Experimenting in Mobile Social Contexts Using JellyNets

Peter Gilbert
Duke University
Durham, NC
gilbert@cs.duke.edu

Eduardo Cuervo
Duke University
Durham, NC
ecuervo@cs.duke.edu

Landon P. Cox
Duke University
Durham, NC
lpcox@cs.duke.edu

ABSTRACT

Programmable consumer devices have placed computation within arm's reach at all times and in all places. Unfortunately, researchers interested in investigating this phenomenon often struggle with the expense, inconvenience, and limited scale of existing experimental platforms. In this paper, we introduce a new experimental platform for mobile and pervasive computing based on *JellyNets*, an abstraction for exposing experiments to arbitrary mobile social contexts.

1. INTRODUCTION

Programmable consumer devices such as smartphones are tightly interwoven with people's daily lives, and are nearly as likely to be carried by their owners as a set of house keys or wallet. The almost constant physical proximity of these devices to their owners creates opportunities for social computing applications that are impossible with bulkier or less powerful mobile computing hardware such as laptops and Palm Pilots. Researchers are beginning to explore these opportunities through a wide range of new *mobile social services* including mobile social networking [11], participatory sensing [5], and micro-blogging [13]. As a result, the mobile research community is facing the question of how to provide an experimental platform for exposing prototype systems to a diverse and dynamic set of mobile social contexts.

Large-scale experimental platforms are an indispensable tool for distributed systems researchers. The widespread embrace of Internet testbeds [26, 31, 12] attests to the value of shared cyber-infrastructure for exposing measurement frameworks and system prototypes to live Internet phenomena. In addition, recent projects aimed at providing experimental platforms for mobile computing and wireless networking [27, 32, 18, 9] have given researchers a larger and more effective toolbox for evaluating system designs. The significant investment in GENI [25] is motivated by a cross-community desire to stitch these existing point solutions into a single coherent computing platform. However, despite this large and distinguished body of work, no system currently allows researchers to fully explore the enormous potential of consumer device-enabled mobile social services.

HotMobile 2009, February 23–24, 2009, Santa Cruz, CA, USA.

In this paper, we take the position that it is feasible and valuable for the mobile and pervasive computing research community to develop a general-purpose experimental platform for exposing experiments 1) to *human mobility* by running code on live consumer devices, and 2) to an arbitrary set of work, residential, and recreational *social contexts* by allowing co-located client bystanders to interact with experiments hosted in their vicinity. The over-arching goal of such a platform would be to allow researchers to perform mobile social experimentation in any social setting where programmable consumer devices are present.

To meet this goal, we are developing a platform based on the *JellyNet* abstraction for mobile and pervasive computing experimentation. All sensor, compute, network, and storage resources underlying a JellyNet are contributed by a federation of co-located, autonomous, volunteer consumer devices. Because JellyNets must support experimentation in arbitrary social contexts, physical-resource pools must be collected opportunistically, and shaped into a unified computing environment in "spineless" settings where dedicated backbone resources and control mechanisms are unavailable. Hosting experiments under these conditions introduces a number of technical and social challenges.

The main technical challenge of establishing a JellyNet is developing a coherent network environment with familiar address-leasing, naming, and routing services using physical resources that can be withdrawn without notice, usually within tens of minutes of being allocated. From the perspective of a developer, programming to the JellyNet abstraction should be nearly indistinguishable from programming a set of dedicated, co-located devices. Co-located client devices cannot be modified, and must interact with experiments executing in a JellyNet through standard managed-network interfaces such as DHCP, DNS, IP, and HTTP.

Social challenges arise from a JellyNet's need for self-interested human volunteers to provide untrusted guest code with battery-draining physical resources and access to their potentially sensitive mobility patterns. Volunteer resource contributers must be sure that hosted experiments will not compromise their location privacy, drain their battery, or monopolize their sensor, compute, network, and storage resources.

This paper describes a basic architecture for implementing JellyNets, focusing on several core challenges and leaving many more for future work. The foundation of our architecture is two self-organizing layers. The bottom layer is an ad-hoc Virtual WiFi [6] network called a *site network* that provides a unified wireless interconnect for all actors within a geographic site. On top of this interconnect,

resource-contributers maintain a fault-tolerant, site-scoped tree structure called a site directory (*s-dir*). S-dir state repositories form the basis of a JellyNet's higher-level features, including address-leasing, naming, and tasking services and *in-situ k-anonymity* location-privacy guarantees. Initial experience with our prototype JellyNet implementation is encouraging. Despite Virtual-WiFi multiplexing and virtualization overhead, co-located client devices experience adequate performance accessing experimental services hosted within a JellyNet.

The rest of the paper is organized as follows: Section 2 provides background information for understanding JellyNets, Section 3 describes the ad-hoc self-organizing lower layers of a JellyNet and the higher-level features they enable, Section 4 describes related work, and Section 5 provides our conclusions.

2. BACKGROUND

In this section we describe three enabling technologies on which our JellyNet architecture relies: programmable consumer devices, pocket hypervisors [10], and Virtual WiFi [6].

A key unknown for building an experimental platform based on JellyNets is whether volunteer consumer devices will have enough excess energy to participate. Preliminary human-battery-interface studies are encouraging. A recent study found that it is common for users to charge their devices with significant residual capacity still available [2]; of the users in the study, more than half charged their phone with half of its battery capacity left, while more than 80% charged their phone with more than 20% capacity left.

Furthermore, nearly half of the device owners in this study used contextual cues such as their location, the time of day, and the need to synchronize with a PC to charge their device, rather than "low" battery capacity. This gives us hope that as battery capacities continue to increase, hardware becomes more power-efficient, and software power-management policies become more sophisticated, consumer devices will have sufficient energy to contribute to a JellyNet. We are also developing distributed power-saving strategies that utilize devices' multiple radios, but a full description of these schemes is beyond the scope of this paper.

Related to the energy constraints of mobile devices is the question of what incentives device carriers should be offered to contribute resources. We imagine a university providing powerful consumer devices to a large population of students and staff for personal and experimental use. Duke University embraced such a model in 2004 when it distributed iPods to all members of its incoming freshman class. Since then, several universities have either adopted or are exploring similar strategies for deploying location-aware services on campus [14].

The JellyNet abstraction must balance developers' need for an expressive, intuitive programming environment and device carriers' need for strong isolation, particularly performance and power isolation. To this end, JellyNets rely on autonomous *pocket hypervisors* [10] to export a virtualized hardware interface to developers and manage devices' physical resources. Pocket hypervisors are identical to desktop hypervisors like Xen and VMware in most ways, but they provide an extended interface to support virtualized wireless communication (e.g., WiFi and Bluetooth), and they contain additional mechanisms and policies to ensure power isolation. Past work on performance isolation has shown that encapsulating application state within the virtual-machine abstraction can reduce the complexity of accounting and enforcement [3, 15].

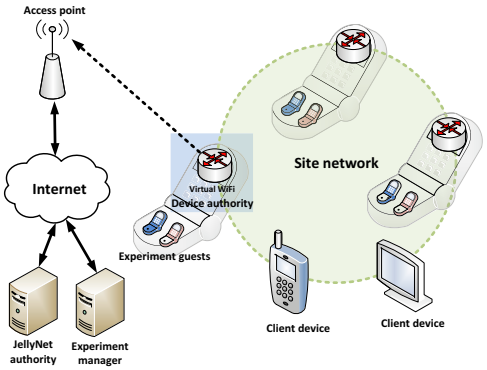

Figure 1: High-level JellyNet

Both commercial [1, 30] and research [4, 16, 19] hypervisors for mobile devices exist, though for various non-technical reasons they are not suitable for our current prototype implementation. Trango and VirtualLogix are proprietary and closed-source, MobiVMM and L4 are incomplete research projects, and Samsung's robust port of Xen for ARM-based mobile phones is only partially public. Our lab is also porting Xen to the Nokia N810 Internet tablet.

The final enabling technology for JellyNets is Virtual WiFi. Virtual WiFi is a set of techniques that allows an operating system to multiplex 802.11 association state from multiple networks across a single physical radio. FatVAP [22] and Juggler [24] recently introduced optimizations for reducing the overhead of switching among networks by taking advantage of the software-MAC implementations in many WiFi drivers. As a result, an OS can switch cards between associations on different channels fast enough (in under 7 ms) to support concurrent TCP streams over different associations. The time to switch between networks on the same channel is even less (around 500 μs).

Virtual WiFi allows JellyNet hypervisors to provide experiments with the illusion of a dedicated WiFi interface without inconveniencing device carriers. Hosted experiment code can scan for and associate with networks concurrently with other experiments on the device and the device carrier. This provides a general mobile-device abstraction for guest code and reduces the opportunity costs of participating for resource contributers.

3. PLATFORM ARCHITECTURE

Our mobile experimental platform consists of five element types: the JellyNet authority, experiment guests, experiment managers, device authorities, and client devices. Figure 1 shows a high-level diagram of these elements.

The *JellyNet authority (JA)* is the only centralized element and is removed from all physical resource allocation decisions to preserve device autonomy. Its primary responsibilities are to segment the private IP address space used within a JellyNet and to register devices and experiments.

The experiments that run within a JellyNet consist of two elements: a set of *experiment guests* and an *experiment manager*. Experiment managers are logical entities that may actually consist of many machines. Experiment guests are the coarsest unit of resource allo-

cation; they consume resources on a volunteer device, implement the experiment's logic, and interact with other co-located elements. Each guest runs in its own unprivileged virtual machine, and no two guests of the same experiment may execute on the same device. The experiment manager is responsible for maintaining an experiment's persistent state by providing off-device services such as database and logging facilities.

Each volunteer device has an associated *device authority (DA)*. DAs have complete control over their device's physical resources and make autonomous allocation and deallocation decisions. DAs are similar in function to the autonomous site authorities in SHARP [7] and Shirako [20]. DA functionality is encapsulated by a hypervisor or privileged virtual machine. The DA interface includes a set of paravirtualized system calls for its hosted guests and a simple RPC interface to communicate with remote elements. A DA's paravirtualized 802.11 and Bluetooth interfaces allow guests to scan for nearby devices, support concurrent associations with in-range networks, and can be used for WiFi localization [23].

In addition to managing physical resources, DAs also maintain 802.11 ad-hoc routing tables and implement site-scoped address-leasing, naming, and tasking services. This distributed state allows co-located elements to discover and access nearby resources and services without relying on fixed infrastructure such as inter-domain routing, DNS, or centralized coordination through the JA. As a result, guests on one device can always communicate with guests on other co-located devices, even in WiFi cold spots or when 3G-cellular networking is too costly.

Client devices can access services exported by JellyNet experiments through a site's ad-hoc network, but do not contribute resources. Client devices include iPods, iPhones, and WiFi-enabled displays. These devices expose experiments to the mobile social context in which they are executing through interactions enabled by a JellyNet. For example, a client device could access the temporary storage offered by a mobile-storage experiment, could register and browse profiles of nearby users through a mobile social networking experiment, or could inject queries directly into a sensing experiment.

Interactions between a device's DA and its experiment guests can be mediated through any number of hardware-virtualization interfaces, although our current implementation uses Xen. Interactions among distributed elements are mediated by a JellyNet's address-leasing, naming, and tasking services. A detailed discussion of how these services are implemented by a JellyNet are beyond the scope of this paper.

The rest of this paper focuses on the foundational layers on which a JellyNet's decentralized network services are built. The bottom layer is an ad-hoc Virtual-WiFi scheme called a *site network* that allows all actors within a geographic site to communicate. Through the site network, DAs maintain a replicated tree structure called a *site directory (s-dir)*. Information embedded in the s-dir allows DAs to implement each of the JellyNet's higher-level features, including address-leasing, naming, device tasking, and location-privacy protection.

3.1 Site networks

Address-leasing, naming, and routing services in a JellyNet are managed at the granularity of a geographic site by a federation of co-located DAs. Sites are defined as a connected 802.11 ad-hoc network with SSID "JNet-⟨coordinate⟩", where the coordinate is set to the network's physical location by the DA who initially seeds the network. This ad-hoc network is called the *site network*. Due to the 30 to 90-meter range of WiFi and humans' tendency to cluster together, site networks are expected to be dense, with each station in direct range of every other. Choosing different SSIDs for each network further reduces the likelihood that sites will extend beyond one or two hops in diameter or merge.

Only DAs and client devices attach directly to a site network; clients can reach experiment guests and the Internet through bridges established by the DAs. Each DA within a site uses its Virtual WiFi driver to join the network while also allowing device owners and experiment guests to concurrently connect to access points. Client devices without Virtual WiFi must dedicate their WiFi radio to the site network, but can still access the Internet whenever a DA advertises an outbound route through the ad-hoc network.

Our current implementation relies on the Optimized Link-State Routing protocol (OLSR) [8] to manage IP routing within a site network, but other algorithms could easily be plugged in. Communication within a site must be reliable enough to support reasonable TCP performance among guests and clients. To ensure the stability of the site network, DAs' Virtual WiFi schedulers privilege the site network relative to all others. Each DA divides time into fixed-length epochs and assigns a fixed percent of every epoch to the site network. Other networks are multiplexed over the remainder, though if there are no other networks to be scheduled, a card could be put into a low-power state until the beginning of the next epoch. In our current implementation, epochs are 200 ms, and the site network is scheduled for the first 100 ms of each epoch. DAs can rely on the scheduling techniques introduced by FatVAP to efficiently schedule networks when a DA is associated with other access points in addition to the site network.

Epochs must be synchronized across DAs since out-of-phase schedules will quickly render the site network unusable. To this end, the JellyNet relies on the 802.11 timer synchronization function (TSF) of the site network to synchronize all stations. 802.11 TSF is accurate to within μs and timer interrupts in Xen can be delivered at the granularity of a single ms[1]. This is more than sufficient for synchronizing Virtual WiFi schedulers managing epochs of hundreds of ms.

Prior work has shown that TSF synchronization in ad-hoc networks begins to break down for networks that are larger than 30 stations [17], but we do not anticipate sites growing to this size. However, if a network begins to grow to a dangerous size, DAs can determine the network's current size from information embedded in the site directory (as described in Section 3.2) and, if needed, start a new site network with its own TSF.

With synchronized Virtual WiFi schedulers, there is no need to induce buffering of site network traffic at other DAs via power-saving mode (PSM). Each DA's Virtual WiFi driver only needs to buffer outbound messages destined for the site network. Unfortunately, client devices are oblivious to the epoch schedule since they are only associated with the site network.

[1]The default granularity for timer interrupts in Xen is 10ms. We changed this to 1ms for our prototype.

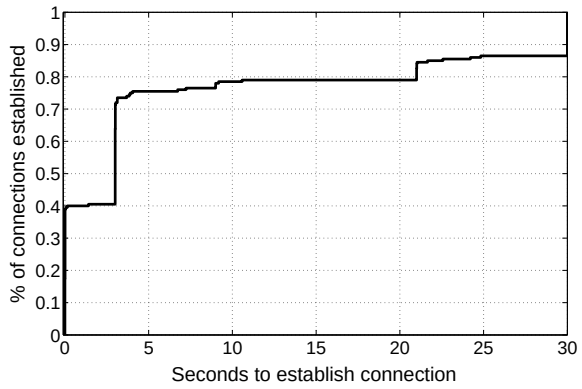

Figure 2: CDF of clients' time to establish a TCP connection with an experiment guest through a site network

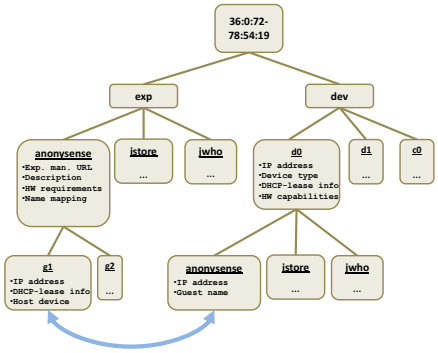

Figure 3: Site directory

Ideally, DAs would be able to broadcast a PSM frame to trick clients into buffering messages destined for the site network. Unfortunately, we have not found any WiFi drivers with a stable implementation of PSM for ad-hoc mode. Thus, when all DAs activate the site network half the time, half of the messages sent by clients are expected to be lost, regardless of whether DAs set the PSM bit between transitions. On the other hand, messages destined for client devices will continue to be buffered properly by both DAs and access points.

This results in a network where half of all messages sent by clients are lost, but very few messages destined for clients are lost. Because TCP employs exponential back-off in retransmitting SYN messages to start a connection, these message losses can lead to long waits for establishing a connection. Figure 2 shows the latency of a client device establishing a TCP connection with an experiment guest in our prototype JellyNet implementation for x86 laptops. DAs in this network were associated with the site network and an access point on a different channel. The distribution reflects the default behavior of TCP in Linux, which is to retransmit SYNs after three, nine, and 21 seconds if necessary. Because a DA typically spends about 75ms of every 200ms epoch active in the site network[2], approximately 40% of connections started immediately. However, more than 25% of connections took longer than 3 seconds to start, and more than 10% were still not established after 30 seconds, which we chose as the timeout value. Once established, TCP connections were capable of remaining connected and transferring megabytes of HTTP data. However, throughput varied widely over the course of lengthy connections due to loss of data packets and ACKs.

JellyNets can eliminate this variability by taking advantage of the fact that clients only contact DAs directly on two occasions: 1) for DNS resolutions, and 2) to route traffic to experiment guests or the Internet. As long as a single DA serves both roles, all of a client's 802.11 frames will be destined for a single gateway station. As a result, JellyNets can improve client TCP performance by keeping gateway DAs' Virtual WiFi interface associated with the site network for the entire epoch. No restriction needs to be placed on traffic during the first half of each epoch, but any client messages received by the gateway during the remainder of the epoch must be buffered. Under such a scheme, outbound client messages are rarely lost, but at the price of preventing experiment guests and the device carrier from associating with other networks. With anticipated site populations close to one or two dozen nodes, we do not believe that it will be difficult to identify a subset of DAs able to serve in this role.

Finally, using the site-network's TSF to synchronize Virtual WiFi schedulers requires DAs to ignore TSF updates from all other networks. This limits a JellyNet's ability to support associations with multiple ad-hoc networks. Though we have not implemented such a scheme, there is no fundamental reason why a set of experiment guests within a site should not be able to form an ad-hoc network using the site-network TSF as long as DAs synchronize the scheduling of these networks in addition to the site network. However, any ad-hoc networks that are not under the complete control of a site's DAs would almost certainly be unable to synchronize their stations.

3.2 Site directories

Services such as address leasing, naming, and tasking are built on the foundation of a stable site network. DAs provide these higher-level services by maintaining additional distributed state in a replicated data structure called a *site directory (s-dir)*. An s-dir is a tree structure that stores information about the resources, services, and experiments available within a geographic site. Managing the consistency of this replicated state in the face of high churn rates is a crucial challenge for implementing JellyNets, and we leave a more detailed discussion of this subject for future work.

Figure 3 shows an example tree. The root of each s-dir is the geographic coordinate of the site. The first DA in a site generates this coordinate and seeds the ad-hoc network. S-dirs have two branches below the root; one branch stores information about the experiments running within a site and the other stores information about the devices present. Each experiment and device node has an associated list of meta-data properties that describes its features.

Leaf nodes representing guest virtual machines sit below the device and experiment nodes. Guest state is stored redundantly under both sub-trees to improve lookup efficiency. In Figure 3, AnonySense guest g1, running on device d0, is represented by the two entries connected by a light arrow.

[2]Switching between infrastructure and ad-hoc networks on different channels takes approximately 25ms using the HAL underlying the MadWiFi driver used in our implementation. An older HAL allowed us to switch within 8ms.

3.3 Higher-level Features

A JellyNet uses information stored in its s-dir to implement higher-level features such as address-leasing, naming, and tasking services and strong location-privacy guarantees.

3.3.1 Address-leasing

JellyNet address-leasing services dynamically bind IP addresses to client devices, DAs, and experiment guests. The JA reserves three disjoint ranges of the private IP space for addressing the three element types. Within these ranges, individual addresses are bound to elements by the DAs within a site; the JA is not involved.

One DA runs a DHCP server for the site network that leases addresses to other DAs, client devices, and experiment guests. Each lease includes an address, a length, a subnet mask, DNS server, and a gateway when appropriate. All leasing state is embedded in the s-dir and fully replicated at all DAs for fault tolerance. If the DA that was providing DHCP service exits the network, any other remaining DA can use its copy of the s-dir to seamlessly pick up where the old server left off. Fault-detection and service fail-over is beyond the scope of this paper. Importantly, client devices only need to run a standard DHCP client in order to obtain an IP address; they do not participate in OLSR. This allows all WiFi-enabled devices such as iPods, iPhones, and public displays to acquire addresses and communicate with experiments hosted by the JellyNet.

3.3.2 Naming

A JellyNet 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, but introduces a new top-level domain called `jnet`. 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 a JellyNet's s-dir. For example, using the s-dir in Figure 3, a lookup of the name "`g1.anonysense.exp.jnet`" 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 `.jnet` names locally.

3.3.3 Tasking

The s-dir is also used by a JellyNet to assign experiment guests to DAs. To protect device carriers' anonymity, DAs must only download guest images from the JA using an anonymizing communication channel such as a MIX-net or public access point. Once an image has been injected into a site, the JellyNet uses *epidemic instantiation* to assign guests to devices.

Under epidemic instantiation, DAs are responsible for instantiating application guests on other co-located devices within a site; in this role, DAs are similar to brokers in Shirako [20]. DAs use their site's s-dir to discover other potential hosts and invoke a well-known RPC interface running under each DA to request that the device run an instance of the guest. Epidemic instantiation also allows the JellyNet to geo-cache experiments. Even if a device running an application guest leaves the site, a new guest can be quickly instantiated on any devices that subsequently arrive. A JellyNet can experience 100% turn-over without interrupting an experiment.

3.3.4 Location privacy

Finally, s-dirs help DAs enforce *in-situ k-anonymization.* K-anonymization ensures that an adversary cannot narrow a device carrier's identity to fewer than k possibilities [29]. AnonySense [9] is an experimental platform for participatory sensing, and provides a useful point of comparison. As with a JellyNet-enabled platform, AnonySense runs experiments on volunteer consumer devices. Neither platform tracks device locations to preserve volunteers' location privacy. However, because AnonySense does not support interactions among co-located elements, it is only able to impose k-anonymity restrictions outside of a physical location by filtering the set of devices that *evaluate* an experiment. It cannot guarantee device carriers that at least k devices within a geographic area will *execute* the experiment.

Consider the following scenario. Using the AnonyTL tasking language an adversary can submit a task to AnonySense that periodically reports its location, but only executes on the devices of Duke professors located in the city of San Diego. AnonySense can ensure that only devices belonging to Duke professors will evaluate the task and that the total number of Duke professors is at least k. However, even if at least k Duke professors' devices evaluate the task, only the Duke professors' devices that are in San Diego will actually *execute* it, and this number may be much smaller than k. For example, if the attacker knows that one Duke professor is likely to be in San Diego for a conference, then once she arrives in San Diego the professor will be the only participant periodically reporting her location to the adversary.

This attack can be defeated if devices are able to ensure that at least k other participants are co-located with them when they execute a task. In the case of the professor, she would only execute the task when k or more of her colleagues were co-located with her at the conference. A JellyNet can accommodate this policy through the s-dir: each DA can consult the s-dir before hosting AnonySense guests to ensure that at least k other devices are nearby. It should be noted that because JellyNets' k-anonymity decisions can only be evaluated in-situ via an s-dir, our platform cannot support less strict policies.

4. RELATED WORK

As with cyberinfrastructure such as PlanetLab [26], EmuLab [31], and DETER [12], our platform aims to help researchers expose experimental systems to realistic conditions, and is complementary to existing mobile platforms such as ORBIT [27, 28], Mobile Emulab [21], CarTel [18], DieselNet [32], and AnonySense [9]. Each of these platforms places important limitations on the experimental setting in one or more of the following ways: restricting guest code to declarative, SQL-like languages [18, 9], utilizing only vehicular [32, 18] or pre-programmed robotic mobility [27, 21], or not supporting interactions between co-located resource-contributors [9]. To the best of our knowledge, no existing system provides a general framework for exposing experiments to arbitrary mobile social contexts.

5. CONCLUSION

This paper has argued that an experimental platform for mobile social contexts based on the JellyNet abstraction is feasible and valuable. JellyNets provide experiments with the illusion of a set of co-located, fully-programmable, dedicated consumer devices and exposes them to arbitrary mobile social contexts by enabling interactions with unmodified client bystanders. We have described solutions to some of social and technical challenges of implementing JellyNets and called attention to other opportunities for future work.

6. REFERENCES

[1] F. Armand, M. Gien, G. Maigné, and G. Mardinian. Shared Device Driver Model For Virtualized Mobile Handsets. In *MobiVirt*, June 2008.

[2] N. Banerjee, A. Rahmait, M. D. Corner, S. Rollins, and L. Zhong. Users and batteries: Interactions and adaptive energy management in mobile systems. In *UbiComp*, September 2007.

[3] 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.

[4] S. bum Suh. Secure architecture and implementation of xen on arm for mobile devices. Xen Summit, Spring 2007, IBM T.J. Watson.

[5] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Rmanathan, S. Reddy, and M. B. Srivastava. Participatory sensing. In *World-Sensor-Web*, 2006.

[6] R. Chandra, P. Bahl, and P. Bahl. MultiNet: Connecting to multiple IEEE 802.11 networks using a single wireless card. In *INFOCOM*, March 2004.

[7] J. Chase, B. Chun, Y. Fu, S. Schwab, and A. Vahdat. SHARP: An architecture for secure resource peering. In *SOSP*, October 2003.

[8] T. Clausen. Optimized link state protocol OLSR. Internet RFC 3626, October 2003.

[9] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos. AnonySense: Privacy-aware people-centric sensing. In *MobiSys*, June 2008.

[10] L. P. Cox and P. Chen. Pocket hypervisors: Opportunities and challenges. In *HotMobile*, February 2007.

[11] L. P. Cox, V. Marupadi, and A. Dalton. Smokescreen: Flexible privacy controls for presence-sharing. In *MobiSys*, May 2007.

[12] DETER network security testbed. http://www.deterlab.net.

[13] S. Gaonkar, J. Li, R. R. Choudhury, L. P. Cox, and A. Schmidt. Micro-Blog: Privacy-aware people-centric sensing. In *MobiSys*, June 2008.

[14] J. D. Glater. Welcome, freshmen. have an ipod. New York Times, August 20, 2008.

[15] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in xen. In *Middleware*, November 2006.

[16] S. Hessel, F. Bruns, A. Bilgic, A. Lackorzynski, H. Haertig, and J. Hausner. Acceleration of the L4/Fiasco Microkernel Using Scratchpad Memory. In *MobiVirt*, June 2008.

[17] L. Huang and T.-H. Lai. On the scalability of IEEE 802.11 ad hoc networks. In *MobiHoc*, June 2002.

[18] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden. CarTel: A distributed mobile sensor computing system. In *SenSys*, November 2006.

[19] S. hwan Yoo, Y. Liu, C.-H. Hong, C. Yoo, and Y. Zhang. MobiVMM: A Virtual Machine Monitor for Mobile Phones. In *MobiVirt*, June 2008.

[20] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, and D. Becker. Sharing networked resources with brokered leases. In *USENIX*, June 2006.

[21] D. Johnson, T. Stack, R. Fish, D. Flickinger, L. Stoller, R. Ricci, and J. Lepreau. Mobile emulab: A robotic wireless and sensor network testbed. In *INFOCOM*, 2006.

[22] S. Kandula, K. C.-J. Lin, T. Badirkhanli, and D. Katabi. FatVAP: Aggregating AP Backhaul Bandwidth. In *NSDI*, April 2008.

[23] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello, and B. Schilit. Place lab: Device positioning using radio beacons in the wild. In *Pervasive*, 2005.

[24] A. J. Nicholson, S. Wolchok, and B. D. Noble. Juggler: Virtual networks for fun and profit. In *University of Michigan, Technical Report CSE-TR-542-08*, April 2008.

[25] L. Peterson, T. Anderson, D. Blumenthal, D. Casey, D. Clark, D. Estrin, J. Evans, D. Raychaudhuri, M. Reiter, J. Rexford, S. Shenker, and J. Wroclawski. GENI design principles. *IEEE Computer*, 39(9), September 2006.

[26] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir. Experiences building PlanetLab. In *OSDI*, November 2006.

[27] 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 *WCNC*, March 2005.

[28] G. Smith, A. Chaturvedi, A. Mishra, and S. Banerjee. Wireless virtualization on commodity 802.11 hardware. In *WinTech*, September 2007.

[29] L. Sweeney. k-Anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5), January 2002.

[30] Trango virtual processors. http://trango-vp.com.

[31] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI*, December 2002.

[32] X. Zhang, J. Kurose, B. N. Levine, D. Towsley, and H. Zhang. Study of a bus-based disruption tolerant network: Mobility modeling and impact on routing. In *MobiCom*, September 2007.