Wireless Virtualization on Commodity 802.11 Hardware

Gregory Smith, Anmol Chaturvedi, Arunesh Mishra, Suman Banerjee [*]
Dept. of Computer Sciences, University of Wisconsin
Madison, WI 53706, USA
{gregory, anmol, arunesh, suman}@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 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.

Categories and Subject Descriptors

C.2.m [**Computer-Communications Networks**]: Miscellaneous

General Terms

Design, Experimentation, Measurement, Performance

Keywords

Wireless Networks, Virtualization, Time Division Multiplexing

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 goal of our work is to design and implement a system that virtualizes a wireless network using a large-scale 802.11 testbed. The objective of virtualization is to allow multiple experiments to co-exist on a wireless experimental facility in an efficient manner.

1.1 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 configured to the same channel parameters, e.g., channel of operation, appropriate setting of transmit power, 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 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.

1.2 Approaches to wireless virtualization

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

- **Space Division Multiplexing (SDM):** This is the simplest 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 trans-

[*]The authors were supported in part by NSF awards CNS-0639434, CNS-0627589, CNS-0627102, and CNS-0520152.

mission 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 paper, 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 building on top of a 802.11-based wireless grid. In section 2, we describe the basic software design and introduce some of the challenges involved in the virtualization effort. Then, in section 3, we present the results of our preliminary integrity and performance benchmarks. Finally, we outline several considerations for future work in 4.

2. DESIGNING TDM-BASED VIRTUALIZATION

Each experimentation node on the wireless grid has a 1 GHz VIA C3 processor, 512 MB RAM, a 20 GB local hard disk, two Atheros mini-PCI 802.11 a/b/g interfaces, and two 100BaseT Ethernet interfaces.

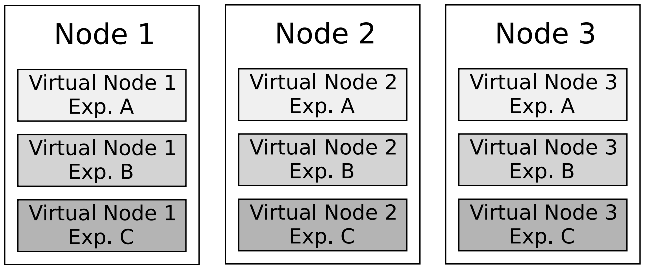

Figure 1: ***Each physical nodes hosts one virtual node per running experiment. At any given time, exactly one experiment is actually active on the grid; the rest sit idle and wait for their next time slice.***

The basic schedulable unit in our TDM virtualization scheme is the experiment. An experiment can be mapped to a set of virtual nodes (a virtual grid). A physical node in the grid hosts a set of virtual nodes, one per running experiment, illustrated in Figure 1. As described above, at most one experiment is active at a time so there is no media contention.

2.1 TDM Context Switches

At the core of TDM is the need for time-synchronization. When the scheduler preempts a running experiment, it is crucial that the physical nodes can act in perfect synchrony. If there are even small delays, packets may be lost: a sender may run too long, or a receiver may be paused too early. Even worse, these stray packets could cause interference for the experiment running in the next time slice. This is particularly catastrophic for reliable protocols, which would be forced to retransmit at the edge of every time slice.

Because of propagation delay, simply stopping all the virtual nodes at the same time is not sufficient. Consider the transmission of a packet from the virtual node, destined for the wireless network. Once the kernel forwards this frame to the 802.11 interface, there is no way to know exactly when the transmission will be 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. It is entirely possible that the scheduler will stop the current experiment while this frame is still trapped on the interface. Because of this, there must be some way to allow for packets like these to leave the interface and reach their destination. The most straightforward way to facilitate this in our system is to insert a short delay, during which the virtual node is paused, but the wireless configuration is unchanged.

Once the delay is over, the physical nodes must prepare themselves for the next experiment. Each virtual node has its last-known wireless configuration as part of its context. When a virtual node is paused, its wireless configuration is check-pointed, and the same configuration is restored prior to resuming the virtual node. Saving the device's configuration is rapid and takes a predictable amount of time, but restoring a configuration is slow and takes a variable amount of time, depending on the which parameters change. During any given context switch, the time spent in reconfiguration could be different for all the nodes on the grid. Therefore, we reserve a slot of time for reconfiguration that will easily accommodate worst possible time needed to reconfigure (about 25 ms).

To summarize, when a context switch is triggered, the virtual nodes corresponding to the current experiment are paused. There is a short delay, to accommodate packets that are *en route*. After this delay, the wireless configuration is saved as part of the context for the recently-paused virtual node. The next virtual node's configuration is restored to the card, and once all physical nodes are guaranteed to be ready, the virtual node is started and the context switch is complete. Because of the overhead involved in a context switch, time slices smaller than 200 ms may not be practical in today's commodity wireless hardware.

2.2 Virtualization Platform

There are many virtualization techniques available, so we must consider which is most appropriate. Our primary goals are:

- Experiments should be maximally *isolated* from one another, so that they do not contend for resources or namespaces.
- The experimenter should be able to use customized kernel or user-level code, to provide flexibility for application and driver development.
- A simple, precise abstraction for accounting.
- As mentioned above, the experiment should be given a consistent view of time: one that does not jump forward at the beginning of each time slice.

These objectives clearly rule out process-level virtualization and motivate the need for a full virtualization platform, such as VMware [3], Xen [1], Linux Vservers, or User Mode Linux (UML).

The Vserver approach is used in PlanetLab's current implementation [7, 2]. However, virtual machines on this platform share the same kernel, violating our goal of isolation.

In this section, we examine the suitability of Xen, UML, and VMware in light of our goals, and illustrate why we chose UML in our implementation. From this point forward, we use the terms "virtual node" (as introduced above) and "virtual machine" interchangeably.

Xen is a *hypervisor* platform, while VMware and UML are *hosted virtualization* platforms (as described in [8][1]). Because it sits right next to the hardware, Xen is capable of passing control of a PCI device exclusively to a single virtual machine. Using this functionality, dubbed *PCI passthrough*, we could push wireless driver code into the virtual machines, so that experimenters can use whatever wireless drivers they want.

However, our wireless drivers (MadWifi) were unacceptably unstable when running inside a guest virtual machine, causing system-wide crashes that were difficult to diagnose. Another difficulty involved the transfer of control, since we needed a way to "hand off" the wireless device from virtual machine to virtual machine. PCI passthrough is still in development and the documentation does not adequately cover the usage we have in mind.

[1]A 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).

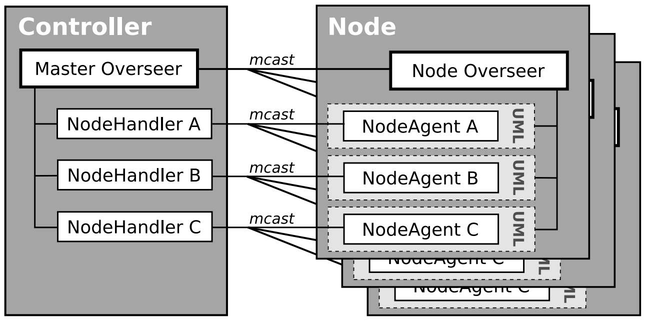

Figure 2: ***An overview of the software infrastructure used for TDM multiplexing, built on top of the existing experiment infrastructure.***

Nonetheless, the virtual node must have a way of accessing and configuring the wireless device. Without PCI passthrough, the next best solution is to tunnel requests from inside the virtual node through to a driver running in the host OS. Since we are only interested in wireless parameters, we have decided to modify the guest kernel's ethernet interface driver to support the wireless extensions. These extra handlers forward ioctl requests to the appropriate wireless interface in the host kernel.

Because these sorts of modifications are rather unorthodox, we prefer to use an open-source virtualization platform that can be modified as necessary to meet our needs. For this reason, we decided that VMware is most likely not the appropriate platform for our projected development course. To break the tie between Xen and UML, we decided that we are simply more comfortable modifying and debugging UML.

2.3 Software Design

As mentioned previously, our implementation is intended to be generic, but we see large-scale 802.11 testbeds as an ideal target platform. Currently, we are porting the system to the ORBIT testbed at Rutgers University, and our system utilizes ORBIT's experimentation infrastructure, which we briefly outline here.

ORBIT is outfitted with software to provide experimenters with a simple interface for running experiments on the grid. This software consists of two primary components, arranged in a one-to-many master/client configuration. The master is called the *node handler*, and the clients are called *node agents*.

The node handler is run as a standalone application. It defines a scripting language for configuring, controlling, and monitoring node behavior. When an experimenter feeds a script through the node handler, the handler translates the script into commands and multicasts them to all the node agents on the grid. Data is collected in a MySQL database for later analysis. This entire process is an *experiment*, which is the basic schedulable unit in the virtualized system.

2.3.1 The Overseers

The overseers are the most important part of the virtualization infrastructure. Since our method of virtualization is TDM, the overseers' primary task is to run experiments in a round-robin fashion. As with the handler and agent, we adopt a master/client model. We refer to the master as the *master overseer* and the client as the *node overseer.* We chose a master/client model over a distributed model, because we can centralize policy in the master overseer. This centralized policy is easier to reason about and is less complex in implementation than a distributed policy.

The master overseer embodies the virtualization policy. It schedules experiments to run and monitors both physical and virtual nodes. The node overseer is purely mechanical: it receives and executes orders from the master. It is in charge of configuring the wireless interface and controlling virtual nodes. Most commonly, the node overseer must perform context switches, as described in Section 2.1.

2.3.2 Synchronization

As presented in Section 2, the most critical aspect of TDM virtualization is synchronization. Therefore, the overseers are designed specifically with this goal in mind. By keeping the nodes' clocks in synch with one another and improving kernel scheduling granularity, we can use *preemptive scheduling* to synchronize the execution of commands across the grid.

In our implementation, the master overseer tags each command with a deadline before multicasting it to the grid. Node overseers will queue the command in a list, ordered by deadline. Meanwhile, a worker thread will sit at the head of the queue and sleep until it is time to execute the next command. This algorithm relies heavily on the ability to sleep for accurate periods and the synchronization of the nodes' clocks.

The node kernel has been patched for high-resolution timers (commonly referred to as hrtimers). This improves scheduling granularity by at least three orders of magnitude. Without the patch, a user-level process trying to sleep for a certain period of time would wake up within 4 ms of the target duration. With the patch, the error is reduced to 40 μs[2].

In order to keep the nodes' clocks in synch, we have configured NTP servers to service the grid and we run NTP daemons on each of the nodes, configured with the smallest polling interval allowed (16 seconds). Unfortunately, the quality of commodity hardware clocks is rather poor. The NTP clients on the node typically only synchronize the clocks within about one millisecond of the source, so the margin of error is roughly 2 milliseconds. This is enough to potentially cause problems, and will most likely receive attention in future work.

2.3.3 Network Configuration

To connect the virtual nodes to the wireless interface, we use a layer 2 software bridge. As implied, all virtual nodes will be effectively present on the same link. This means that experiments will need to operate in reserved address spaces in order to avoid crosstalk between experiments.

2.3.4 Wireless Configuration

Virtual machines in UML have a generic network driver that is used for all network interfaces. In order to grant virtual nodes the ability to configure and read statistics from the wireless device, we implemented the wireless extensions in the virtual network driver. As implemented, these extensions simply forward ioctl requests to the real network driver in the host kernel. Thus, the wireless extensions are a transparent tunnel from the virtual machine to the wireless card driver. We justify this approach for TDM, simply because we can guarantee that only one virtual machine will be using the wireless card at a time.

As described above, it is necessary to save the wireless device's configuration at the end of every time slice and store the parameters as context for the running experiment. By restoring these parameters later, the virtual node will have a coherent view of the wireless configuration. The node overseer is in charge of this operation.

A significant drawback in our implementation, is that the node overseer does not deal with the card's state in full generality. That is to say, it only saves and restores basic parameters such as essid, channel, and mode. From the virtual node's perspective, the wireless interface's buffers, registers, and bookkeeping information will be put into an indeterminate state at the beginning of each time slice. The integrity analysis presented in Section 3.2 will showcase some of the more visible effects of this shortcoming.

3. INTEGRITY AND PERFORMANCE

In evaluating our system, we are interested in examining the coherence and scalability of our design. Our primary concern is that the virtualized environment is a suitable emulation of the real thing, so that applications running in experiments see minimal side effects resulting from the TDM virtualization. As a secondary goal, we would like to ensure that our system can scale to accommodate many concurrent experiments utilizing many physical nodes.

In this paper, we primarily concentrate on the integrity of the system, because we see this as the most daunting challenge in implementing TDM virtualization. However, we do briefly address the scalability of the system in section 3.4.

3.1 Experimental Setup

For the results in this section, we had an isolated off-grid node (the master) issuing commands to grid nodes. In order to measure network integrity, we are primarily interested in characterizing data flows through the virtualized wireless medium. We feel that a single data flow between a source and sink node will provide us with good baseline measurements. Thus, the results in this section involve only two grid nodes.

For measuring the integrity of the medium, we run two experiments concurrently. One experiment involves a source-to-sink data flow from one node to the other. The data flow is generated by iperf: a common Unix utility for measuring network throughput by saturating the medium. During the transfer, we run tcpdump in both the host and guest OSes, and use these dumps to characterize the flow of data. All of the results obtained in section 3.2 are the result of these dumps. For consistency, all data flow tests were performed using 802.11a.

The second experiment is a "null experiment": it simply introduces another schedulable unit into the system, so that we can witness the effects of TDM virtualization on the data

[2]These numbers were measured while the node's processor was mostly idle. Under load, both numbers increase, but the factor of improvement offered by hrtimers is roughly the same. We have yet to see a situation in which the hrtimers are off by more then a hundred microseconds.

flow created in the first experiment. To ensure that the system is functioning properly, this experiment configures the wireless interface differently than the first experiment. Thus, if any configuration overruns occur, they will factor into our integrity results. However, these occurred very rarely, and hence they did not have a noticeable effect in our tests.

3.2 Network Streams

Because our project is concerned with virtualizing the wireless medium, we are most interested in how accurately the virtualized network emulates the real-world network. We present our findings in this section.

3.2.1 UDP

The UDP tests were initially quite surprising: in the virtualized environment, there are fewer packets lost, and throughput is much greater than in the real world. The magnitude of the effect is directly related to the frequency of context switches (or inversely related to the length of the time slice).

The reason that UDP streams perform better inside the virtualized environment entirely has to do with the congestion of the medium. Keep in mind that the iperf utility saturates the medium with packets. This means that 802.11 frames will queue up on the sender's wireless interface waiting for CTS. So, at the end of a timeslice for an experiment running iperf, there are conceivably several packets still waiting for transmission. More importantly, these are lost (not sent) when the wireless configuration is changed, and when the experiment starts its next time slice, the medium is completely clear.

Therefore, when the time slice is relatively long, iperf has enough time during the time slice to saturate the medium to the point where packets begin getting lost. The frequency of the context switches dictates how often the medium is wiped clear of congestion, from the virtual node's point of view. We refer to this phenomenon as the "clear medium" problem.

3.2.2 TCP

For the same reason that UDP experiences improved throughput in the virtualization environment, TCP suffers, as seen in Figures 3 and 4. Again, frames queue up on the 802.11 interface and buffers are cleared at the end of each time slice. Since TCP is a reliable protocol, these lost packets must be detected and retransmitted, resulting in under-usage of the medium.

The wireless medium is cleared at the start of the experiment's next time slice, so all of the data points in Figure 3 show that TCP has increased throughput in the virtualized environment. However, as the trend shows, the throughput decreases as the frequency of context switches increases. This effect is opposite of that observed with UDP streams, because TCP must retransmit lost packets.

Figure 4 shows the percentage of packets received by the sink that were out of order. We define the *post-slice pause* as the delay at the end of each time slice. Recall that this delay is intended to catch packets that are queued on 802.11 interfaces or in the air. From observation, all out-of-order packets are retransmission of packets that were lost during context switches.

Increasing the duration of the post-slice pause increases the probability that a late packet will reach its destination during a context switch. However, as the plot shows, this

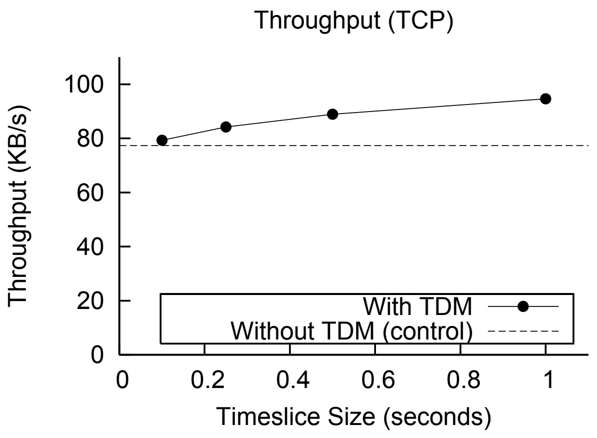

Figure 3: ***Throughput for iperf TCP tests.***

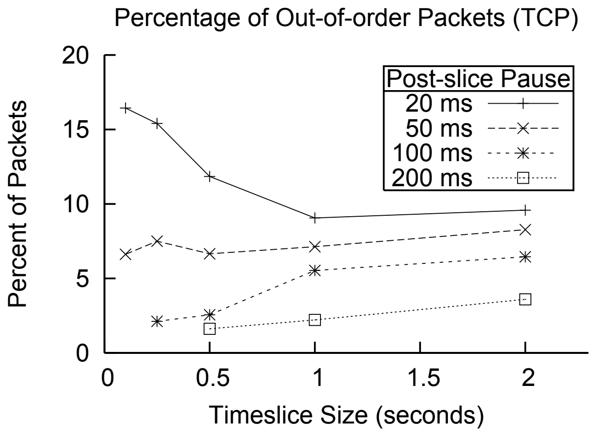

Figure 4: ***Packet of packets that arrived out of order during iperf TCP tests.***

is clearly not enough; even if we wait 200 ms at the end of a timeslice, roughly 1.5% of packets arrive out of order. Increasing the length of the post-slice pause to 500 ms did not exhibit a noticeable improvement.

Another curiosity is the effect that the timeslice duration has on the percentage of out-of-order packets (as shown by the lines in the plot). The data gathered for this plot was very noisy, varying by as much as 3-4%. We hypothesize that this effect stems from inaccurate node synchronization and the "clear medium" problem, outlined in section 3.2.1.

3.3 Time slice size

Figure 5 presents the calculated (theoretical) overhead due to context switching. Each line represents a particular duration for the context switch. Exactly 25 ms are allocated for wireless configuration, and the rest is dedicated to the post-slice pause. Clearly, the amount of overhead gets out of hand rather rapidly. In fact, if we increase the post-slice pause sufficiently to eliminate out-of-order packets, sub-second time slices are no longer feasible, due to the expense of context switching.

In Section 4, we will discuss a number of approaches by which we can alleviate this cost. Such approaches will re-

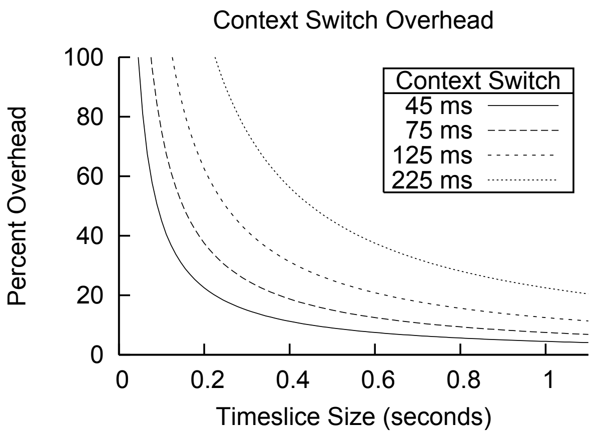

Figure 5: ***This plot shows the percentage of time that is lost to overhead, as a function of time slice size and several different context switch durations.***

duce the time overheads of mitigating out-of-order packets. This will lead to an overall improvement in efficiency of the system.

3.4 Scalability

One of the appealing results of our research is that the implementation scales nicely with respect to the number of experiments running. All of the data obtained in the previous sections were verified for two, three, and four concurrent experiments, and the results were always identical. However, this is not to say that the number of experiments can scale indefinitely. The primary limiting factor is the computing power of the physical nodes themselves. Only equipped with 512 MB RAM and 1 GHz processors, they are simply not fit to accommodate large numbers of experiments running concurrently.

We've tested up to five concurrent virtual machines without seeing any ill side-effects, with the exception that the nodes are under extremely heavy load while the VMs are booting. Once the boot is complete, the strain on the physical nodes is significantly less. The node overseer's ability to stop accurately did not diminish under this many VMs once they are booted. In cases where many concurrent experiments are desirable, the VMs need not be booted out of band. In such a configuration, the node overseer should retain its ability to execute commands at precise times.

For this paper, we have not conducted a thorough test of the system's ability to scale in terms of testbed size. This will be a subject of future research. We anticipate that node clock synchronization will figure prominently in the extent to which the system suffers under a large number of nodes. Another major factor will be system integrity: the effects characterized in the previous sections will obviously extend to situations where multiple flows are in place. For multihop routing, we expect integrity to degrade as a function of the number of nodes involved in the route.

4. FUTURE WORK

There are several items slated for future work that will improve the performance and coherence of the system. Once these modifications are implemented and tested, we will have a better understanding of which types of experiments are well-suited to running in a virtualized environment.

4.1 TDM on a Large-scale Testbed

We are in the process of porting the virtualization infrastructure to ORBIT: a 400-node testbed at Rutgers University. Once integrated, this environment will allow us to more thoroughly test our scalability hypotheses. ORBIT is an ideal target application for our system. The testbed is in very high demand, and an experimenter allocating time on the grid must register the entire grid at once. Obviously, this leads to an under-utilization of resources. In addition to maximizing grid utilization, a strong virtualization infrastructure will also provide a mechanism for accounting and prioritization.

4.2 Dual Interface

Because wireless configurations take an unpredictable amount of time, we have no choice but to reserve a full 25 ms during every context switch for wireless device configuration, when in the common case, configuration should take no more than 4 to 5 milliseconds. Since the physical nodes are outfitted with two wireless interfaces, we propose the following solution to this problem. Each experiment is allowed to use only one interface. When a virtual node starts its timeslice, it is bridged to one of the two interfaces, while the other interface is configured for the upcoming timeslice. Then, during a context switch, the node overseer bridges the next virtual node to the pre-configured wireless interface, so that there is no real-time overhead for wireless configuration.

4.3 Clock Synchronization Improvements

Because commodity computer hardware is prone to inaccurate timekeeping, NTP alone is not a suitable solution for sub-millisecond clock synchronization in a grid environment. Through conversations with systems administrators who deal with such issues, we have learned that a more appropriate solution might involve using a highly reliable time source (such as GPS) that generates PPS[3] signals, which are broadcast to all nodes on the testbed. NTP running on the nodes can use these 1 Hz signals to synchronize their clocks far more accurately than they can using standard stratum 2 NTP servers. Making an improvement of this magnitude would mean that fewer packets are lost during context switches and that virtual nodes clocks' remain more closely synchronized with other virtual nodes from the same experiment.

4.4 Wireless Configuration Management

A fatal issue, addressed above, is that we simply do not have the required power to snapshot *all* of the state from the wireless device. As shown particularly clearly in Section 3, the effects are dramatic. Ideally, we want the ability to save and restore the actual buffers and registers on the device. Such a powerful mechanism would improve the coherence of the virtualized environment and grant the experimenter the ability to poll the device for accurate statistics. Currently, stateful and time-sensitive statistics (such as RTT) will inevitably be muddied by the TDM approach.

[3]Pulse per second.

5. CONCLUSIONS

The ability to cleanly virtualize a wireless network has important practical implications for experimentation facilities. In addition to increasing grid utilization, it provides an accounting abstraction of the correct granularity.

However, as this paper shows, TDM virtualization of a wireless network is a difficult task. We identified two major challenges in our implementation: node synchronization and device state. As a group, nodes must have a coherent and accurate view of the clock; individually, they must be able to perform tasks at specific times. To meet these goals, we used NTP and high resolution kernel timers, respectively. However, NTP without a high-accuracy local clock source cannot provide the accuracy we need. Secondly, we noted that we have only limited access to the state stored on the wireless device.

It is because of these limitations that our current implementation struggles with integrity tests. These two factors are the culprits that induce the "clear medium" problem (section 3.2.1) and TCP retransmission. As seen in section 3.2.2, small adjustments are not sufficient in eradicating these symptoms. To get small gains, we must make large sacrifices. On the other hand, if we were able to snapshot the state of the wireless device and keep node clocks in synchrony, these unwanted side effects would be significantly reduced. We expect that further research in these areas will be fruitful.

6. 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 Operatiing 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.