Implementation and Evaluation of a Mobile PlanetLab Node

Keon Jang[†] Sangman Kim[†] Geoffrey M. Voelker[‡] Sue Moon[†]

[†]KAIST [‡]University of California, San Diego

ABSTRACT

With the introduction of cellular phones, mobile wireless communication has become an integral part of day-to-day life in this century. A testbed is crucial in continuing cutting-edge wireless technology research and development since physical and MAC layers in mobile wireless communication have many parameters pertinent to performance that are often not tractable in analysis or simulation alone. Although many wireless testbeds have been built, most are not general purpose, have a limited access policy, or focus on single technologies like WiFi. In this paper, we present the implementation and evaluation of a mobile testbed node. This node will be the founding block of a mobile wireless testbed that we envision to be as open as PlanetLab to researchers.

1. INTRODUCTION

With the introduction of cellular phones, mobile wireless communication has become an integral part of day-to-day life in this century. The competition for the mobile wireless communication market is fierce; so are the investment and drive for new mobile wireless technologies with lower latency and higher bandwidth. As physical and MAC layers in mobile wireless communication have many parameters pertinent to performance that are often not tractable in analysis or simulation alone, a testbed is an integral part in cutting-edge wireless technology development.

A wide range of wireless testbeds have been built in the past: WiFi mesh networks [6, 4], sensor networks [18, 10], long-range multi-hop wireless networks [21], just to name a few. But only a few include mobility in the design. Explorebots [9] and Mobile Emulab [17] offer indoor testbeds with robots mounted with wireless communication interfaces. The Kiosknet project at the University of Waterloo [23], the CarTel project at MIT [13], and the DieselNet project at the University of Massachusetts at Amherst [1] all use WiFi devices mounted on vehicles and have specific design goals. Ormont *et al.* have built a city-wide vehicular infrastructure and conducted wide-area wireless measurement experiments [20]. They mounted laptops on a bus, and the laptops are equipped with an EV-DO (EVolution-Data Only) 3G modem and a WiFi interface.

The outdoor testbeds, with the exception of Mobile Emulab, are not general purpose and have a limited access policy, typically to the individual research group. For those who are used to the easy access of PlanetLab or Emulab, obtaining access is a major deterrent to utilizing such testbeds. This difficulty has motivated us to build a mobile wireless testbed that is as open and accessible as PlanetLab and Emulab.

Most wireless testbeds focus on WiFi, and only a small number of testbeds support commercial mobile wireless technologies. We design our wireless testbed node to support various wireless technologies and easily support new technologies as they come to the market. Due to the prevalence of Windows in the world, many commercial wireless services provide modems only with Windows drivers and led us to support Windows in our mobile node.

Virtualization is a key technology enabling an open and shared testbed like PlanetLab or Emulab. PlanetLab uses container-based virtualization to share a single physical machine with many users by isolating each user in a sliver. Container-based virtualization provides high scalability over hypervisor-based VMMs or hosted VMMs [24], but is limited to using a single, shared OS kernel for all users. Our mobile node needs to run Windows because some target wireless technologies provide only Windows drivers. We design our mobile node to use a combination of hosted VMM technology and container-based technology. The basic idea is to run PlanetLab on top of the Windows OS via VMware Workstation. This way we do not compromise scalability and eliminate development cost, but does raise the question of overhead.

In this paper we present our mobile node design and evaluate its performance. Mobile wireless technologies are yet to support network speeds of 1 Gbps or higher, and modern multi-core PCs are capable of handling 10Gbps. We evaluate our node design using a gigabit Ethernet interface and show that it can be used to support various wireless experiments simultaneously. This node will be the founding block of a mobile wireless testbed that we envision to deploy in the near future.

The rest of paper is organized as follow. We discuss major challenges and requirements of a mobile node and present our design of a prototype in Section 2. In Section 3, we ex-

plain details of our implementation and in Section 4 present preliminary evaluation results. In Section 5 we conclude with discussions and future work.

2. MOBILE PLANETLAB NODE DESIGN

In this section we outline the design of our mobile wireless testbed node, discussing alternatives and the motivations for our design choices.

2.1 Our Design

As outlined in [16], building a mobile tesbed has different challenges from a wired testbed. First, the cost of deploying and maintaining a mobile wireless testbed is very high and labor intensive. Thus it makes more sense to allow multiple network interfaces, homogeneous or not, on a single node, and take advantage of the deployed infrastructure for maximum utilization. For the same reason, we should make the testbed as open and publicly available as possible to the broad research community.

Second, the control channel between a mobile node and the rest of the testbed or the central control node, if any, should be reliable, but we cannot depend on reliable communication in mobile wireless networks. GPRS and 3G networks exhibit highly variable latency and frequent blackouts [7, 8]. To address this issue, we consider aggregating multiple interfaces into one virtual channel. Rodriguez *et al.* have proposed a Mobile Access Router (MAR) design that exploits diversities in technology, networks, and channels, and demonstrated that MAR achieves zero blackout and high bandwidth through multiple interface aggregation [22]. In this work we choose to use a single interface for the control channel as a first step, and leave multiple interface aggregation for future work.

PlanetLab is one of the most successful network testbeds with over 1000 nodes deployed at about 500 sites around the globe. The simple management framework based on container-based virtualization and its "bring your own lunch" participation model have enabled an explosive growth in usage, now hosting thousands of services. Its extensive user base and our familiarity with its management framework have appealed to us strongly, and we have chosen to use PlanetLab as a base management framework.

Transplanting PlanetLab into a mobile wireless network is not straightforward, however. First, PlanetLab was originally designed with wired networks in mind, and expects a static IP per node at an Ethernet interface. Most mobile ISPs offer dynamically allocated IP addresses, and do not offer static IP addresses at all. We choose to use a *proxy node* to relay communication between a mobile wireless PlanetLab node and MyPLC (a central management node). The proxy node has a static IP address that MyPLC sees. The details of the proxy node operation are in Section 3.

The choice of PlanetLab limits us to only a handful of mobile wireless technologies. PlanetLab is Linux-based, but many mobile wireless modems come with Windows device drivers and connection managers only and no Linux support. In particular, WiBro,[1] which is the mobile Internet service with the highest bandwidth at this point, does not provide a Linux driver or a connection manager.[2] Worse yet, some HSDPA modems are configured as serial modems and do not work as an Ethernet interface; as a result, PlanetLab nodes cannot boot up through those HSDPA modems.

The prevalence of Windows-based devices has led us to run Windows and PlanetLab simultaneously on a single machine. We choose to run PlanetLab on VMware Workstation [2] in Windows and operate wireless modems in Windows. VMware Workstation is a hosted VMM (Virtual Machine Monitor) that allows multiple operating systems running on a host OS. An advantage of this architecture is that we no longer have to worry about acquiring Linux-based device drivers. The downside is that we compromise the ease of remote system management since we have to maintain Windows updates as well as those for PlanetLab. However, most people would find mobile wireless nodes very hard to deploy and our testbed is likely to be under single administrative control. Installation and management of VMware Workstation and Windows incur extra overhead compared to vanilla PlanetLab, but are out of the scope of this paper.

We expect users of our mobile testbed to demand exclusive access to modems, if not to the node. To enforce exclusive access per modem, we need network stack isolation. PlanetLab offers port-based network stack isolation, but not per modem. Trellis is a PlanetLab-based software platform used to host multiple virtual networks and implements network stack virtualization. It adopts NetNS [5] and provides each "container" with its own in-kernel virtual devices and related data structures. Trellis maps a sliver to a namespace one-to-one and a namespace can claim any network interfaces. Under Trellis, multiple slivers can share a physical network interface as well. We have chosen to use NetNS and allow only one NetNS per VNIC. This way we enforce network stack virtualization and allow per-user exclusive access to modems.

2.2 Design Alternatives

A hypervisor-based VMM is another way of running multiple instances of OSes in a single machine. In hypervisor-based VMMs, there exists a host VM that manages hardware and multiplexes it to guest OSes. Unlike hosted VMM, the hypervisor runs below the host OS, and schedules all OSes. In the hosted VMM's case, the host OS manages scheduling and hardware, and the VMM itself is a process running in the host OS. Thus a hypervisor-based VMM is thinner and

[1]WiBro is a WiMax-compatible standard developed and deployed in South Korea. KT launched WiBro service in 2006, and it currently covers the greater metropolitan area of Seoul and a few hotspots in Daejeon and Pohang.

[2]Running a Windows NDIS driver in Linux using NDIS Wrapper (http://sourceforge.net/projects/ndiswrapper/) may enable bringing up the interface, but connecting to the ISPs network requires a Connection Manager.

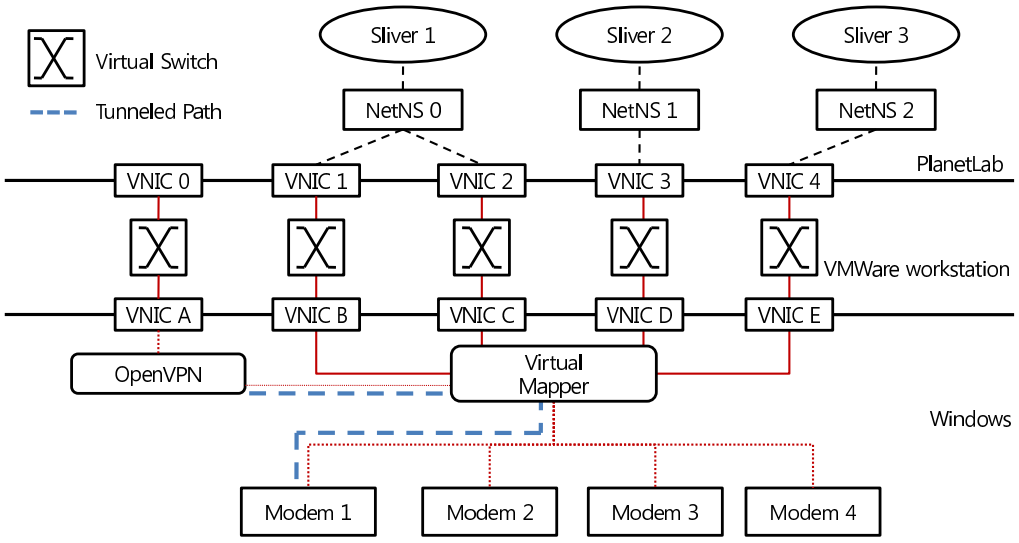

Figure 1: Network subsystem of a mobile PlanetLab node

more efficient than hosted VMMs in general.

We choose to use a hosted VMM over a hypervisor-based VMM for its efficiency in I/O. Wireless modems mostly use USB these days, and running an USB modem in a virtualized environment incurs overhead of emulating an USB controller. Iwamatsu reported that an USB 100 Mbps Ethernet modem in a Xen guest domain requires more than three times the CPU utilization of running it in the native OS while achieving 60∼70% of native throughput [15]. Our design of running Windows as a host OS and forwarding network packets back and forth to a PlanetLab VM avoids the overhead of USB I/O forwarding.

If individual users can run their own kernel, one can custom-configure the kernel as necessary. CoreLab is one example using a hosted VMM, but memory and computation overhead is substantial when the number of VMMs increases [19]. Although CoreLab's design provides a flexible environment for users, we choose to use container-based virtualization for better scalability.

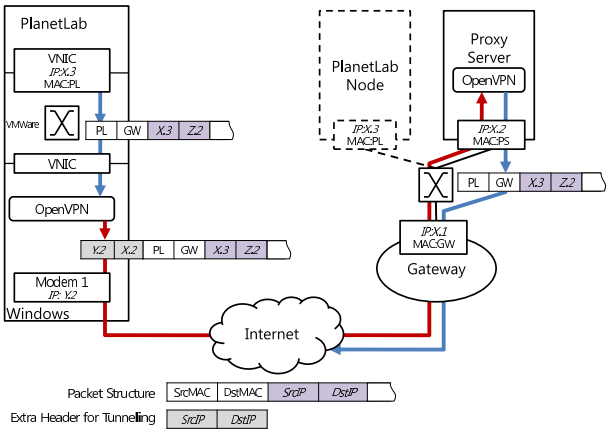

Figure 2: Packet encapsulation and decapsulation through a proxy server

3. IMPLEMENTATION

In this section we describe the implementation details of our mobile PlanetLab node design. For concreteness, we assume a node running PlanetLab v4.2 inside VMware Workstation v6.5 on Windows XP as a host operating system.

The node can have multiple modems and we configure as many VNICs in VMware one-to-one. We configure one extra VNIC that is used to communicate with MyPLC and provide remote access for users. PlanetLab selects the first VNIC as its main interface to communicate with MyPLC. We use OpenVPN[3] to create an Ethernet tunnel between a PlanetLab node and a proxy server.

We illustrate the above implementation through an example in Figure 1. In this example our node has 4 modems and thus 5 VNICs in VMware. We have implemented the switching software in Windows that multiplexes and demultiplexes packets between modems and VNICs and call it a *virtual mapper*. To map the VNICs in VMware to modems in Windows, we could employ NAT between them. However, NAT makes it difficult to accept incoming connections. Another approach is to assign IP addresses from the same subnet as the modems' IP addresses to VNICs, bridge them to modems, and run the modems in promiscuous mode. However, no ISPs would allow multiple IP addresses to a single modem. We have decided to assign the IP address of a modem to a corresponding VNIC, and let the virtual mapper intercept packets before they reach Windows and forward accordingly. We implemented the virtual mapper as an Win-

[3]http://openvpn.net

dows NDIS intermediate driver.[4]

Use of the virtual switch in VMware and the virtual mapper in Windows introduces multiplexing and demultiplexing overhead in packet processing. Bhatia *et al.* report 10 to 1 performance degradation from raw Linux packet forwarding to that through Xen [3]. Sugerman *et al.* have experimented with virtual I/O devices on VMware Workstation and seen five times higher CPU utilization for the same level of throughput as raw Linux [26]. Their performance improvement comes from various optimizations to reduce world switches. Although our system employs two levels of bridging, the virtual mapper bridging does not involve any context switching and hence we expect to see negligible effect on performance.

To relay packets with correct IP addresses, we place a proxy server with a static IP address somewhere in the Internet and then assign an IP address from the same subnet as the proxy server to the first VNIC of the PlanetLab node. This way the PlanetLab node behaves as if it were attached directly to the same subnet as the proxy server.

We include Figure 2 to demonstrate the operation of a proxy server. It shows the flow path of a packet from our node to the destination via the proxy. The packet leaves PlanetLab with the source IP address of $X.3$ and the destination $Z.2$, and with the source MAC address of PL and destination MAC address of GW, as it is headed towards the gateway. When the packet passes through OpenVPN, OpenVPN adds Modem 1's IP address $Y.2$ as the source IP address and the proxy server's IP address $X.2$ as the destination IP address. When the packet arrives at the proxy server, OpenVPN strips off the extra header and transmits the packet back to the Internet with the original header. As the proxy server transmits a packet with a source IP address different from its own, its NIC should operate in promiscuous mode. Communication with MyPLC or for login goes through the proxy server, but the rest of the experiment traffic is sent out directly to a corresponding wireless modem without encapsulation.

4. EVALUATION

In previous sections we have presented the network subsystem design of our mobile PlanetLab node and its implementation details. In this section we evaluate the performance of our mobile node. The key issue is the performance overhead incurred by running PlanetLab over VMware and Windows.

The control channel between our mobile PlanetLab node and MyPLC goes through not only VMware Workstation and Windows, but also through a proxy server. We do not treat the control channel as a performance critical path and we omit its evaluation in this work.

The goal of evaluation is to characterize the additional overhead that our node implementation incurs by VMware Workstation, and yet have good networking performance as a mobile wireless node. We measure CPU usage, network throughput, and latency. Table 4 summarizes the hardware specification of the machine we used in the experiment. To measure the latency we use the $RDTSC$ instruction of the x86 architecture that returns the number of CPU cycles since machine startup. Modern computers operate at higher than 2 Ghz, and this method gives us latency in nanosecond resolution.

[4]http://msdn.microsoft.com/en-us/library/aa504394.aspx

Item	Specification
CPU	Intel Q9300 2.66Ghz Quad Core
Memory	4 GB
VM Memory	2 GB
NIC	Intel EXPI9400PT 1 GbE

Table 1: Hardware Specification

	UDP		TCP	
	Thruput (Mbps)	CPU (%)	Thruput (Mbps)	CPU (%)
PL in VMware	876	204	398	200
Ubuntu in VMware	793	112	525	160
native Ubuntu	954	11	926	24

Table 2: Maximum Achievable Throughput Measured with and without VMware (100% CPU utlization indicates full utilization of one core)

We begin our evaluation with a 1 Gbps Ethernet (GbE) link instead of wireless modems as a stress test. We connect a native Ubuntu machine to our mobile node directly via a cross cable. We use *iperf* to generate UDP and TCP traffic and measure their throughputs and CPU utilization. For comparison, we evaluate the same traffic in native PlanetLab, native Ubuntu, and Ubuntu in VMware systems. Table 2 summarizes the results of our experiment. All throughput is received throughput measured at PlanetLab in VMware. We ran each experiment with 2 GB five times and calculated the average. We configured VMware to use 2 virtual CPUs. As expected, native Ubuntu has the best performance with throughput above 900 Mbps for both UDP and TCP, and with low CPU utilization. Next, native PlanetLab's performance is close to that of native Ubuntu's. Our design shows comparable performance (about 83% of native Ubuntu's) in UDP, but only 43% in TCP. The high CPU utilization in the case of TCP indicates that computational overhead in TCP processing is the cause behind low TCP performance. Although VMware incurs non-negligible overhead as we expected, the bandwidth of about 400 Mbps translates to more than 1 802.11n cards, about 40 WiBro modems [11], or more than 50 HSDPA modems [14]. It is unlikely to have more than one WiFi modem on a node, for the testbed is for mobile technologies and WiFi is likely to serve the devices on a vehicle. Thus we deem the maximum achievable throughput of about 400 Mbps reasonable. However, the low TCP performance of our design and Ubuntu in VMware needs

# of processes		0	1	2	3	4	5
send latency (μs)	mean	62	59	62	61	56	60
	median	61	55	58	57	56	56
	90th %	74	65	72	69	66	66
recv latency (μs)	mean	163	171	229	206	239	653
	median	156	147	158	155	157	159
	90th %	200	189	227	199	205	209

Table 3: Send and receive latency under high CPU load

Traffic(Mbps)		0	100	200	300	400	500
send latency (μs)	mean	62	49	62	55	49	46
	median	61	38	45	44	40	35
	90th %	74	73	93	79	69	54
recv latency (μs)	mean	163	178	205	293	276	278
	median	156	137	168	217	214	232
	90th %	200	246	323	449	424	500

Table 4: Send and receive latency under high network load

further investigation.

Next we investigate the VMware overhead on packet processing. We modified `ping` to add timestamps and transmitted one ICMP packet per second for 300 s. Table 3 shows the measured latency between a packet's arrival at the virtual mapper to the application within a VM (receive latency) or the other way around (send latency). We ran 1 to 10 CPU-intensive background processes. The CPU utilization is 100% when 1 process is running, and 200% for the rest of the cases. We observe that the send latency is consistently smaller than the receive latency; the median send latency is about a third of receive latency. This outcome is consistent with previously reported results on native PlanetLab [25]. Send or receive latency is mostly not affected by other processes, but outliers do exist that drive the mean receive latency high. These outliers are due to timestamping at the application layer, which introduces large variation in scheduling. The results demonstrate that the VMware overhead per packet under high CPU load is not significant.

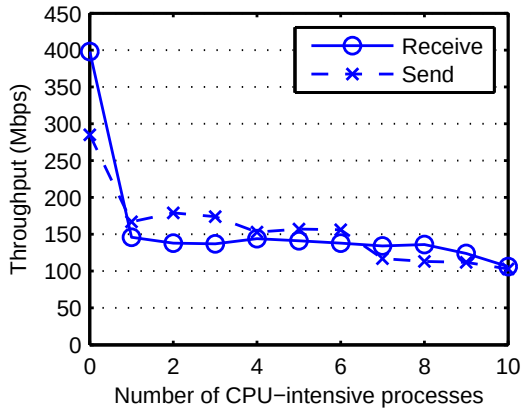

Figure 3: TCP throughput under high CPU load

We then investigate the TCP throughput with the same background processes. Figure 3 shows the measured throughput from the experiment. Although the receive latency is higher than the send latency, the receive throughput is higher than the send throughput when there is no background process. VMware Workstation implements packet aggregation on the receive path to decrease the number of interrupts, while on the send path this optimization function should lie in the guest OS driver. We have not included such an optimized driver in our PlanetLab code. This remains yet to be confirmed. As soon as we add a CPU-intensive background process, the throughput drops to about 150 Mbps for both send and receive sides, and it gradually drops to 100 Mbps as the number of background processes increases to 10. The number of background processes does not have a big impact on the TCP throughput since the I/O-intensive processes receive higher priority in scheduling.

To evaluate the performance of multiple flows through a single interface, we generate UDP traffic from 0 to 500 Mbps by 100 Mbps increments, and measure the latency of `ping`. Table 4 shows the latency between the virtual mapper and the application running in a VM in microseconds. Send latency is not affected by background traffic regardless of its volume. Receive latency increases as the background traffic increases, but for 90% of cases it stays below 500 μs. Wireless technologies other than WiFi typically have latency over tens of milliseconds, and latency of 500 μs should contribute less than 5% in end-to-end delay.

5. SUMMARY AND FUTURE WORK

We have listed challenges of building a mobile wireless PlanetLab node and addressed them in our prototype implementation. Our prototype node uses a combination of PlanetLab with VMware Workstation and Windows and provides network stack isolation to users. The adoption of Windows as the host OS provides immediate access to a wide range of mobile wireless modems.

Our VMware-based approach adds overhead of running PlanetLab as every I/O operation goes through VMware, requiring a context switch in the CPU. We have evaluated the performance overhead from VMware by comparing with native Linux performance using a gigabit Ethernet and a WiFi modem. There is non-negligible overhead in terms of TCP throughput degradation and receive latency. Still, a single machine stands to support a good number of wireless modems.

Running PlanetLab in VMware incurs extra overhead in installing and managing VMware and Windows in comparison to running simple vanilla PlanetLab. Using commercial software such as Windows and VMware Workstation adds another dimension. Automating installation and configuration of all necessary software including Windows and VMware Workstation is not straightforward due to software

license issues.

We have not addressed the mapping between slivers and namespaces and between namespaces and NICs. As only a small number of modems can be installed on a node, not all user demand can be met all the time. We are considering a reservation system to guarantee time-based exclusive access to a user. Also if we adopt network aggregation for the control channel, then we need coordinate control channel traffic with other users' demand.

We will install commercial wireless modems on a node. The cost of wireless services is expensive due to lack of flat-rate service in South Korea. We are designing an accounting system to charge users based on their usage. PlanetLab currently records all network activity using PlanetFlow [12] and PlanetFlow is a good candidate component for recording network usage in our accounting system.

Finally, our next step is to deploy a node in the real world. We are targeting buses for installation because buses run on a fixed route, which helps to repeat an experiment, and buses normally run on a open road and we can use GPS to track them. We will make our code public so that any researcher who wants to deploy their own mobile testbed can use our software.

6. ACKNOWLEDGEMENTS

We would like to thank Mikhail Afanasyev, Yuvraj Agarwal, Stefan Savage, Alex Snoeren, and Patrick Verkaik for their comments. This paper is one of the results from the project (2009-F-050-01), "Development of the core technology and virtualized programmable platform for the Future Internet" that is sponsored by MKE and KCC. We would like to express our gratitude for their full support of the research and development of the project.

7. REFERENCES

[1] Dieselnet. http://prisms.cs.umass.edu/dome/umassdieselnet.
[2] VMWare Workstation. http://www.vmware.com/product/ws/.
[3] S. Bhatia, M. Motiwala, W. Muhlbauer, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford. Hosting Virtual Networks on Commodity Hardware. In *Tech Report GT-CS-07-10*, Georgia Institute of Technology, Atlanta, GA, November 2007.
[4] J. Bicket, D. Aguayo, S. Biswas, and R. Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *MobiCom '05*, pages 31–42, New York, NY, USA, 2005. ACM.
[5] E. Biederman. NetNS. https://lists.linux-foundation.org/pipermail/containers/2007-September/007097.html.
[6] J. Camp, J. Robinson, C. Steger, and E. Knightly. Measurement driven deployment of a two-tier urban mesh access network. In *MobiSys '06*, pages 96–109, New York, NY, USA, 2006. ACM.
[7] R. Chakravorty, A. Clark, and I. Pratt. GPRSWeb: optimizing the web for GPRS links. In *MobiSys '03*, pages 317–330, New York, NY, USA, 2003. ACM.
[8] M. C. Chan and R. Ramjee. TCP/IP performance over 3G wireless links with rate and delay variation. In *MobiCom '02*, pages 71–82, New York, NY, USA, September 2002. ACM.
[9] T. A. Dahlberg, A. Nasipuri, and C. Taylor. Explorebots: a mobile network experimentation testbed. In *E-WIND '05*, New York, NY, USA. ACM.
[10] E. Ertin, A. Arora, R. Ramnath, V. Naik, S. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, H. Cao, and M. Nesterenko. Kansei: a testbed for sensing at scale. In *IPSN '06*, New York, NY, USA. ACM.
[11] M. Han, Y. Lee, S. Moon, K. Jang, and D. Lee. Evaluation of VoIP Quality over WiBro. In M. Claypool and S. Uhlig, editors, *PAM 2008*, volume 4979. Springer Berlin / Heidelberg, April 2008.
[12] M. Huang, A. Bavier, and L. Peterson. PlanetFlow: maintaining accountability for network services. *SIGOPS Oper. Syst. Rev.*, 40(1).
[13] 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 '06*, pages 125–138, New York, NY, USA, 2006. ACM.
[14] H. Ishii, T. Sao, S. Tanaka, S. Ogawa, Y. Iizuka, T. Nakamori, and T. Nakamura. Experiments on HSDPA Throughput Performance in W-CDMA Systems. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E89-A(7).
[15] N. Iwamatsu. Paravirtualized USB Support for Xen Status Update, 2009.
[16] K. Jang, S. Woo, and S. Moon. Design considerations for a mobile testbed. In *CFI'08: Proceedings of the 3rd International Conference on Future Internet Technologies*, 2008.
[17] D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau. Mobile Emulab: A Robotic Wireless and Sensor Network Testbed. pages 1–12, April 2006.
[18] R. Murty, G. Mainland, I. Rose, A. Chowdhury, A. Gosain, J. Bers, and M. Welsh. CitySense: An Urban-Scale Wireless Sensor Network and Testbed. pages 583–588, May 2008.
[19] A. Nakao, R. Ozaki, and Y. Nishida. CoreLab: an emerging network testbed employing hosted virtual machine monitor. In *CONEXT '08*, pages 1–6, New York, NY, USA, 2008. ACM.
[20] J. Ormont, J. Walker, S. Banerjee, A. Sridharan, M. Seshadri, and S. Machiraju. A city-wide vehicular infrastructure for wide-area wireless experimentation. In *WiNTECH '08*, New York, NY, USA. ACM.
[21] T. Pozar and M. Peterson. 802.11 wireless WAN's & BSD, 2003.
[22] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt, and S. Banerjee. MAR: A Commuter Router Infrastructure for the Mobile Internet. In *Mobisys '04*, pages 217–230, 2004.
[23] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav. Low-cost communication for rural internet kiosks using mechanical backhaul. In *MobiCom '06*, New York, NY, USA. ACM.
[24] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *EuroSys '07*, New York, NY, USA. ACM.
[25] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using PlanetLab for network research: myths, realities, and best practices. *SIGOPS Oper. Syst. Rev.*, 40(1).
[26] J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, Berkeley, CA, USA. USENIX Association.