] Benjamin A. Chambers. The Grid Roofnet: A Rooftop Ad Hoc Wireless Network. Technical report, MIT Master's Thesis, Jun 2002.

[5] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press, 2005.

[6] Lars Cremean, William Dunbar, David van Gogh, Jason Hickey, Eric Klavins, Jason Meltzer, and Richard M. Murray. The Caltech Multi-Vehicle Wireless Testbed. In *Proc. of Conference on Decision and Control*, 2002.

[7] Pradipta De, Ashish Raniwala, Rupa Krishnan, Krishna Tatvarthi, Jatan Modi, Nadeem Ahmed Syed, Srikant Sharma, and Tzi cker Chiueh. MiNT-m: An Autonomous Mobile Wireless Experimentation Platform. In *Proceedings of Mobisys*, 2006.

[8] Pradipta De, Ashish Raniwala, Srikant Sharma, and Tzi cker Chiueh. MiNT: A Miniaturized Network Testbed for Mobile Wireless Research. In *Proceedings of Infocom*, 2005.

[9] Emulab - Network Emulation Testbed Home. http://www.emulab.net/.

[10] Scott Graham and P. R. Kumar. The Convergence of Control, Communication, and Computation. In *Proceedings of Personal Wireless Communication (PWC)*, 2003.

[11] J. Heidemann, N. Bulusu, and J. Elson. Effects of Detail in Wireless Network Simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, January 2001.

[12] David Johnson, Tim Stack, Russ Fish, Dan Flickinger, Rob Ricci, and Jay Lepreau. Mobile Emulab: A Robotic Wireless and Sensor Network Testbed. In *Proceedings of Infocom*, 2006.

[13] R. Karrer, A. Sabharwal, and E. Knightly. Enabling Large-scale Wireless Broadband: The Case for TAPs. In *Proceedings of Hotnets Workshop*, Nov 2003.

[14] H. Lunndgren, D. Lundberg, J. Nielsen, E. Nordstrom, and C. Tscudin. A Large-scale Testbed for Reproducible Ad Hoc Protocol Evaluations. In *Proceedings of WCNC*, 2002.

[15] D. Maltz, J. Broch, and D. Johnson. Experiences Designing and Building a Multi-Hop Wireless Ad-Hoc Network Testbed. In *CMU TR99-116*, 1999.

[16] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The Cricket Location-Support System. In *Proc. 6th ACM MOBICOM,*, Aug 2000.

[17] Ashish Raniwala and Tzicker Chiueh. Architecture and Algorithms for an IEEE 802.11-based Multi-channel Wireless Mesh Network. In *Proc. of IEEE Infocom*, 2005.

[18] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols. In *To appear at the Wireless Communications and Networking Conference (WCNC'05)*, Mar 2005.

[19] Roomba Discovery Robotic Vacuum Cleaner. http://www.irobot.com/consumer/product_detail. cfm?prodid=18.

[20] Roomba Serial Command Interface. http://www.irobot.com/images/consumer/hacker/ Roomba_SCI_Spec_Manual.pdf.

[21] Universal Infrared Remote Control From Any PC. http://www.innotechsystems.com/spitfire6001.htm.

[22] M. Takai, R. Bagrodia, M. Gerla, B. Daneshrad, M. P. Fitz, M. B. Srivastava, E. M. Belding-Royer, S. V. Krishnamurthy, M. Molle, P. Mohapatra, R. R. Rao, U. Mitra, C. Shen, and J. B. Evans. Scalable Testbed for Next-Generation Wireless Networking Technologies. In *Proc. of IEEE Tridentcom*, Feb 2005.

[23] Video for Linux (v4l) Resources. http://www.exploits.org/v4l/.

[24] Brian White, Jay Lepreau, and Shashi Guruprasad. Lowering the Barrier to Wireless and Mobile Experimentation. In *Proceedings of Hotnets Workshop*, 2002.