4.2 Project Title: PlanetLab Integration (Senior Personnel: Dr. George Hadjichristofi, Mr. Ivan Seskar, Prof. Marco Gruteser, Prof. Dipankar Raychaudhuri)

While PlanetLab serves as the baseline model for programming and virtualization in wired GENI [1-3], the model needs to be significantly extended to accommodate: device heterogeneity (e.g. wireless access points, ad hoc radios, sensors), a broader range of experiment types (e.g. short-term network performance experiments running on selected network nodes vs. long-term slices used in PlanetLab), and alternative end-user support requirements (e.g. experienced programmers needing little if any experimental support vs. protocol analysts who might prefer tools for higher-level programming and execution management). Integrating PlanetLab with a large-scale wireless testbed like ORBIT aims to yield important design insights on the issues of device heterogeneity and necessary extensions to control and management protocols for effective support of wireless networks as an integral part of the experimental system.

Progress: We provide proof of concept for the integration of a wireless and wired testbeds by deriving two baseline models of integration that allow:(1) Scheduled access to wireless ORBIT nodes from slivers used by PlanetLab experimenters, and(2) Long-running "ORBIT slice" in PlanetLab nodes that can be accessed by ORBIT experimenters. Based on these models, we developed working prototypes of an integrated system.

For the first model, the integration was facilitated through the use of a PlanetLab-Orbit proxy node as indicated in Figure 30. The proxy enables PlanetLab users to include ORBIT nodes in their experiments. PlanetLab users can log into their slivers and start their experiments. Traffic generated from the PlanetLab slivers and directed to the PlanetLab-Orbit proxy is forwarded to the ORBIT nodes corresponding to those slivers. This set up was achieved through the use of GRE tunnels from PlanetLab nodes to the PlanetLab-Orbit proxy and from the proxy node to the ORBIT grid. As shown in Figure 30 (bottom), traffic from 3 PlanetLab slivers representing 3 separate experiments is redirected by the proxy to 3 different groups of ORBIT nodes (assuming SDMA virtualization is used).
For the second model, we extended the ORBIT control framework to include PlanetLab nodes. Typically, during an ORBIT experiment the NodeHandler, which is the central experiment controller in ORBIT, would be used to deploy an experiment (see Figure 31). The ORBIT users would provide an experimental script to the NodeHandler that defined the experiment. The NodeHandler would then parse the script and communicate with the NodeAgents running on each active node on the grid. Applications would then be loaded and executed by the NodeAgents on the nodes based on the instructions from the NodeHandler. To enable ORBIT users to include PlanetLab nodes in their experiments, we have added the same functionality to PlanetLab nodes.

The ORBIT NodeAgent software was modified and installed on PlanetLab slices. The NodeHandler was also modified to support the new NodeAgent software running on the PlanetLab nodes. The mode of communication between the NodeHandler and NodeAgents was changed from multicast on ORBIT nodes to unicast TCP connections based on the fact that Internet does not guarantee delivery with multicast traffic. The NodeHandler now provides a single programming interface with extended capability to start applications on the PlanetLab nodes. Thus, we avoid the previous method of setting PlanetLab-Orbit experiments by manually connecting with “ssh” into the PlanetLab nodes to configure and start the applications. This integrated control framework provides abstraction for the experimenters and facilitates reproducible experiments.

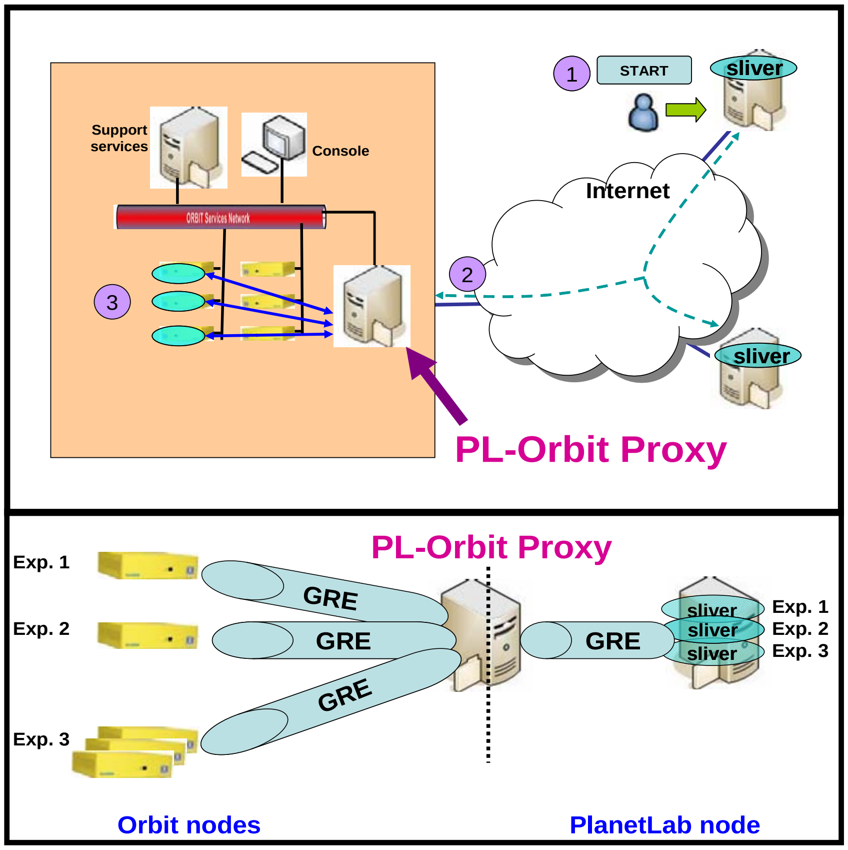


Figure 30. PlanetLab to ORBIT Integration.

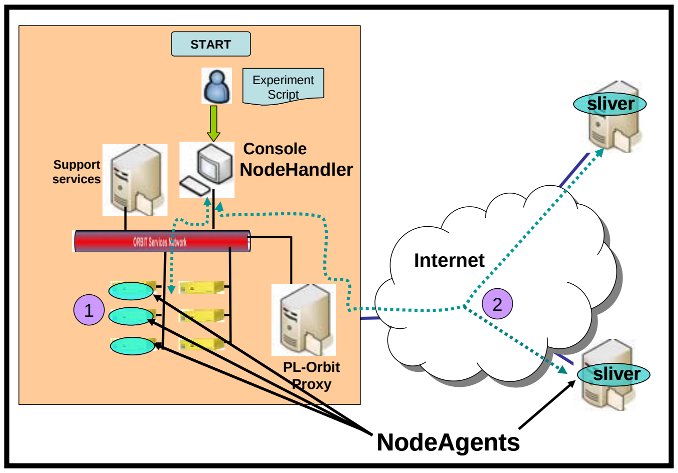


Figure 31. ORBIT to PlanetLab Integration.

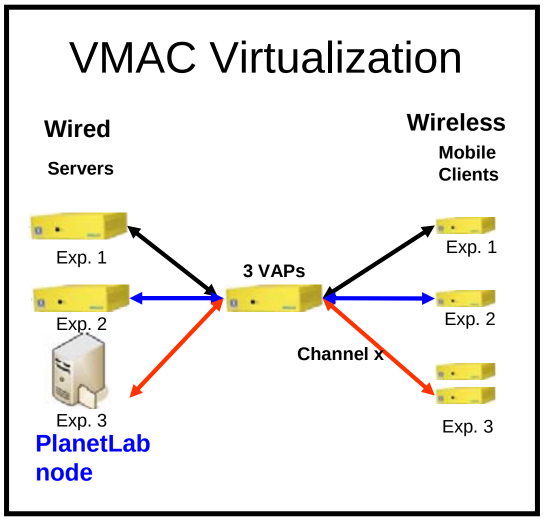
A snippet of the experimental script is shown in Figure 32. This script is parsed by the NodeHandler to configure the PlanetLab nodes. The modules in italics have been developed to extend the ORBIT control framework to include PlanetLab Nodes. The purpose of each module is as follows:
defPNodes: It defines the PlanetLab nodes that are a part of the experiment.
defPApplication: It defines the experiment applications to be started on the PlanetLab nodes. The NodeHandler also reports success or error after execution of these applications on the PlanetLab nodes.

WhenPLReady: It waits for the NodeAgents on the PL nodes to report to the NodeHandler.
PLexpdone: This facilitates a clean slice after the experiment is done.

```
#DEFINE THE NODES
defPNodes('planet.cc.gt.atl.ga.us','planetlab01.cs.washington.edu')

# START THE APPLS. ON THE PL NODES
WhenPLReady(){
defPApplication('bash /home/orbit_pkamat/PLDEMO1'){}
wait 195
defPApplication('bash /home/orbit_pkamat/PLDEMO2'){}
PLexpdone() }
```

Figure 32. Script Snippet.



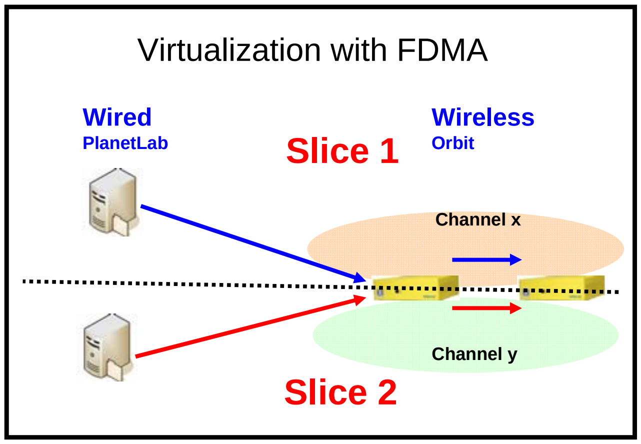


Figure 33. Integration and virtualization of wired-wireless networks

As part of proof-of-concept of integrated and virtualized wired-wireless experiments we have demonstrated the FDMA and VMAC methods of virtualization and PlanetLab-Orbit integration at a series of events and workshops held at WINLAB. Figures 33 shows some integrated scenarios that were demonstrated. The important aspect of these configurations is that we have extended the notion of virtualized wired slices to wireless networks and provided a wired-wireless network slice that can also support both short-term and long-term running experiments while better utilizing wireless resources.

Conclusions and Future Work: This project has enabled proof-of-concept integration between wired and wireless testbed. The ORBIT to PlanetLab integration has provided ORBIT users with a single programming interface and experimental methodology that enables the dynamic inclusion of wired nodes during the execution of experiments. On the other hand, the PlanetLab to ORBIT integration has enabled PlanetLab users to include ORBIT wireless nodes in their experiments. It should be noted that from the perspective of PlanetLab users the capability to dynamically deploy and set up experiments is missing, since PlanetLab has no control framework that will provide such services. Open research issues include the dynamic set up of GRE tunnels to connect PlanetLab slivers with wireless resources as well as the allocation of such resources.

A paper describing the integration approach and initial results is currently in preparation.

References for Sec 3.2:

[1] L. Peterson, GENI: Global Environment for Network Investigations, *ACM SIGCOMM '05*, August 2005.
[2] GENI Design Principles, *http://www.geni.net/design_principles.pdf*
[3] NSF Global Environment for Networking Initiatives, *http://www.geni.net/*

4.3 VINI-ORBIT Integration (Senior Personnel: Dr. George Hadjichristofi, Mr. Ivan Seskar, Prof. Marco Gruteser, Prof. Dipankar Raychaudhuri)

Project Background/Rationale:
VINI is a virtual network infrastructure within PlanetLab that allows network researchers to evaluate their protocols in a realistic environment, while providing a high degree of control over network conditions [1][2]. VINI leverages a number of technologies which have been integrated together, such as UML, XORP [3], Click [4][5], and OpenVPN [6]. Figure 34 shows the basic components that are utilized in a VINI virtual node. UML runs as a user-space process and creates a virtual environment complete with network devices. The UML instance is started from within the experimenter's slice on PlanetLab. Each virtual environment communicates with other VINI environments by creating an overlay topology with UML interfaces as the end points. XORP, running within UML, is an open source routing protocol suite that implements a number of routing protocols, such as BGP and OSPF. It manipulates routes in the data plane through the Click modular software router. XORP generally assumes that each link to a neighboring router is associated with a physical interface. For example, if the MIT sliver in Figure 34 is to be connected to Alaska, packets sent out via eth2 of the UML would be sent to Alaska.

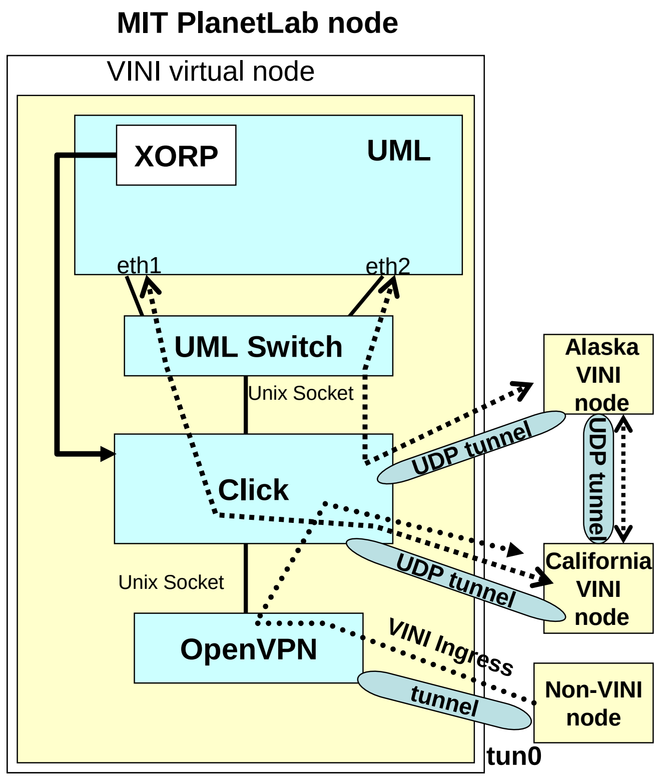


Figure 34. VINI software architecture.

In reality, all packets going in and out of UML are sent first through the UML switch and Click modules at the lower layers. The UML switch is a virtual switch used to connect UML and Click through Unix sockets. Click inspects, modifies, and route all types of packets. OpenVPN was introduced into the VINI software architecture with the main scope of connecting an outside machine to a VINI experiment as an edge node allowing for IP traffic injection into VINI.

The objective of this project is the integration of VINI and ORBIT while satisfying the following goals:

• No packet type restrictions: Any type of Ethernet encapsulated packet should be able to propagate between the two networks. These packet types include both IP, non-IP packets, as well as broadcast traffic.

• Arbitrary topology creation: The solution should provide researchers with the capability to connect any wireless node to any wired node in different combinations and carry Layer 3 and above experiments. Figure 35 shows a sample network configuration where wireless nodes (e.g., ORBIT nodes) are connected to wired (e.g., VINI) nodes. One or more wireless nodes are connected to multiple wired nodes in the integrated overlay network.

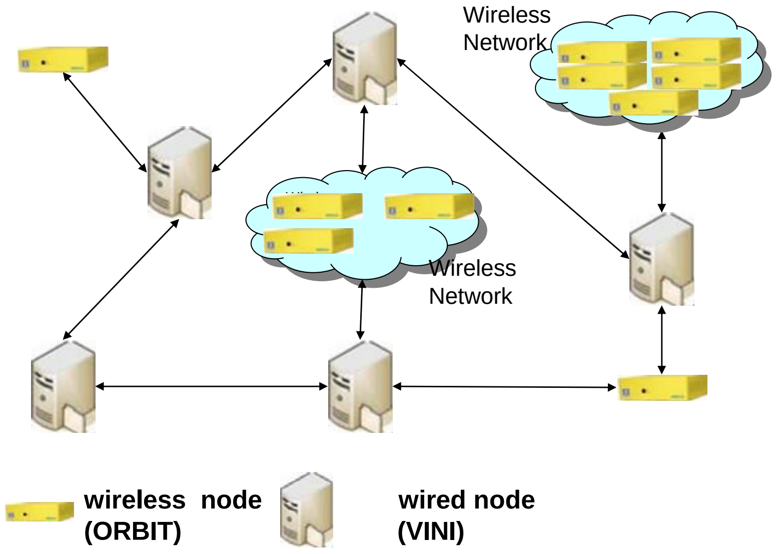


Figure 35. Sample configuration of the integrated testbed.

Progress: A series of modifications were made on the existing VINI architecture to accommodate this integration. At the higher layer, UML was configured with an additional interface to accommodate traffic to the ORBIT testbed. The interface was linked with a TAP interface, which was associated with OpenVPN. The OpenVPN module was then configured to carry Ethernet tunnels (i.e., Ethernet packets encapsulated in IP) instead of IP tunnels. Click was also modified to link UML traffic to the OpenVPN application by simply passing packets without processing them for forwarding purposes. This configuration allowed the delivery of Ethernet packets directly to the UML enabling Layer 3 experimentation.

While VINI provides a powerful platform to create controlled network topologies, it's automatically generated underlying configuration files require intimate knowledge of VINI's inner workings. As described above, VINI links together the Click modular software router, UML, UML Switch, XORP BGP/OSPF routing software and OpenVPN using a system of Unix sockets, UDP sockets and Linux Tap/Tun interfaces. By carrying Ethernet traffic to the user space we allow users with basic knowledge in Linux and Linux networking to use the integrated testbed without knowing the details of the lower layers of virtualization on the nodes (e.g., Click) or having to modify the underlying system. Therefore, transparency aids the user to expedite the deployment of an integrated network layer experiment.

On the ORBIT side, there was more flexibility since ORBIT nodes do not have a pre-defined software architecture as compared to VINI nodes. An ORBIT baseline image was used as a foundation (Debian GNU with Linux kernel 2.6.12) and OpenVPN was compiled and installed along with other supporting packages. The Linux kernel was recompiled with the Tunneling and Bridging options to enable the creation of TAP interfaces for OpenVPN Ethernet tunnels. ORBIT nodes were configured based on a Router or Bridge configuration, which provided different topology characteristics. In the Router configuration, an ORBIT node is set up as a router that can handle any routing protocol used by XORP. This set up can be visualized as adding nodes and extending the existing VINI core network, while providing access to a wireless networks. Typically, this mode can be utilized to enable integration of multi-hop wireless networks with a wired testbed. In the Bridge configuration, an ORBIT node bridges the OpenVPN interface with the wireless interfaces and removes the need to carry routing. Such a set up allows for experiments where multiple wireless end nodes are attached to VINI nodes and can be visualized as adding a physically disjoint wireless interface to a VINI node. Typically, this mode can be utilized to enable the integration of access point functionality on the wired testbed nodes (i.e., one hop wireless connectivity). Even though IPv4 was used in our tests, this framework will allow non-IP traffic. For both the Bridge and Router configurations, the VINI scripts were set to automatically generate commands for the ORBIT control framework to image, power on, and configure the ORBIT Bridge and Router nodes. Thus, automatic topology creation during experiments was facilitated.

As a proof-of concept of the integrated architecture we deployed a Layer 3 experiment over VINI and ORBIT. The objective of this experiment was to investigate hand-off issues between access points that may belong to different Internet Service Providers (ISPs). Figure 36 represents the topology that is used in the experiment. Three VINI nodes are physically located in Berkeley, California Tech, and MIT and communicate with each other via UDP tunnels. A Video Server is linked to the California Tech VINI node by using an ORBIT node with the Bridge configuration as previously described. We then attach two access points, A and B, to the other VINI nodes and configure them in the Router mode. The ath0 wireless interfaces and 172.16.X.X IP addresses on the four ORBIT nodes are manually configured to have the proper channel, essid, frequency and IP addresses. The TAP/Bridge interfaces and 192.168.X.X IP configurations on the ORBIT nodes are configured automatically. OSPF is utilized to automatically set up connectivity between nodes. The execution steps of the experiment are as follows; initially, video is streamed from the Video Server to the Mobile Client through access point A. As the Mobile Client moves away from access point A connectivity breaks and the video freezes. Access point A senses that the link is broken and through OSPF advertises that it is no longer in the routing path of the Mobile Client, and that information is propagated and reflected in all the nodes. Meanwhile the Mobile Client establishes connectivity to the network via access point B, which in turn advertises the new link to the Mobile Client. Once the new routing information is propagated in the network the video is restored. It is important to note that this investigation is different from Mobile IP because in the case of mobile IP access point A would relay the packets of the Mobile Client to access point B after the hand-off. In this scenario, the packets do not traverse the Berkleley and access point A overlay nodes and do not get redirected to access point B. They instead go through California Tech, MIT, and access point B and are delivered to the Mobile Client.

Initial results of our measurements with the Iperf tool have shown that the packet delay with Mobile IP is 443 msecs as compared to 225 msecs with our set up. Furthermore, by manual triggering the change in the routing paths when the connectivity of the Mobile Client between access point A and B changed, we have found that it takes approximately 4.5 seconds for the new routes to propagate in the network and for the video to get restored.

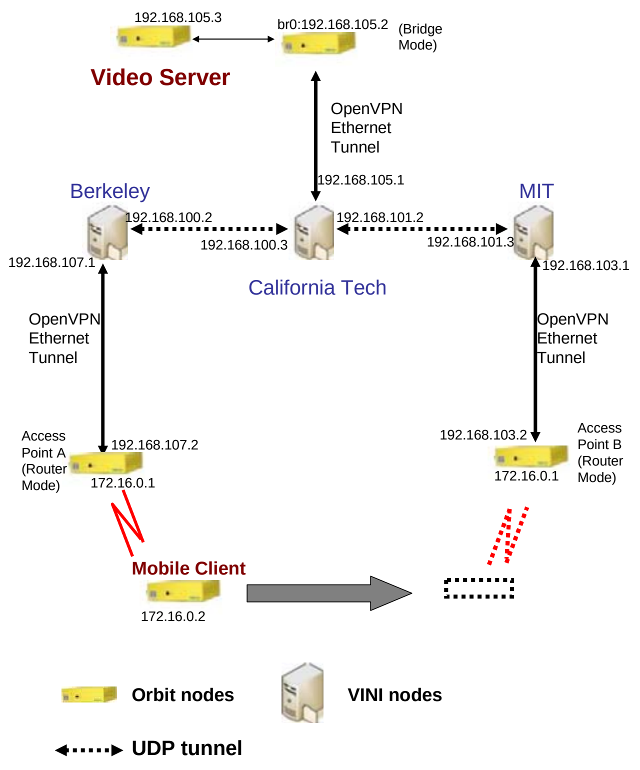


Figure 36. Integrated architecture to test hand-off in a mobility scenario.

Conclusions and Future Work: We developed and tested an integrated architecture solution that enables network layer experiments over wired and wireless networks on existing Internet links with realistic background traffic. Our solution provides an abstraction of the underlying software architecture that simplifies the configuration complexity of setting up experiments. In addition, it supports non-IP traffic and broadcast traffic, as well as any- to-any host connectivity. VINI nodes use virtualization to accommodate multiple users. The cost of providing Layer 3 experiments through virtual internetworking at the UML level is, however, a lower performance since forwarding data packets in the UML kernel incurs nearly 15% additional overhead [7]. As future work, we intend to continue our integration efforts and exploit some of the new features that will be available with VINI, such as fixed bandwidth guarantees over Internet2 links.

A paper describing the integration approach and initial results was presented at WinTech 07 [8].

References for Sec 3.3:

[1] "Understanding VINI" , https://www.vini-veritas.net/documentation/pl-vini/user/understand, available May 10, 2007
[2] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI Veritas: Realistic and Controlled Network Experimentation," in ACM SIGCOMM, Vol. 36, No. 4, pp. 3-14, October 2006.
[3] "XORP: Open Source IP Router," http://www.xorp.org/, available May 17, 2007.
[4] "Click Modular Router," http://pdos.csail.mit.edu/click/, available May 20, 2007.
[5] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," ACM Transactions on Computer Systems, vol. 18, pp. 263–297, August 2000.
[6] "OpenVPN: An open source SSL VPN solution," http://openvpn.net/.

[7] X. Jiang and D. Xu, “Violin: Virtual internetworking on overlay infrastructure,” in Proc. International Symposium on Parallel and Distributed Processing and Applications, pp. 937–946, 2004.
[8] George C. Hadjichristofi, Avi Brender, Marco Gruteser, Rajesh Mahindra, Ivan Seskar, ” A Wired-Wireless Testbed Architecture for Network Layer Experimentation Based on ORBIT and VINI”, Proc. WinTech 07, Sept 2007, Montreal.