Roomba MADNeT:
a Mobile Ad-hoc Delay Tolerant Network Testbed

Joshua Reich[a] *reich@cs.columbia.edu*
Vishal Misra[a] *misra@cs.columbia.edu*
Dan Rubenstein[b] *danr@cs.columbia.edu*
[a]CS / [b]EE Department, Columbia University, New York, NY, USA

I. Introduction

We have built a mobile, ad-hoc, delay tolerant, network testbed (MADNeT). Our testbed is geared towards enabling the exploration of highly disconnected networks whose nodes must store and forward information (multi-hop paths are very unlikely), while allowing us to explore connected network scenarios as well. Using our MADNeT, we can emulate a wide range of mobile networking scenarios and execute real-world (albeit somewhat simplified) data collection missions. Our system is built from commercial off-the-shelf (COTS) hardware running a paired-down Linux OS, yielding a relatively inexpensive but fairly flexible mobile network testbed. We have implemented replication-based information diffusion protocols and a distributed opportunistic surveillance application on the testbed as proof-of-concept. Currently, we are refining our software tool-set and instrumenting our nodes to test both our protocols and application.

II. Testbed Design

As we began to explore various mobile networking scenarios, it became clear to us that being able to test our protocols and applications in hardware would bring significant value to our research. Not only would implementation in hardware provide strong evidence that our techniques could be applied in real-world environments, but the process of implementation and evaluation might also help us refine our assumptions as to what a reasonable semi-autonomous mobile wireless node could do, along with potentially highlighting new directions for exploration. MADNeT was motivated by our need for a system whose nodes could be flexibly reconfigured in both hardware and software, while allowing (although not enforcing) the use of standard wireless protocol families such as 802.11 - at a price point that was relatively affordable (see Figure 2 for sample list of parts and prices). Given that the thrust of our research is in networks, not mobile robotics, vision, or hardware design - we wanted to build our system out of building blocks,

Figure 1: a single MADNeT node

both hardware and software, that would provide as much basic functionality as possible. Consequently, it seemed the most economical way to achieve our goal would be to use COTS hardware and open-source software.

Specifically, we needed a mobility platform capable of robust movement over indoor (e.g., floor, carpet) and outdoor (e.g., cobblestone, grass) terrain. We also needed a processing platform capable of allowing us to program in a high-level language and make use of already developed libraries for image processing and the like. An equally important issue was that this unit could be interfaced with a standardized commercially available hardware such as wireless radio, web-cam, GPS, USB memory and Bluetooth adapters.

II.A. Hardware

After examining several potential setups, we determined that the combination of a iRobot Roomba Create mobility / sensing platform with a Linksys WRTSL54GS wireless router running OpenWrt (an open-source Linux distribution for embedded devices), along the lines outlined in [6], would provide the ideal balance between capability, cost, and (relative) ease of development. The iRobot Create provided us with a fully functioning mobility platform with the bonus of including on board sensors and actuators. Access to these was easily available through

the iRobot Open Interface [4]. The WRTSL54GS router complemented the Create nicely, providing an integrated unit featuring a Broadcom 4704 processor running at 266MHz, 8MB flash, 32MB RAM, an integrated Broadcom wireless 802.11 radio (unfortunately, the proprietary Broadcom driver provided with OpenWrt limits access to much link-layer information), BCM5325 switch, and a USB 2.0 port. At the time of implementation the WRTSL54GS was the router most fully supported by OpenWrt (support has declined somewhat in the newer Kamikaze branch of the distribution) [1].

With minor modification to the 7-pin DIN to 9-pin Serial cable included with the Create, we were able to tap directly into the Create's on-board battery which provides 2700 mAh at 15V. Consequently we were able to power the entire system (router, Roomba, and peripherals) from the Create's battery. While one could obtain increased runtime by mounting a separate battery to power the router and peripherals, powering everything off the Create's battery has the significant advantage of enabling a more highly-integrated system. The entire unit can be turned on and off with the press of a single button and charged with little effort. In fact the unit can be programmed to automatically find a nearby docking station and charge itself when low on battery (the Open Interface a command which will initiate the Create's built-in find-dock-and-charge routine). We found that our applications drew from 1000-2000 mAh on the average, although on standby the system could run on far less in 300-400 mAh range. On a full charge the entire system ran between 1 and 2.5 hours, dependent on the application, and hardware configuration. For those who need more power, it is possible to custom-build replacement batteries that hold in excess of 3300 mAh [5].

II.B. Software

We are currently using the WhiteRussian 0.9 build of OpenWrt Linux. Along with the basic functionality native to the Linux kernel, OpenWrt features a functional web-server, firewall, ssh server/client, and most of the basic tools one would expect. An extensive and growing collection of ported packages is easily installable through the ipkg package management system. While micro versions of several programming languages are available, by far the most robust method for programming is to cross-compile C or C++ programs. In WhiteRussian this process is somewhat more difficult than necessary (only certain LINUX/UNIX systems will be able to execute cross-compilation) and certain libraries are unavailable, but overall one can write fairly sophisticated applications as we will see in III.B.

Part	**Cost***
iRobot Create Programmable Robot*	$115[!]
iRobot APS Battery	$40
iRobot APS Charger	$40
Linksys WRTSL54GS	$100
Creative Labs WebCam Instant***	$23
OEM USB Hub	$6
OEM USB Flash Drive	$14
Generic USB-to-serial adaptor	$12
9V Battery Snap Connector Radio Shack #207-325	$2
Power Plug, standard barrel Radio Shack #274-1569	$1
Double-side Velcro	$10
Total	$365

* prices reflective of University vendors, all sums in USD

* 7-pin DIN to 9-pin Serial cable & 12AA Battery Case, included

[!] iRobot offers an educator's 10-pack for $1000

** for the various compatible web-cam / driver combinations see [3]

Figure 2: part list for MADNeT nodes

The basic development process consists of implementing the code on a development machine. This code is then cross-complied and packaged using the OpenWrt SDK. The complied packages are then copied to the Linksys unit where they are installed with the ipkg system [2]. This can result in a somewhat tedious debugging process, prompting us make liberal use of optionally compiled debug messages.

III. Utilizing the Testbed

To demonstrate the functionality of our testbed, we have developed a prototype application for distributed opportunistic surveillance in a mobile Delay Tolerant Network (DTN) environment. The goal of our application is to collect data on subjects of interest through the use of passively mobile nodes (nodes which while moving, are not in control of their own movement). In such a setting, nodes can only communicate with peers as opportunity allows. As a concrete example: consider a mobile DTN comprising service robots, vehicles with mounted nodes, and perhaps devices carried by rangers and visitors spread across the expanse of a national park (we have emulated the key ingredients of this scenario using MADNeT nodes opportunistically surveilling a red Roomba). Researchers wishing to collect data on animal populations could task these nodes to opportunistically record relevant data and images when recognized (within a sufficient degree of probability). The question then becomes

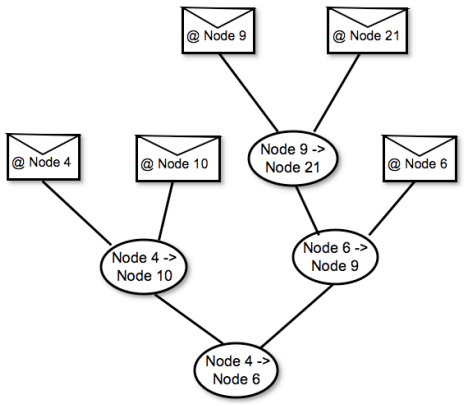

Figure 3: virtual tree for a single message

how to pass this data around the network so as to make it available to the interested parties.

III.A. Balanced Replication

To make this information available, we used a novel technique for balanced replication using distributed counter values. This technique for balanced replication works to spread data produced by nodes evenly throughout the network. By spreading data evenly, our system both helps to minimize the average time a data item takes to reach an information gatherer and also increase the robustness of the system to failure of a subset of its nodes.

In our protocol when two nodes meet they exchange meta-data, comparing their message caches. Each message in the cache is marked with a counter value. After deciding which messages should be replicated (we need to be selective since our nodes only have limited memory and may be bounded in the time they have to exchange replicas as well) the nodes replicate each others' chosen messages and increment the counter value on both the initial copy of the message and replica. The counter values possessed by replicas of a given message can be envisioned as forming the leaves of a virtual binary tree spread throughout the network (Figure 3) with one tree per unique message. By encouraging even growth of the branches through selective replication of lower counter values, we can ensure an asymptotic balance in the number of copies of each message stored throughout the network (under certain random mobility assumptions). Consequently, within a moderate number of nodal contacts, an information gatherer will be able to draw a large portion of the relevant data from our mobile DTN.

III.B. Implementation

To implement this application we wrote a robust multi-threaded, socket-based peer detection and message replication engine. This engine worked with data produced by our image recognition routines. For this purpose, we developed a fairly basic image processing engine (detecting our proxy subject, a red Roomba). We have also adapted code provided through [6] to perform I/O operations on the Create platform (in this application used primarily to monitor power readings) and we additionally developed a logging facility which wrote logs to attached USB memory.

IV. Future Work

We are currently in the process of instrumenting and extending our software with the goal of collecting data on the performance of both our counter-based replication protocol and our opportunistic surveillance application, along with potential extensions of our counter technique to network coding. We also have several exciting applications we are considering developing on our testbed, including techniques for maintaining network connectivity by controlled or partially controlled mobility, and transitioning network topologies between delay-tolerant / high-disconnected and connected regimes. Finally, we are looking into interfacing additional hardware including GPS, digital compass, and RFID card readers.

V. Acknowledgments

We acknowledge Arpan Saurabh Soparkar for help with the equipment and Matt Yu-Ming Chang for help with the image processing code.

References

[1] `http://openwrt.org/`, September 2007.

[2] `http://wiki.openwrt.org/buildroot`, October 2007.

[3] `http://wiki.openwrt.org/webcam`, October 2007.

[4] `http://www.irobot.com/sp.cfm?pageid=248`, October 2007.

[5] `http://www.roombareview.com/hack/battery.shtml`, October 2007.

[6] T. E. Kurt. *Hacking Roomba*. Wiley, 2007.