

Wireless Virtualization on commodity 802.11 hardware

Suman Banerjee, Arunesh Mishra, Anmol Chaturvedi, Gregory Smith *
University of Wisconsin-Madison
{suman,arunesh,anmol,gregory}@cs.wisc.edu

Abstract

In this paper we describe specific challenges in virtualizing a wireless network and multiple strategies to address them. Among different possible wireless virtualization strategies, our current work in this domain is focussed on a Time-Division Multiplexing (TDM) approach. Hence, we present our experiences in the design and implementation of such TDM-based wireless virtualization. Our wireless virtualization system is specifically targeted for multiplexing experiments on a large-scale 802.11 wireless testbed facility.

1 Introduction

Experimentation facilities need to be an integral part of research endeavors. While initial research ideas can be evaluated through analysis, simulations, and emulations, implementation and deployment of these ideas on a realistic environment helps in systematically identifying and addressing many practical problems that systems typically encounter. As pointed out in the 2002 report [4] of an NSF workshop on network research testbeds, “There is no substitute for a real life environment to capture the complexity of multipath and fading that is inherent to wireless.” Therefore, understanding wireless environments requires extensive prototyping and experimentation, in order to uncover new insights that lead to improvements in system design.

The NSF-led GENI (Global Environment for Network Innovations) initiative in the US aims to build such an experimental facility that allows “fundamental innovations in networking and distributed systems” — innovations that can potentially lead to the design and development of the future Internet architecture. To meet this objective, GENI needs to be designed for maximum flexibility allowing for clean-slate designs and experimentation of such designs at scale to a large number of experimental researchers *simultaneously*. In a January 2006 document, the GENI working group identified four important techniques to achieve this goal — virtualization of the physical hardware, programmability that allows experimentation with clean-slate designs, controlled experimentation to allow many experiments to co-exist at their level of controlled-risk without affecting the others, and modularity to accommodate new building blocks and components over time.

The goal of our work is to demonstrate a key capability that will be needed to realize GENI’s wireless facility — virtualization of a wireless network using a large-scale 802.11 facility as an example. The objective of virtualization is to allow multiple experiments to co-exist on a wireless experimental facility in an efficient manner.

Challenges to wireless virtualization: Virtualizing a wireless network presents some unique challenges that are not observable in a wired network. The biggest challenge in this domain is to virtualize the wireless link. To establish a wireless link, a transmitter-receiver pair has to be configured to the same channel parameters, e.g., channel of operation, appropriate setting of transmit power,

*The authors are supported by NSF SGER grant CNS-0639434.

receiver sensitivity, etc. Now consider two different experiments (A and B), each with its own definitions of wireless links and communication patterns spanning the physical network. If these two experiments are to co-exist on the same hardware, communication activities from one experiment should not affect *any* reception behavior on the second experiment, and vice versa. This observation translates to two important requirements: (i) *Coherence*: When a transmitter of one experiment is active, all of the corresponding receivers and potential sources of interference as defined by the experiment should be simultaneously active on their appropriate channels of operation, and (ii) *Isolation*: When a node belonging to one experiment is receiving some signal pertinent to the experiment, no transmitter of a different experiment within the communication range of the receiver should be active in the same or a partially-overlapping channel [6]. To enforce such requirements, we need careful scheduling of transmission activities across different experiments. We next discuss different approaches to meet these goals.

1.1 Approaches to wireless virtualization

Virtualization of the wireless medium can be achieved in multiple ways. They are:

- **Space Division Multiplexing (SDM)**: This is the most simple approach, where physical resources are partitioned in space. Each experiment is assigned a set of physical nodes such that transmitting nodes from different experiments do not interfere with each other. Such an approach is possible because any wireless transmission has very little impact beyond a certain interference perimeter. The size of the region bounded by this perimeter depends on many factors such as transmission power and channel characteristics. The exact choice of transmit power at each node, thus, plays a significant role in the design of this virtualization approach.
- **Frequency Division Multiplexing (FDM)**: In this approach, different experiments are partitioned in the frequency domain for their communication needs. It can be implemented as follows. Each physical node in the facility is equipped with multiple wireless interfaces. Multiple virtual nodes — one corresponding to a single node from each experiment — is hosted in each such physical node. Different virtual nodes use distinct wireless interfaces, each configured with the frequencies allocated to the corresponding experiment. Interference between the different experiments are avoided by ensuring that different experiments are assigned non-interfering channels.
- **Code Division Multiplexing (CDM)**: This approach is analogous to FDM, except that different experiments use different orthogonal codes for their communication. The choice of codes in this approach is critical.
- **Time Division Multiplexing (TDM)**: In this case, the entire wireless network is partitioned in time across the different experiments. Each experiment is assigned a time slot during which each physical node in the system activates the virtual node corresponding to this particular experiment. This is somewhat analogous to processor sharing mechanisms that are widely adopted in current operating systems to multiplex different processes on the same hardware. However, since an experiment can potentially run on a large set of physical nodes, this approach calls for synchronized operations between them.
- **Hybrid approaches**: It is possible to envision a number of virtualization approaches that combine one or more techniques among SDM, FDM, CDM, and TDM. For example, we can combine the FDM and TDM approach to design a technique called *Frequency-hopping*, where each experiment hops through a unique sequence of frequencies (channels) over different time

slots. Our prior work has shown that such dynamic frequency allocation schemes can lead to significant throughput gains over static FDM approaches [5].

In the rest of this extended abstract, we describe the design and implementation of one of these approaches — namely Time Division Multiplexing for wireless virtualization. Our implementation effort is fairly general in nature. However, to demonstrate these capabilities, we are using the ORBIT facility (please see www.orbit-lab.org) and are working in close collaboration with our colleagues at Rutgers University on this project. Hence, the virtualization capabilities that we are developing are currently being built into ORBIT’s 802.11-based wireless grid.

2 Designing TDM-based virtualization in ORBIT

The ORBIT facility presents users with the ability for remote experimentation. Each experimentation node in ORBIT has a 1 GHz VIA C3 processor, 512 MB RAM, a 20 GB local hard disk, two wireless mini-PCI 802.11 a/b/g interfaces, and two 100BaseT Ethernet ports. In absence of virtualization, if an experimenter wants to run a controlled experiment on this facility, he needs to reserve the entire 400 node testbed for his exclusive use. To overcome this current limitation, we are working with our colleagues at Rutgers University to develop different wireless virtualization techniques for ORBIT that will provide automated and finer-grained resource sharing among multiple experimenters. Our TDM-based virtualization is one of these approaches.

Time slot granularity and implications: At the core of TDM is the need for time-synchronization. When one virtual node corresponding to one experiment has configured the wireless interface to transmit, it is necessary that the wireless interface at the receiving virtual node be configured with the same channel parameters to receive this transmission. In addition, all potential sources of interference that are part of this experiment, also needs to correspondingly transmit their interfering signals at the same time to faithfully imitate the experimenter’s objectives. Finally, no other experiment should either add or mitigate the interference effects of the first experiment.

Now consider the case where the transmitting virtual node of the experiment sends data into the kernel for transmission out of the 802.11 wireless interface. Once the kernel forwards this frame to the 802.11 interface for transmission, there is no explicit way of knowing when exactly this transmission is successfully completed. Sometimes the transmission of this frame can get significantly delayed in the wireless interface itself due to specific implementation of the interface’s transmission logic.

In addition, if two different experiments are operating using different channel parameters, e.g., different channels of operations, transmit power, etc., there can be significant delays incurred in re-setting the interface parameters when experiments are swapped. These delays can be as large as 10 ms.

A consequence of these inaccuracies is that time slots smaller than 100 ms may not be practical in today’s commodity wireless hardware. Large TDM time slots can adversely affect the performance of stateful protocols such as TCP. Consider two experiments, A and B , each with two virtual nodes A_1, A_2 and B_1, B_2 respectively. Let us assume that both of these virtual node pairs have an active TCP connection. Let us also assume that an experiment swap (from say, A to B) on the testbed happens right after the TCP sender on A_1 queued a segment for transmission to A_2 , but before the corresponding frame could be sent to the wireless interface. In this case, the segment will need to be delayed in node A_1 for an entire time slot. With respect to TCP, this will manifest as a increased delay on the path, resulting possibly in sender timeout, congestion avoidance, and significant losses in perceived TCP throughputs.

Therefore, TDM based virtualization implemented on commodity 802.11 based wireless systems would require mechanisms that completely freezes the virtual connection (in case of TCP) and all such state at the higher layers.

Proposed TDM approach: A large number of machine virtualization techniques are known today each with different properties. Examples include VMWare [3], Denali [9], Xen [1], Linux Vservers, and User Mode Linux (UML). The Vserver approach is used in PlanetLab’s current implementation [7, 2]. Since multiple virtualized entities share the same kernel in their design, it is inadequate for our desired level of isolation between different experiments. In our work, we have considered the suitability of the Xen and the UML platforms to meet our needs. We will compare all the solutions and try to deduce the most feasible one.

- *UML over Xen:* Xen 3.0 offers pure-hypervisor virtualization and allows a guest operating system to run without modification. On every node, each experiment has its own virtual machine. By constricting each machine’s memory usage (to say, 16MB), we expect to keep as many as 25 Virtual Machines (VMs) resident in memory. Importantly, beneath this threshold, we do not anticipate the need to suspend and restore VMs to disk — an expensive operation. Rather, they can be paused and resumed very quickly because they reside in main memory. Because Xen is developed with performance in mind, we expect that this operation will be faster in Xen than in UML.

However, we had issues with the wireless drivers under Xen. Because each experiment will be modifying wireless parameters, it is desirable to give each VM direct access to the PCI wireless card. A very new feature in Xen offers this functionality (dubbed *PCI passthrough*), but the wireless drivers were unacceptably unstable when running inside the guest VM and caused system-wide crashes. We believe that this instability is related to the experimental nature of PCI passthrough.

As a design choice, we have decided to pursue a scheme that allows configuration by proxy, wherein a virtualized wireless driver in the guest VM relays requests to the real driver. We focus specifically on creating a virtualized driver (instead of some separate configuration utility) in the interest of transparency: by implementing wireless extensions on a virtual device, binaries operating within the VM need not be modified to run in our system. Because kernel debugging and driver development on UML is considerably simpler, in our initial efforts we decided that UML is the best place to work on such a solution.

- *UML over VMWare:* VMWare and UML are very analogous in function. Both offer *hosted virtualization*, as described in [8]¹ Both provide partitioning, isolation, and encapsulation among multiple running machines on the same host. They both provide virtual devices, such as switches and TAPs. The way they differ is that VMWare can virtualize most Intel architecture based operating systems while UML only virtualizes Linux (on any Linux-compatible architecture).

Both UML and VMWare do not need to modify the host while Xen needs to modify the host kernel so that the so called hypervisor can let the virtualized machines have direct hardware access according to some policies.

The reason to choose UML or Xen over VMWare is because of its open source nature. Every interface exposed inside a VMWare guest is a generic Ethernet interface without wireless

¹A hosted virtualization architecture is one in which the virtualization software runs inside a host OS and only accesses hardware through the drivers provided by the host. This differs from pure-hypervisor virtualization like Xen, which runs on the bare metal and provides device drivers through a dedicated domain (dom0).

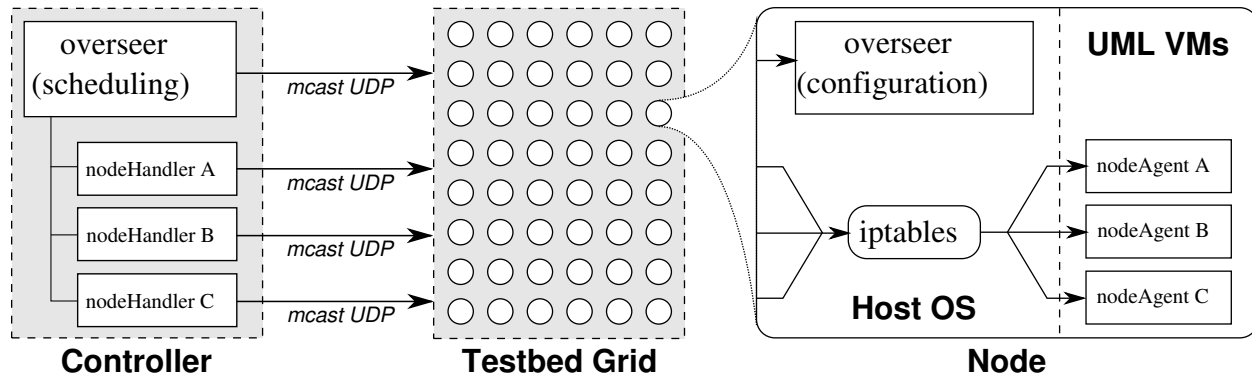


Figure 1: The overview of software architecture to enable experiment multiplexing within the context of ORBIT.

capabilities, which is similar to UML. The same is true of UML and Xen, but because these are open source they may be modified to meet our needs. Under VMWare, we would need to create some extra utility that contacts a configuration proxy running under the host OS. All binaries that need to make changes to the wireless configuration would need to be aware of this utility and its usage.

Therefore, we choose UML as our virtualization architecture, because we believe it will be the most suitable environment for driver development.

Software Design: The complete architecture can be divided into sub-groups.

- The existing ORBIT infrastructure:
 - *The NodeHandler:* The *NodeHandler*, located in a dedicated server machine, directly controls actions across of all nodes through a well-defined API.
 - *The NodeAgent:* The *NodeAgent* is a daemon running on each node that receives commands from the *NodeHandler*. Commands typically involve (but are not limited to) wireless configuration and the execution of binaries on the node. An experiment, then, is defined by a sequence of synchronized commands, issued to nodes in the grid.
 - *Gridservices:* This runs on a dedicated machine and provides services like DNS, DHCP, CMC (Node power control service), PXE boot and Frisbee (Disk imaging daemon) to the entire testbed.
 - *Applications/OTG/OTR:* As mentioned above, the *NodeAgent* has the ability to run arbitrary binaries on the node. These binaries may be custom-crafted and sent to the node or they may reside on the node by default. The canonical examples of such binaries are OTG (ORBIT Traffic Generator) and OTR (ORBIT Traffic Receiver). These are C applications which can send and receive traffic over various protocols like TCP and UDP. They are linked against OML libraries, so they can report statistics back to the database.
 - *OML:* This is the testbed measurement framework. It is essentially a library interface to a MySQL database backend, providing a simple means for binaries to record empirical data in a centralized location.

- Virtualization Infrastructure:

- *The overseer*: The overseer is a distributed service that, in topology, closely resembles the Handler/Agent framework. A single *scheduling* daemon (akin to the Handler) makes scheduling decisions and sends commands to a corresponding *configuration* daemon (akin to the Agent) on each node. A node’s configuration daemon is in charge of switching experiments on that node. Consequently, this daemon is also responsible for saving and restoring an experiment’s context, which includes the current configuration of the wireless card.
- *The host proxy server*: As stated above, threads running inside a UML kernel do not have direct access to wireless card. Therefore, the host proxy server is a simple UDP server that runs in the host OS and listens for configuration commands from UML kernels. Binaries running within UML send commands through a *virtualized wireless driver* (VWD), as described below. In summary, the flow looks like:

UML Application $\rightarrow_{\text{syscall}}$ VWD \rightarrow_{UDP} Proxy Server

- *The virtualized wireless driver*: Since the UML kernel does not have direct access to the hardware, it cannot make *ioctl*s to change wireless parameters. However, it is presented with a virtual generic ethernet interface, which offers a line of communication with the host OS. The host proxy server described above exploits this communication path, since it gives UML-level processes the ability to configure the wireless card by proxy.

The VWD extends UML’s generic ethernet interface by implementing wireless extensions. On the receipt of any *ioctl* call that falls in this category, the VWD will pass the command over a UDP socket to the proxy server in the host machine. The proxy server will then repeat the *ioctl* in the host environment, where it will be directed to the wireless device. A response will, in turn, be propagated back along the same path.

Current Status: As of today, the TDM virtualization infrastructure in ORBIT has been implemented, and we are still in the process of conducting a detailed performance evaluation study.

3 Acknowledgments

The work presented in this paper is being done in collaboration with our colleagues managing the ORBIT facility at Rutgers University. In particular, we are grateful Dipankar Raychaudhuri, Ivan Seskar, and Sanjoy Paul for multiple discussions and different forms of assistance in this effort.

References

- [1] 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 *Symposium of Operating Systems Principles*, October 2003.
- [2] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services. In *Symposium on Networked Systems Design and Implementation*, March 2004.
- [3] S. Devine, E. Bugnion, and M. Rosenblum. Virtualization system including a virtual machine monitor for a computer with a segmented architecture. US Patent, 6397242, October 1988.

- [4] Bob Aiken et. al. Report of nsf workshop on network research testbeds, November 2002.
- [5] A. Mishra, D. Agrawal, V. Shrivastava, S. Banerjee, and S. Ganguly. Distributed channel management in uncoordinated wireless environments. In *ACM Mobicom*, September 2006.
- [6] A. Mishra, V. Srivastava, S. Banerjee, and W. Arbaugh. Partially overlapped channels not considered harmful. In *ACM Sigmetrics*, June 2006.
- [7] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *ACM Workshop on Hot Topics in Networks (HotNets)*, October 2002.
- [8] J. Sugeman, G. Venkitachalam, and B.-H. Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *USENIX Annual Technical Conference*, 2002.
- [9] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and performance in the denali isolation kernel. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.