

CrowdLab: An Architecture for Volunteer Mobile Testbeds

Eduardo Cuervo, Peter Gilbert, Bi Wu and Landon P. Cox
Department of Computer Science
Duke University
Durham, NC 27705

Abstract—Researchers investigating mobile and wireless systems can run experiments on many testbeds, but no existing option supports experimentation “in the wild” without sacrificing features such as access to low-level wireless state and efficient scheduling of co-local guests. To fill this void, we present a new architecture for mobile testbeds called CrowdLab. CrowdLab allows researchers to run guest virtual machines on volunteer mobile nodes and ensures efficient use of testbed resources through a new dual-mode networking abstraction and a weakly-consistent, replicated state store called a site directory.

We have implemented two CrowdLab prototypes, one for x86 laptops and one for ARM-based Nokia N810 Internet Tablets, and evaluated them using power measurements, micro-benchmarks, and trace-driven emulation. Our evaluation demonstrates that handheld users can contribute 2.5 hours per day to CrowdLab and still have over 12.5 hours of idle time remaining. In addition, emulated mobility-trace replays show that CrowdLab’s fault-tolerance mechanisms allow experiments to run uninterrupted, even in the face of high churn rates.

I. INTRODUCTION

Large-scale community testbeds have become an indispensable tool for experimental distributed systems. The widespread embrace of Internet [23], [31], [9] and wireless [25], [33], [15], [11] testbeds attests to the value of shared cyberinfrastructure for exposing measurement frameworks and system prototypes to live network phenomena. GENI [24] represents a cross-community effort to standardize networked testbed abstractions and control frameworks.

However, to continue to meet the needs of researchers, testbed architectures must evolve in tandem with technology. In particular, as mobile and wireless systems become increasingly central to the lives of millions of users, it is critical that testbeds allow researchers to investigate the unique challenges of this emerging computing environment. Researchers need tools that give them first-hand observations of how mobility and location affect system behavior at all levels of the stack, including the operating system, drivers, and applications. Unfortunately, existing testbeds do not give researchers all of the features they require.

Testbeds such as ORBIT [25] and Dieselnet [33] provide two important features: 1) programmatic control of low-level wireless state, and 2) concurrent scheduling of co-local instances. Many classes of experiments rely on access to wireless driver code to investigate the behavior of wireless protocols, while interactions among nodes in a distributed system cannot

be observed unless nodes execute concurrently. However, these features are currently only supported through centralized control structures in which dedicated resources (e.g., processors, sensors, and NICs) are tethered to provisioned infrastructure (e.g., mobile robots and campus buses).

The drawback of tethering resources to infrastructure is that experiments cannot be exposed to live human mobility or a diverse set of locations. Support for human mobility and geographic diversity requires opportunistic use of *volunteer* resources from personal devices such as netbooks, laptops, and smartphones. However, utilizing mobile volunteers requires a decentralized control structure due to the security and privacy concerns of donors [8], [1], [14]. First, because experiments must not corrupt or leak host devices’ data, hosts do not give experiment code low-level control. Second, to prevent mobile volunteers from being tracked, donor machines schedule experiments independently and make no guarantees to researchers that nodes will be able to communicate.

To better balance the concerns of researchers and volunteers, we present a new testbed architecture called *CrowdLab*. CrowdLab is the first testbed architecture to utilize volunteer mobile resources without sacrificing features common to infrastructure-bound testbeds. CrowdLab protects volunteers’ data by isolating guest code through hardware virtualization and protects volunteers’ location privacy by decentralizing testbed services. CrowdLab experiments are granted low-level access to a host’s Wifi through a new *dual-mode network abstraction* that allows guest code and a host resource manager to take turns controlling the radio. Services such as tasking and scheduling are performed in situ by federations of co-local devices, and are implemented on top of a fault-tolerant distributed state store called a *site directory*.

To summarize, this paper makes the following contributions:

- CrowdLab is the first testbed architecture to support low-level access to wireless state and concurrent scheduling of co-local instances on volunteer mobile devices.
- CrowdLab introduces a coarsely-synchronized, dual-mode wireless networking abstraction that supports accurate wireless measurement, is power-efficient, and enables in-situ coordination.
- CrowdLab introduces a fault-tolerant distributed data store called a site directory that allows the testbed to operate efficiently in the face of device churn.

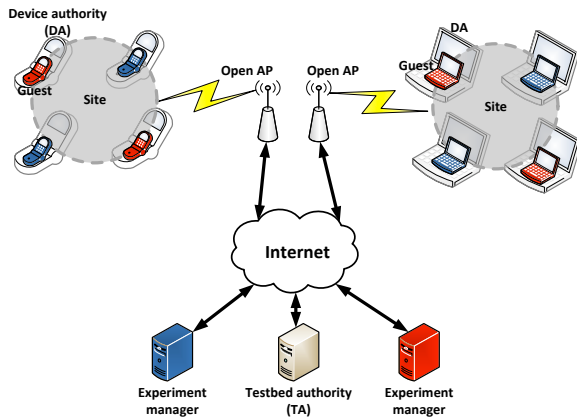


Fig. 1. High-level view of CrowdLab.

- We demonstrate the feasibility of CrowdLab through power measurements, micro-benchmarks, and trace-driven experiments using two prototype implementations, one for x86 laptops and one for Nokia N810 mobile devices.

The rest of the paper is organized as follows: Section II provides an architectural overview of CrowdLab, Section III describes the programming interface provided to researchers, Section IV describes the design and implementation of site directories and the higher-level features they enable, Section V describes our prototype implementation, Section VI provides an evaluation of our prototype, Section VII describes related work, and Section VIII provides our conclusions.

II. OVERVIEW

This section provides a high-level overview of the CrowdLab architecture. Figure 1 shows each major component type and its relationship to the others.

The only centralized component in CrowdLab is the *testbed authority (TA)*. The TA registers experiments and resource contributors and provides a root of trust. Because preserving the location privacy of mobile contributors is a first-order concern, the TA cannot know volunteer devices' locations or how to reach them. As a result, testbed functions such as scheduling and tasking are handled in a decentralized manner within a *site*. A site is an opportunistic federation of independent, co-located resource managers called *device authorities (DAs)* within a small geographic area (e.g. in the same building). Each DA is bound to a single physical device (e.g., a mobile phone or laptop), and has complete control over its device's physical resources. In our current CrowdLab implementation, DA functionality is encapsulated within a privileged domain virtual machine (Domain-0 in Xen [2]), although hypervisor technologies such as VMware, Parallels, or KVM would also be sufficient. These hypervisors are increasingly common on laptops of users who wish to run multiple operating systems, and have the potential to be deployed on future smartphones [30]. It may also be feasible for a university to deploy CrowdLab by offering VMware or Parallels licenses to students and staff.

Volunteers contribute resources to CrowdLab in the same way that users contribute spare resources to SETI@home but a micro-payments model similar to that of Amazon's Mechanical Turk could be applied. Because energy for mobile devices is precious, the DA enforces limits on when and how much experiments consume. First, a device will not participate in CrowdLab if the owner is actively using it. There are a number of ways for a DA to recognize when a device is not in use, such as waiting for long periods between user input. In addition, users provide their device's DA with a daily resource budget for running experiments. Budgets are coarsely defined as a percent of battery capacity (e.g., no more than 25% of a device's battery per full charge) or as a period of participation (e.g., no more than two hours per day). Energy accounting on consumer platforms is notoriously difficult, and DAs' enforcement of these budgets is best-effort. CrowdLab aims to use contributors' resources conservatively, and users can always reduce their contribution if participation interferes with normal device usage.

To task devices and schedule experiments in locations without provisioned infrastructure, co-local DAs must coordinate through self-organizing, ad-hoc networks called *site networks*. Each ad-hoc network is maintained independently of all others and participation defines the site membership. Site networks are expected to exhibit a dense topology on the order of dozens of nodes, with a network diameter of at most two hops in between nodes. Sites may perform logical admission control to ensure that these constraints are met: nodes attempting to participate in a site that is too populous or too wide may not be allowed to register in the site directory and in consequence be forced to start a separate site.

CrowdLab experiments are composed of two component types: sets of *guests* embedded within one or more sites and an off-site *experiment manager*. Guests are hosted by DAs, consume resources on volunteer devices, and implement an experiment's logic. Each guest runs in an isolated execution environment under the supervision of a DA, and only one guest executes on a device at a time. In our current CrowdLab implementation, guests execute as unprivileged (Xen Domain-U) virtual machines.

Communication between guests and their off-site managers requires connectivity through an open 802.11 access point, and may be intermittent and unpredictable. Experiment managers are responsible for maintaining an experiment's long-term state by providing off-device services such as remote logging. During periods of disconnection guests can use the site network to propagate results to each other in a delay-tolerant fashion.

III. PROGRAMMING INTERFACE

CrowdLab must balance developers' need for an expressive, intuitive programming model and device carriers' need for strong isolation. To balance these concerns, CrowdLab relies on hypervisors to export a virtualized hardware interface to programmers. Encapsulating an experiment's state within a low-level virtual-machine abstraction reduces the complexity

of protecting host devices without over-constraining developers.

AnySense [8], Scylla [28], Maté [21], and Java also offer virtualized environments for mobile devices, but provide a more restrictive computational model and API than hardware virtualization. All of these environments rely on an interposing code interpreter to mediate access to the hardware and are incompatible with other programming languages and native code. In contrast, hypervisors export a hardware interface with support for native legacy code, including operating systems and drivers. Thus, rather than forcing applications to be written in a single language and precluding access to low-level state, CrowdLab allows developers to re-use existing code written in Python, C, or any other language.

As with other hardware virtual-machine interfaces, CrowdLab guests have access to virtualized I/O peripherals such as a disk, Ethernet NIC, and Bluetooth radio. However, a more complex interface to the device’s Wifi is required to efficiently share the radio between the host DA and guest virtual machine. As noted in Section II, DAs coordinate through the ad-hoc site network. Guests may want to communicate with other site components, but will mostly want to use Wifi to interact with and measure their environment. For example, guests may want to profile open access points or communicate with their manager.

JellyNets previously proposed transparently multiplexing a device’s radio between a host and guest through Virtual Wifi [14], [5], but this approach has two drawbacks: accuracy and energy. First, the overhead of rapidly switching between host and guest networks makes accurate measurement of network characteristics such as bandwidth and latency impossible. Rapid switching also leaves no time for the Wifi card to enter true power-save mode (PSM), which makes running experiments extremely power intensive.

Instead, CrowdLab coarsely multiplexes access to Wifi by dividing time into fixed-length *epochs*. Epochs begin and end at well-known times, and are split into two phases: a *managed mode*, in which a DA controls the radio and helps maintain the site network, and an *unmanaged mode*, in which a guest controls the radio. Note that a DA can always regain control of the radio in unmanaged mode if it needs to. In our current implementation, epochs last 5 or 10 minutes, depending on the platform, with the first 40% spent in managed mode and the next 60% spent in unmanaged mode. DAs only need to be coarsely synchronized, since epochs are expected to start on the hour and every 5 or 10 minutes thereafter. To ensure that DAs do not drift too far from each other, they sync their clocks via a local implementation of a NTP-like protocol at the beginning of each managed phase.

Coarse-grained switching addresses the accuracy and energy problems of fine-grained multiplexing. Since guests have sole access to Wifi in unmanaged mode, bandwidth and latency measurements are identical to those taken by a dedicated device. In addition, if a guest does not need to use Wifi for stretches of the unmanaged phase, the radio can take advantage of PSM or be powered off.

The semantics of a wireless interface in managed and unmanaged mode are very different. If a wireless interface is in managed mode, communication through the virtual device is limited to the site. In unmanaged mode the set of reachable machines depends on the network with which the guest associates the radio. Each guest and DA is assigned a unique IP address in the private IP space and can communicate with other components through the site network during managed mode using DA-supported routing (e.g., OLSR [7]).

While the primary motivation for managed mode is to allow DAs to coordinate, guests of the same experiment can also interact with each other during these periods for experiment specific collaborations. To help guests discover each other, host DAs support a *getCoordinator* call that returns the IP address of a guest in the site from the same experiment. CrowdLab guarantees that all calls to *getCoordinator* will return the same answer, regardless of which guest makes the request. This allows experiments to bootstrap intra-experiment coordination by serializing communication through a *coordinator* guest. If the call returns the guest’s own IP address, then it has been chosen as the coordinator for the experiment.

Because the semantics of a wireless interface in managed and unmanaged mode are different, guests’ behavior must change according to the state of the Wifi radio. Guests can monitor the epoch through two calls into their DA: *isManaged*, which returns true if the epoch is in managed mode and false otherwise, and *timeToNextEpoch*, which returns the number of seconds until the beginning of the next epoch.

The level of control a guest has over Wifi in unmanaged mode depends on factors such as the features of the host hypervisor and the chipset of the Wifi card. If proper support is in place, the DA can hand over low-level control of the wireless card using the PCI passthrough and hot-plug features present in Xen and other hypervisors. This allows guests to test custom Wifi drivers and directly interact with access points. To ensure safe use of the radio, hypervisor support for virtualized PCI hot-plug allows the DA to regain control of the radio before switching back to managed mode. DieselNet [33] currently uses PCI passthrough to give a guest virtual machine low-level control of a wireless card at boot time, but the platform does not take advantage of PCI hot-plug.

If the host device does not support PCI passthrough and hot-plug, guests control the Wifi radio in unmanaged mode through a simple message-passing interface: *wifi_status* returns whether the radio is turned on or off, *wifi_scan* performs a scan and returns a list of SSID, MAC-address, and signal-strength tuples, *wifi_connect* takes an SSID and MAC address and attempts to associate with a network, and *wifi_shutdown* turns the radio off. If a guest tries to connect to a network using these calls, the DA uses DHCP or ZeroConf [32] and forwards guest communication using NAT.

In addition to the virtual execution environment, researchers must help CrowdLab by providing a certificate signed by the TA that lists their experiment’s properties. This certificate includes a hash of the virtual machine image, a URL where the image can be obtained, any requirements for the image

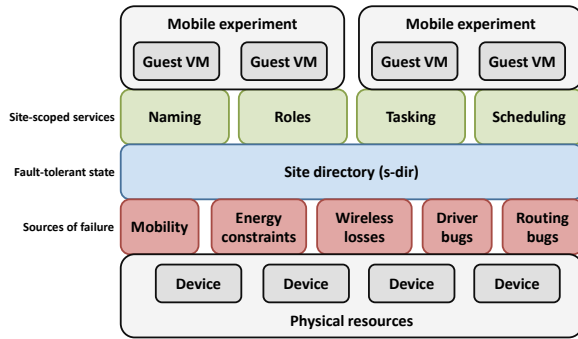


Fig. 2. Layered view of site resources and services.

(e.g., Wifi chipset, CPU instruction set, or support for PCI passthrough and hot-plug), and the minimum number of guests that must be scheduled concurrently within a site. Hardware requirements allow DAs to assign guest images to host devices, while the number of minimum guests allows DAs to gang schedule guests within a site.

One danger that device owners may face in giving up control of Wifi to guest code is that the card may be used to access sensitive resources. CrowdLab DAs do not share any WPA2 secrets with guests, but some networks use MAC addresses for access control, even though this provides poor security with or without hosted code.

IV. SITE SERVICES

Figure 2 shows a layered view of the actors and services within a site. The bottom layer represents the physical resources that are contributed by volunteer devices and managed by individual DAs. The top layer represents experiments composed of multiple guests. In order for guests to make good use of the available resources, the site must provide addressing and naming services, task devices, assign roles (e.g., experiment coordinator), and schedule guests. The primary challenge of providing these services is that they must be executed in a coordinated manner. Calls to *getCoordinator* must return the same answer, guest images should be disseminated efficiently, and the concurrent scheduling requirements of experiments must be satisfied. At the same time, these services must be implemented on top of rapidly-churning resource pools. CrowdLab addresses these issues by building site services on top of a replicated state store called a *site directory (s-dir)*. S-dirs are collectively maintained by a site’s DAs and allow a site to function correctly in the face of resource churn.

S-dirs are structured as trees and store information about a site, including IP-address and DNS mappings, device up-times and capabilities, the identities of experiment coordinators, the locations of guest images, and where guests are running. Figure 3 shows an example tree. The root of each s-dir is the geographic coordinate of the network. The first DA to arrive at a site generates this coordinate, and seeds the site network. S-dirs have two branches below the root; one branch stores information about the experiments that are available within a site and the other stores information about the devices present.

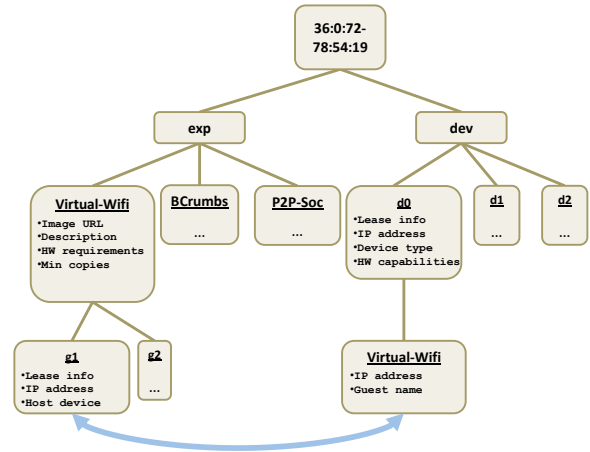


Fig. 3. Example site directory

Each experiment and device node has an associated list of meta-data properties that describes its features. Leaf nodes represent guest virtual machines and sit below the device (*dev*) and experiment branches (*exp*). Descriptions of guests are stored redundantly under both sub-trees: in Figure 3, Virtual-Wifi guest *g1*, running on device *d0*, is represented by the two entries connected by a light arrow.

In the rest of this section, we describe how s-dirs are maintained and how they can be used by DAs to implement site-wide services.

A. S-dir maintenance

In mobile environments, failures can occur at all levels of the stack: human mobility leads to node churn on the order of minutes, wireless networks frequently lose messages, and energy-budget constraints can lead to sudden resource withdrawal. As a result, s-dirs must be stored redundantly to provide any modicum of fault tolerance. S-dirs are fully replicated at all DAs within a site. Full replication allows any DA to use its local copy of the s-dir to implement any network service. At the same time, enforcing strong consistency among replicas would be impractical given the likely number of transient failures and rate at which nodes will enter and leave a site.

Managing weakly-consistent replicated state among a set of failure-prone mobile nodes is not a new problem. The Coda distributed file system [19] and Bayou distributed database [29] are among the best known investigations of this space, and both systems rely on optimistic concurrency to improve availability and performance. However, Coda and Bayou state-management algorithms were designed for data services that could easily tolerate temporarily divergent views. S-dirs exhibit a high degree of read and write sharing within a site and divergent views can waste precious resources.

Consider the case of four guests of the same experiment, *A*, *B*, *C*, and *D*, in which an inconsistency leads *C* to believe that *A* is the experiment coordinator and *D* to believe that *B* is the coordinator, while *A* and *B* each believe themselves to be the coordinator. In this case, *C* will look to *A* to coordinate

and D will look to B , leading to erroneous decisions and incorrect results.

Similarly, consider an experiment e that requires at least two concurrent guests, and an inconsistency leads X to believe that Y and Z are running an instance of e , and Y and Z to believe that X and Y are running an instance of e . In this case, X , Y , and Z will all believe that the scheduling constraint of e has been met even though only a single instance of e is being run by Y .

To prevent such waste, sites maintain weakly-consistent s-dir replicas through *consensus*, *leases*, and *hierarchy* [20] rather than an optimistic concurrency-control scheme. Similar to registry updates in Grapevine [3], s-dir updates are propagated to all replicas through the site network during an epoch’s managed phase using unreliable multicast. However, due to the relatively high loss rates in wireless networks, sites rely on three simple but effective techniques to expedite anti-entropy.

First, s-dir updates consist of the entire tree plus a version number. Each updated s-dir reflects all prior updates received by the sender so that whenever a node detects a missed update, it can integrate the new updates into its local copy. DAs compress updates before transmitting them to reduce the communication overhead of shipping the entire tree. Second, each DA that receives an update repeats the update back through its multicast tree. The size of an s-dir grows linearly with the size of the network, but the number of transmissions needed to re-multicast an update grows slowly due to the site network’s dense topology. As a result, each s-dir update generates approximately one transmission per DA. Third, s-dir updates are piggy-backed on any unicast traffic between DAs since wireless unicast is much more reliable than multicast. We have found that it is difficult to maintain consistent views of the s-dir over even short periods without the additional redundancy provided by these techniques.

Sites manage s-dir replicas through a single-writer, all-reader approach: all updates to the s-dir are serialized through a single *master DA* that is granted a write lock over the entire s-dir as long as it serves in that role. Our current implementation elects masters via a modified version of the Bully algorithm [13], that is executed at the beginning of each managed epoch, using multicast for the election messages and using as a metric for choosing the coordinator the time at which the device joined the site, winning the one that joined first and breaking ties with the device’s id.

Divergent views of the master’s identity are detected when a node that believes itself to be the master receives updates indicating a different master. Nodes resolve such conflicts by selecting the node that has spent the most time in the site as the master. Similarly, an absent master is detected at the beginning of an epoch, when the other DAs return to managed mode and attempt to contact the master. When this happens, each DA elects itself as master and any resulting conflicts are resolved as in the first case when these self elected DAs multicast their updates.

Changes to the s-dir only occur during the managed phase of an epoch. Thus, even though the site may experience churn

while the network is in unmanaged mode, the state of the s-dir will remain unchanged during these phases. When the site network transitions back to managed mode, the master is responsible for updating the s-dir so that it reflects “ground truth” and for propagating updates to participating DAs.

While simplifying concurrency control, the danger of relying on a single master to issue s-dir updates is that it creates a single point of failure. Sites can still provide naming services, route traffic, and run virtual machines without a master, but new addressing, role-assignment, scheduling, and tasking decisions cannot be made without a master. As a result, it is important for DAs to quickly identify when the master has failed and call a new election. One option is for non-master DAs to send periodic keep-alives to the master. This approach may quickly detect failures, but it is prone to false positives in wireless networks where ICMP messages can easily be lost and will burn energy on mostly unnecessary transmissions.

Instead, all node failures, including master failures, are detected through expiring *leases*. Under this scheme, all master and non-master roles are leased, and a node is considered to have failed if its lease is not renewed. Lease lengths are set to an epoch cycle, so that every node must renew its leases at the beginning of the managed phase of an epoch. The master renews its lease internally, and propagates its renewed lease terms by piggy-backing the update on unicast renewal requests from non-masters and via periodic multicast. If other nodes in the site do not receive the master’s lease renewal when updating their own leases, they assume that the master has failed and attempt to elect a new one. New masters use their copy of the s-dir to seamlessly pick up where the old one left off.

B. Service implementations

State stored in an s-dir provides a stable foundation for a number of fault-tolerant services including addressing, naming, tasking, role assignment, and scheduling. We now describe how each of these services can be implemented using the information embedded in an s-dir.

1) *Addressing and naming*: CrowdLab addressing and naming services dynamically bind IP addresses and DNS names to DAs and experiment guests. IP address assignment is critical for bootstrapping a site. Technologies such as ZeroConf allow hosts to self-assign IP addresses and DNS names through protocols such as IPv4LL [6] and mDNS [22], but CrowdLab takes advantage of the natural mapping from the S-dir naming leases used to detect node failures to the DHCP addressing leases for provisioning the IP space. Thus, rather than relying on ZeroConf, the master DA of each site runs a DHCP server and leases IP addresses to DAs and experiment guests. Each lease includes an address, a length, a subnet mask, and a DNS server. The master leases its IP address internally to itself.

Other DAs must obtain a DHCP lease before they can contribute resources. Newly arriving DAs initially check to see if the current time falls into the managed phase of an epoch, and, if not, wait until the next managed phase. Recall that

DAs can infer the timing of managed and unmanaged modes because epochs start at well-known wall-clock times (e.g., at the beginning of each hour and every 10 minutes thereafter), and mode transitions are coarsely synchronized on the order of minutes. Once within the managed phase of an epoch, new DAs look for the CrowdLab ESSID identifying a site network. If they find a site network, the new DAs join and submit a DHCP request. If the DHCP request fails, DAs will retry after a randomized pause.

If the second DHCP request fails or if a DA does not find a site network in the first place, it will start a new network, assign itself an IP address, designate itself master, and multicast a new *s-dir*. This may cause a conflict if another master exists, but CrowdLab’s conflict-detection and master-election mechanisms allow the site to converge.

All IP-address leasing state is embedded in the *s-dir* and fully replicated at all DAs for fault tolerance. After an address is leased to a DA, the master inserts a new node into the *dev* branch of the *s-dir* containing the lease terms and multicasts the new tree to the rest of the DAs. If a lease expires, the master removes the associated node from the *s-dir* and multicasts the new tree.

Lease requests for guests are handled similarly. After the master identifies a potential host DA, it requests that it run an instance of the experiment. If the host DA agrees, then the master reserves an address for the new experiment guest, and returns the lease terms to the host DA. The master then inserts a new leaf node below the *exp* branch containing the terms of the lease and inserts a new leaf node below the *dev* branch under the host DA’s node. These updates are then multicast to the other DAs.

A downside of batching updates to the *exp* and *dev* branches is that there is often a delay of minutes between the time when a guest’s addresses are leased and the time when a guest has actually booted. For that reason, the *s-dir* node with a guest’s information carries an attribute describing its status. A guest is originally marked as booting by the master and the update is multicast. Once the virtual machine boots successfully the host notifies the master. The master then updates the *s-dir* by marking the guest as ready and multicasting the update. This approach prevents devices from trying to contact a virtual machine before it is ready.

In addition to managing the IP space of a site through the *s-dir*, the master also provides a site-scoped hierarchical naming scheme for mapping human-readable names to IP addresses. The namespace uses a dotted notation that complements the conventional DNS space by introducing a new top-level domain called *site*. Below this level are two more, *exp* and *dev*, that correspond to the experiments and devices available within a site. The mappings for this service fall directly out of the *s-dir*. For example, using the *s-dir* in Figure 3, a lookup of the name “*g1.virtual-wifi.exp.site*” would return the IP address of the guest virtual machine executing on device *d0*. Since the *s-dir* is fully replicated, all DAs can resolve *.site* names locally.

The naming service is implemented as a lightweight server running in each DA. These servers collectively intercept every

DNS lookup request within a site and resolve the name using the *s-dir*. For convenience, sites’ DNS services also support automated redirection conventions. For example, performing a lookup of the name *virtual-wifi.exp.site* returns the coordinator of the Virtual Wifi experiment. This is often an easier way to identify experiment coordinators than through an RPC to the DA.

2) *Tasking, scheduling, and role assignment*: The *s-dir* can also be used to assign guests to DAs. To protect device carriers’ anonymity, DAs can download experiment meta-data and guest images using an anonymizing communication channel such as a MIX-net or public access point. Once an experiment image has been injected into a site, the site master must task devices and schedule the experiment. All tasking, scheduling, and role-assignment decisions are made by the master at the beginning of the managed mode of an epoch or when a new device joins the site.

There are many ways to schedule virtual machines, and CrowdLab can support any number of algorithms through policy plug-ins. Once a schedule has been chosen, the master is responsible for coordinating the instantiation of guests on other devices; in this role, the master is similar to brokers in ORCA [16]. To increase deployment speed all DAs help spread experiments. In order to avoid more than one DA sending the same experiment to the same target, DA’s will ask the master for permission. This makes the master a point of synchronization during tasking. A master uses the *s-dir* to match guest requirements to the features of a site’s hosts. To execute a schedule, the master invokes a well-known RPC interface running at each DA to request that the target DA run an instance of the guest. If the image is not already stored locally, the master will tell the target DA where to get the image. Tasking within a site allows experiments to survive faults. Even if a device running a guest leaves the site, a new guest can be instantiated during the next managed phase on another device. Thus, a network can experience 100% turnover without interrupting an experiment.

Tasking must complete before the managed phase ends and experiment guests are handed control of their hosts’ NICs. DAs reduce the latency and overhead of tasking by minimizing the data transferred between devices. Each DA stores an identical base disk image and transfers only the differences between the guest image and the base. This reduces the amount of communication needed to transfer an image to the size of the application-specific state. In our experience, we have found that guest images are often under 10 megabytes, and can be transferred in under a minute.

Finally, experiment coordinators are chosen by the master using a similar metric used to elect the master: the guest which has been running for the longest becomes the experiment coordinator. As long as all DAs have the same view of the *s-dir*, each will identify the same set of experiment coordinators.

V. IMPLEMENTATION

In this section we describe our CrowdLab prototypes for laptops and handheld devices as well as three representative experiments that demonstrate the features of CrowdLab.

A. Prototype testbeds

CrowdLab is implemented for both ARM-based Nokia N810 Internet Tablets and commodity x86 laptops. Both prototype testbeds rely on Xen for virtualization; the laptops run Xen 3.4, while the N810s run a heavily customized version of Xen based on Xen-ARM 3.0.1. PCI passthrough and hot-plug are currently available only on our laptop implementation, and as a result, guests running on the N810s cannot load custom Wifi drivers. Instead, they must use the message-passing interface to access Wifi. Routing in the ad-hoc site network is handled by OLSRD version 0.5.6 with the Basic Multicast Forwarding Plugin version 1.6.1.

For both testbeds, a daemon running in Domain-0 implements the DA and is responsible for leasing, naming, scheduling, tasking, maintaining s-dir consistency, and service failover. The daemon is written in Java, and runs under Sun's VM on the x86 laptops and CacaoVM on the N810s. We chose Java as it is a platform independent language that allowed us to have a common codebase among smartphones and laptops. The DA daemon consists of approximately 15,000 lines of Java code. The TA consists of approximately 1,000 lines of Java and runs on a dedicated server. It manages experiment certificates and meta-data in an XML database, and offers a web interface for developers to register and sign experiment images.

We also implemented a trace simulation engine that takes traces of user activity from a Wifi hotspot as input and converts the events in the traces to actions performed by real DAs executing on our laptop testbed. Every time an association event occurs in the trace, an unassigned laptop in our testbed "joins" the site by taking on a new MAC address, assuming the id of the associating device in the trace, and starting execution of the DA daemon. In the same way, when a disassociation event appears in the trace, the DA is shut down and the wireless driver disabled, effectively disconnecting the device from the site. With this scheme we are able to emulate sites with as many concurrent devices as laptops in our testbed. Furthermore, it was necessary to design the emulation framework to be decentralized: the laptops in the testbed run the actual CrowdLab system, and network switching within an epoch makes it impossible to centrally control DAs.

The laptop and N810 testbeds use different epoch lengths due to the slowness of the N810s' ARM 11 based 400 MHz TI OMAP 2420 processors. We found that two minutes was often not long enough for the DA daemon to complete all of its tasks, largely because Java on the N810 is very slow. As a result, we doubled the lengths of the managed and unmanaged phases for the N810 testbed. However, given that several recent smartphones feature GHz processors and highly-optimized Java virtual machines, these limitations are unlikely to persist in the future. We expect newer smartphones to share the same epoch lengths as laptops allowing sites to have both types of devices.

B. Mobile hypervisor

A major challenge in enabling CrowdLab to work on mobile devices was providing a suitable virtualization platform for ARM, the dominant CPU architecture for smartphones and PDAs. ARM, a RISC instruction set architecture, is very different from x86, which has served as the target platform for several mature virtualization systems. System-level virtualization has only recently become available for ARM devices. Samsung released a stable port of Xen to a prototype device featuring an ARM9 processor in 2009 [4]. At the same time, we had also been working with collaborators at the Nokia Research Center in Palo Alto to port Xen to the N810 Internet Device.

When Samsung's port was publicly released, we shifted our focus to adding support for the N810 to Samsung's code, now the official ARM Xen port. This required significant coding effort, and it is worth noting that, as far as we know, ours is the only version of Xen running on a consumer ARM device.

We ported Xen from a FreeScale i.MX21 prototype board to an off-the-shelf Nokia N810. Much of the effort was devoted to creating a Maemo (Nokia Linux) Domain-0 and a Domain-U, since no guest domains had been released by Samsung. The port of Xen to the N810 and the creation of the Domain-U involved merging source from Maemo, Xen and Linux, plus about 700 lines of new C and ARM assembly code.

C. Experiments

To demonstrate CrowdLab's features, we have written three representative experiments: a prototype Virtual-Wifi driver (virtualwifi), a Wifi profiler (C-scan), and a gang-scheduled social sensing experiment that uses Bluetooth (C-who). These applications represent three different categories of experiments which CrowdLab can support: wireless-driver experimentation, measurement, and sensing. These applications also demonstrate CrowdLab's support for hosting experiments coded in a variety of languages: our Wifi profiler was written in C, the Virtual-Wifi controller was written in Python, the Virtual-Wifi driver itself was written in C, and the bluetooth social-sensing application was written in Java.

VI. EVALUATION

In evaluating the CrowdLab architecture, we wanted to answer the following questions:

- Does coarse-grained Wifi switching allow experiments to save energy during unmanaged mode?
- How much time might mobile device owners be willing to contribute to CrowdLab?
- How well does CrowdLab function under churn?

To answer these questions we performed micro-benchmarks on our N810 testbed and used mobility emulation to replay a mobility trace through our laptop testbed.

A. Coarse-grained Wifi switching

The motivation for CrowdLab's Wifi-switching approach is threefold: 1) to allow local coordination via the ad-hoc site network, 2) to give experiments complete control of the radio

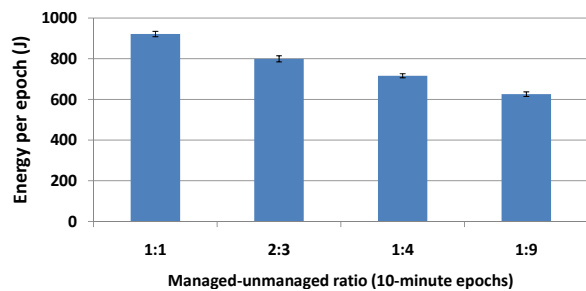


Fig. 4. Energy consumed per epoch (Nokia N810).

during the unmanaged phase, and 3) to take advantage of idle periods when radios can be powered off or placed in PSM mode. To evaluate the effectiveness of allowing radios to operate in PSM mode or to be powered down during the unmanaged phase, we measured the energy consumed by an N810 over a ten-minute period on our handheld testbed for different epoch configurations.

We measured energy by connecting an ammeter in series with the device and battery and sampling the current drawn by the device to calculate power. The monitored N810 participated in a site of two and ran an instance of the C-scan experiment during the tests. During each unmanaged phase, C-scan connected to nearby APs and performed latency and bandwidth measurements, typically lasting four to five minutes. After completing its measurements, the device powered down its radio for the remainder of the unmanaged phase.

Figure 4 shows the energy consumed by the device during one 10-minute epoch for different configurations of the managed-unmanaged ratio. As expected, spending more time in unmanaged mode provided more opportunities to save energy by enabling PSM or shutting down the NIC. Changing the ratio of managed-to-unmanaged mode from 1:1 to 1:9 reduced the energy consumed during an epoch by approximately 30%.

B. Contribution levels

To be feasible CrowdLab contributors must have sufficient excess battery capacity to run experiments. To approximate how much time a device owner may be able to contribute between charges, we measured the battery lifetime of our N810 testbed for various periods of participation. Each N810 began with a fully-charged 1500 mAh Li-Polymer battery.

When contributing resources, each N810 participated in a five-device site with a 10-minute epoch, with four minutes spent in managed mode and six minutes spent in unmanaged mode. To drive the experiment, we ran the C-scan experiment on all five devices. During the unmanaged phase, C-scan guests used the Wifi radio to profile nearby APs. To approximate nomadic users, devices remained in the same location for the duration of the experiment, providing a relatively small, constant set of available APs. APs that had already been tested were retested periodically, and work was scheduled infrequently throughout much of the experiment. As a result, devices were able to keep their radios powered down for much of the time spent in unmanaged mode. After contributing the

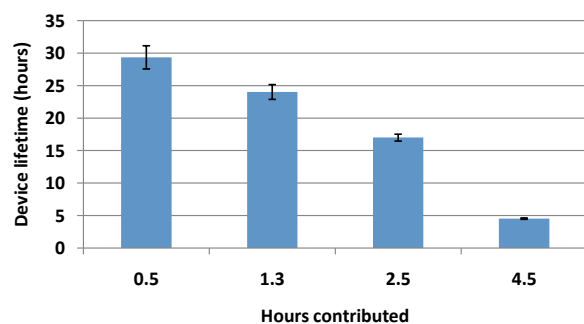


Fig. 5. Time contributed and device lifetime (Nokia N810).

specified number of hours, each device was left in an idle state until its battery was exhausted.

Figure 5 shows the average battery lifetime for different periods of participation. Each bar reports an average over the five devices participating in the site, with error bars showing the standard deviation. The figure shows that an N810 owner could run experiments for 2.5 hours and still expect 12.5 more hours of standby time. On the other hand, the battery was exhausted by approximately 4.5 hours of constant participation. Though these results are hardly definitive, they are encouraging. Even with the older hardware of our N810s, device owners could feasibly contribute 1 hour of experimental time per 24 hours of running on their battery, or 2.5 hours of experimental time per 16 hours running on their battery.

C. Handling churn

1) *Mobility trace-driven emulations*: To evaluate our laptop-based CrowdLab prototype under realistic levels of mobility-induced churn, we used our mobility-trace emulation framework. Over the course of a replay, any of the 10 laptops (only two of which had PCI hot-plug support) in the prototype testbed in our lab could emulate many nodes from a trace by changing their 802.11 MAC address and reconnecting to the site network. Because our prototype consists of 10 laptops, we could only emulate a maximum of 10 concurrent sessions. However, our prototype can emulate traces with dozens of sessions over the course of hours or days.

We used a trace taken from WiFi user logs collected from 2004–2007 by the Ile Sans Fil project, which operates 140 free access points in Montreal, mostly located in restaurants and cafes. The data set is available from crawdad.org. We selected one of the busiest access points from the data set and used associations with the access point to model CrowdLab sessions. The day that we chose to emulate represents an average day for that access point. For this emulation we deployed our three CrowdLab experiments: a Virtual Wifi driver experiment, C-scan, and C-who. Experiment patches were between 1 MB and 14 MB. The Ile Sans Fil trace had a maximum site size of 10 concurrent devices and a total of 45 sessions over eight hours. By performing live replays of this trace on our prototype we sought to characterize how well a CrowdLab site would operate under in the face of faults. During emulation, each laptop recorded detailed logs of all events, s-dir contents, and outbound network traffic.

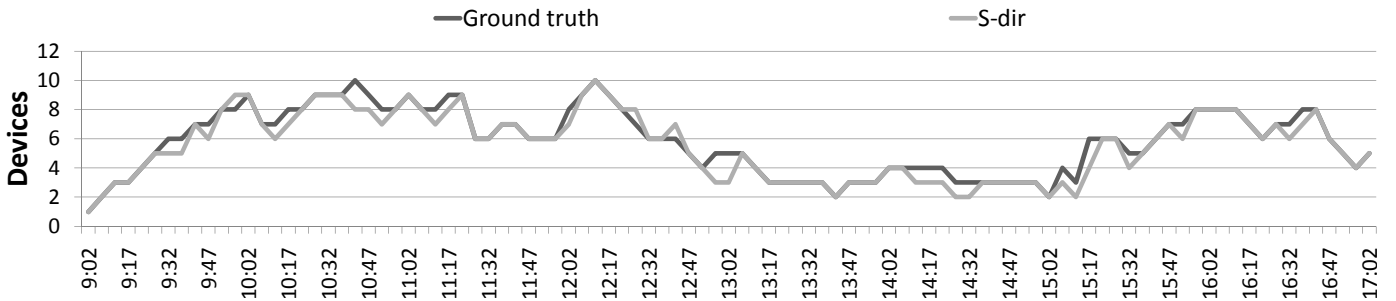


Fig. 6. Accuracy of resources reflected in the s-dir.

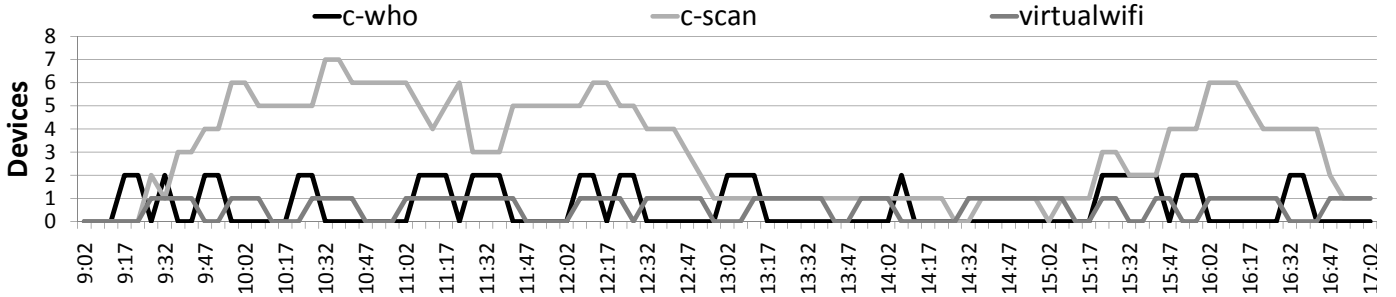


Fig. 7. Number of experiment guests.

2) *Results:* Our results suggest that CrowdLab can provide a consistent view of the site and survive under different types of failures. Figure 6 shows the site size over time, as reflected in the Ile Sans Fil trace (“Ground truth”) and as reflected by the s-dir of the master DA. The results demonstrate that for our emulated replay, the s-dir closely matched ground truth most of the time. Differences between the s-dir and ground truth were caused by: 1) devices arriving or leaving during the unmanaged epoch and not being detected by the master until the next managed epoch, 2) missed multicast requests from new devices to register, or 3) wireless-driver errors. Despite these sources of failure, the s-dir over- or under- counted the size of the site by at most two devices and did so for no longer than a single epoch.

Figure 7 shows the number of guests for each of the three experiments during the replay. These results are consistent with the following scheduling goals:

- 1) Devices are tasked within one epoch of joining the site.
- 2) Experiments are matched to devices equipped with the required hardware specifications (i.e., PCI hot-plug support for Virtual Wifi).
- 3) Experiments requiring gang-scheduling (e.g., C-who) only execute when there are a sufficient number of hosts running or ready to run the same experiment.

The first goal is demonstrated by comparing the s-dir size in Figure 6 to the guests deployed in Figure 7. When the site grows, the number of guests also increases. It should be noted that the total number of guests is often one fewer than the total site size. This is due to gang-scheduling constraints that we will discuss shortly.

Our second goal is demonstrated by the Virtual Wifi line in Figure 7. In our prototype only two of the 10 laptops

are equipped with PCI passthrough and hot-plug. For most of the replay, only one of those machines participated in the site at any given time. Unfortunately, hot-plugging a wireless driver in Xen occasionally causes a Domain-U virtual machine to crash, which leads the DA to restart the VM. This is encouraging since it demonstrates how host devices can survive even severe bugs in an experiment guest. However, as a result of the limited number of machines supporting PCI hot-plug and driver bugs, there is never more than one Virtual Wifi guest.

Finally the C-who line demonstrates our third goal, since it must be gang-scheduled. Throughout the replay, C-who only has more than one guest or no guests at all. When the number of C-who guests would have dropped to one, all instances were removed. However, our current scheduler also leaves a spare host for C-who so that when an additional host arrives C-who can be run immediately. This is why the total number of guests is often one less than the site size.

VII. RELATED WORK

As with cyberinfrastructure such as PlanetLab [23], EmuLab [31], and DETER [9], CrowdLab aims to help researchers expose experimental systems to realistic conditions, and is complementary to existing mobile platforms such as ORBIT [25], [26], Mobile Emulab [18], CarTel [15], DieselNet [33], MiNT-m [11], AnonySense [8], PRISM [10] and efforts to port cyberinfrastructure to the mobile domain [17]. GENI represents an effort to create a more general framework for using these and other point solutions [12].

Each of these testbeds limits experiments in one or more of the following ways: by restricting guest code to declarative, SQL-like languages [15], [8], by tethering experiments to

vehicular [33], [15] or pre-programmed robotic mobility [25], [18], [11], or by not supporting coordination among co-located devices and experiments [8].

The systems that are closest in spirit to CrowdLab are AnonySense, PRISM and DieselNet. In many ways, AnonySense represents a language-based solution (rather than a hardware virtualization solution) to the same problem we are trying to solve. The AnonySense client runs on volunteer smartphones, periodically downloads experimental work, and ships reports back to a researcher. Many of the design choices made by AnonySense are appealing: tasks are lightweight, isolation is enforced through the language interpreter, and participation does not require hooks into the low-level resource manager. In contrast to Anonymsense, PRISM uses a system call interposition based solution. Many of the trade-offs for these features have been discussed earlier, and particularly because of their lightness, we believe that there is a place for systems like AnonySense and PRISM as well as CrowdLab. Similarly, DieselNet provides the scheduling and low-level access features of CrowdLab, but restricts experiments to bus routes in Amherst, Massachusetts. However, by utilizing volunteers' personal devices, CrowdLab has the potential to run guests in a much wider range of environments.

BlueMonarch [27] is a system for testing Bluetooth applications and is aimed at helping researchers test prototypes "in the wild." BlueMonarch has the very attractive quality of only requiring access to a single machine instead of large pools of volunteer devices. The tester's machine probes its environment through the Bluetooth discovery protocol and uses those measurements to build a model of the link between devices; these models can then be used to emulate more complex interactions. The main drawback of BlueMonarch is that it requires a researcher to be in control of whatever device is running the BlueMonarch emulator. We believe that CrowdLab and BlueMonarch are ultimately complementary and that by adding support in CrowdLab for a low-level Bluetooth interface and running BlueMonarch as a guest, the number of environments in which Bluetooth applications could be tested would be greatly increased.

VIII. CONCLUSION

In this paper, we have presented a new architecture for mobile testbeds called CrowdLab. CrowdLab allows researchers to run guest code on volunteer mobile devices without sacrificing the features of other infrastructure-bound testbeds. Our experiments show that many mobile device users should be able to contribute several hours of experimental time to CrowdLab each day, and that CrowdLab provides a stable testbed environment in the face of mobility-induced churn.

ACKNOWLEDGEMENTS

This research was supported in part by IBM, the National Science Foundation (CNS-0720717), and Nokia Research. We would also like to thank Luis Tobon for assisting with the power measurements and the anonymous reviewers for their feedback.

REFERENCES

- [1] Y. Anokwa *et al.*, "Open source data collection in the developing world," *Computer*, vol. 42, pp. 97–99, 2009.
- [2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP*, October 2003.
- [3] A. Birrell *et al.*, "Grapevine: an exercise in distributed computing," in *CACM*, 1982.
- [4] S. bum Suh, "Secure architecture and implementation of xen on arm for mobile devices," Xen Summit, Spring 2007, IBM T.J. Watson.
- [5] R. Chandra *et al.*, "MultiNet: Connecting to multiple IEEE 802.11 networks using a single wireless card," in *INFOCOM*, March 2004.
- [6] S. Cheshire *et al.*, "Dynamic configuration of ipv4 link-local addresses," Internet RFC 3927, May 2005.
- [7] T. Clausen, "Optimized link state protocol OLSR," Internet RFC 3626, October 2003.
- [8] C. Cornelius *et al.*, "AnonySense: Privacy-aware people-centric sensing," in *MobiSys*, June 2008.
- [9] "DETER network security testbed," <http://www.deterlab.net>.
- [10] T. Das *et al.*, "Prism: platform for remote sensing using smartphones," in *MobiSys*, June 2010.
- [11] P. De *et al.*, "MiNT-m: An autonomous mobile wireless experimentation platform," in *MobiSys*, June 2006.
- [12] "GENI: Global environment for network innovations," <http://www.geni.net>.
- [13] H. Garcia-Molina, "Elections in a distributed computing system," *Computers, IEEE Transactions on*, vol. C-31, no. 1, pp. 48–59, jan. 1982.
- [14] P. Gilbert, E. Cuervo, and L. P. Cox, "Experimenting in mobile social contexts using jellynets," in *HotMobile*, February 2009.
- [15] B. Hull *et al.*, "CarTel: A distributed mobile sensor computing system," in *SenSys*, November 2006.
- [16] D. Irwin *et al.*, "Sharing networked resources with brokered leases," in *USENIX*, June 2006.
- [17] K. Jang *et al.*, "Implementation and evaluation of a mobile planetlab node," in *ROADS*, October 2009.
- [18] D. Johnson *et al.*, "Mobile emulab: A robotic wireless and sensor network testbed," in *INFOCOM*, 2006.
- [19] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," *ACM Transactions on Computer Systems*, vol. 10, pp. 3–25, 1992.
- [20] B. W. Lamson, "How to build a highly available system using consensus," in *WDAG*. Springer-Verlag, 1996, pp. 1–17.
- [21] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," in *ASPLOS*, October 2002.
- [22] "Multicast dns," <http://www.multicastdns.org>.
- [23] L. Peterson *et al.*, "Experiences building PlanetLab," in *OSDI*, November 2006.
- [24] —, "GENI design principles," *IEEE Computer*, vol. 39, no. 9, September 2006.
- [25] D. Raychaudhuri *et al.*, "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," in *WCNC*, March 2005.
- [26] G. Smith *et al.*, "Wireless virtualization on commodity 802.11 hardware," in *WinTech*, September 2007.
- [27] T. J. Smith, S. Saroiu, and A. Wolman, "Bluemonarch: a system for evaluating bluetooth applications in the wild," in *MobiSys '09*, June 2009.
- [28] P. Stanly-Marbell and L. Iftode, "Scylla: A smart virtual machine for mobile embedded systems," in *HotMobile*, December 2000.
- [29] D. B. Terry *et al.*, "Managing update conflicts in bayou, a weakly connected replicated storage system," in *SOSP*, 1995, pp. 172–183.
- [30] "VMware MVP," <http://www.vmware.com/products/mobile>.
- [31] B. White *et al.*, "An integrated experimental environment for distributed systems and networks," in *OSDI*, December 2002.
- [32] A. Williams, "Requirements for automatic configuration of ip hosts," Internet-Draft draft-ietf-zeroconf-reqts-12, September 2002.
- [33] X. Zhang *et al.*, "Study of a bus-based disruption tolerant network: Mobility modeling and impact on routing," in *MobiCom*, September 2007.