

# Using Virtualization and Live Migration in a Scalable Mobile Wireless Testbed

Dongwoon Hahn Ginnah Lee  
Electrical and Computer Engineering  
University of Maryland  
College Park, MD, USA  
{dhahn, ginnah}@umd.edu

Brenton Walker Matt Beecher  
Laboratory for  
Telecommunications Sciences  
College Park, MD, USA  
{brenton, beecher}@ltsnet.net

Padma Mundur  
UMIACS  
University of Maryland  
College Park, MD, USA  
pmundur@umiacs.umd.edu

**Abstract**—Laboratory-based mobile wireless testbeds such as MeshTest and the CMU Wireless Emulator are powerful platforms that allow users to perform controlled, repeatable, mobile wireless experiments in the lab. Unfortunately such systems can only accommodate 10-20 nodes in an experiment. We have designed and built a prototype of a system that uses software virtualization and live migration to facilitate experiments involving intermittently connected networks with many multiples of the number of physical nodes available on such a testbed.

Building a system like this presents many technical and research challenges. In this paper, we will describe the physical construction and software architecture of the system while providing a discussion on the research issues that we are currently addressing.

## I. INTRODUCTION

Scalability, control, and realism are critical issues in wireless network performance studies. Simulation tools like NS2 [1] or the Opportunistic Network Environment (ONE) [2] provide users with control, scalability and repeatability, however they lack realism, both in the sense that the hardware, firmware, and operating system must be modeled in software, and because the wireless propagation and interference models tend to be very simplified. Field tests, on the other hand, achieve realism by deploying real implementations running on real hardware, often in their intended environment. However, live field tests involving mobile wireless devices are expensive, difficult to control and monitor, and may not be reproducible. As an intermediate solution many researchers have developed laboratory-based mobile wireless test systems [3], [4], [5]. Such systems use real wireless devices which operate in an emulated RF environment. These laboratory-based systems, however, are limited to experiments involving 10-20 wireless nodes. We have developed a system which uses virtualization and live migration to share access to an emulated mobile RF environment, facilitating scalable experiments on such laboratory-based mobile wireless testbeds.

Virtualization has been used as a tool in a variety of testbeds and emulators. The most popular application seems to be facilitating experiments involving more nodes than are physically available. For instance, Emulab supports virtual nodes in its experiments using FreeBSD Jail [6]. In a more light-weight example, the Network Emulation Testbed (NET) uses Virtual Routing to run wired network experiments with a very high

ratio of virtual to physical nodes [7]. Other researchers use virtualization for a similar purpose, but perform experiments over a simulated wireless channel. For example, in [8] Virtual User Mode Linux is used to experiment with several mobile wireless Delay-Tolerant Network (DTN) scenarios.

Virtualization has also been used to maximize concurrent experiments on limited hardware resources. One recent example of such a study is [9] in which researchers describe how they use virtualization to support multiple simultaneous over-the-air wireless experiments. Their approach is to use time-division-multiplexing. That is, to run the individual experiments in separate time slices without interference. Another study along these lines, regarding the Orbit testbed [10], looks at using the OS-level virtualization tool, OpenVZ to support concurrent experiments.

In our research, we use node virtualization and migration for a different and novel purpose. Our focus is performance evaluation of intermittently connected mobile wireless networks. This class of networks could span the spectrum from Mobile Ad-hoc Networks which are occasionally partitioned, to very sparse sensor networks with mobile data mules. In situations where a network is partitioned we are able to conduct a large experiment as a collection of smaller scale mobile interactions. In our testbed the limited resource is the number of inputs to the emulated RF environment. If a node in a mobile scenario is geographically isolated, it does not require access to the RF environment. Therefore, as a mobile network scenario evolves, only those nodes that are within communication range of other nodes are given access to the wireless medium. It is not practical to physically swap devices connected to the emulated RF environment, so we use software virtualization tools to move running node images on to and off of the wireless testbed nodes during an experiment. In this way, we can experiment with a network that has more nodes than could be supported on a non-virtualized system.

## II. TESTBED DESIGN

Figure 1 shows a high level schematic of the premise behind the proposed testbed. The underlying physical scenario used in network evaluation consists of arbitrary arrangements of nodes and a mobility pattern; there are nodes that are within

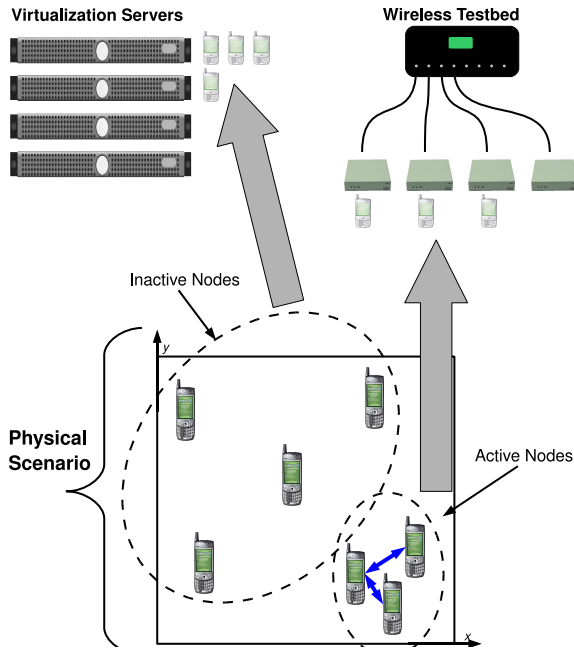


Fig. 1. A schematic for the Testbed Design.

communication range of one another and geographically isolated nodes. In our system each node involved in the physical scenario corresponds to a virtual node or *vnode*. The actual computers that the *vnode* images run on are physical nodes or *pnodes*. The wireless devices in the testbed are *pnodes* and as are several powerful rack-mount servers which have no wireless hardware. Each *vnode* has a running image on some *pnode*, and the *vnode* images are moved from one *pnode* to another as necessary to achieve the desired physical scenario.

*Vnode* images that are currently involved in wireless interactions are moved to the wireless *pnodes* in the testbed. *Vnodes* that are geographically isolated in the physical scenario are moved to the rack-mount servers. Furthermore, *vnodes* involved in separate isolated wireless interactions can be moved to *pnodes* connected to different RF switches. This means that the testbed can be expanded to support more *vnodes* by increasing the number of RF switches and *pnodes*, as long as no individual cluster of wireless nodes becomes too large. The evolution of the underlying physical scenario due to node mobility and the migration of *vnode* images are controlled by a new mobility/migration management system we developed.

The testbed architecture consists of three main components:

- 1) An RF matrix switch and a collection of wireless devices which host *vnodes* that are within communication range of other nodes.
- 2) Rack-mount servers that host isolated *vnodes*
- 3) A mobility/migration management system that moves *vnode* images between these two physical entities and manipulates attenuation settings within the RF switch

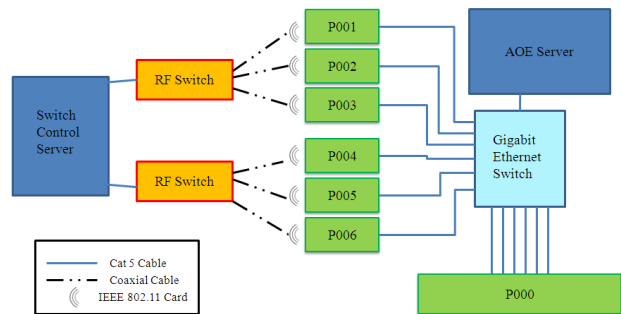


Fig. 2. Hardware Architecture.

based on the underlying physical scenario.

### III. TESTBED IMPLEMENTATION

We have built a working prototype of the testbed that covers all phases of its development lifecycle from design to implementation and testing. Using the testbed we have been able to test and evaluate several of our target network scenarios. There are however, several research oriented issues that need to be addressed. We will discuss those issues in detail in Section IV. In this section we describe hardware and software configuration used in the testbed implementation.

#### A. Hardware architecture

The high-level testbed hardware architecture is shown in Figure 2. The setup consists of two JFW 50PA-338 8-port RF matrix switches, six Dell Studio Hybrid computers, an AOE server, and a virtualization server all connected through a gigabit Ethernet switch. The AOE server and virtualization servers are Penguin Relion 1600SC machines. Each Studio Hybrid is equipped with an Atheros 802.11 a/b/g card with Mini PCI Express as host interface and a frequency band of 2.4 GHz and 5 GHz. The device driver used is MadWiFi version 0.9.4 in ad-hoc mode fixed on channel 6. The Studio Hybrids are similar to Dell Studio-series laptops, but repackaged as a small form-factor desktop. We chose them because the small form factor allows them to fit into our shielded enclosures, they support gigabit ethernet and MiniPCI express Wi-Fi cards, and they are available with a reasonably capable processor that supports Intel VT (Intel Core 2 Duo T8100). The Wi-Fi cards were chosen because of their availability and their documented compatibility with the MadWiFi drivers.

#### B. Wireless Testbed and RF Switch Control Issues

In our prototype implementation, the wireless testbed part consists of computers in shielded enclosures, two RF matrix switches and a switch control server. The RF energy from each computer's Wi-Fi card is cabled out of the enclosures and into the matrix switch of programmable attenuators. The enclosures prevent inadvertent cross-talk, and the matrix switch allows us to arbitrarily control the attenuation between the wireless devices.

The program that controls the RF switch at the heart of the Virtual MeshTest system is a daemon called *xsimd*. The *xsimd*

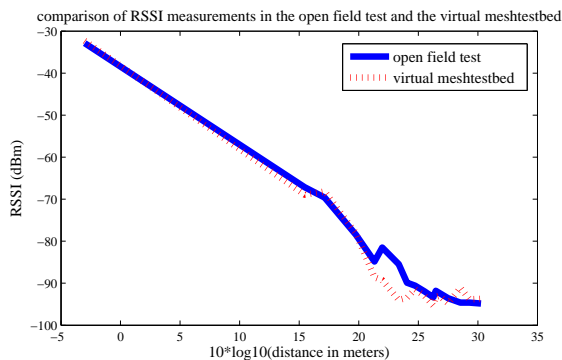


Fig. 3. Comparison of RSSI measurements in the open field test to that of the testbed.

program for RF switch control was developed for the MeshTest testbed effort at LTS cited in [3] and is used with some modifications in the current prototype. The daemon accepts physical scenarios from the mobility management system, computes the appropriate attenuator settings, and applies them to the RF matrix switch. Physical scenarios consist of an XML-encoded list of node location updates for all nodes in the experiment. By repeatedly sending physical scenarios to xsimd at regular intervals the wireless testbed subjects devices to the simulated effects of mobility.

We calibrated the RF switch using results from a field test we performed. In the field experiment we placed a pair of stationary nodes at a fixed distance and let them transmit beacons. We repeated the experiment at seventeen locations for varying distances, from 0.5 meters to 1100 meters. We used two Dell laptops with the same Atheros 802.11 cards used in the virtual testbed. All driver configuration parameters were also the same as those used in the virtual testbed, including disabling antenna diversity. At each location, we ran *tcpdump* for about 30 seconds, or until at least twenty frames were received and the RSSI values were recorded. The RSSI values were extracted by dumping the radiotap headers in the *tcpdump* output. Also, throughput and jitter measurements were made using *iperf*.

In the corresponding testbed-based experiment, RSSI measurements for 200 packets were taken at each distance used in the field test. By adjusting the insertion loss values, we could fit the path loss characteristics of the Virtual MeshTest system to those of the open field test. Figure 3 shows a comparison of the field test results with those obtained in the testbed.

### C. Node Migrations using Virtualization Software Xen

The core idea for the proposed testbed design depends on node virtualization and migration. The underlying design requirement is fast and seamless migration of vnodes between pnodes. There are three different ways of executing node migration using most virtualization software that support node migrations. This terminology seems to not be standardized, so to be clear we define:

- *Cold Migration* Complete shutdown and reboot of the target VM
- *Warm Migration* Suspend, copy, and resume execution of the target VM
- *Live Migration* keep an instance of the VM running on the source pnode while copying to the destination pnode

Both warm and live migrations exhibit minimal downtime and seamless migration whereas cold migration can result in noticeable downtime and considerable clock drift. However there are certain hardware requirements for warm and live migrations to be successful, whereas cold migration is more tolerant of hardware changes on the target host. In this paper all of our tests and experiments involve warm or live migration.

We considered two virtualization technologies for our testbed, Xen [11], [12], [13] and OpenVZ [14], both of which support live migration. Xen is a paravirtualization technology for several architectures that provides a hardware abstraction similar but not identical to the raw hardware through its Virtual Machine Monitor (VMM) called a “hypervisor”. The modified guest OSes called domains run on top of the hypervisor. In contrast, OpenVZ is an OS-level virtualization technology for Linux kernels. While some users have found it to have better performance than Xen [15], the restriction of not being able to run different guest OSes gave us a strong disincentive to adopt it for our experiments. Because of its flexibility to accommodate different OSes on a single uniform host OS, we decided to use Xen in our testbed implementation.

Integrating Xen with wireless networks is one of the more interesting challenges we came across as we constructed the Virtual MeshTest testbed. Xen provides primarily three network configurations which are bridging mode, routing mode and NAT mode. NAT mode was not considered because the nodes IP address is not visible to external networks and it is not appropriate for the experiments we were considering.

Each virtual machine can have logical network interfaces. The MAC addresses of the logical interfaces can be either manually assigned or randomly generated. In bridging mode, the MAC address of the logical interface is visible to external networks. In routing mode, the IP address of the logical interface is visible to external networks. Xen also creates virtual network interface (vif) which is a pair of connected virtual Ethernet interfaces with one end in guest OS (also called domainU) and the other in host OS (also called domain0). We used routing mode for wireless interfaces in the experiments we ran using the prototype.

### D. Mobility Management Software

The software components in our testbed which control node movement and migration are depicted in figure 4. The system takes as input a set of parameters describing a physical mobility scenario. For example, we have tested the system with mobility patterns for DataMULE and several variants of Random Direction (random), Mini-Beltway and three node drive-by (deterministic) models. The *Mobility Generator* produces an XML file encoding the desired mobile scenario, which can be fed into the *Preprocessor*. The Preprocessor

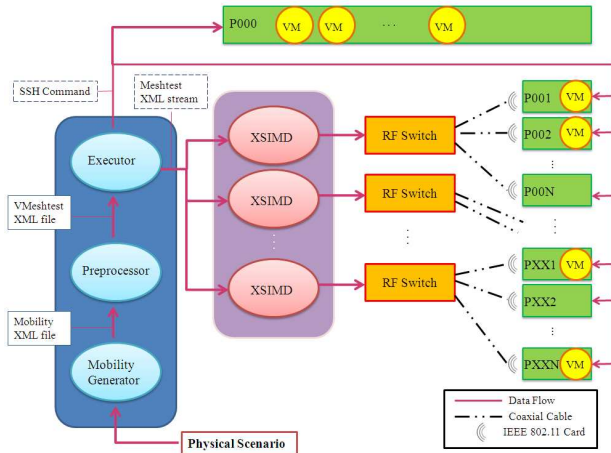


Fig. 4. Mobility Management Architecture.

determines what node migrations should take place, and when, and encodes what physical scenarios should be sent to the xsimd daemons. This last step can get somewhat convoluted because the mapping between nodes in the original scenario and switch inputs changes throughout an experiment. The experiment is started by invoking the *Executor*. The *Executor* maintains real time during the experiment, sending physical scenarios to xsimd and issuing Xen commands to the pnodes.

The migration algorithm implemented in the *Preprocessor* is a hard decision algorithm based on a unit disk model. When two nodes are within some distance  $D$ , called the *migration range*, of each other in the physical scenario, the *Preprocessor* must assign them to the same RF switch. If a node is more than distance  $D$  from any other node it is considered isolated, and is scheduled to be migrated to the offline virtualization server. For nodes that are not isolated, the *Preprocessor* uses a depth first search to identify connected clusters of nodes. The cluster composition and the associated time slots are used as the basis for migration. If a given mobility scenario has communicating clusters that requires more resources than are available (more switches, or more ports on a given switch), the *Preprocessor* will reject the scenario.

The *Executor* reads the XML file generated by the *Preprocessor* and executes the migration commands which move vnodes from either the Hybrids or the Penguin server. It also sends the current physical scenario to xsimd every second. When a vnode migrates from the penguin server to a hybrid, the corresponding attenuation settings between it and the nodes it is in range of should be updated as soon as the migration is complete. The issuing of xsimd commands is fairly straightforward, but the procedure for migrating vnodes can become somewhat complicated. We discuss this issue in more detail in Section IV.

#### IV. RESEARCH ISSUES

There are a number of issues that we will investigate going forward. One significant question is what nodes and when to migrate which leads to interesting algorithmic developments

on migration strategies. A second equally significant activity we are working on is to integrate Emulab as a front-end user interface to our testbed.

#### A. Migration Algorithms and Strategies

Our current system builds a migration schedule in advance by analyzing a mobility scenario for an entire experiment. It initiates node migrations when the distance between virtual nodes passes a hard boundary. While this type of migration strategy is adequate for some purposes, there are better strategies.

Many simulation studies assume a hard boundary or a fixed wireless communication range of as little as 250m [16]. However, we have found that commercial 802.11 hardware can communicate at over 1000m, though the throughput and jitter deteriorate as the distance increases. In the hard-decision migration algorithm, increasing the communication range increases the size of the node clusters. Since all nodes in a cluster must be migrated to the same RF switch, the hard-decision algorithm will fail for scenarios where node clusters become too large. We are experimenting with the use of a soft decision algorithms to monitor all the possible wireless interactions and perform migrations that minimize the amount of lost communication opportunity. Such lost opportunities could cause errors in performance estimates which need to be quantified.

An example of one such an algorithm we evaluated is what we call the “shortest link first”. This algorithm assigns pairs of nodes to an RF switch by prioritizing links based on distance. In our preliminary analysis, this strategy achieved migration patterns which were very close to optimal. There are also other factors to consider, such as the nodes that may already be on the switch from prior interactions and limiting node migration frequency that could be used in the design of a migration algorithm.

Besides the migration algorithm, there are two general techniques for performing migrations which we have descriptively named *sequential migration* and *parallel migration*. In sequential migration mode *all* the virtual machines are paused during each migration and resumed after the migration is complete. Therefore the time spent on migrations must be tracked and taken into account when analyzing experimental data.

Parallel migration is the more intuitive migration technique, but much more difficult to implement. In parallel migration mode, when (or possibly before) a vnode is due to begin execution on a new pnode, a live migration is initiated. Since migrations are generally quick (a few seconds for our vnodes) the new vnode will be active on the correct switch port with little delay and very little impact on the experiment. The benefit of parallel migration is that it allows a user to interact with an experiment in real-time, and allows us to include non-virtualized nodes in experiments.

In parallel migration mode we need to synchronize updates to the RF switch with migrations to make sure that the vnode is running on the destination pnode when xsimd starts to

update its location. Furthermore, because there may be many migrations in progress at the same time, parallel migration also introduces a distributed coordination problem. In general we do not want two vnodes executing on the same wireless pnode at any time. For one thing, the current RF switch settings can only be correct for one of the vnodes, and since the time a migration takes will not be known in advance, there will be contention for configuring the wireless interface. A more severe problem arises if the Executor attempts to migrate a vnode that has not yet finished its previous migration. We are addressing these and other challenges as we develop our migration algorithms and control architecture.

### B. Integration with Emulab

In this paper, we have controlled our experiments and gathered the results with a variety home-made scripts. To make our system/work relevant to a broader community of people we plan to use Emulab for experimental control. However, Emulab does not support all of the functionality we intend to provide. For example, Emulab has no notion of mobility control, and while the development version of Emulab does have support for virtualization with Xen, it does not support the notion of migrating virtual machines between physical nodes during an experiment. Because of this we have built our own mobility and migration control system. Our challenge is to integrate it with Emulab while maintaining its ability to function independent of Emulab. This approach is similar to that taken by the CMULab developers in their mobility control system. Our problem is more challenging, however, because the act of migrating virtual nodes during an experiment has consequences for Emulab's internal data structures and database.

### C. Experimentation

We conducted DataMULE experiments [17] to test the implementation of the testbed. We adopted a three-tier architecture which includes stationary access points (AP), sensors, and mobile data MULEs. We used the DTN2 static routing module with the following two forwarding rules: Sensors forward data to MULEs only and MULEs forward bundles to the AP.

Each data MULE periodically sends out discovery announcements, and all nodes listen for these announcements. When a node detects a MULE, it opens an opportunistic link. All opportunistic links remain in the routing table until dtnd is restarted, though they are marked as Unavailable once the MULE is out of range. Since we implement the Xen *routing mode* and the two virtual machines are on different subnets, the broadcast neighbor discovery announcement cannot be delivered between the MULEs and the sensors or the access points. Therefore all neighbor discovery announcements for each neighbor (sensors and access points) by the MULEs were made by the unicast announcement every second. Since unicast neighbor discovery is used, the access points and sensors cannot find each other which works well with our forwarding rule that only MULEs are allowed to forward bundles to the

AP. The results we obtained are comparable to earlier test results obtained with the MeshTest system [18]. We omit a discussion on results for space consideration.

As the project continues, one of the goals is to validate the testbed with field experiments conducted on other real wireless testbeds like the DOME project [19].

## V. CONCLUSION

In this paper we have described the design and construction of a laboratory-based mobile wireless testbed, VMT, that uses virtualization and live migration to facilitate experiments that are larger than would otherwise be possible. The system uses a new experimental control system that manages the migration and mobility of virtual nodes during an experiment. We have calibrated the system against a live field test, and have used the system to run DataMULE experiments with results comparable to a similar experiment done without virtualization. We are in the process of drastically expanding the system. In the coming months we will continue to develop more sophisticated migration algorithms and techniques, and integrate our mobility management system with Emulab.

## REFERENCES

- [1] "The Network Simulator NS-2." <http://www.isi.edu/nsnam/ns/>.
- [2] A. Kernen, J. Ott, and T. Krkkinen, "The one simulator for dtn protocol evaluation.," in *SimuTools*, p. 55, ICST, 2009.
- [3] T. C. Clancy and B. D. Walker, "Meshtest: Laboratory-based wireless testbed for large topologies," in *IEEE TridentCom 2007*, pp. 1–6, 2007.
- [4] G. Judd and P. Steenkiste, "Repeatable and realistic wireless experimentation through physical emulation," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 63–68, 2004.
- [5] "Carnegie Mellon University Wireless Emulator." <http://www.cs.cmu.edu/~emulator/>.
- [6] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Gurururasad, T. Stack, K. Webb, and J. Lapreau, "Large-scale visualization in the emulab network testbed," in *Usenix*, 2008.
- [7] S. Maier, D. Herrscher, and K. Rothermel, "Storage routing for dtn congestion control," *Computer Communications*, vol. 30, pp. 943–956, 2007.
- [8] L. R. Amondaray and J. S. Pascual, "Delay tolerant network simulation with vnuml," in *ACM CHANTS*, 2008.
- [9] G. Smith, A. Chaturvedi, A. Mishra, and S. Banerjee, "Wireless virtualization on commodity 802.11 hardware," in *WINTECH*, 2007.
- [10] G. Bhanage, I. Seskar, Y. Zhang, and D. Raychaudhuri, "Evaluation of openvz based wireless testbed virtualization," Tech. Rep. WINLAB-TR-331, Rutgers University, 2008.
- [11] "The Official Xen Project Site." <http://www.xen.org/>.
- [12] 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 *ACM SOSP*, 2003.
- [13] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Usenix NSDI*, 2005.
- [14] "OpenVZ." <http://wiki.openvz.org/>.
- [15] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin, "Performance evaluation of virtualization technologies for server consolidation," Tech. Rep. HPL-2007-59R1, HP Laboratories, 2008.
- [16] U. Lee, E. Magistretti, B. Zhou, M. Gerla, aolo Bellavista, and A. Corradi, "Mobeyes: Smart mobs for urban monitoring with vehicular sensor networks," *IEEE Wireless Communications*, 2006.
- [17] R. Shah, S. Roy, S. Jain, and W. Brunette, "Data MULEs: Modeling a Three-tier Architecture for Sparse Sensor Networks," in *IEEE SNPA*, 2003.
- [18] M. Seligman, B. D. Walker, and T. C. Clancy, "Delay-tolerant network experiments on the meshtest wireless testbed," in *CHANTS '08*, (New York, NY, USA), pp. 49–56, ACM, 2008.
- [19] "UMassDieselNet." <http://prisms.cs.umass.edu/dome/>.