Setting Up a Highly Configurable, Scalable Nimbus Cloud Test Bed Running on a MANET

Master's Project Report

By Joshua McKee (Advisor: Patricia J. Teller, Ph.D.)

Department of Computer Science

The University of Texas at El Paso

December 2014

Acknowledgements

This work would not have been possible without the software and technical assistance provided by FutureGrid. In particular, we extend our sincere thanks to Kate Keahey and Pierre Riteau of the University of Chicago for their assistance. Also, we wish to acknowledge the economic support provided by the Army Research Laboratory through the Army High Performance Computing Research Center (AHPCRC), Stanford University Award 60300261-107307-B.

Contents

Chapter 1: Introduction

Cloud computing is a relatively new and, therefore, rapidly evolving field in computer science. Consequently, much research is still being done with regards to harnessing clouds for certain applications. For instance, the use of mobile devices by soldiers or people involved in emergency, evacuation, or relief operations to perform important field computations is becoming prevalent. However, despite the rapid increase in the processing capability of mobile devices, there are many cases where such computations are too large for these devices to handle – either due to memory capacity or processing capability limitations. In addition, resource-intensive computations tend to increase power consumption, which is still a major issue for mobile devices. Although most of these devices have remote access to alternate computing resources via satellite, latency is a major constraint. Because of this, it is sometimes necessary for these devices to harness the processing and storage power of nearby mobile and static devices to perform computations in a timely manner. Cloud computing can assist with this as it provides a means of utilizing other computational resources without the user having to be concerned about the types and locations of the resources being used.

The goal of this project is to create a highly configurable, scalable test bed that consists of various heterogeneous hardware devices, both static and mobile, connected over a mobile ad hoc network (MANET). This test bed will allow us to measure the effectiveness of such a system, for example under different scheduling strategies. To this end, we implemented a cloud test bed, called aNTRuM (a Nimbus Test bed Running on a MANET), which consists of the hardware and software described in Chapter 3 and specified in Appendix B, and in which compute platforms are connected via a MANET. The Nimbus Infrastructure [1], an open-source Infrastructure as a Service (IaaS) cloud software, was used to implement the cloud infrastructure and OLSRd [2], an implementation of the Optimized Link State Routing (OLSR) protocol, was used to implement the MANET. Furthermore, we incorporated a technique

for individual devices in the test bed to make the status of certain resources, such as battery life, available to the cloud scheduler, which will facilitate future planned modifications to the cloud scheduler. We also set up Nimbus Phantom, an open-source multi-cloud scaling tool, in the test bed to be available for future work. To ease the creation of a cloud test bed of this type, we created automation scripts for easy test bed setup, configuration, and scaling.

Our main contributions include:

1. a methodology for implementing a cloud test bed where resources are connected via a MANET;
2. automation scripts for easy test bed setup, configuration, and scaling; and
3. a technique for detecting the state of resources, in particular, the remaining battery life of mobile devices in the cloud.

The first contribution largely can be attributed to the fact that the underlying network of our cloud test bed is a MANET – we have not found any work that has set up a cloud test bed in which compute components communicate in this way. Our second contribution automates the setup, removal, and configuration of the test bed and its devices. Our final contribution facilitates the scheduling of a cloud that contains mobile devices.

This report is organized as follows. This chapter introduced the purpose of this project, our goals, and our contributions. Chapter 2 outlines the differences between our work and other work done with regards to cloud computing test beds, and provides an overview of the FutureGrid test bed and our collaborations with the FutureGrid team. Chapter 3 presents the design of aNTRuM and particulars regarding our choices of the software that was used for its implementation. Chapter 4 reveals implementation details, specifically, how we set up our test bed to run on a MANET and how we added resource awareness to our test bed. Chapter 5 provides a flowchart of the setup of a Nimbus test bed using our automation scripts, as well as description of each script. Chapter 6 demonstrates the

functionality offered by aNTRuM and reports the results of our tests. Chapter 7 presents our conclusions and discusses possible future work. Appendix A provides definitions of abbreviations used in the report. Appendix B provides the details of aNTRuM, including software versions and network setup. Appendix C provides a tutorial on using the automation scripts for creating a setup similar to aNTRuM as well as information about when the scripts were last tested and issues that arose during testing. Appendix D presents the scripts described in Chapter 5. Appendix E presents the modifications made to the software used by aNTRuM and scripts created for use in aNTRuM. Finally, Appendix F presents the directory structure of aNTRuM's scripts.

Chapter 2: Related Work

In general, the term “mobile cloud computing” tends to carry the connotation of a set of mobile devices that utilize cloud resources, either through a cellular network or via Wi-Fi. More recently, much research has been done in the area of *cloudlets*, which serve as intermediary compute or storage resources between sets of mobile devices and a cloud. The nodes (devices) within a cloudlet usually are located closer to a mobile device than are the nodes of a cloud, thus, the communication latency between the mobile device and the resources of a cloudlet is lower than the latency between the mobile device and the resources of a cloud. For example, the MOCHA Project [3] is studying the performance of fielded applications using a cloudlet comprised of a small cluster in a vehicle as both a compute resource and arbitrator between mobile devices and a cloud. Since high latency is an issue when using cloud resources, a cloudlet can assist in reducing this latency and potentially provide sufficient resources to avoid having to communicate with a cloud altogether. This is especially true when the computational requirements of an application are too large for a mobile device but are not large enough to merit the use of remote cloud resources.

In terms of energy awareness in clouds, work, such as [4], has been done, but is limited to traditional clouds, making it less relevant to aNTRuM. For example, the authors of [4] use load balancing and virtual machine (VM) consolidation to minimize power usage, but their approach does not take into account the potentially higher delay of transferring VMs from one device to another via a MANET (due to wireless generally being slower than wired networks), or of the fact that some devices may have battery-life constraints, as is the case with aNTRuM.

This chapter discusses: (1) other cloud computing test beds (including the MOCHA Project) and (2) the FutureGrid project and our collaborations with its team.

2.1 Cloud Computing Test Beds

The cloudlet test bed used by the Mocha Project is the only one that we found in the literature that employs mobile devices and a mobile cloudlet. In contrast, our cloudlet (aNTRuM), currently not connected to a cloud, is composed of various nearby heterogeneous compute resources, rather than a single device or cluster. This adds new dynamics, such as the potential unreliability of the aforementioned compute resources due to damage, distance, battery-life constraints, etc. As mentioned in Chapter 3: (1) OSLRd, a network routing protocol that is used to manage traffic in the MANET that connects aNTRuM's devices, can assist with including networking reliability into cloud scheduler decision-making, and (2) our technique for making the status of a devices' resources available can assist with including power and other non-traditional resource constraints into the scheduler's decision-making as well.

Various other cloud test beds exist, but again do not have the same focus and goals as we do for aNTRuM. The Open Cloud Testbed (OCT) [5], for example, is a distributed cloud test bed with one focus being network provisioning. The OCT allows for provisioning resources across data centers. However, in contrast to aNTRuM, it focuses on providing a high performance (10Gb/s) network for handling extremely large data sets. Open Cirrus [6] is a much larger cloud computing test bed with one of its four main foci being systems-level research. Consequently, it provides direct access to its physical resources and, thus, allows for a greater level of control. However, there is no provision for wireless networks. Emulab [7], on the other hand, provides a wireless test bed, but only for the purpose of network-specific research. Other cloud computing test beds exist, but are not systems or network (other than high speed or reliability) focused.

Additionally, two new test beds, CloudLab [8] and ChameleonCloud [9] have recently been announced. These two test beds seek to provide cloud computing test beds that are configurable at all levels and

support special hardware, which includes programmable networks. However, there is little to no indication that these test beds will provide real wireless hardware.

2.2 FutureGrid

One of our collaborators for this project is the FutureGrid project team (the FutureGrid project recently came to a close). FutureGrid [10] is a test bed developed for the purpose of allowing scientists to experiment with new approaches to cloud computing, as well as parallel and grid computing. FutureGrid has a set of heterogeneous computing systems, a data management system, and a dedicated network. Eucalyptus, Nimbus, and OpenStack were the three open-source cloud IaaS offering on FutureGrid. The FutureGrid team has indicated to us that they believe that our project is novel, mainly because of the underlying network infrastructure of aNTRuM.

Chapter 3: Design

As mentioned in Chapter 1, the goal of this project is to create a highly configurable, scalable cloud test bed that consists of various heterogeneous hardware devices, both static and mobile, connected over a mobile ad hoc network (MANET). To this end, we implemented a test bed, called *aNTRuM* (A Nimbus Test bed Running on a MANET), in which the cloud hardware components, specifically, the head node, virtual machine monitor (VMM) nodes, and client nodes are connected via a MANET. aNTRuM also consists of a set of software components that compose the cloud infrastructure, namely, the cloud, hypervisor, and optimized link state routing (OLSR) protocol software, as well as a multi-cloud scaling tool. This chapter describes: (1) the functionality of each software component and how each component interacts with the other software components of aNTRuM, and (2) the software used to implement each of the components. A description of the development and actual implementation of the test bed is presented in Chapter 4.

3.1 High-level Design

Figure 3.1 presents a high-level diagram that depicts the cloud, hypervisor, optimized link state routing (OLSR) protocol, and multi-cloud scaling software of aNTRuM and their interactions. As shown in the figure, aNTRuM is comprised of a head node, VMM nodes, and client nodes. The head node handles the scheduling of the virtual machines (VMs) that run on the VMM nodes, the management of cloud hardware resources, and interfacing with clients. Installed on the head node are the cloud IaaS and OLSR software. The VMM nodes, which handle the provisioning of VMs, contain the cloud control agent, hypervisor, and OLSR software. The cloud IaaS, cloud control agent, and cloud client software are all parts of the cloud software. The client uses the cloud client software to interface with the IaaS software on the head node. By interfacing with the cloud control agent software on a VMM node, which, in turn interfaces with the VMM node's hypervisor, the head node's cloud IaaS software creates and manages

VMs. The cloud scaling software runs on a VM in the cloud, and allows for easily managing a set of resources deployed across multiple clouds. The capability provided by this software will be used in future work.

Because the devices in aNTRuM are connected via a MANET, the OLSR software on the head node, VMM nodes, and client nodes is used to facilitate communication among the nodes of the cloud. The information provided by OLSR also will be used in future work to automate some functions of the head node, which we describe in more detail in the next section.

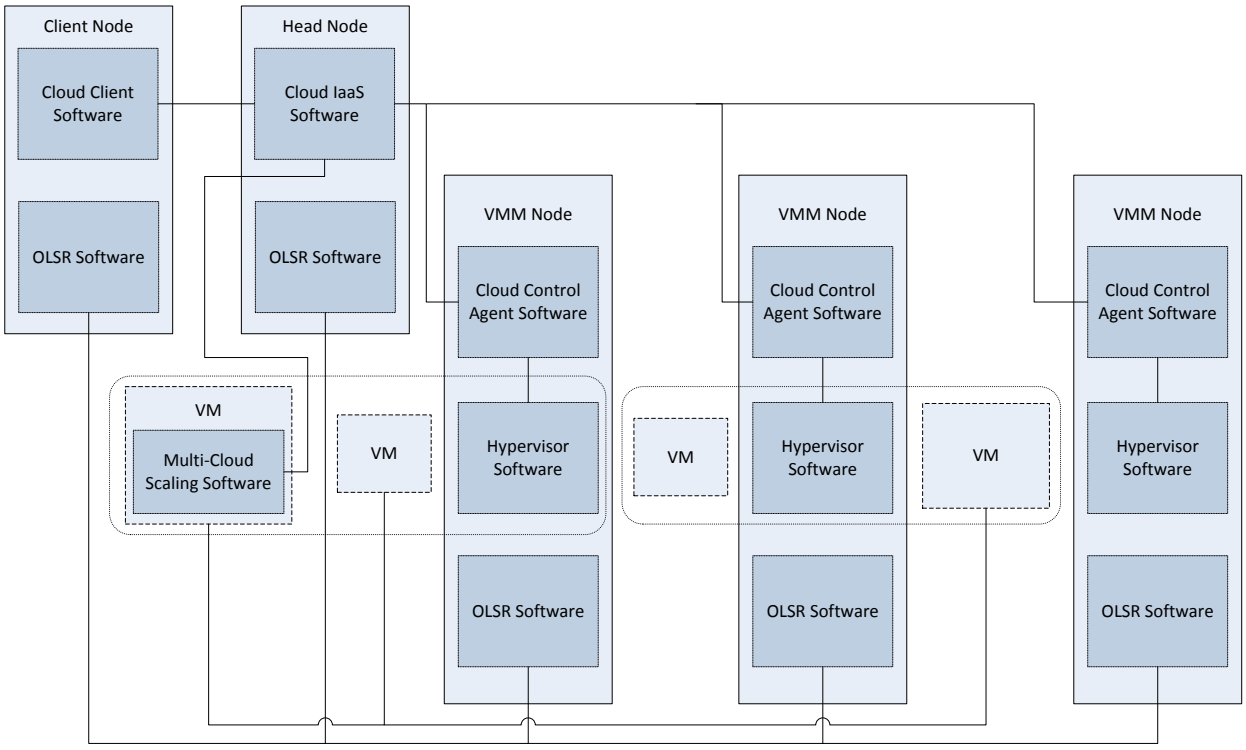

Figure 3.1: High-level architecture of aNTRuM.

3.2 Software Selected to Implement High-level Design

Below we present the software that was selected to implement each component of the high-level architecture of aNTRuM shown in Figure 3.1. In addition, we provide support for our selection decisions.

Cloud Software: The *Nimbus* infrastructure [1] was used to implement the cloud IaaS and control components of aNTRuM. The Nimbus infrastructure is open-source IaaS cloud software that provides resources to users in the form of VMs, allocated and configured according to user requirements. A system running the Nimbus infrastructure needs to have a service node (head node) running the cloud IaaS software and one or more VMM nodes, each running the cloud control agent software. Nimbus also provides client software that can be used with the Nimbus infrastructure. For simplicity, the Nimbus infrastructure will be referred to as Nimbus in the remainder of this document.

Nimbus uses a service called *Cumulus* to manage the VM images available for use in the cloud. These images can be uploaded by either the cloud administrator via the IaaS software or by users via the Nimbus client software or other EC2 clients. Normally, Cumulus stores the images in the service node, but it can be configured to store them across the various VMM nodes.

Nimbus was selected for the IaaS cloud software for the following two reasons: (1) An IaaS cloud allows us to use a MANET as the interconnection network for aNTRuM and it permits us to extend, in the future, the cloud's scheduling capabilities. This is because IaaS is the lowest level of cloud infrastructure (it provides the most fundamental cloud computing services, i.e., VMs, storage, network, etc.), and changing something as foundational to the cloud as the network configuration requires changes at the lowest levels. (2) According to [11], which provides a comparison of the main open-source IaaS software currently available, Nimbus offers the option of using either a centralized or local (per node) DHCP server for assigning IP addresses to VMs. Since most solutions for setting up DHCP on MANETs are not for VMs, and since the only need for a DHCP server in our test bed was for assigning IP address to VMs, it was necessary to have local (per VMM node) DHCP servers.

Hypervisor Software: The purpose of hypervisor software is simply to create and run VMs. Nimbus currently interfaces with either the XEN [12] or KVM [13] hypervisor. We chose to use KVM, over XEN,

for a number of reasons but mainly because XEN is a type 1 (bare metal) hypervisor. This means that it runs directly on a node's hardware with the control domain (`dom0`) running above it; this requires modifications to the Linux kernel. Since we want aNTRuM to have the capability of running on devices that are not completely dedicated to the cloud, using XEN would defeat this purpose. In contrast, KVM is built into the Linux kernel, giving it the ability to act as a type 1 hypervisor but still reside within the Linux operating system. However, KVM requires hardware virtualization support, making it less friendly to some ARM-based devices that don't include such virtualization support.

Multi-Cloud Scaling Software: *Phantom* [14], open-source software that allows users to manage resources distributed over multiple clouds via its auto-scaling capabilities, was used as the multi-cloud scaling software. Phantom can be extended by adding new decision engines; Phantom's decision engines are what determine its behavior. It uses OpenTSDB [15] for monitoring VM resources, and can collect performance-related data from VMs, such as available memory, load, etc., which in turn can be used in Phantom's decision engines. Originally, Phantom was going to be used for detecting the state of resources in our test bed. However, since Phantom's level of control is limited to making requests to clouds, and does not have the capability of determining where and how VMs are scheduled in a particular cloud, it was not practical to use Phantom in that way. In Chapter 4, we discuss this in more detail and describe how we added resource awareness into our test bed. In Chapter 7, we mention the possibility of using Phantom in future work to manage multiple clouds.

OLSR Software: We decided to use the OLSR daemon (*OLSRd*) [2] running on a mobile ad-hoc network (MANET) for the OLSR protocol software portion of aNTRuM. This allows us to mimic fielded device connectivity and to align our research with the Army's interest in routing protocols for wireless networks, such as the Optimized Link State Routing (OLSR) protocol. *OLSRd* is an implementation of the OLSR protocol with the additional feature of link quality sensing, a feature that is lacking in the OLSR

protocol. As mentioned in Chapter 7, in future work the data provided by OLSRd can provide data that may allow us to modify Nimbus to detect when nodes disappear from and reappear in the network and to handle those situations accordingly.

Operating System: For the underlying operating system (OS) of the aNTRuM nodes, we chose to use *Ubuntu 14.04 LTS* [16]. This selection was due to our familiarity with this OS and its popularity. Version 14.04 was chosen because it is the latest Long-Term Support (LTS) version of Ubuntu to date. Additionally, the development of Ubuntu for phones and tablets is currently in progress. We mention in Chapter 7 that we plan to later expand aNTRuM to include mobile devices such as tablets and phones (ARM devices) -- having the same OS distribution on the various nodes of aNTRuM will ease the process of doing so. This component of the test bed was not mentioned above since it is assumed that each machine will be running an OS and none of the components are Linux-distribution dependent.

Chapter 4: Implementation

In this chapter, we describe the details of our implementation of aNTRuM. In the first section we reveal how we modified Nimbus to run on a MANET. In the second section we discuss how we made the state of device resources available to the Nimbus cloud scheduler.

4.1 Setting up Nimbus on a MANET

The following sections describe our process of setting VM bridging using wireless cards and setting up Nimbus to run on a MANET, as well as the challenges of doing so.

4.2.1 Setting up VM Bridging using Wireless Cards

In any IaaS cloud, for the client to be able to connect to the VMs, a network bridge is required on the VMM node hosting the VM. One of the challenges with setting up aNTRuM to run on a MANET is that most wireless cards to not natively support bridging like Ethernet cards do. Because of this, it was necessary to use a workaround that involved the following five steps for each of aNTRuM's VMM nodes:

1. create a TAP interface,
2. enable kernel IP forwarding,
3. enable Proxy ARP for the WLAN and TAP interfaces,
4. assign the TAP interface a certain IP address, and
5. add routes from the host to the guests.

Step 1: A TAP device is a virtual-network kernel device that simulates a link-layer device, and is used to create network bridges. By attaching the VM to a unique TAP device created for that VM, traffic is bridged between the wireless card on the VMM hosting the VM and the VM itself.

Step 2: By enabling kernel IP forwarding on a VMM node the kernel can forward the traffic that arrives via the wireless card to the VMs running on the node, assuming the destination subnet for said traffic is the same as that of the VMs.

Step 3: Address resolution protocol (ARP) requests allow determination of the MAC address of a machine using the machine's IP address via an ARP broadcast message. An ARP proxy will respond to requests for MAC addresses that are not on the network, but for which it knows the locations, and will return its own MAC address. Enabling this technology on a VMM node for both the wireless interface and the TAP interfaces assigned to the various VMs allows for the wireless interface to respond to ARP requests directed towards VMs running on the host, as well as from the VMs to the host's network. ARP functions at the link layer.

Step 4: By assigning an IP address to a TAP device in the same subnet as the IP address of a VM attached to the TAP device, traffic can flow between the network card of the VMM node on which the VM is running and the VM by creating a route to the network to which the VM belongs.

Step 5: By adding a route to a VMM node's routing table for the IP address of a VM running on the node and the TAP device to which the VM is attached, traffic destined for that VM can be routed to it.

It is important to note that the setup described above encompasses only the necessary steps taken to set up bridging between a VMM node's wireless card and VMs running the VMM node; it does not account for security issues that may appear as a result of such a setup. Because aNTRuM was set up as a cloud test bed, and since our focus was on setting up a working test bed with specific features, we did not focus on setting up a firewall on the VMM nodes to protect our bridging setup. This is left for future work.

4.1.2 Configuring Nimbus to Run on a MANET

Since the Nimbus software is built to run on devices connected via a wired network, it was necessary to make some changes to the software and its configuration to be able to run on devices connected via a MANET, with a bridging setup as described in Section 4.1.2. This section describes those software modifications; the content of the files themselves can be found in Appendix E. Note that the following three paragraphs refer to modifications done to software running on the VMM nodes of the test bed.

workspace-control.sh: Because the workaround described in Section 4.1.2 cannot be achieved by a normal configuration of the Nimbus cloud software, it was necessary to modify the script called `workspace-control.sh`. This is the main script for handling requests for VM creation and destruction. It sends commands and parameters as large strings to a Python script, which handles the rest. Essentially, the modification captures the command and parameters before they are sent to the Python script, and runs one of two auxiliary scripts, depending on the captured command. One of the scripts sets up one or more TAP interfaces to attach to the VM that will be created and the other tears down the TAP interface when the VM will be terminated. The parameters that are captured are the name of the VM and its IP address; these parameters are used in the setup script.

libvirt_template.xml: Another file that was modified to allow for using wireless bridging was the template for the VM XML definition used by libvirt (`libvirt_template.xml`), a hypervisor management tool used by Nimbus to manage KVM. The template was modified for the VM to use a generic Ethernet device, which allows the VM to attach to a TAP interface and to not use the standard VM network startup and shutdown scripts that are triggered by using a generic interface definition.

qemu.conf: A few changes to a configuration file (`qemu.conf`) were also necessary to allow for libvirt to use a generic Ethernet device. These modifications include

- setting `clear_emulator_capabilities` to `0`,

- setting `user` to "`root`",
- setting `group` to "`root`", and
- adding "`/dev/net/tun`" to `cgroup_device_acl`.

Since aNTRuM runs Ubuntu, which uses apparmor (not SELinux), it was necessary to remove apparmor in order to allow the above setup to work, thus, increasing potential security issues. This is because apparmor blocks the VM from attaching to the TAP device, even though it is allowed in the configuration file.

As mentioned in Chapter 3, it was necessary to set up a local DHCP server on each VMM node. Fortunately, this option is available in Nimbus. Another option available within Nimbus is the ability to cache a VM image on a VMM after the first time a VM based on that image is created on the VMM. This was important for our setup since the transfer of large files, such as VM images, is generally slower across a MANET than across a wired network. The option to store the VM images on various VMMs in the first place by disturbing Cumulus' image storage is also available in Nimbus, but it is much more complicated to set up and was unnecessary for our purposes. Thus, this is left as a future work item.

cloud-client.sh: Because aNTRuM runs on a MANET, it does not have a domain name system (DNS) available for its devices and VMs. Therefore, the main script of the Nimbus client software (`cloud-client.sh`) was modified to call auxiliary scripts to add and remove entries for VMs to the `hosts` file on the device on which the Nimbus client software is installed. This allows users to access their VMs from the device running the Nimbus client software using the VM's hostname, not just its IP address. However, this modification only works when interfacing with the test bed's cloud via the Nimbus client software, and not via the EC2 interface, which is used by a user when he or she sets up Phantom and by Phantom to communicate with the test bed's cloud. Hence, an additional workaround was needed to overcome this limitation when setting up Phantom, which consists of creating a VM through the Nimbus

client software, getting its IP and hostname information, adding the information to the hosts file, and destroying the VM. This workaround assumes that no other VMs will be created or destroyed between when it is finished and prior to setting up Phantom.

4.2 Making Nimbus Resource Aware

As mentioned in Chapter 3, we originally planned on using Phantom to make the states of resources available to our test bed's cloud scheduler. However, we were not able to do this because Phantom only is able to manage VMs across clouds, not within a single cloud. Even though Phantom's VM management and scaling capabilities far exceed those of the Nimbus cloud scheduler, these capabilities are only useable across multiple clouds (see [17, 18] for more details on Phantom). As a result and since the scope of this project is limited to a single cloud, this is left as future work. Nonetheless, Phantom is included as part of aNTRuM.

Because the Nimbus cloud scheduler is limited to simply providing the VM resources requested by the user, it was impractical to make it provide any more than the current state of resources of its available VMM nodes, for which it knows only basic information such as the maximum amount of RAM allocated for VMs to use and how many CPU cores to allot to VMs. The devices in our test bed have other constraints such as battery life and a widely variable range of currently available RAM (since they simulate VMM nodes that may not be dedicated to only that purpose). Thus, we created a small script to run on each VMM node that outputs this information to a file in a specific location on the VMM node after a certain interval of time. Because the cloud software on the head node uses secure shell (SSH) for requesting resources on its VMM nodes, it could easily access that file and use the resident information in its scheduler's decision process. For example, it could use the available battery life of a mobile VMM node and the allotted running time of a VM to decide whether to schedule on that VMM node. VMM nodes also use SHH to notify the head node that a VM is propagated, so they could alert the cloud

software on the head node that its battery life is low or depleting. This may seem rudimentary, but it is consistent with the current capabilities of the scheduler. In this case, the extra capabilities provided by Phantom would be useful if Phantom had control of individual devices of a cloud.

The following is a description of the information provided by the script that collects the system resource information (the script itself can be found in Appendix E).

For memory information:

- Total memory (in kB): `memory.memtotal`
- Free memory (in kB): `memory.memfree`

For device information:

- Device has battery(s) (value can be 0 or 1): `device.hasbattery`

(The following will only be provided if the device has a battery)

- Device is plugged in (value can be 0 or 1): `device.pluggedin`

For battery information:

- For each battery:
    - Battery status (values can be 'full', 'charging', 'discharging'): `battery_[battery_number].status`
    - Battery capacity (in %): `battery_[battery_number].capacity`

Chapter 5: Automatic Configuration

A computer running independently of the cloud is set up to function as the test bed management (master) node of aNTRuM. Using a set of automation scripts, the master node allows easy setup, configuration, and scaling of a Nimbus cloud, while not interfering with it otherwise. The following sections provide a flowchart of the automated setup process and describe the functionality of the scripts.

5.1 Flowcharts

Figure 5.1 describes the steps to follow, using the automation scripts that run on the master node of aNTRuM, to set up a resource-aware Nimbus cloud test bed that runs on a wireless ad-hoc network that employs OSLRd.

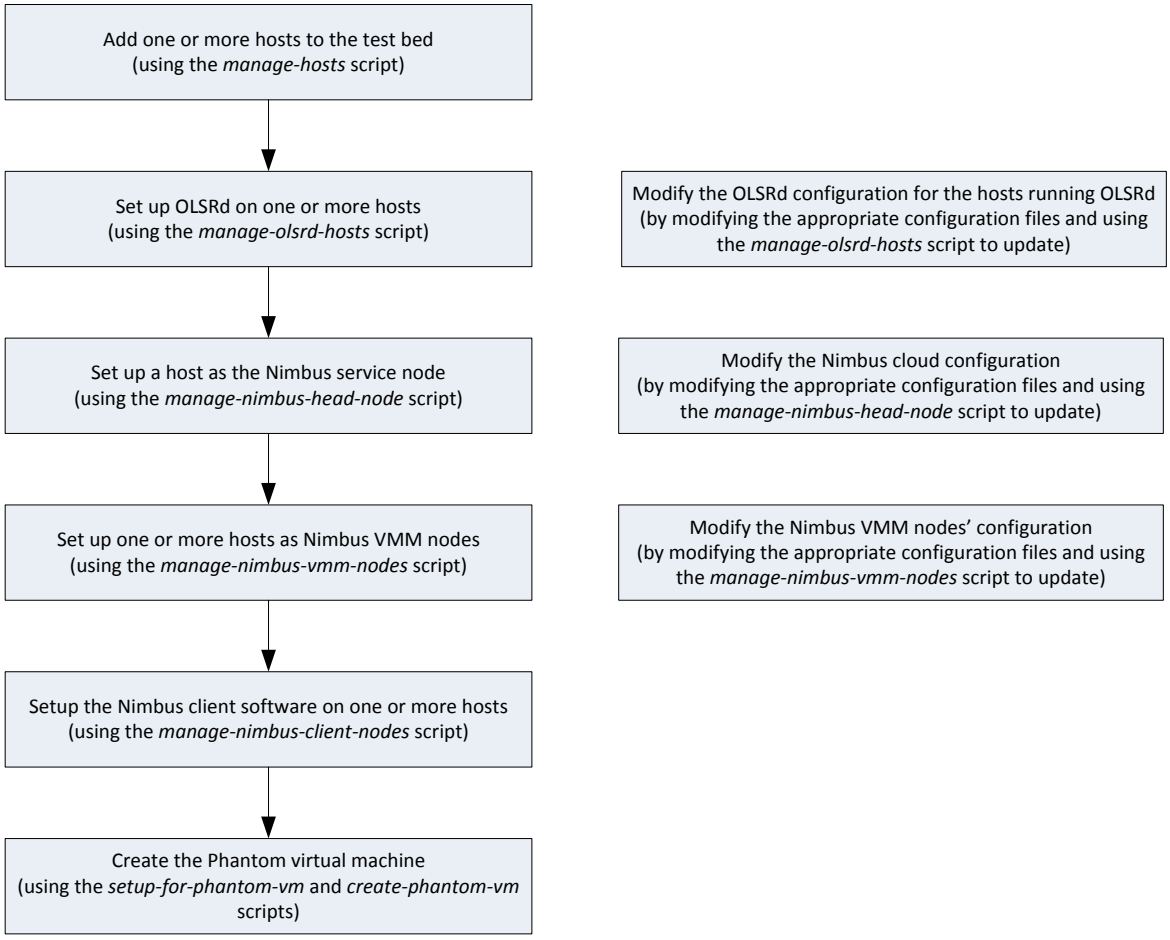

Figure 5.1: aNTRuM setup and modification using automation scripts

The scripts also allow for easy reconfiguration of hosts from one purpose to another, as well as easy removal of hosts from the test bed and even a complete teardown of the test bed. They also allow for easily propagating changes to the settings of the software used in the test bed.

5.2 Automation Scripts

This section describes the function of each script that is executed on the test bed management node. A tutorial that takes a user through each step in the aNTRuM setup flowchart shown in Figure 5.1, i.e., describes how to use the scripts to set up a test bed similar to aNTRuM, is provided in Appendix C.

5.2.1 Test Bed Management Scripts

Resident on the master node are scripts that call the core scripts described in Section 5.2.2; these "primary scripts" allow the user to manage the test bed. The following table lists the core scripts called by each primary script.

Table 5.1: Core scripts called by primary scripts. The first row lists the primary scripts and the following rows name the core scripts that are called by each primary script.

manage-hosts	manage-olsrd-hosts	manage-nimbus-head-node	manage-nimbus-vmm-nodes		manage-nimbus-client-nodes	manage-phantom-vm
add-host	add-olsrd-host	setup-nimbus-head-node	add-nimbus-vmm-node	connect-nimbus-vmm-to-head	add-nimbus-client-node	setup-for-phantom-vm
remove-host	remove-olsrd-host	remove-nimbus-head-node	remove-nimbus-vmm-node	disconnect-nimbus-vmm-from-head	remove-nimbus-client-node	remove-phantom-vm-setup
update-host	update-olsrd-hosts	update-nimbus-head-node	update-nimbus-vmm-nodes	is-nimbus-vmm-connected-to-head	update-nimbus-client-nodes	create-phantom-vm
list-hosts	list-olsrd-hosts	list-nimbus-head-node	list-nimbus-vmm-nodes		list-nimbus-client-nodes	

5.2.2 Test Bed Management Core Scripts

The following are descriptions of the core scripts listed in Figure 5.1, which are executed on the test bed management node.

5.2.2.1 manage-host core scripts

5.2.2.1.1 add-host

Description: Sets up a new host, host A, on the test bed.

Prerequisites: Host A does not already belong to the test bed.

Script outline:

1. Adds the public key of the management node to the `authorized_keys` file of host A.
2. Runs a script to set up password-less `sudo` for the user on host A. This is done by creating a new file for the user in host A's `sudoers.d` directory with the `NOPASSWD` parameter set in the file.
3. Adds an entry for host A to the `testbed_hosts` file.
4. Adds an entry for host A to the management node's `hosts` file.
5. Runs the *update-hosts* script.

5.2.2.1.2 remove-host

Description: Removes a host, host A, from the test bed.

Prerequisites: Host A belongs to the test bed.

Script outline:

1. If the `--force` parameter is not set, then:
    a. Runs the *remove-olsrd-host* script if OLSRd has been set up on the host A.
    b. Removes the entries for the other test bed hosts from host A's `hosts` file.
    c. Removes password-less `sudo` for the user by removing the user file created by *add-hosts* from the `sudoers.d` directory on host A.
    d. Removes the public key of the management node from the `authorized_keys` file of host A.
2. Removes the entries for host A from the management node's `hosts` file.
3. Removes the entry for host A from the `authorized_keys` file of the management node.
4. Removes the entry for host A from the `testbed_hosts` file.
5. Runs the *update-hosts* script.

5.2.2.1.3 update-hosts

Description: Updates the `hosts` file on all hosts that belong to the test bed.

Prerequisites: The test bed contains at least one host.

Script outline:

1. For each host, host A, in the test bed:
    a. Copies the test bed sections of the management node's `hosts` file to host A's `hosts` file.

5.2.2.1.4 list-hosts

Description: Lists the hosts belonging to the test bed.

Prerequisites: The test bed contains at least one host.

Script outline:

1. Prints the `testbed_hosts` file.

5.2.2.2 manage-olsrd-hosts core scripts

5.2.2.2.1 add-olsrd-host

Description: Sets up the OLSRd software to run on a host, host A.

Prerequisites:

- Host A is part of the test bed.
- OLSRd is not already set up on the host.

Script outline:

1. Copies the OSLRd setup files to host A.
2. Installs the packages required to run OLSRd on host A, and reboots host A if necessary.

3. Builds and installs OLSRd on host A.
4. Runs the *update-olsrd-host* script for host A.
5. Runs the *update-hosts* script.

5.2.2.2.2 remove-olsrd-host

Description: Removes the OLSRd software from a host, host A.

Prerequisites:

- Host A is part of the test bed.
- OLSRd is set up on host A.

Script outline:

1. Kills the `olsrd` process on host A.
2. Removes the configuration to set up an ad hoc connection on the wireless card from the `network/interfaces` file on host A.
3. Uninstalls OLSRd on host A, deregisters OSLRd from `update-rc.d`, and removes the OLSRd startup script from `init.d` on host A.
4. If user approves, uninstalls the packages required to run OLSRd from host A, and reboots host A if necessary.
5. Removes the OLSRd data from the entry for host A in the `testbed_hosts` file.
6. Removes the OLSRd data from the entries for host A in the management node's `hosts` file.
7. Runs the *update-hosts* script.

5.2.2.2.3 update-olsrd-hosts

Description: Runs the *update-olsrd-host* script for each host running OLSRd in the test bed.

Prerequisites: At least one host in the test bed runs OSLRd.

Script outline:

1. For each host, host A, running OLSRd in the test bed:
    a. Runs the *update-olsrd-host* script for host A.
2. Runs the *update-hosts* script.

5.2.2.2.4 update-olsrd-host

Description: Updates the OLSRd ad hoc setup script, the OLSRd configuration file, and the wireless-ad-hoc-connection-related entries in the `network/interfaces` file on a host, host A, then restarts OLSRd on host A.

Prerequisites:

- Host A is part of the test bed.
- OLSRd is set up on host A.

Script outline:

1. Copies the OLSRd ad hoc setup script to the OLSRd directory on host A and runs it. This setup script configures the available wireless card to run an ad hoc connection.
2. Adds the ad hoc connection data to the entry for host A in the `testbed_hosts` file.
3. Adds the ad hoc connection data to the entries for host A in the management node's `hosts` file.
4. Kills the `olsrd` process on host A.
5. Copies the OLSRd configuration file to host A.
6. Starts OLSRd on host A.
7. Adds a configuration to set up the wireless card to run the ad hoc connection set up by the OLSRd ad hoc setup script to the `network/interfaces` file on host A.

8. Adds a startup script to `init.d` on host A to run OLSRd on startup and registers the script with `update-rc.d`.

5.2.2.2.5 list-olsrd-hosts

Description: Lists the hosts belonging to the test bed running OLSRd.

Prerequisites: At least one host in the test bed runs OSLRd.

Script outline:

1. Prints the `testbed_hosts` file and filters the results to display hosts running OLSRd.

5.2.2.3 manage-nimbus-head-node core scripts

5.2.2.3.1 setup-nimbus-head-node

Description: Sets up a host, host A, as the Nimbus service node (head node) of the test bed.

Prerequisites:

- Host A is part of the test bed.
- OLSRd is set up on host A.
- No other host is set up as the service node in the test bed.

Script outline:

1. Copies the Nimbus IaaS setup files to host A.
2. Installs the packages required to run the Nimbus service node software on host A, and reboots host A if necessary.
3. Performs an operation on host A to work around a bug in Python virtualenv.
4. Runs the Nimbus service node install script on host A.
5. Generates an RSA key pair on host A.
6. Sets host A as the Nimbus service node in the `testbed_hosts` file.

7. Modifies `rc.local` to run Nimbus on startup on host A.
8. Runs the *update-nimbus-head-node* script.
9. If user approves, for each host set up as a Nimbus VMM node in the test bed (host B):
    a. Runs the *connect-nimbus-vmm-to-head* script for host B.

5.2.2.3.2 remove-nimbus-head-node

Description: Removes the Nimbus service node (head node) software from the test bed host, host A, which is set up as the Nimbus service node.

Prerequisites: The test bed contains a host set up as the Nimbus service node.

Script outline:

1. If user approves, for each host, host B, set up as a Nimbus VMM node in the test bed:
    a. Runs the *remove-nimbus-vmm-node* script for host B.

   Otherwise, for each host, host B, set up as a Nimbus VMM node in the test bed:
    a. If host B is registered as a VMM node on host A, runs the *disconnect-nimbus-vmm-from-head* script for host B.
2. If user approves, for each host set up as a Nimbus client node in the test bed (host C):
    a. Runs the *remove-nimbus-client-node* script for host C.
3. If the test bed contains a host configured to set up a VM running Phantom and if user approves, runs the *remove-phantom-vm-setup* script.
4. Stops the Nimbus service on host A.
5. Removes the lines that run Nimbus on startup from `rc.local` on host A.
6. Removes the Nimbus directories from host A.
7. Uninstalls the packages required to run the Nimbus service node software from host A, and reboots host A if necessary.

8. Unsets host A as the Nimbus service node in the `testbed_hosts` file.

5.2.2.3.3 update-nimbus-head-node

Description: Updates the Nimbus service node (head node) software configuration files on the test bed host, host A, which is set up as the Nimbus service node.

Prerequisites: The test bed contains a host that is set up as the Nimbus service node.

Script outline:

1. Creates public and private network pool files for Nimbus to use when assigning network configurations to VMs on host A.
2. Copies the network pool files and Nimbus service node configuration files to host A.
3. If no DNS server is specified in the main configuration file, adds a fake DNS value to the network configuration sample file on host A.
4. Restarts Nimbus on host A.

5.2.2.4.8 list-nimbus-head-node

Description: Lists the host that belongs to the test bed and is set up as the Nimbus service node (head node).

Prerequisites: The test bed contains a host that is set up as the Nimbus service node.

Script outline:

1. Prints the `testbed_hosts` file and filters the results to display the host that is set up as the Nimbus service node.

5.2.2.4 manage-nimbus-vmm-node core scripts

5.2.2.4.1 add-nimbus-vmm-node

Description: Sets up the host, host A, as a Nimbus VMM node.

Prerequisites:

- Host A is part of the test bed.
- OLSRd is set up on host A.
- The test bed contains a host that is set up as the Nimbus service node.
- Host A is not already set up as a Nimbus VMM node.

Script outline:

1. Returns an error if host A does not support hardware virtualization (or if it is not enabled).
2. Returns warnings if host A does not have a 64-bit CPU or does not have a 64-bit OS installed.
3. Copies the Nimbus VMM software to host A.
4. Installs the packages required to run the Nimbus VMM software on host A (including KVM and libvirt), and reboots host A if necessary.
5. Adds user to libvirt and KVM groups on host A.
6. Returns an error if hardware acceleration is not enabled in the BIOS on host A.
7. Copies configuration files for KVM and libvirt to host A and restarts libvirt on host A.
8. Installs the Nimbus VMM software on host A.
9. Runs the *update-nimbus-vmm-node* script for host A.
10. Sets the file and folder permissions in the Nimbus VMM installation directory on host A as needed.
11. Runs a script (`test-dependencies.sh`) on host A that does a dependencies check for the Nimbus VMM software, and returns an error if the script fails.
12. Removes apparmor from host A, and reboots host A.
13. Tests the creation of a VM on host A by:

a. Running `control-test.sh` (from Nimbus) on host A using a sample image and sample configuration, and returning an error if unsuccessful.
b. Trying to ping the VM, and returning an error if unsuccessful.

14. Runs `destroy-control-test.sh` (from Nimbus) to destroy the test VM on host A.
15. Sets host A as a Nimbus VMM node in the `testbed_hosts` file.
16. Runs the *connect-nimbus-vmm-to-head* script for host A.

5.2.2.4.2 remove-nimbus-vmm-node

Description: Removes the Nimbus VMM software from a host, host A.

Prerequisites:

- Host A is part of the test bed.
- Host A is set up as a Nimbus VMM node.

Script outline:

1. If a host, host B, is set up as the Nimbus service node and host B has host A registered as one of its VMM nodes, runs *disconnect-nimbus-vmm-from-head* for host A.
2. Removes the Nimbus VMM software directories from host A.
3. Uninstalls the system resource metric collector script from host A, deregisters it from `update-rc.d`, and removes the system resource metric collector startup script from `init.d` on host A.
4. If user approves, reinstalls apparmor, and reboots host A if necessary.
5. If user approves, uninstalls the packages required to run the Nimbus VMM software from host A, and reboots host A if necessary.
6. Unsets host A as a Nimbus VMM node in the `testbed_hosts` file.

5.2.2.4.3 update-nimbus-vmm-nodes

Description: Runs the *update-nimbus-vmm-node* script for each host set up as a Nimbus VMM node in the test bed.

Prerequisites: At least one host in the test bed is set up as a Nimbus VMM node.

Script outline:

1. For each host, host A, set up as a Nimbus VMM node:
    a. Runs the *update-nimbus-vmm-node* script for host A.

5.2.2.4.4 update-nimbus-vmm-node

Description: Updates Nimbus' `workspace-control` script, the DHCP and DHCPd configuration files, and the Nimbus VMM software configuration files on a host, host A.

Prerequisites:

- Host A is part of the test bed.
- Host A is set up as a Nimbus VMM node.

Script outline:

1. Copies a modified version of the Nimbus VMM `workspace-control` script and the additional files used by that version to host A (see Appendix E for details on the modifications to that script).
2. Copies the DHPCd and DHPC configuration files to host A.
3. Copies the Nimbus VMM software configuration files to host A.
4. Copies the system resource metric collector script (see Chapter 4 for details) to host A and installs it.

5. Adds a startup script to `init.d` on host A to run the system resource metric collector script on startup, registers the script with `update-rc.d`, and starts the system resource metric collector script.

5.2.2.4.5 connect-nimbus-vmm-to-head

Description: Registers a host, host A, that is set up as a Nimbus VMM node on the Nimbus service node (head node).

Prerequisites:

- Host A is part of the test bed.
- Host A is set up as a Nimbus VMM node.
- The test bed contains a host set up as the Nimbus service node.
- Host A is not already registered as a VMM node on the Nimbus service node.

Script outline:

1. Exchanges public keys between host A and the Nimbus service node.
2. Registers host A as a VMM node on the Nimbus service node.
3. Prompts user to enter the amount of RAM to be allocated to VMs on host A. If response is invalid, defaults to maximum available RAM.
4. If host A is the first VMM node added to the Nimbus service node:
    a. Populates the Nimbus auto-configuration decisions file and copies it to the Nimbus service node.
    b. Runs the Nimbus auto-configuration script on the Nimbus service node.
    c. Restarts Nimbus on the Nimbus service node.

    Otherwise:
    a. Registers host A as a VMM node on the Nimbus service node.

5.2.2.4.6 disconnect-nimbus-vmm-from-head

Description: Deregisters a host, host A, that is set up as a Nimbus VMM node on the Nimbus service node (head node).

Prerequisites:

- Host A is part of the test bed.
- Host A is set up as a Nimbus VMM node.
- The test bed contains a host that is set up as the Nimbus service node.
- Host A is registered as a VMM node on the Nimbus service node.

Script outline:

1. If VMs are running on host A and user approves, terminates those VMs.
2. Deregisters host A as a VMM node on the Nimbus service node.
3. Removes the public key of the Nimbus service node from the `authorized_keys` file of host A, and vise-versa.

5.2.2.4.7 is-nimbus-vmm-connected-to-head

Description: Check if a host, host A, that is set up as a Nimbus VMM node is registered on the Nimbus service node (head node).

Prerequisites:

- Host A is part of the test bed.
- Host A is set up as a Nimbus VMM node.
- The test bed contains a host set up as the Nimbus service node.

Script outline:

1. Checks if host A is registered as a VMM node on the Nimbus head node, and outputs the result.

5.2.2.4.8 list-nimbus-vmm-nodes

Description: Lists the hosts belonging to the test bed that are set up as Nimbus VMM nodes.

Prerequisites: At least one host in the test bed is set up as a Nimbus VMM node.

Script outline:

1. Prints the `testbed_hosts` file and filters the results to display the hosts in the test bed that are set up as Nimbus VMM nodes.

5.2.2.5 manage-nimbus-client-nodes core scripts

5.2.2.5.1 add-nimbus-client-node

Description: Sets up the nimbus client software on a host, host A.

Prerequisites:

- Host A is part of the test bed.
- OLSRd is set up on host A.
- The test bed contains a host that is set up as the Nimbus service node.
- The Nimbus client software is not already set up on host A.

Script outline:

1. Copies the Nimbus client software to host A.
2. Installs the packages required to run the Nimbus client software on host A, and reboots host A if necessary.
3. Generates an RSA key pair on host A if one does not already exist.
4. Adds a new Nimbus cloud user on the Nimbus service node.
5. Unpacks the Nimbus client software on host A.

6. Copies the necessary cloud and new user configuration files to host A from the Nimbus service node.
7. Sets host A as a Nimbus client node in the `testbed_hosts` file.
8. Runs the *update-nimbus-client-node* script for host A.

5.2.2.5.2 remove-nimbus-client-node

Description: Removes the Nimbus client software from a host, host A.

Prerequisites:

- Host A is part of the test bed.
- The Nimbus client software is set up on host A.

Script outline:

1. Removes Nimbus client software directories from host A.
2. Removes the Nimbus cloud user associated with host A on the Nimbus service node.
3. Removes entries for VMs pertaining to the Nimbus cloud user associated with host A from host A's `hosts` file.
4. Unsets host A as a Nimbus client node in the *`testbed`*`_hosts` file.

5.2.2.5.3 update-nimbus-client-nodes

Description: Runs the *update-nimbus-client-node* script for each host with the Nimbus client software set up on it.

Prerequisites: At least one host in the test bed has the Nimbus client software set up on it.

Script outline:

1. For each host, host A, with the Nimbus client software set up on it:
    a. Runs the *update-nimbus-client-node* script for host A.

5.2.2.5.4 update-nimbus-client-node

Description: Updates Nimbus' `cloud-client.sh` script on a host, host A.

Prerequisites:

- Host A is part of the test bed.
- The Nimbus client software is set up on host A.

Script outline:

1. Copies a modified version of the Nimbus client `cloud-client.sh` script and the additional files used by that version to host A (see Appendix E for details on the modifications to that script).

5.2.2.5.5 list-nimbus-client-nodes

Description: Lists the hosts belonging to the test bed with the Nimbus client software set up on them.

Prerequisites: At least one host in the test bed has the Nimbus client software set up on it.

Script outline:

1. Prints the `testbed_hosts` file and filters the results to display the hosts with the Nimbus client software set up on them.

5.2.2.6 manage-phantom-vm core scripts

5.2.2.6.1 setup-for-phantom-vm

Description: Configures a host, host A, to be ready to set up a VM running Phantom in the Nimbus cloud in the test bed.

Prerequisites:

- Host A is part of the test bed.

- The Nimbus client software is set up on host A.
- The test bed contains a host set up as the Nimbus service node.
- The test bed contains at least one host set up as a Nimbus VMM node.
- No other host is configured to set up a VM running Phantom.

Script outline:

1. Creates a folder for Phantom on host A.
2. Installs the packages required to set up the Phantom VM from host A, and reboots host A if necessary.
3. Downloads the files necessary to set up the Phantom VM to host A.
4. Performs an operation to work around a bug in Java, then reboots host A.
5. Adds the VM image on which Phantom will be installed to the Nimbus cloud image repository.
6. Creates a directory on host A for the Nimbus access keys for the client associated with host A.
7. Sets host A as the one setting up the Phantom VM in the `testbed_hosts` file.

5.2.2.6.3 remove-phantom-vm-setup

Description: Removes the configuration from the host, host A, which is ready to set up a VM running Phantom in the Nimbus cloud in the test bed.

Prerequisites: The test bed contains a host configured to set up a VM running Phantom.

Script outline:

1. Removes the Phantom setup directories from host A.
2. Removes the image associated with Phantom from the Nimbus cloud image repository.
3. If user approves, uninstalls the packages required to set up the Phantom VM from host A, and reboots host A if necessary.

4. Unsets host A as the host setting up the Phantom VM in the `testbed_hosts` file.

5.2.2.6.2 create-phantom-vm

Description: Uses the host, host A, which is configured to set up the Phantom VM in the Nimbus cloud in the test bed, to create the Phantom VM.

Prerequisites:

- Host A is configured to set up the Phantom VM.
- The test bed contains a host that is set up as the Nimbus service node.
- The test bed contains at least one host that is set up as a Nimbus VMM node.
- A DNS server is specified in the main configuration file.

Script outline:

1. Runs the *setup-adhoc-gateway* script for all VMM nodes in the test bed.
2. Copies the Phantom credentials file to host A.
3. Creates a python virtual environment for the Phantom setup on host A and installs `cloudinitsd` in it.
4. Runs a script on host A that registers a key pair for Phantom to use with the Nimbus cloud.
5. Prepares the image on which Phantom will be installed by:
    a. Creating a VM instance of the image in the Nimbus cloud.
    b. Adding the host entry for the VM into the VM's `hosts` file.
    c. Saving the VM into the Nimbus cloud's image repository.
    d. Adding the host entry for the VM into the host's `hosts` file.
6. Modifies the Phantom creation plan on host A to remove the VM image generator part due to an issue with it.
7. Runs cloudinitsd on host A in order to create the Phantom VM.

8. If the Phantom VM is created successfully, then:
    a. Adds modified versions of the files used for adding a user to Phantom and connecting Phantom to clouds to the Phantom VM, and runs them.
    b. Prints the new Phantom user's credentials that can be used to access Phantom.

5.2.2.6.4 setup-adhoc-gateway

Description: Sets up a host, host A, that is specified either as a parameter or in the main configuration file as a gateway node for the test bed's ad hoc network.

Prerequisites: The specified host is part of the test bed.

Script outline:

1. Sets up host A as a gateway for the test bed's ad hoc network by enabling IP forwarding and configuring iptables, if not already done (see Appendix D for details).

5.2.3 Other Test Bed Management Scripts

5.2.3.1 Setup

5.2.3.1.1 setup

Description: Sets up the host, host A, to be the test bed management (master) node.

Prerequisites: The setup script has not already been run on host A.

Script outline:

1. Runs a script to set up password-less `sudo` for the user on host A. This is done by creating a new file for the user in host A's `sudoers.d` directory with the `NOPASSWD` parameter set in the file.
2. Installs the packages required to run the management scripts on host A.
3. Generates an RSA key pair on host A if one does not already exist.

4. Sets up the main configuration file on host A if it does not already exist.
5. Creates the `testbed_hosts` file on host A.
6. Downloads the setup files for OLSRd and Nimbus on host A.

Chapter 6: Demonstration and Observations

This chapter demonstrates the functionality offered by aNTRuM and reports observations made while testing aNTRuM.

6.1 Functionality Offered by aNTRuM

As mentioned in Chapter 1, our main contributions include:

1. a methodology for implementing a cloud test bed where resources are connected via a MANET;
2. automation scripts for easy test bed setup, configuration, and scaling; and
3. a technique to detect the state of a resource on a device in the cloud, in particular, to detect the remaining battery life of mobile devices in the cloud.

The first contribution alludes to the fact that a MANET is the underlying network of aNTRuM. aNTRuM contains a fully functional Nimbus cloud that runs on a MANET, instead of a wired network. (Chapter 4 describes the details of this part of the setup.)

The second contribution refers to the automation scripts (described in Chapter 5) that are executed on the management node of aNTRuM, which allow for easy setup of a Nimbus cloud from scratch, and for easy addition and removal of nodes, as well as easy propagation of configuration changes to the test bed nodes.

The final contribution mentions the detection of the state of resources on the devices in the Nimbus cloud in aNTRuM. aNTRuM uses a script written for this purpose, described in Chapter 4, to accomplish this goal.

Hence, aNTRuM offers all the aforementioned functionalities, and in Chapter 7 we discuss the possibilities offered by such a test bed for future work.

6.2 Observations

The following sections are observations made during the testing of the aNTRuM test bed.

6.2.1 Network Communication Observations

Because aNTRuM is designed to run on a MANET, a myriad of new potential issues is presented with regards to the reliability of using such a network, including:

- An increase in delay in communication and file transfer times: Because wireless networks are generally slower than wired networks, especially those in a cluster, a noticeable increase in the delay of transferring large files (such as VM images) is observable. This is the reason image caching is used on the VMM nodes in aNTRuM (see Chapter 4 for details). This is also why aNTRuM includes a wired network for the purpose of setting up the test bed (see Appendix B for more details).
- An increase in potential security issues: Because wireless networks are less secure than wired networks, attacks such as signal jamming/blocking and packing sniffing may be used against the systems on the network more easily than on a wired network. Also, as discussed in Chapter 4, the setup of aNTRuM to use wireless bridging significantly decreases the security of the systems involved. This is because the cloud software and virtualization software it uses are not designed to support such a network in a secure fashion. As noted in Chapter 1, the purpose of aNTRuM is for experimentation with a cloud where resources are connected by a MANET that employs OLSRd, and not to provide such a design.
- An increase in the complexity of the network: MANETs are inherently more complex than traditional wired networks. Thus, important network services, such as DHCP and DNS, are far more complex to implement in a MANET environment. Although work, such as [19, 20], has been done regarding ways implement both services in a MANET in a robust way, even MANET protocols (e.g., OLSR) are still under research and development. This is why, as mentioned in

Chapter 4, aNTRuM is set up with DHCP servers on each VMM node and with workarounds to make up for the lack of a DNS server on the MANET; for the purpose of allowing the cloud to work properly. But, again, the purpose of aNTRuM is as an experimental test bed meant to function as if such issues were already resolved.

- A decrease in reliability of the cloud: Because nodes of aNTRuM's cloud could, in theory, be mobile devices that are not guaranteed to be available at all times, the need for fault tolerance, such as VM redundancy, is much higher. But with redundancy, the demand on resources is much higher, which in itself could cause issues.

6.2.2 Device Load Observations

One of the purposes of aNTRuM is to allow for the inclusion of mobile devices as nodes of the cloud. However, because of the way aNTRuM was set up (some of the reasons are mentioned in Section 6.2.1), other problems are created, such as:

- An overwhelming storage and resource demand on mobile VMM nodes: Because mobile devices usually have much more limited resources than do non-mobile devices, caching VM images on mobile VMM nodes could quickly fill up storage space. Even just running VMs on mobile devices requires them to have the resources to do so. Also, the need for all the extra software that is required to run VMs on a device cloud potentially quickly drain the available resources, especially the power, of the device.
- The need to transfer VM instances to other VMM nodes due to power constraints: If a mobile VMM node is running low on power, the need may arise to transfer its VMs to other VMM nodes. However, the cost of doing so may be quite high due to the reasons described in Section 6.2.1 and the above paragraph, in which case the use of the mobile VMM may not be merited unless no other resources are available.

Chapter 7: Conclusions and Future Work

This chapter presents our conclusions and potential future work.

7.1 Conclusions

aNTRuM was created with the goal of providing an easily scalable and configurable, power-aware cloud test bed running on a MANET. We created this test bed to be a foundational platform to conduct interesting research, some of which is mentioned in the next section. One of the valuable aspects of aNTRuM is that it is a real hardware-based test bed, rather than a simulation test bed. Conducting experiments on real hardware, rather than or in addition to simulations, can provide insights of greater value and practicality, especially when dealing with networks and cloud infrastructure at a low level.

Another valuable aspect of aNTRuM is that it is, to the best of our knowledge, unique in the way it is set up. Having a cloud run on top of a MANET really deviates from the norm in terms of cloud computing, since generally clouds are thought of as highly reliable systems. As a consequence, a whole new set of challenges is presented with regards to MANET performance, reliability, and security.

An additional research area that is drawing attention in cloud computing is power and energy usage. Since clouds are usually implemented on clusters, cloud computing facilities tend to consume high amounts of energy. As mentioned in Chapter 2, research had been and is being done in the area of energy awareness in cloud computing. However, the nature of our cloud provides a whole new set of challenges in terms of power and energy awareness. Because aNTRuM is composed of compute resources that may not be dedicated to virtualization, there is much less control over power usage aside from that consumed by the VMs themselves. Of far greater concern is the need for power monitoring due to the energy and battery-life constraints of the compute resources.

7.2 Future Work

A few of the many possibilities for future work that are facilitated by aNTRuM are:

- Modification of the Nimbus cloud scheduler to be resource-aware. Since our test bed already provides battery and memory metrics, a modification to the scheduler would be all that is needed to take advantage of these metrics.
- Addition of an ARM-powered device to the cloud. Currently, aNTRuM supports only x86 devices. However, as mentioned in Chapter 3, aNTRuM was designed with mobile devices in mind. As Ubuntu for ARM devices and hardware virtualization support in ARM processors is becoming more main stream, such a goal lies not too far in the future.
- Modification of the cloud scheduler for connectivity awareness. Since aNTRuM is connected via a MANET, the cloud should be aware of when nodes come into and out of range, as well as the link quality of the connection and the number of hops between nodes. OLSRd already provides the information for such detection, thus, modification of the scheduler to take advantage of such information is the next logical step.
- Using Phantom for managing VM resources across multiple aNTRuM-like clouds. Currently, our test bed only consists of a single cloud. Setting up multiple smaller clouds that are like aNTRuM would allow for using Phantom to its full potential and assessing its effectiveness in such a scenario. Phantom's ability to constantly monitor and manage VMs is highly beneficial in a setup such as aNTRuM's.

Bibliography

[1] Nimbus. Accessed December 9, 2014. http://www.nimbusproject.org.

[2] olsrd. Accessed December 9, 2014. http://olsr.org.

[3] Soyata, Tolga, Rajani Muraleedharan, Jonathan Langdon, Colin Funai, Scott Ames, Minseok Kwon, and Wendi Heinzelman. "COMBAT: Mobile-Cloud-based COmpute/coMmunications Infrastructure for BATtlefield Applications." In *Proceedings of SPIE* 8403 (2012): 84030K-13.

[4] Miles, Alan, Yan Bai, Donald Chinn, and Bharat Bhargava. "An Experimental Study of Hybrid Energy-aware Scheduling in a Cloud Testbed." In *Global Information Infrastructure and Networking Symposium (GIIS), 2014*, 1-6. IEEE, 2014.

[5] Grossman, Robert, Yunhong Gu, Michal Sabala, Collin Bennett, Jonathan Seidman, and Joe Mambretti. "The Open Cloud Testbed: A Wide Area Testbed for Cloud Computing Utilizing High Performance Network Services." (2009). http://arxiv.org/abs/0907.4810.

[6] Avetisyan, Arutyun, Roy Campbell, Indranil Gupta, Michael Heath, Steven Ko, Gregory Ganger, Michael Kozuch, David O'Hallaron, Marcel Kunze, Thomas Kwan, Kevin Lai, Martha Lyons, Dejan Milojicic, Hing Yan Lee, Yeng Chai Soh, Ng Kwang Ming, Jing-Yuan Luke, and Han Namgoong. "Open Cirrus: A Global Cloud Computing Testbed." *Computer* 43, no. 4 (2010): 35-43.

[7] Emulab. Accessed December 9, 2014. https://www.emulab.net.

[8] CloudLab. Accessed December 9, 2014. http://www.cloudlab.us.

[9] Chameleon Cloud. Accessed December 9, 2014. http://www.chameleoncloud.org.

[10] FutureGrid. Accessed December 9, 2014. https://portal.futuregrid.org.

[11] Von Laszewsk, Gregor, Javier Diaz, Fugang Wang, and Geoffrey Fox. "Comparison of Multiple Cloud Frameworks." In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, 734-741. IEEE, 2012.

[12] XEN Project. Accessed December 9, 2014. http://www.xenproject.org.

[13] KVM. Accessed December 9, 2014. http://www.linux-kvm.org.

[14] Nimbus Phantom Documentation. Accessed December 9, 2014. http://www.nimbusproject.org/doc/phantom/latest/.

[15] OpenTSDB. Accessed December 9, 2014. http://opentsdb.net.

[16] Ubuntu. Accessed December 9, 2014. http://www.ubuntu.com.

[17] Keahey, Kate, Patrick Armstrong, John Bresnahan, David LaBissoniere, and Pierre Riteau. "Infrastructure Outsourcing in Multi-cloud Environment." In *Proceedings of the 2012 Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit*, 33-38. ACM, 2012.

[18] Duplyakin, Dmitry, Paul Marshall, Kate Keahey, Henry Tufo, and Ali Alzabarah. "Rebalancing in a Multi-cloud Environment." In *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*, 21-28. ACM, 2013.

[19] Hsu, Yuan-Ying, and Chien-Chao Tseng. "Prime DHCP: A Prime Numbering Address Allocation Mechanism for MANETs." In *IEEE Communications Letters*, 712-714. Vol. 9. No. 8. IEEE, 2005.

[20] Ahn, Sanghyun, and Yujin Lim. "A Modified Centralized DNS Approach for the Dynamic MANET Environment." In *9th International Symposium on Communications and Information Technology, 2009*, 1506-1510. IEEE, 2009.

Appendix A: Abbreviations

Abbreviation	Definition
aNTRuM	A Nimbus Test bed Running on a MANET
ARP	Address Resolution Protocol
IaaS	Infrastructure-as-a-Service
MANET	Mobile Ad Hoc Network
OLSR	Optimized Link State Routing Protocol
OLSRd	OLSR daemon
OS	Operating System
VM	Virtual Machine
VMM	Virtual Machine Manager

Appendix B: Test Bed Details

This appendix describes the details of the software that was used in the aNTRuM test bed, as well as its network layout.

B.1 Software Versions

The following table lists the versions of the main software used in aNTRuM.

Software	Version
Ubuntu	14.04.1
OLSRd	0.7.7.1
Nimbus IaaS	2.10.1
Nimbus Cloud Client	022

B.2 Network Layout

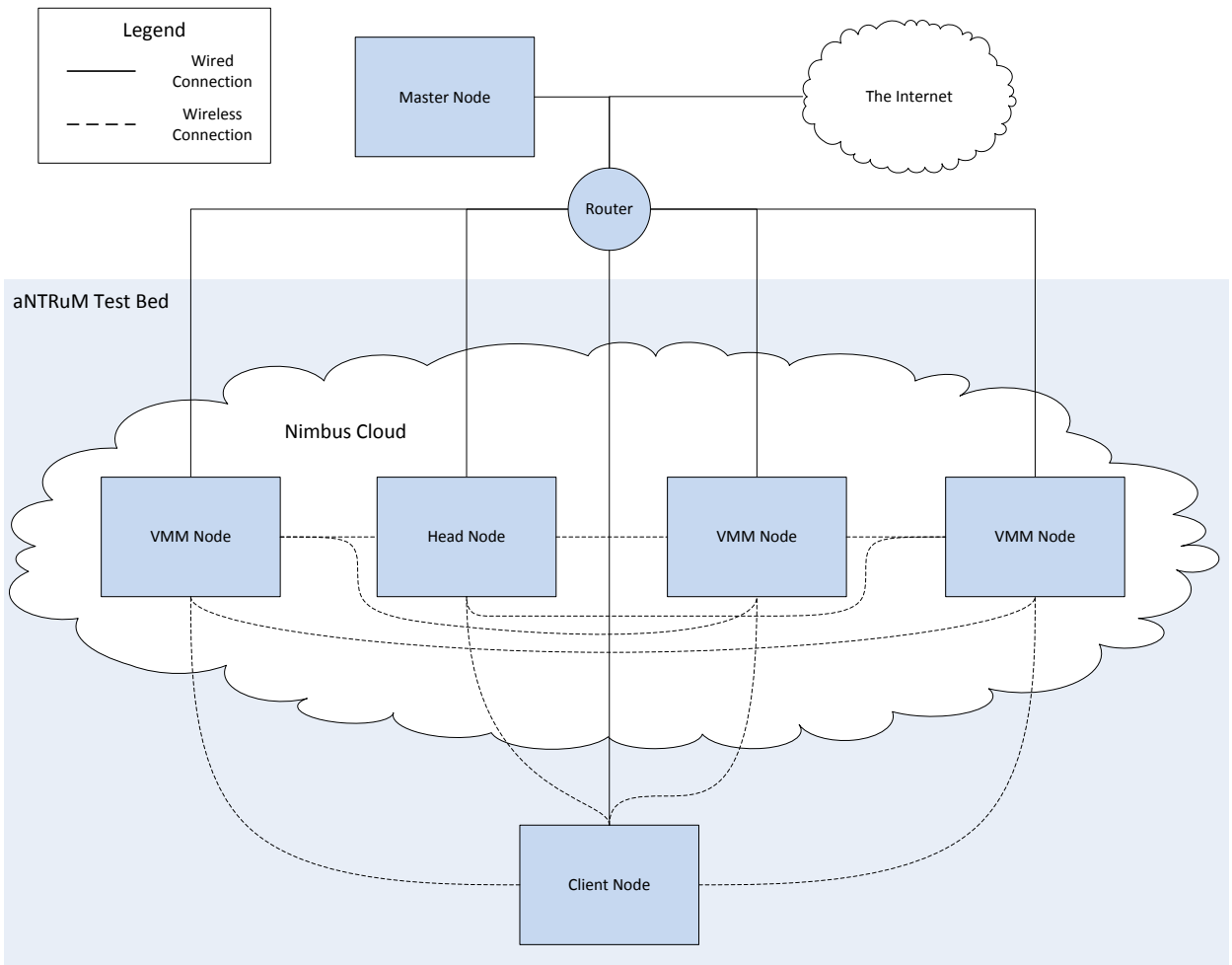

Figure B.1: Network layout of aNTRuM

Figure B.1 presents the network layout of aNTRuM. aNTRuM uses both of the following network structures:

- All test bed nodes are connected to a router, which has full connectivity to the Internet. This network allows the master node to run the management scripts for all the test bed nodes, and it allows the nodes to have access to the Internet for the purpose of downloading the necessary software for set up.
- All test bed nodes (minus the master node) also have Linux kernel-compatible wireless cards (a TP-LINK TL-WN722N N150 High Gain Wireless USB Adapter was used for each device in aNTRuM), which are set up to run a MANET with OLSRd managing it. This network is used by the cloud software running in the test bed. Although the nodes themselves have Internet connectivity via the wired network, the VMs running in the test bed do not, unless one of the nodes is set up as a gateway.

Appendix C: Test Bed Set Up Tutorial

The following sections describe how to set up a test bed similar to aNTRuM using the aNTRuM test bed management scripts.

NOTE: These scripts are meant to be used for creating a test bed. It is recommended that dedicated machines on a private network be used for this purpose since the aNTRuM scripts will modify system files and will enable password-less super-user permissions. Some of the modifications made by the scripts may introduce vulnerabilities to the system. Also, the scripts are written with workarounds for bugs/issues in the version of Linux used in aNTRuM that may not be present on other operating systems or OS versions, so they may not function properly on these.

C.1 Hardware Requirements

The network setup described in Appendix B is recommended. Two or more machines are required: one to function as the master node and the rest to function as the nodes of the test bed. Additionally, all machines must have:

- An administrator account with the same username as that of the other test bed nodes (except for the master node). The default username in the main configuration file is "nimbus".
- Static IP addresses.

Any machine that will function as a virtual machine manager (VMM) node will need:

- A processor that supports hardware virtualization. Hardware virtualization must be enabled in the BIOS as well.
- A 64-bit processor and OS. A 32-bit OS may function but will limit the VM capacity.

C.2 Software Requirements

SSH server must be installed on all test bed nodes. A compatible operating system is required for the scripts to work. The management scripts were designed to work with the particular versions of Ubuntu, OLSRd, and Nimbus described in Appendix B. They include workarounds for certain bugs and issues specific to these versions, thus, changing the version may or may not break the scripts.

C.3 Setting up the Master Node

The aNTRuM test bed management scripts can be found at: https://github.com/hipersys/antrum. Use the `git clone` command on the machine that will serve as the master node to retrieve them. This node will manage the test bed nodes via the management scripts.

Step 1 - Create the configuration file, an example of which can be found in `$ANTRUM_DIR/management/etc/`: The configuration file must be named `main.conf`. *The* easiest way to do this is to make a copy of the example configuration file. Note that for many of the configuration parameters, especially those pertaining to file/folder paths, changes shouldn't be made

once that part of the test bed has been set up, otherwise the scripts will not function properly. If the VMs are to have connectivity to the Internet, a DNS server will need to be specified.

Step 2 - Run the setup script (`$ANTRUM_DIR/management/setup/setup`), which will set up the master node, as well as download the files necessary for setting up OLSRd, Nimbus, and Phantom.

C.4 Setting up the Test Bed Nodes

The management scripts are contained in `$ANTRUM_DIR/management/bin/`. For details on each script and their functionalities see Chapter 5. See Appendix D for the scripts as they appear in the repository. Running the script with the `-h` parameter will show the available commands for that script.

C.4.1 Adding Hosts

Step 3 - Add a node to test bed: `manage-hosts` is for managing machines that are part of the test bed. To add a node to the test bed, run `./manage-hosts -a <host_ip>`. The script prompts the user for a password twice; once for the initial SSH connection to the host, and the other for running a `sudo` command. The script sets up the authentication keys and password-less `sudo`, therefore, the password will not be requested after that. It also adds the hostname to the list of test bed hosts, which is used by the rest of the management scripts. This step is necessary for each machine that is to be part of the test bed.

To list all the nodes in the test bed and their roles, run `./manage-hosts -l`. To remove a node from the test bed, run `./manage-hosts -r <host_name>` (or `./manage-hosts -f <host_name>` if the host is no longer connected to the rest of the test bed).

NOTE: Because the test bed is set up to run on a MANET, there is no DNS server for the hosts and VMs in the test bed. Because of this, the scripts are set up to modify the `hosts` file of each individual test bed node as necessary. In this way, hostnames can be used as in a traditional network setup.

C.4.2 Setting up OLSRd

Step 4 - Add OLSRd to a host: `manage-olsrd-hosts` is for managing OLSRd on the test bed nodes. To add OLSRd to a host, run `./manage-olsrd-hosts -a <host_name>`. The wireless card on the machine is set up to run in ad-hoc mode with an appropriate IP address by the script. OLSRd is also set up and run on the host, and it is set up to run on startup; in this way, it always will be running. Note that OLSRd is expected to be running on all machines that are part of the test bed, thus, this step is also necessary. This is because all communication within the cloud infrastructure will be via the MANET on which OLSRd runs.

The files in `$ANTRUM_DIR/management/dist/olsrd/` can be modified to change how OLSRd runs. Run `./manage-olsrd-hosts -u` to propagate changes to all the test bed nodes. To remove OLSRd from a host, run `./manage-olsrd-hosts -r <host_name>`.

More information regarding OLSRd can be found at http://olsr.org/.

NOTE: Due to each host having two IP addresses, one for connectivity to the master node and the Internet, and the other for the MANET that the test bed uses for connectivity, the hosts files on all the test bed nodes are set up to have the wireless IP address paired with the hostname of each machine, and for the other IP address to be associated with the hostname with a string ("x" by default) tagged on to the end.

C.4.3 Setting up Nimbus

The Nimbus infrastructure is the IaaS cloud used on the aNTRuM test bed. For more information regarding Nimbus visit http://www.nimbusproject.org/.

Note that all of the following can be set up on one or more nodes, but there can only be one service (head) node.

C.4.3.1 Setting up the Service Node

Step 5 - Set up the service node: `manage-nimbus-head-node` is for managing the host that will serve as the Nimbus service node (head node). Run `./manage-nimbus-head-node -s <host_name>` to set up a host as the service node. The script sets up the host to run the Nimbus IaaS software, which includes the cloud resource provisioner and which interfaces with clients (see Chapter 3 for more details).

The configuration files for the service node can be found in `$ANTRUM_DIR/management/dist/nimbus/head/`. To propagate configuration changes to the head node once it is set up, use the `./manage-nimbus-head-node -u` command. To remove the Nimbus service node software from the head node, run `./manage-nimbus-head-node -r`.

Once the service node is set up, cloud administrator features can be found on the service node in `$HOME/nimbus/bin/` (unless the default install location has been changed). For example, `nimbus-admin` is for managing VMs, `nimbus-nodes` is for managing VMM nodes, and `nimbusctl` is for managing the Nimbus and Cumulus services. For further information, the Nimbus cloud administrator guide can be found at http://www.nimbusproject.org/docs/2.10.1/admin/index.html.

C.4.3.2 Setting up the VMM nodes

Step 6 - Set up VMM nodes: `manage-nimbus-vmm-nodes` is for managing the hosts that will function as the Nimbus virtual machine manager (VMM) nodes. Running `./manage-nimbus-vmm-nodes -a <host_name>` sets up the specified host to serve as a VMM node. The Nimbus cloud control software is installed on the host, as well as KVM, libvirt, and DHCP server (since there is no central DHCP server in the setup), to name a few. The host is also set up to use wireless bridging for VM connectivity. Furthermore, the host will be automatically connected to the Nimbus head node.

Note that the setup script additionally sets up a script on the host that collects information regarding certain system resources and outputs them to a file (this script is called `output-resource-status.py` and can be found in `$ANTRUM_DIR/management/dist/nimbus/vmm/`). This information is available for future incorporation into the cloud scheduler on the service node.

The configuration files for the service nodes can be found in `$ANTRUM_DIR/management/dist/nimbus/vmm/`. Running `./manage-nimbus-vmm-nodes -u` will propagate configuration changes to those files to all VMM nodes that are set up. To remove the VMM software from a test bed node, run `./manage-nimbus-vmm-nodes -r <host_name>`.

C.4.3.3 Setting up the Client Nodes

Step 7 - Set up client nodes: `manage-nimbus-client-nodes` is for managing the hosts that will run the Nimbus client software. The Nimbus client software is set up on a host by running `./manage-nimbus-client-nodes -a <host_name>`. The client is automatically connected to the service node by the script. The client software can be set up on as many hosts as desired (each will function as a separate client). Although it is possible to set up more than one client on a single host, the scripts are only set up to handle one client per host.

To remove the Nimbus client software from a host, run `./manage-nimbus-client-nodes -r <host_name>`.

Once the cloud client software is set up, the client can be used with the cloud. The client is called `cloud-client.sh` and can be found in `$HOME/nimbus/nimbus-cloud-client-022/bin/` (unless the default install location or client version has been changed). Running `./cloud-client.sh -h` will cause a list of available commands to be printed. These include running a VM and transferring a new VM image. An advanced user reference for the Nimbus client software can be found at http://www.nimbusproject.org/docs/2.10.1/clouds/appendix.html.

C.4.4 Testing the setup

To ensure that everything is properly set up, the following actions should be taken on a client node:

1. Add a VM image to the cloud's image repository using the command `$HOME/nimbus/nimbus-cloud-client-022/bin/cloud-client.sh --transfer --sourcefile <image_location>`. For the sake of the test, the VM image file used for setting up Phantom can be used; it can be found on the master node at `$ANTRUM_DIR/management/dist/phantom/phantom-ubuntu.gz`. To learn how to generate VM images, see http://scienceclouds.org/ecosystem/generation-of-virtual-machine-images/ (note that the VM images must be compatible with KVM).
2. Create a VM on the cloud using the command `$HOME/nimbus/nimbus-cloud-client-022/bin/bin/cloud-client.sh --run --name <image_name> --hours 1`. Because of the network structure of this setup, the transfer of the image may take a while, but this is normal.
3. Wait about a minute after the VM is created for it to boot up, then try to SSH into the VM using `ssh root@<vm_name>` or `ssh root@<vm_ip>`.
4. If everything works without error, the setup is working properly. To terminate the VM that was created, use `$HOME/nimbus/nimbus-cloud-client-022/bin/cloud-client.sh --terminate --handle <vm-handle>`.

NOTE: As mentioned previously, for VMs to have connectivity to the Internet, a DNS server must be specified in the main configuration file on the master node. Additionally, a host must be set up as a gateway node. A script is available in the `sbin` folder called `setup-adhoc-gateway`, which can be used to set up a host as the gateway (the host that will function as the gateway is also specified in the main configuration file). Note that the VMMs that the VMs are running on must also be configured to route to this gateway, or the VMs will not be able to reach it.

Note also that aNTRuM was set up and tested using particular versions of software. Because of this, using other versions of the core or prerequisite software could cause issues. Steps 1-7 were last tested in their entirety in December 2014. Any changes made after this test were tested enough to ensure they worked properly. The first part of step 8 was last tested in March 2015, but, because of technical issues, the rest of step 8 was last tested in December 2014. Thus, step 8 has not been tested in its entirety. The following are issues that presented themselves during testing:

1. Ad hoc connectivity issue: Sometimes, when one of the test bed nodes was restarted, it was assigned a cell number (for the ad hoc connection) that was different than that of the rest of the test bed nodes and, therefore, it could not communicate with them via the ad hoc network. This problem can be fixed by running `./manage-olsrd-hosts -u`, which will reset the ad hoc connection on all the nodes in the test bed, causing all of them to have the same cell number.
2. Phantom VM setup issues: Various errors can occur when `cloudinit.d` is run to set up Phantom on a VM. Sometimes these errors are not consistent, making their cause difficult to pinpoint. Hence, this portion of the setup can be difficult and unreliable. These errors are the reason that step 8 was not tested in its entirety.

C.4.5 Setting up Phantom (Optional)

Step 8 - Set up Nimbus Phantom: `manage-phantom-vm` is used to set up a host with the Nimbus client software on it and to set up and run Phantom. Before this, a VM image must be created and placed in the location specified in the main configuration file. This image should be a KVM compatible Ubuntu 13.10 server VM image with Git and Chef installed on it. The host is set up to run cloudinit.d, which launches and sets up a VM to run Phantom, by running `./manage-phantom-vm -s <host_name>`. Next, `./manage-phantom-vm -c` is executed to run cloudinit.d. This creates a VM that is running Phantom, and the user is added to Phantom. If an error occurs while running the launch plan, the cloudinit.d logs, located at `$HOME/.cloudinitd/<run_value>/`, can be checked. If the error is an issue with VM connectivity, ensure that the VM has access to the Internet. If not, try running `./manage-phantom-vm -c` again, (unless the error occurs on level 9 of the launch plan, in which case everything should work). Note that when an error occurs, the VM for Phantom will be left running, and can to be terminated via `nimbus-admin` on the service node (see Section C.4.3.1).

To remove the Phantom setup, run `./manage-phantom-vm -r` (this will not terminate the Phantom VM if it is already running).

Even though Phantom is open-source software, it is currently not released as software, but rather is available as a service. As such, its functionality as provided by the setup script is limited in the following ways:

- A limited run time: The Phantom VM will run as long as allowed by the test bed cloud's default VM run time limit configuration. For the test bed cloud, this limit has been changed to one month. To adjust this (and other EC2 interface settings), modify the appropriate configuration files in `$ANTRUM_DIR/management/dist/nimbus/head/elastic/` on the master node.
- A lack of persistence: The Phantom launch configuration, which provides persistence to the data in Phantom, did not work in the setup. Therefore, the launch plan that is used does not provide this persistence. Hence, the Phantom service cannot be restarted, or the data will be lost.
- A lack of VM image generation: Although Phantom does provide image generators, there were issues installing that portion of Phantom in the test bed, thus, this feature is not available in the test bed.
- A lack of hostname resolution for VMs created using Phantom: As mentioned above, there is no DNS server in the test bed, and a workaround has not been implemented for the test bed cloud's EC2 interface as it was for the client software. Therefore, VMs created using Phantom must be accessed using their IP addresses.
- A lack of SSH access into the VMs created by Phantom.

To check if Phantom is running properly, try accessing Phantom through its web interface. From a browser on a host in the test bed (excluding the master node), go to `http://<phantom_vm_ip>`, and attempt to log in with the credentials printed when `./manage-phantom-vm -c` completes running. Upon logging in, notice that multiple clouds are listed, along with aNTRuM, as available to use with Phantom. This is because, as mentioned previously, Phantom does not have a formal release, and so all the other clouds are added during its setup. However, they cannot be used without the proper credentials for them.

Phantom also can be accessed programmatically through its autoscale and HTTP APIs. For information on using those interfaces, on collecting metrics from VMs using Phantom, and on building a decision engine to use with Phantom, see http://www.nimbusproject.org/doc/phantom/latest/advanced.html. Note that this documentation refers to Phantom service available as part of the Nimbus project, thus, the URLs must be changed to those of the Phantom VM in the test bed.

More information regarding Phantom can be found at http://www.nimbusproject.org/phantom. More information regarding cloudinit.d (a Nimbus platform tool) can be found at http://www.nimbusproject.org/doc/cloudinitd/latest/.

C.4.6 Beyond aNTRuM

Nimbus also provides a "Context Broker" for coordinating large groups of VMs. Although this software is beyond the scope of aNTRuM, it is potentially useful for future work. For more information, visit http://www.nimbusproject.org/docs/current/faq.html#ctxbroker.

Appendix D: Test Bed Management Scripts

This Appendix provides the aNTRuM management scripts found at https://github.com/hipersys/antrum.

D.1 Primary Scripts

The following are the primary scripts, contained in `antrum/management/bin/`.

D.1.1 manage-hosts

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"

show_help() {
    echo -e " General:"
    echo -e "\t-h, --help\t\t\tDisplay this help information"
    echo -e " Actions:"
    echo -e "\t-a, --add [host ip address]\tAdd a host"
    echo -e "\t-l, --list\t\t\tList current hosts"
    echo -e "\t-r, --remove [hostname]\t\tRemove a host"
    echo -e "\t-f, --force-remove [hostname]\tRemove a host without
removing files from host"
    echo -e "\t-u, --update\t\t\tUpdate all hosts"
    exit 1
}

ARGS=$(getopt -o a:lr:f:uh -l add:,list,remove:,force-
remove:,update,help -n $0 -- $@)

if [ $? -ne 0 ]; then
    exit 1
fi

eval set -- $ARGS

while true; do
    case $1 in
      -a|--add)
          /bin/bash $TESTBED_MGT_DIR/sbin/add-host $2
                exit 0
          ;;
      -l|--list)
          /bin/bash $TESTBED_MGT_DIR/sbin/list-hosts
          exit 0
          ;;
      -r|--remove)
          /bin/bash $TESTBED_MGT_DIR/sbin/remove-host $2
```

```
            exit 0
            ;;
        -f|--force-remove)
            /bin/bash $TESTBED_MGT_DIR/sbin/remove-host $2 "-f"
            exit 0
            ;;
        -u|--update)
            /bin/bash $TESTBED_MGT_DIR/sbin/update-hosts
            exit 0
            ;;
        -h|--help)
            show_help
            exit 0
            ;;
        *)
            echo "No action specified (--help for help)"
            exit 1
            ;;
    esac
done
```

D.1.2 manage-nimbus-client-nodes

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"

show_help() {
    echo -e " General:"
    echo -e "\t-h, --help\t\t\tDisplay this help information"
    echo -e " Actions:"
    echo -e "\t-a, --add [hostname]\t\tAdd a Nimbus client node"
    echo -e "\t-l, --list\t\t\tList current Nimbus client nodes"
    echo -e "\t-r, --remove [hostname]\t\tRemove a Nimbus client node"
    echo -e "\t-u, --update\t\t\tUpdate the Nimbus client nodes"
    exit 1
}

ARGS=$(getopt -o a:lr:uh -l add:,list,remove:,update,help -n $0 -- $@)

if [ $? -ne 0 ]; then
    exit 1
fi

eval set -- $ARGS

while true; do
```

```
    case $1 in
      -a|--add)
          /bin/bash $TESTBED_MGT_DIR/sbin/add-nimbus-client-node $2
                exit 0
          ;;
      -l|--list)
          /bin/bash $TESTBED_MGT_DIR/sbin/list-nimbus-client-nodes
          exit 0
          ;;
      -r|--remove)
          /bin/bash $TESTBED_MGT_DIR/sbin/remove-nimbus-client-node $2
          exit 0
          ;;
      -u|--update)
          /bin/bash $TESTBED_MGT_DIR/sbin/update-nimbus-client-nodes
          exit 0
          ;;
      -h|--help)
          show_help
          exit 0
          ;;
      *)
          echo "No action specified (--help for help)"
          exit 1
          ;;
    esac
done
```

D.1.3 manage-nimbus-head-node

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"

show_help() {
    echo -e " General:"
    echo -e "\t-h, --help\t\t\tDisplay this help information"
    echo -e " Actions:"
    echo -e "\t-s, --setup [hostname]\t\tSetup the Nimbus head node"
    echo -e "\t-l, --list\t\t\tList current Nimbus head node"
    echo -e "\t-r, --remove\t\t\tRemove the Nimbus head node"
    echo -e "\t-u, --update\t\t\tUpdate the Nimbus head node"
    exit 1
}

ARGS=$(getopt -o s:lruh -l setup:,list,remove,update,help -n $0 -- $@)
```

```
if [ $? -ne 0 ]; then
    exit 1
fi

eval set -- $ARGS

while true; do
    case $1 in
      -s|--setup)
          /bin/bash $TESTBED_MGT_DIR/sbin/setup-nimbus-head-node $2
               exit 0
          ;;
      -l|--list)
          /bin/bash $TESTBED_MGT_DIR/sbin/list-nimbus-head-node
          exit 0
          ;;
      -r|--remove)
          /bin/bash $TESTBED_MGT_DIR/sbin/remove-nimbus-head-node
          exit 0
          ;;
      -u|--update)
          /bin/bash $TESTBED_MGT_DIR/sbin/update-nimbus-head-node
          exit 0
          ;;
      -h|--help)
          show_help
          exit 0
          ;;
      *)
          echo "No action specified (--help for help)"
          exit 1
          ;;
    esac
done
```

D.1.4 manage-nimbus-vmm-nodes

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"

show_help() {
    echo -e " General:"
    echo -e "\t-h, --help\t\t\tDisplay this help information"
    echo -e " Actions:"
    echo -e "\t-a, --add [hostname]\t\tAdd a Nimbus VMM node"
    echo -e "\t-l, --list\t\t\tList current Nimbus VMM nodes"
```

```
    echo -e "\t-r, --remove [hostname]\t\tRemove a Nimbus VMM node"
    echo -e "\t-u, --update\t\t\tUpdate the Nimbus VMM nodes"
    echo -e " Nimbus:"
    echo -e "\t-c, --connect [hostname]\tConnect a Nimbus VMM node to
the Nimbus head node"
    echo -e "\t-d, --disconnect [hostname]\tDisconnect a Nimbus VMM
node from the Nimbus head node"
    echo -e "\t-i, --is-connected [hostname]\tCheck if a Nimbus VMM
node is connected to the Nimbus head node"
    exit 1
}

ARGS=$(getopt -o a:lr:uc:d:i:h -l
add:,list,remove:,update,connect:,disconnect:,is-connected:,help -n $0
-- $@)

if [ $? -ne 0 ]; then
    exit 1
fi

eval set -- $ARGS

while true; do
    case $1 in
      -a|--add)
          /bin/bash $TESTBED_MGT_DIR/sbin/add-nimbus-vmm-node $2
               exit 0
          ;;
      -l|--list)
          /bin/bash $TESTBED_MGT_DIR/sbin/list-nimbus-vmm-nodes
          exit 0
          ;;
      -r|--remove)
          /bin/bash $TESTBED_MGT_DIR/sbin/remove-nimbus-vmm-node $2
          exit 0
          ;;
      -u|--update)
          /bin/bash $TESTBED_MGT_DIR/sbin/update-nimbus-vmm-nodes
          exit 0
          ;;
      -c|--connect)
          /bin/bash $TESTBED_MGT_DIR/sbin/connect-nimbus-vmm-to-head $2
          exit 0
          ;;
      -d|--disconnect)
          /bin/bash $TESTBED_MGT_DIR/sbin/disconnect-nimbus-vmm-from-
head $2
          exit 0
          ;;
      -i|--is-connected)
          /bin/bash $TESTBED_MGT_DIR/sbin/is-nimbus-vmm-connected-to-
head $2
```

```
            exit 0
            ;;
        -h|--help)
            show_help
            exit 0
            ;;
        *)
            echo "No action specified (--help for help)"
            exit 1
            ;;
    esac
done
```

D.1.5 manage-olsrd-hosts

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"

show_help() {
    echo -e " General:"
    echo -e "\t-h, --help\t\t\tDisplay this help information"
    echo -e " Actions:"
    echo -e "\t-a, --add [hostname]\t\tAdd an OLSRd host"
    echo -e "\t-l, --list\t\t\tList current OLSRd hosts"
    echo -e "\t-r, --remove [hostname]\t\tRemove an OLSRd host"
    echo -e "\t-u, --update\t\t\tUpdate all OLSRd hosts"
    exit 1
}

ARGS=$(getopt -o a:lr:uh -l add:,list,remove:,update,help -n $0 -- $@)

if [ $? -ne 0 ]; then
    exit 1
fi

eval set -- $ARGS

while true; do
    case $1 in
      -a|--add)
          /bin/bash $TESTBED_MGT_DIR/sbin/add-olsrd-host $2
                exit 0
          ;;
      -l|--list)
          /bin/bash $TESTBED_MGT_DIR/sbin/list-olsrd-hosts
          exit 0
```

```
            ;;
        -r|--remove)
            /bin/bash $TESTBED_MGT_DIR/sbin/remove-olsrd-host $2
            exit 0
            ;;
        -u|--update)
            /bin/bash $TESTBED_MGT_DIR/sbin/update-olsrd-hosts
            exit 0
            ;;
        -h|--help)
            show_help
            exit 0
            ;;
        *)
            echo "No action specified (--help for help)"
            exit 1
            ;;
    esac
done
```

D.1.6 manage-phantom-vm

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"

show_help() {
    echo -e " General:"
    echo -e "\t-h, --help\t\t\tDisplay this help information"
    echo -e " Actions:"
    echo -e "\t-s, --setup [hostname]\t\tSetup to create the Phantom
VM using the specified host"
    echo -e "\t-c, --create\t\t\tCreate the Phantom VM"
    echo -e "\t-r, --remove\t\t\tRemove the Phantom VM setup"
    exit 1
}

ARGS=$(getopt -o s:crh -l setup:,create,remove,help -n $0 -- $@)

if [ $? -ne 0 ]; then
    exit 1
fi

eval set -- $ARGS

while true; do
    case $1 in
```

```
    -s|--setup)
        /bin/bash $TESTBED_MGT_DIR/sbin/setup-for-phantom-vm $2
             exit 0
        ;;
    -c|--create)
        /bin/bash $TESTBED_MGT_DIR/sbin/create-phantom-vm
        exit 0
        ;;
    -r|--remove)
        /bin/bash $TESTBED_MGT_DIR/sbin/remove-phantom-vm-setup
        exit 0
        ;;
    -h|--help)
        show_help
        exit 0
        ;;
    *)
        echo "No action specified (--help for help)"
        exit 1
        ;;
  esac
done
```

D.2 Core Scripts

The following are the core scripts, contained in `antrum/management/sbin/`.

D.2.1 add-host

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

NEW_HOST_IP=$1

if ! is_valid_ip $NEW_HOST_IP; then
    echo "Invalid IP address"
    exit 1
fi

if is_testbed_host $NEW_HOST_IP; then
    echo "Host already added"
    exit 1
fi
```

```
echo "adding public key to authorized keys of new host..."
cat $HOME/.ssh/id_rsa.pub | ssh $TB_HOSTS_USERNAME@$NEW_HOST_IP \
    "[ -d \$HOME/.ssh ] || mkdir \$HOME/.ssh
     [ -f \$HOME/.ssh/authorized_keys ] || touch
\$HOME/.ssh/authorized_keys
     sed -i \"/^.*$(echo $USER)@$(hostname)\$/d\"
\$HOME/.ssh/authorized_keys;
     cat - >> \$HOME/.ssh/authorized_keys;"

echo "modifying sudoers file for passwordless sudo..."
scp -q $TB_MGT_DIR/dist/set-pwdless-sudo
$TB_HOSTS_USERNAME@$NEW_HOST_IP:/tmp/sps
ssh -t $TB_HOSTS_USERNAME@$NEW_HOST_IP "sudo /tmp/sps
$TB_HOSTS_USERNAME"

echo "adding new host to hosts lists..."
NEW_HOST_NAME=`ssh $TB_HOSTS_USERNAME@$NEW_HOST_IP "hostname"`

if is_testbed_host $NEW_HOST_NAME; then
    echo "Host already added"
    exit 1
fi

echo "updating host files..."
add_testbed_host $NEW_HOST_NAME
set_tb_host_wired_ip $NEW_HOST_NAME $NEW_HOST_IP
grep -Fqw "$NEW_HOST_IP$TB_HOSTS_WIRED_IP_APPEND" /etc/hosts || \
    sudo sed -i "/testbed wired network/a
$NEW_HOST_IP\t$NEW_HOST_NAME$TB_HOSTS_WIRED_IP_APPEND" /etc/hosts

echo "cleaning up..."
ssh $TB_HOSTS_USERNAME@$NEW_HOST_IP "rm -f /tmp/sps"

echo "new host \"$NEW_HOST_NAME\" added to testbed"

echo "updating hosts..."
$BASH_EXE $TB_MGT_DIR/sbin/update-hosts

exit 0
```

D.2.2 add-nimbus-client-node

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
```

```
source $TB_BASH_FUNCTIONS_LIB_FILE

HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

if ! tb_host_has_olsr_ip $HOST_NAME; then
    echo "OLSRd not set up on the host"
    exit 1
fi

if ! tb_host_head_node_exists; then
    echo "No head node found in testbed"
    exit 1
fi

if tb_host_is_client_node $HOST_NAME; then
    echo "Nimbus client already set up on host"
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

HEAD_NODE_IP=$(get_tb_host_wired_ip $(get_testbed_head_node))

echo "copying nimbus client files..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "mkdir -p $TB_HOSTS_NIMBUS_DIR
     mkdir -p $TB_HOSTS_HOME_DIR/.nimbus"
scp -q $TB_NIMBUS_CLIENT_TAR_FILE
$TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_NIMBUS_CLIENT_DIR.tar.gz

echo "installing dependencies..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "$TB_PKG_MGR_UPDATE_CMD
     $TB_PKG_MGR_INSTALL_CMD openjdk-6-jre" &> /dev/null
reboot_and_wait_if_needed $HOST_NAME

echo "setting up rsa keys on host..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "[ -f \$HOME/.ssh/id_rsa ] || ssh-keygen -f \$HOME/.ssh/id_rsa -N
'' " > /dev/null

echo "creating new user on head node..."
tmp_newuser_dir=`mktemp -d /tmp/tbmancn.XXXXXXX`
ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP \
    "$TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbus-new-user -d $tmp_newuser_dir
$HOST_NAME@$TB_NIMBUS_CLIENT_DOMAIN" \
```

```
    > /dev/null

echo "setting up nimbus client..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "tar -xf $TB_HOSTS_NIMBUS_CLIENT_DIR.tar.gz -C
$TB_HOSTS_NIMBUS_DIR"

echo "copying cloud files..."
scp -q $TB_HOSTS_USERNAME@$HEAD_NODE_IP:$tmp_newuser_dir/*
$tmp_newuser_dir/
scp -q $tmp_newuser_dir/cloud.properties \
    $TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_NIMBUS_CLIENT_DIR/conf/
scp -q $tmp_newuser_dir/*.pem
$TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_HOME_DIR/.nimbus/
tmp_certs_dir=`mktemp -d /tmp/tbmancn.XXXXXXXX`
scp -q
$TB_HOSTS_USERNAME@$HEAD_NODE_IP:$TB_HOSTS_NIMBUS_HEAD_DIR/var/ca/trus
ted-certs/* \
    $tmp_certs_dir/
scp -q $tmp_certs_dir/*
$TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_NIMBUS_CLIENT_DIR/lib/certs/

echo "updating hosts file..."
echo -e "\n# testbed vms" | ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "cat - | sudo tee -a /etc/hosts > /dev/null"

echo "updating host list..."
set_tb_host_as_client_node $HOST_NAME

echo "cleaning up..."
ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP "rm -rf $tmp_newuser_dir"
rm -rf $tmp_newuser_dir
rm -rf $tmp_certs_dir

echo "nimbus client set up on \"$HOST_NAME\""

echo "updating client node settings..."
$BASH_EXE $TB_MGT_DIR/sbin/update-nimbus-client-node $HOST_NAME

exit 0
```

D.2.3 add-nimbus-vmm-node

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
```

```
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

if ! tb_host_has_olsr_ip $HOST_NAME; then
    echo "OLSRd not set up on the host"
    exit 1
fi

if ! tb_host_head_node_exists; then
    echo "No head node found in testbed"
    exit 1
fi

if tb_host_is_vmm_node $HOST_NAME; then
    echo "Nimbus vmm already set up on host"
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

HEAD_NODE_NAME=$(get_testbed_head_node)
HEAD_NODE_IP=$(get_tb_host_wired_ip $HEAD_NODE_NAME)

if [[ $(ssh $TB_HOSTS_USERNAME@$HOST_IP "egrep -c '(vmx|svm)'
/proc/cpuinfo") == 0 ]]; then
    echo "Error: Host does not support hardware virtualization"
    exit 1
fi

if [[ $(ssh $TB_HOSTS_USERNAME@$HOST_IP "egrep -c ' lm '
/proc/cpuinfo") == 0 ]] && \
    ! prompt_accepted "WARNING: Host does not have 64-bit CPU!
Continue anyway?"; then
    exit 1
fi

if ! $(ssh $TB_HOSTS_USERNAME@$HOST_IP "uname -m | grep -Fqw
\"x86_64\"") && \
    ! prompt_accepted "WARNING: Host does not have 64-bit OS
installed! Continue anyway?"; then
    exit 1
fi

echo "copying nimbus client files..."
```

```
ssh $TB_HOSTS_USERNAME@$HOST_IP "mkdir -p $TB_HOSTS_NIMBUS_SRC_DIR"
scp -q $TB_NIMBUS_VMM_TAR_FILE
$TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_NIMBUS_VMM_DIR.tar.gz

echo "installing dependencies..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "$TB_PKG_MGR_UPDATE_CMD
     $TB_PKG_MGR_INSTALL_CMD python-dev
     command -v kvm > /dev/null 2>&1 || $TB_PKG_MGR_INSTALL_CMD qemu-
kvm
     command -v libvirtd > /dev/null 2>&1 || $TB_PKG_MGR_INSTALL_CMD
libvirt-bin
     command -v ubuntu-vm-builder > /dev/null 2>&1 ||
$TB_PKG_MGR_INSTALL_CMD ubuntu-vm-builder
     command -v tunctl > /dev/null 2>&1 || $TB_PKG_MGR_INSTALL_CMD
uml-utilities
     command -v dhcpd > /dev/null 2>&1 || $TB_PKG_MGR_INSTALL_CMD isc-
dhcp-server
     $TB_PKG_MGR_INSTALL_CMD pm-utils" \
      &> /dev/null
reboot_and_wait_if_needed $HOST_NAME

echo "adding user to libvirt and kvm groups..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sudo adduser $TB_HOSTS_USERNAME libvirtd
     sudo adduser $TB_HOSTS_USERNAME kvm" > /dev/null

if ! $(ssh $TB_HOSTS_USERNAME@$HOST_IP "kvm-ok | grep 'KVM
acceleration can be used' > /dev/null"); then
    echo "Error: Hardware virtualization not enabled in the BIOS"
    ssh $TB_HOSTS_USERNAME@$HOST_IP "kvm-ok"
    exit 1
fi

echo "copying kvm/libvirtd configuration files..."
tmp_libvirt_dir=$(ssh $TB_HOSTS_USERNAME@$HOST_IP "mktemp -d
/tmp/tbmanvn.XXXXXXX")
scp -q $TB_MGT_DIR/dist/nimbus/vmm/libvirt/*
$TB_HOSTS_USERNAME@$HOST_IP:$tmp_libvirt_dir/
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sudo cp $tmp_libvirt_dir/* /etc/libvirt/
     sudo restart libvirt-bin"

echo "setting up nimbus vmm..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "tar -xf $TB_HOSTS_NIMBUS_VMM_DIR.tar.gz -C
$TB_HOSTS_NIMBUS_SRC_DIR
     sudo mkdir -p /opt/nimbus
     sudo mv $TB_HOSTS_NIMBUS_VMM_DIR/workspace-control/*
/opt/nimbus/"

echo "updating vmm node settings..."
```

```
set_tb_host_as_vmm_node $HOST_NAME
$BASH_EXE $TB_MGT_DIR/sbin/update-nimbus-vmm-node $HOST_NAME
unset_tb_host_as_vmm_node $HOST_NAME

echo "setting permissions..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "cd /opt/nimbus/
     sudo chown -R root bin etc lib libexec src
     sudo chown -R $TB_HOSTS_USERNAME var
     sudo find . -type d -exec chmod 775 {} \;
     sudo find . -type f -exec chmod 664 {} \;
     sudo find bin sbin libexec -iname \"*sh\" -exec chmod 755 {} \;"

echo "testing dependencies..."
if ! ssh $TB_HOSTS_USERNAME@$HOST_IP "/opt/nimbus/sbin/test-
dependencies.sh" > /dev/null; then
    echo "Error: Dependency script failed"
    exit 1
fi

# Workaround for issue described here (except Ubuntu uses apparmor
instead of SELinux):
# http://wiki.libvirt.org/page/Guest_won%27t_start_-
_warning:_could_not_open_/dev/net/tun_%28%27generic_ethernet%27_interf
ace%29
echo "removing apparmor..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "command -v apparmor_status > /dev/null 2>&1 &&
$TB_PKG_MGR_REMOVE_CMD apparmor
     true" &> /dev/null
reboot_and_wait $HOST_NAME

echo "testing VM creation..."
ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP \
    "cat
$TB_HOSTS_NIMBUS_HEAD_DIR/services/var/nimbus/control.netsample.txt" |
\
    ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "cat - > /tmp/control.netsample.txt"
scp -q $TB_MGT_DIR/dist/nimbus/vmm/ubuntu10.10.gz
$TB_HOSTS_USERNAME@$HOST_IP:/tmp/
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "cd /tmp/
     gunzip -f ubuntu10.10.gz"
if ! $(ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "/opt/nimbus/sbin/control-test.sh --image /tmp/ubuntu10.10 --
netsample \
         /tmp/control.netsample.txt --memory 256 --mountpoint hda" &>
/dev/null); then
    ssh $TB_HOSTS_USERNAME@$HOST_IP "/opt/nimbus/sbin/destroy-control-
test.sh" &> /dev/null
    echo "Error: control-test.sh failed to run properly"
```

```
    exit 1
fi
sleep 10 # Wait for VM to boot
if ! $(ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "ping -c 1 \$(grep ip /tmp/control.netsample.txt | awk '{print
\$2}')" > /dev/null); then
    ssh $TB_HOSTS_USERNAME@$HOST_IP "/opt/nimbus/sbin/destroy-control-
test.sh" &> /dev/null
    echo "Error: Failed to ping test VM"
    exit 1
fi
ssh $TB_HOSTS_USERNAME@$HOST_IP "/opt/nimbus/sbin/destroy-control-
test.sh" &> /dev/null

echo "updating host list..."
set_tb_host_as_vmm_node $HOST_NAME

echo "cleaning up..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "rm -rf $tmp_libvirt_dir
     rm -f /tmp/control.netsample.txt
     rm -f /tmp/ubuntu10.10"

echo "nimbus vmm set up on \"$HOST_NAME\""

$BASH_EXE $TB_MGT_DIR/sbin/connect-nimbus-vmm-to-head $HOST_NAME

exit 0
```

D.2.4 add-olsrd-host

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

if tb_host_has_olsr_ip $HOST_NAME; then
    echo "OLSRd already set up on host"
```

```
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

echo "copying olsrd files..."
ssh $TB_HOSTS_USERNAME@$HOST_IP "mkdir -p $TB_HOSTS_OLSR_DIR"
scp -q $TB_OLSRD_TAR_FILE
$TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_OLSRD_DIR.tar.gz

echo "installing dependencies..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "$TB_PKG_MGR_UPDATE_CMD
     command -v make > /dev/null 2>&1 || $TB_PKG_MGR_INSTALL_CMD make
     command -v bison > /dev/null 2>&1 || $TB_PKG_MGR_INSTALL_CMD
bison
     command -v flex > /dev/null 2>&1 || $TB_PKG_MGR_INSTALL_CMD flex"
&> /dev/null
reboot_and_wait_if_needed $HOST_NAME

echo "setting up olsrd..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "tar -xf $TB_HOSTS_OLSRD_DIR.tar.gz -C $TB_HOSTS_OLSR_DIR
     cd $TB_HOSTS_OLSRD_DIR; make;
     sudo ln $TB_HOSTS_OLSRD_DIR/olsrd /usr/bin/olsrd" > /dev/null

echo "updating host..."
set_tb_host_olsr_ip $HOST_NAME "TMPNE"
$BASH_EXE $TB_MGT_DIR/sbin/update-olsrd-host $HOST_NAME

echo "olsrd set up on \"$HOST_NAME\""

echo "updating hosts..."
$BASH_EXE $TB_MGT_DIR/sbin/update-hosts

exit 0
```

D.2.5 connect-nimbus-vmm-to-head

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE
```

```
HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

if ! tb_host_is_vmm_node $HOST_NAME; then
    echo "Nimbus vmm not set up on host"
    exit 1
fi

if ! tb_host_head_node_exists; then
    echo "No head node found in testbed"
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

HEAD_NODE_NAME=$(get_testbed_head_node)
HEAD_NODE_IP=$(get_tb_host_wired_ip $HEAD_NODE_NAME)

if is_vmm_connected_to_head $HOST_NAME; then
   echo "host \"$HOST_NAME\" already connected to head node"
   exit 1
fi

echo "exchanging public keys..."
ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP \
    "echo -e \"Host $HOST_NAME\n\tStrictHostKeyChecking no\" >>
\$HOME/.ssh/config"
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "echo -e \"Host $HEAD_NODE_NAME\n\tStrictHostKeyChecking no\" >>
\$HOME/.ssh/config
     [ -f \$HOME/.ssh/id_rsa ] || ssh-keygen -f \$HOME/.ssh/id_rsa -N
'\"" > /dev/null
ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP "cat \$HOME/.ssh/id_rsa.pub" | \
    ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sed -i \"/^.*$TB_HOSTS_USERNAME@$HEAD_NODE_NAME\$/d\"
\$HOME/.ssh/authorized_keys
     cat - >> \$HOME/.ssh/authorized_keys"
ssh $TB_HOSTS_USERNAME@$HOST_IP "cat \$HOME/.ssh/id_rsa.pub" | \
    ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP \
    "sed -i \"/^.*$TB_HOSTS_USERNAME@$HOST_NAME\$/d\"
\$HOME/.ssh/authorized_keys
     cat - >> \$HOME/.ssh/authorized_keys"

echo "connecting host \"$HOST_NAME\" to head node..."
host_mem=`echo $(($(ssh $TB_HOSTS_USERNAME@$HOST_IP "cat
/proc/meminfo" \
    | grep 'MemTotal' | awk '{print $2}') / 1024))`
```

```
read -p "Enter the amount of memory to allocate to VMs (maximum is
$host_mem): " input_host_mem
if [[ $input_host_mem -lt $host_mem && $input_host_mem -gt 0 ]]; then
    host_mem=$input_host_mem
else
    echo "Invalid response, defaulting to $host_mem"
fi
if ! $(ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP \
    "[ -f $TB_HOSTS_NIMBUS_HEAD_DIR/services/share/nimbus-
autoconfig/autoconfig-decisions.sh ]"); then
    tmp_autoconf_vals=`mktemp /tmp/tbmcnvth.XXXXXXX`
    cp $TB_MGT_DIR/dist/nimbus/head/autoconfig-decisions.sh.template
$tmp_autoconf_vals
    sed -i "s/@USER@/$TB_HOSTS_USERNAME/g" $tmp_autoconf_vals
    sed -i "s/@HOST@/$HEAD_NODE_NAME/g" $tmp_autoconf_vals
    sed -i "s/@VMM_HOST@/$HOST_NAME/g" $tmp_autoconf_vals
    sed -i "s/@MAX_RAM@/$host_mem/g" $tmp_autoconf_vals
    scp -q $tmp_autoconf_vals \
      $TB_HOSTS_USERNAME@$HEAD_NODE_IP:$TB_HOSTS_NIMBUS_HEAD_DIR/servic
es/share/nimbus-autoconfig/autoconfig-decisions.sh
    ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP \
      "[[ \$($TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl services status |
\
             awk '{print \$3}') == \"running\" ]] || \
             $TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl services start
         $TB_HOSTS_NIMBUS_HEAD_DIR/services/share/nimbus-
autoconfig/autoconfig-adjustments.sh > /dev/null
         $TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl services restart"
else
    ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP \
      "[[ \$($TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl services status |
\
             awk '{print \$3}') == \"running\" ]] || \
             $TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl services start
         $TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbus-nodes --add $HOST_NAME -
-memory $host_mem" > /dev/null
fi

echo "cleaning up..."
rm -f $tmp_autoconf_vals

echo "vmm node \"$HOST_NAME\" connected to head node"

exit 0
```

D.2.6 create-phantom-vm

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

if ! tb_host_phantom_vm_exists; then
    echo "Phantom vm not set up"
    exit 1
fi

if ! tb_host_head_node_exists; then
    echo "No head node found in testbed"
    exit 1
fi

if ! tb_host_vmm_node_exists; then
    echo "No vmm node found in testbed"
    exit 1
fi

if [[ $TB_NIMBUS_VMS_DNS_SERVER == 'none' ]]; then
    echo "No DNS server specified (in main.conf)"
    exit 1
fi

HOST_NAME=$(get_testbed_phantom_vm_node)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

HEAD_NODE_NAME=$(get_testbed_head_node)
HEAD_NODE_IP=$(get_tb_host_wired_ip $HEAD_NODE_NAME)

echo "setting up all VMM nodes as gateways for their VMs..."
for host in $(get_testbed_vmm_nodes); do
    $BASH_EXE $TB_MGT_DIR/sbin/setup-adhoc-gateway $host
done

echo "setting up the credentials file..."
tmp_creds=`mktemp /tmp/tbmcpv.XXXXXXX`
cp $TB_MGT_DIR/dist/phantom/phantom-creds.template $tmp_creds
sed -i "s/@HOST@/$HEAD_NODE_NAME/g" $tmp_creds
sed -i "s/@USER@/$TB_HOSTS_USERNAME/g" $tmp_creds
cat $tmp_creds | ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "cat - > $TB_HOSTS_PHANTOM_DIR/phantom-creds"

echo "setting up the python virtual environment..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "cd $TB_HOSTS_PHANTOM_DIR
     virtualenv .phantom
     source $TB_HOSTS_PHANTOM_DIR/.phantom/bin/activate
     pip install -U boto==2.34.0
```

```
     pip install cloudinitd" > /dev/null

echo "registering a nimbus keypair for phantom..."
scp -q $TB_MGT_DIR/dist/phantom/nimbus-register-keypair
$TB_HOSTS_USERNAME@$HOST_IP:/tmp/nrkp
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "source $TB_HOSTS_PHANTOM_DIR/.phantom/bin/activate
     NIMBUS_HOSTNAME=\"$HEAD_NODE_NAME\" \
     NIMBUS_IAAS_ACCESS_KEY=\$(cat
\$HOME/.secrets/NIMBUS_ACCESS_KEY_ID) \
     NIMBUS_IAAS_SECRET_KEY=\$(cat
\$HOME/.secrets/NIMBUS_SECRET_ACCESS_KEY) \
     /tmp/nrkp \"phantom\" \"\$HOME/.ssh/id_rsa.pub\"" > /dev/null

echo "setting up hosts files..."
tmp_vm_info=`mktemp /tmp/tbmcpv.XXXXXXXX`
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "$TB_HOSTS_NIMBUS_CLIENT_DIR/bin/cloud-client.sh --run --name
phantom.gz --hours 1" \
    > $tmp_vm_info
vm_handle=$(cat $tmp_vm_info | grep "Creating workspace" | awk '{print
$3}' | \
    sed -e 's/^"//' -e 's/"...$//')
vm_ip=$(cat $tmp_vm_info | grep "IP address" | awk '{print $3}')
vm_hostname=$(cat $tmp_vm_info | grep "Hostname" | awk '{print $2}')
sleep 60 # wait for VM to boot
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "echo -e \"\n$vm_ip\t$vm_hostname\" | \
         ssh -o StrictHostKeyChecking=no root@$vm_ip \"cat - >>
/etc/hosts\"
     $TB_HOSTS_NIMBUS_CLIENT_DIR/bin/cloud-client.sh \
         --save --handle $vm_handle --newname phantom-ready.gz &>
/dev/null
     sudo sed -i \"/testbed vms/a $vm_ip\t$vm_hostname\t# phantom-vm\"
/etc/hosts"

echo "extra prep..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "grep -q \"#.*packer\.conf\" $TB_HOSTS_PHANTOM_CONF_FILE ||
     sudo sed -i \"s/\(^.*packer\.conf$\)/# \1/\"
$TB_HOSTS_PHANTOM_CONF_FILE"

echo "running cloudinit.d..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "source $TB_HOSTS_PHANTOM_DIR/.phantom/bin/activate
     source $TB_HOSTS_PHANTOM_DIR/phantom-creds && cloudinitd -v boot
$TB_HOSTS_PHANTOM_CONF_FILE" || :

if $(ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "ssh -o StrictHostKeyChecking=no root@$vm_ip \
         \"[ -f /tmp/nimbusready/newuser/newuser.sh ]\""); then
    echo "configuring Phantom to work with Nimbus on test bed..."
```

```
    nimbus_factory_id=$(ssh $TB_HOSTS_USERNAME@$HOST_IP \
      "cat $TB_HOSTS_NIMBUS_CLIENT_DIR/conf/cloud.properties" | \
          grep vws.factory.identity | sed 's/vws.factory.identity=//')
    tmp_vm_files=`mktemp -d /tmp/tbmcpv.XXXXXXXXX`
    cp $TB_MGT_DIR/dist/phantom/vm/* $tmp_vm_files
    sed -i "s/@HEAD_NODE@/$(get_tb_host_olsr_ip $HEAD_NODE_NAME)/g" \
      $tmp_vm_files/*.template
    sed -i "s|@FACTORY_ID@|$nimbus_factory_id|g"
$tmp_vm_files/antrum.yml.template
    mv $tmp_vm_files/test_add_user.py.template
$tmp_vm_files/test_add_user.py
    mv $tmp_vm_files/add_users.py.template $tmp_vm_files/add_users.py
    mv $tmp_vm_files/antrum.yml.template $tmp_vm_files/antrum.yml
    tmp_vm_files_remote=$(ssh $TB_HOSTS_USERNAME@$HOST_IP "mktemp -d
/tmp/tbmcpv.XXXXXXXXXX")
    scp -q $tmp_vm_files/*
$TB_HOSTS_USERNAME@$HOST_IP:$tmp_vm_files_remote/
    ssh $TB_HOSTS_USERNAME@$HOST_IP \
      "scp -q -o StrictHostKeyChecking=no \
              $tmp_vm_files_remote/*
root@$vm_ip:/home/epu/phantom/sandbox/FG/
          ssh -o StrictHostKeyChecking=no root@$vm_ip \
              \"cd /tmp/nimbusready/newuser
                ./newuser.sh > /dev/null\"" \
    ssh $TB_HOSTS_USERNAME@$HOST_IP \
      "echo \"Your Phantom credentials are:\"
          scp -q -o StrictHostKeyChecking=no \
              root@$vm_ip:/tmp/nimbusready/newuser/bootconf.json
$tmp_vm_files_remote/
          echo -n \"username: \"
          cat $tmp_vm_files_remote/bootconf.json | grep
PHANTOM_USERNAME | \
              tr '\"' ' ' | awk '{ print \$3 }'
          echo -n \"password: \"
          cat $tmp_vm_files_remote/bootconf.json | grep
PHANTOM_IAAS_SECRET_KEY | \
              tr '\"' ' ' | awk '{ print \$3 }'"
fi

echo "cleaning up..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "$TB_HOSTS_NIMBUS_CLIENT_DIR/bin/cloud-client.sh --delete --name
phantom-ready.gz" > /dev/null
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "rm -f /tmp/nrkp
     rm -rf $tmp_vm_files_remote"
rm -f $tmp_vm_info
rm -rf $tmp_vm_files

echo "script complete"
exit 0
```

D.2.7 disconnect-nimbus-vmm-from-head

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

if ! tb_host_is_vmm_node $HOST_NAME; then
    echo "Nimbus vmm not set up on host"
    exit 1
fi

if ! tb_host_head_node_exists; then
    echo "No head node found in testbed"
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

HEAD_NODE_NAME=$(get_testbed_head_node)
HEAD_NODE_IP=$(get_tb_host_wired_ip $HEAD_NODE_NAME)

if ! is_vmm_connected_to_head $HOST_NAME; then
   echo "host \"$HOST_NAME\" not connected to head node"
   exit 1
fi

echo "removing host \"$HOST_NAME\" from head node list..."
if [[ $(ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP \
    "$TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbus-admin --list --host
$HOST_NAME 2>&1") != \
    "No vms with host $HOST_NAME found" ]]; then
    if ! prompt_accepted "WARNING: VMs running on host \"$HOST_NAME\"!
Continue anyway (and destroy VMs)?"; then
     exit 1
    fi
    ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP \
```

```
     "[[ \$($TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl services status |
\
             awk '{print \$3}') == \"running\" ]] || \
             $TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl services start
         $TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbus-admin --shutdown --host
$HOST_NAME --force"
fi
ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP \
    "[[ \$($TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl services status | \
         awk '{print \$3}') == \"running\" ]] || \
         $TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl services start
     $TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbus-nodes --remove $HOST_NAME" >
/dev/null

echo "removing public keys from authorized keys of hosts..."
ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP \
    "sed -i \"/^.*$TB_HOSTS_USERNAME@$HOST_NAME\$/d\"
\$HOME/.ssh/authorized_keys
     sed -i \"/ $HOST_NAME$/,+1d\" \$HOME/.ssh/config
     [ ! -f \$HOME/.ssh/known_hosts ] || ssh-keygen -R $HOST_NAME" &>
/dev/null
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sed -i \"/^.*$TB_HOSTS_USERNAME@$HEAD_NODE_NAME\$/d\"
\$HOME/.ssh/authorized_keys
     sed -i \"/ $HEAD_NODE_NAME$/,+1d\" \$HOME/.ssh/config
     [ ! -f \$HOME/.ssh/known_hosts ] || ssh-keygen -R
$HEAD_NODE_NAME" &> /dev/null

echo "vmm node \"$HOST_NAME\" disconnected from head node"

exit 0
```

D.2.8 is-nimbus-vmm-connected-to-head

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi
```

```
if ! tb_host_is_vmm_node $HOST_NAME; then
    echo "Nimbus vmm not set up on host"
    exit 1
fi

if ! tb_host_head_node_exists; then
    echo "No head node found in testbed"
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

if is_vmm_connected_to_head $HOST_NAME; then
    echo "Host \"$HOST_NAME\" connected to head node"
else
    echo "Host \"$HOST_NAME\" not connected to head node"
fi

exit 0
```

D.2.9 list-hosts

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf

echo -e "HOST\tWIRED IP\tOLSR IP\t\tOLSR IFACE\tNODE TYPE"
cat $TB_HOSTS_FILE

exit 0
```

D.2.10 list-nimbus-client-nodes

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf

echo -e "HOST\tWIRED IP\tOLSR IP\t\tOLSR IFACE\tNODE TYPE"
cat $TB_HOSTS_FILE | grep -Fw "CLIENT"
```

```
exit 0
```

D.2.11 list-nimbus-head-node

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf

echo -e "HOST\tWIRED IP\tOLSR IP\t\tOLSR IFACE\tNODE TYPE"
cat $TB_HOSTS_FILE | grep -Fw "HEAD"

exit 0
```

D.2.12 list-nimbus-vmm-nodes

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf

echo -e "HOST\tWIRED IP\tOLSR IP\t\tOLSR IFACE\tNODE TYPE"
cat $TB_HOSTS_FILE | grep -Fw "VMM"

exit 0
```

D.2.13 list-olsrd-hosts

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf

echo -e "HOST\tWIRED IP\tOLSR IP\t\tOLSR IFACE\tNODE TYPE"
cat $TB_HOSTS_FILE | grep "172.29"

exit 0
```

D.2.14 remove-host

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

if ! prompt_accepted "Remove host \"$HOST_NAME\" from testbed?"; then
    exit 1
fi

if [[ $2 != "-f" ]]; then

    if tb_host_has_olsr_ip $HOST_NAME; then
      $BASH_EXE $TB_MGT_DIR/sbin/remove-olsrd-host $HOST_NAME || :
    fi

    echo "updating hosts file..."
    ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sudo sed -i \"/testbed wired network/,/^[[:blank:]]*$/d\"
/etc/hosts
     sudo sed -i \"/testbed olsr network/,/^[[:blank:]]*$/d\"
/etc/hosts"

    echo "removing passwordless sudo option..."
    ssh $TB_HOSTS_USERNAME@$HOST_IP "sudo rm -f
/etc/sudoers.d/$TB_HOSTS_USERNAME"

    echo "removing public key from authorized keys of host..."
    ssh $TB_HOSTS_USERNAME@$HOST_IP \
      "sed -i \"/^.*$(echo $USER)@$(hostname)\$/d\"
\$HOME/.ssh/authorized_keys"

fi

echo "removing host from hosts lists..."
```

```
sudo sed -i "/^.*$HOST_NAME$TB_HOSTS_WIRED_IP_APPEND\?$/d" /etc/hosts
ssh-keygen -R $HOST_NAME &> /dev/null
ssh-keygen -R $HOST_IP &> /dev/null
remove_testbed_host $HOST_NAME

echo "host \"$HOST_NAME\" removed from testbed"

echo "updating hosts..."
$BASH_EXE $TB_MGT_DIR/sbin/update-hosts

exit 0
```

D.2.15 remove-nimbus-client-node

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

if ! tb_host_is_client_node $HOST_NAME; then
    echo "Nimbus client not set up on host"
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

HEAD_NODE_IP=$(get_tb_host_wired_ip $(get_testbed_head_node))

if ! prompt_accepted "Remove Nimbus client from host \"$HOST_NAME\"?";
then
    exit 1
fi

echo "removing files..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "rm -rf $TB_HOSTS_NIMBUS_CLIENT_DIR
     rm -rf $TB_HOSTS_HOME_DIR/.nimbus
```

```
     rm -rf $TB_HOSTS_NIMBUS_CLIENT_DIR.tar.gz
     sudo rm -f $TB_HOSTS_EXTRA_SCRIPTS_DIR/*-host-entry"

if tb_host_head_node_exists; then
    echo "removing user from head node..."
    ssh $TB_HOSTS_USERNAME@$HEAD_NODE_IP \
     "$TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbus-remove-user
$HOST_NAME@$TB_NIMBUS_CLIENT_DOMAIN"
fi

echo "updating hosts file..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sudo sed -i \"/testbed vms/,/^[[:blank:]]*$/d\" /etc/hosts"

echo "updating host list..."
unset_tb_host_as_client_node $HOST_NAME

echo "nimbus client removed from \"$HOST_NAME\""

echo "script complete"
exit 0
```

D.2.16 remove-nimbus-head-node

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

if ! tb_host_head_node_exists; then
    echo "No head node found in testbed"
    exit 1
fi

HOST_NAME=$(get_testbed_head_node)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

if ! prompt_accepted "Remove Nimbus head node?"; then
    exit 1
fi

if tb_host_vmm_node_exists && prompt_accepted "VMM nodes exist, remove
them?"; then
    for host in $(get_testbed_vmm_nodes); do
     echo "removing Nimbus VMM from host \"$host\""
```

```
      $BASH_EXE $TB_MGT_DIR/sbin/remove-nimbus-vmm-node $host
    done
else
    for host in $(get_testbed_vmm_nodes); do
      if is_vmm_connected_to_head $HOST_NAME; then
          echo "for host \"$host\":"
          $BASH_EXE $TB_MGT_DIR/sbin/disconnect-nimbus-vmm-from-head
$host
      fi
    done
fi

if tb_host_client_node_exists && ( prompt_accepted "Client nodes
exist, remove them?" || \
    prompt_accepted "Current client nodes cannot be reconnected to a
new head node. Remove them?" ); then
    for host in $(get_testbed_client_nodes); do
      echo "removing Nimbus client from host \"$host\""
      $BASH_EXE $TB_MGT_DIR/sbin/remove-nimbus-client-node $host
    done
fi

if tb_host_phantom_vm_exists && ( prompt_accepted "Phantom setup
exists, remove it?" || \
    prompt_accepted "Current Phantom setup will not work with a new
head node. Remove it?" ); then
    echo "removing phantom vm setup"
    $BASH_EXE $TB_MGT_DIR/sbin/remove-phantom-vm-setup
fi

echo "stopping nimbus..."
ssh $TB_HOSTS_USERNAME@$HOST_IP
"$TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl stop"

echo "unsetting up nimbus to run on startup..."
ssh $TB_HOSTS_USERNAME@$HOST_IP "sudo sed -i \"/nimbusctl start/d\"
/etc/rc.local"

echo "removing files..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "rm -rf $TB_HOSTS_NIMBUS_HEAD_DIR
     rm -rf $TB_HOSTS_NIMBUS_SRC_DIR"

echo "removing dependencies..."
if prompt_accepted "ant-optional was installed for Nimbus, but is no
longer needed. Remove?"; then
    echo "removing ant-optional..."
    ssh $TB_HOSTS_USERNAME@$HOST_IP "$TB_PKG_MGR_REMOVE_CMD ant-
optional" &> /dev/null
    reboot_and_wait_if_needed $HOST_NAME
fi
```

```
echo "updating host list..."
unset_tb_host_as_head_node $HOST_NAME

echo "nimbus removed from \"$HOST_NAME\""

echo "script complete"
exit 0
```

D.2.17 remove-nimbus-vmm-node

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

if ! tb_host_is_vmm_node $HOST_NAME; then
    echo "Nimbus VMM not set up on host"
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

if ! prompt_accepted "Remove Nimbus vmm from host \"$HOST_NAME\"?";
then
    exit 1
fi

if tb_host_head_node_exists && is_vmm_connected_to_head $HOST_NAME;
then
    $BASH_EXE $TB_MGT_DIR/sbin/disconnect-nimbus-vmm-from-head
$HOST_NAME
fi

echo "removing files..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "rm -rf $TB_HOSTS_NIMBUS_VMM_DIR
     sudo rm -rf /opt/nimbus
```

```
      rm -rf $TB_HOSTS_NIMBUS_VMM_DIR.tar.gz
      sudo rm -f $TB_HOSTS_EXTRA_SCRIPTS_DIR/*-vm-network
      sudo rm -f /usr/bin/ors
      sudo update-rc.d -f ors remove > /dev/null
      sudo rm -f /etc/init.d/ors"

if prompt_accepted "apparmor was removed for Nimbus VMM. Reinstall it
(Recommended)?"; then
    echo "reinstalling apparmor..."
    ssh $TB_HOSTS_USERNAME@$HOST_IP \
      "$TB_PKG_MGR_UPDATE_CMD
          $TB_PKG_MGR_INSTALL_CMD apparmor" &> /dev/null
    reboot_and_wait_if_needed $HOST_NAME
fi

echo "removing dependencies..."
if prompt_accepted "libvirt, kvm, ubuntu-vm-builder, libcap, tunctl,
and isc-dhcp-server were installed for Nimbus VMM, but are no longer
needed. Remove?"; then
    echo "removing libvirt, kvm, ubuntu-vm-builder, libcap, tunctl,
and isc-dhcp-server..."
    ssh $TB_HOSTS_USERNAME@$HOST_IP \
      "$TB_PKG_MGR_REMOVE_CMD ubuntu-vm-builder libvirt-bin qemu-kvm
libcap2-bin isc-dhcp-server uml-utilities" \
      &> /dev/null
    reboot_and_wait_if_needed $HOST_NAME
fi

echo "updating host list..."
unset_tb_host_as_vmm_node $HOST_NAME

echo "nimbus vmm removed from \"$HOST_NAME\""

echo "script complete"
exit 0
```

D.2.18 remove-olsrd-host

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

HOST_NAME=$1
```

```
if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

if ! tb_host_has_olsr_ip $HOST_NAME; then
    echo "OLSRd not set up on the host"
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

if ! prompt_accepted "Remove OLSRd from host \"$HOST_NAME\"?"; then
    exit 1
fi

echo "killing olsrd..."
ssh $TB_HOSTS_USERNAME@$HOST_IP "pgrep olsrd > /dev/null && sudo
killall olsrd" || true

echo "updating network interfaces file..."
ssh $TB_HOSTS_USERNAME@$HOST_IP "sudo sed -i \"/olsr network
interface/,/^[[:blank:]]*$/d\" /etc/network/interfaces"

echo "removing files..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sudo rm -f /usr/bin/olsrd
     sudo update-rc.d -f olsrd remove
     sudo rm -f /etc/init.d/olsrd
     rm -rf $TB_HOSTS_OLSR_DIR" > /dev/null

echo "removing dependencies..."
if prompt_accepted "bison and flex were installed for OLSRd, but are
no longer needed. Remove?"; then
    echo "removing bison and flex..."
    ssh $TB_HOSTS_USERNAME@$HOST_IP "$TB_PKG_MGR_REMOVE_CMD bison
flex" &> /dev/null
    reboot_and_wait_if_needed $HOST_NAME
fi

echo "removing host olsrd data from hosts lists..."
set_tb_host_olsr_ip $HOST_NAME "NONE"
set_tb_host_wifi_iface $HOST_NAME "NONE"
sudo sed -i "/^.*$HOST_NAME$/d" /etc/hosts

echo "olsrd removed on \"$HOST_NAME\""

echo "updating hosts..."
$BASH_EXE $TB_MGT_DIR/sbin/update-hosts

exit 0
```

D.2.19 remove-phantom-vm-setup

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

if ! tb_host_phantom_vm_exists; then
    echo "Phantom vm not set up"
    exit 1
fi

HOST_NAME=$(get_testbed_phantom_vm_node)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

HEAD_NODE_NAME=$(get_testbed_head_node)
HEAD_NODE_IP=$(get_tb_host_wired_ip $HEAD_NODE_NAME)

if ! prompt_accepted "Remove Phantom VM setup?"; then
    exit 1
fi

echo "removing files..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "rm -rf $TB_HOSTS_PHANTOM_DIR
     rm -rf \$HOME/.secrets"

if tb_host_head_node_exists; then
    echo "removing the phantom vm images from nimbus..."
    ssh $TB_HOSTS_USERNAME@$HOST_IP \
      "$TB_HOSTS_NIMBUS_CLIENT_DIR/bin/cloud-client.sh --delete --name
phantom.gz --common" > /dev/null
fi

echo "removing dependencies..."
if prompt_accepted "virtualenv and git were installed for Phantom, but
are no longer needed. Remove?"; then
    echo "removing virtualenv..."
    ssh $TB_HOSTS_USERNAME@$HOST_IP \
      "$TB_PKG_MGR_REMOVE_CMD python-virtualenv git" \
      &> /dev/null
    reboot_and_wait_if_needed $HOST_NAME
fi
```

```
echo "updating host list..."
unset_tb_host_as_phantom_vm_node $HOST_NAME

echo "phantom vm setup has been removed from \"$HOST_NAME\""

echo "script complete"
exit 0
```

D.2.20 setup-adhoc-gateway

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

if [ $# -eq 0 ]; then
    GW_IP=$TB_NIMBUS_VMS_GATEWAY
else
    GW_IP=$1
fi

if [[ $GW_IP == "default" ]]; then
    if ! tb_host_head_node_exists; then
      echo "No head node found in testbed"
      exit 1
    fi
    HOST_NAME=$(get_testbed_head_node)
    HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)
else
    if ! is_testbed_host $GW_IP; then
      echo "Host to be gateway not found in testbed"
      exit 1
    fi
    HOST_NAME=$(get_tb_host_name $GW_IP)
    HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)
fi

echo "setting up gateway..."
iface=$(ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "ifconfig | grep -B 1 $HOST_IP | head -1 | awk '{print \$1}'")
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sudo sysctl -w net.ipv4.ip_forward=1 > /dev/null
     sudo iptables -t nat -C POSTROUTING -s 172.29.0.0/14 -o $iface -j
MASQUERADE &> /dev/null ||
```

```
     sudo iptables -t nat -A POSTROUTING -s 172.29.0.0/14 -o $iface -j
MASQUERADE"

echo "host \"$HOST_NAME\" set up as adhoc gateway"

exit 0
```

D.2.21 setup-for-phantom-vm

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

if ! tb_host_is_client_node $HOST_NAME; then
    echo "The host is not a client node"
    exit 1
fi

if ! tb_host_head_node_exists; then
    echo "No head node found in testbed"
    exit 1
fi

if ! tb_host_vmm_node_exists; then
    echo "No vmm node found in testbed"
    exit 1
fi

if tb_host_phantom_vm_exists; then
    echo "Phantom vm already running"
    exit 1
fi

if [ ! -f $TB_PHANTOM_VM_IMAGE_FILE ]; then
    echo "ERROR: The image file to be used for Phantom does not exist.
Please
create a KVM compatible, Ubuntu 13.10 64-bit server VM image, and
```

```
place it at the following location:
$TB_PHANTOM_VM_IMAGE_FILE
The image must have git and chef installed on it."
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

HEAD_NODE_NAME=$(get_testbed_head_node)
HEAD_NODE_IP=$(get_tb_host_wired_ip $HEAD_NODE_NAME)

echo "creating the phantom folder..."
ssh $TB_HOSTS_USERNAME@$HOST_IP "mkdir -p $TB_HOSTS_PHANTOM_DIR"

echo "installing dependencies..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "$TB_PKG_MGR_UPDATE_CMD
     $TB_PKG_MGR_INSTALL_CMD python-dev
     $TB_PKG_MGR_INSTALL_CMD git
     command -v virtualenv > /dev/null 2>&1 || $TB_PKG_MGR_INSTALL_CMD
python-virtualenv" \
      &> /dev/null
reboot_and_wait_if_needed $HOST_NAME

echo "downloading the Phantom setup files..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "cd $TB_HOSTS_PHANTOM_DIR
     git clone -q https://github.com/nimbusproject/Phantom.git
     cd $TB_HOSTS_PHANTOM_DIR/Phantom
     git checkout -q ubuntu"

# Workaround for bug described here:
https://bugs.launchpad.net/ubuntu/+source/openjdk-6/+bug/1006776
if ! $(ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "grep -q \"#.*security.provider.9\" /etc/java-*-
openjdk/security/java.security"); then
    ssh $TB_HOSTS_USERNAME@$HOST_IP \
      "sudo sed -i \"s/\(^security.provider.9.*$\)/# \1/\" /etc/java-*-
openjdk/security/java.security"
    reboot_and_wait $HOST_NAME
fi

echo "adding the phantom vm image to nimbus..."
scp -q $TB_PHANTOM_VM_IMAGE_FILE
$TB_HOSTS_USERNAME@$HOST_IP:/tmp/phantom.gz
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "$TB_HOSTS_NIMBUS_CLIENT_DIR/bin/cloud-client.sh --transfer --
sourcefile /tmp/phantom.gz --common" > /dev/null

echo "setting up access keys..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
```

```
    "[ -d \$HOME/.secrets ] || mkdir \$HOME/.secrets"
ssh $TB_HOSTS_USERNAME@$HOST_IP "cat
$TB_HOSTS_NIMBUS_CLIENT_DIR/conf/cloud.properties" | \
    grep s3id | sed 's/.*=//' | ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "cat - | tee \$HOME/.secrets/NIMBUS_ACCESS_KEY_ID > /dev/null"
ssh $TB_HOSTS_USERNAME@$HOST_IP "cat
$TB_HOSTS_NIMBUS_CLIENT_DIR/conf/cloud.properties" | \
    grep s3key | sed 's/.*=//' | ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "cat - | tee \$HOME/.secrets/NIMBUS_SECRET_ACCESS_KEY > /dev/null"

echo "cleaning up..."
ssh $TB_HOSTS_USERNAME@$HOST_IP "rm -f /tmp/phantom.gz"
rm -f $tmp_creds

echo "updating host list..."
set_tb_host_as_phantom_vm_node $HOST_NAME

echo "phantom is setup to run using \"$HOST_NAME\""

exit 0
```

D.2.22 setup-nimbus-head-node

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

if ! tb_host_has_olsr_ip $HOST_NAME; then
    echo "OLSRd not set up on the host"
    exit 1
fi

if tb_host_head_node_exists; then
    echo "Head node already exists"
    exit 1
fi
```

```
HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

echo "copying nimbus iaas files..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "mkdir -p $TB_HOSTS_NIMBUS_DIR
     mkdir -p $TB_HOSTS_NIMBUS_HEAD_DIR
     mkdir -p $TB_HOSTS_NIMBUS_SRC_DIR"
scp -q $TB_NIMBUS_HEAD_TAR_FILE
$TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_NIMBUS_HEAD_SRC_DIR.tar.gz

echo "installing dependencies..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "$TB_PKG_MGR_UPDATE_CMD
     $TB_PKG_MGR_INSTALL_CMD openjdk-6-jdk
     $TB_PKG_MGR_INSTALL_CMD python-dev
     $TB_PKG_MGR_INSTALL_CMD python-twisted-web
     $TB_PKG_MGR_INSTALL_CMD sqlite3
     $TB_PKG_MGR_INSTALL_CMD gcc
     $TB_PKG_MGR_INSTALL_CMD libssl-dev
     $TB_PKG_MGR_INSTALL_CMD ant-optional" &> /dev/null
reboot_and_wait_if_needed $HOST_NAME

# Workaround for bug described here:
https://bugs.launchpad.net/ubuntu/+source/python2.7/+bug/1115466
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sudo ln -s /usr/lib/python2.7/plat-*/_sysconfigdata_nd.py
/usr/lib/python2.7/" &> /dev/null || :

echo "installing nimbus..."
# Workaround for issue where prompts aren't outputted until input is
received
while ! prompt_accepted \
    "During the following setup, the script will pause to wait for
input
(right after \"Configuring installed services\"), but will not output
a prompt.
Simply press Enter twice at this point to allow the script to proceed.
Understand?"; do
    : # loop until prompt is accepted
done
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "tar -xf $TB_HOSTS_NIMBUS_HEAD_SRC_DIR.tar.gz -C
$TB_HOSTS_NIMBUS_SRC_DIR
     cd $TB_HOSTS_NIMBUS_HEAD_SRC_DIR; ./install
$TB_HOSTS_NIMBUS_HEAD_DIR"

echo "setting up rsa key pair..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "[ -f \$HOME/.ssh/id_rsa ] || ssh-keygen -f \$HOME/.ssh/id_rsa -N
'"' > /dev/null
```

```
echo "updating host list..."
set_tb_host_as_head_node $HOST_NAME

echo "\"$HOST_NAME\" set up as head node"

echo "setting up nimbus to run on startup..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sudo sed -i \"s|^exit 0$|su $TB_HOSTS_USERNAME -c
'$TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl start'\n&|\" /etc/rc.local"

echo "updating head node settings..."
$BASH_EXE $TB_MGT_DIR/sbin/update-nimbus-head-node

if tb_host_vmm_node_exists && prompt_accepted "VMM nodes exist,
connect to them?"; then
    for host in $(get_testbed_vmm_nodes); do
      echo "for host \"$host\":"
      $BASH_EXE $TB_MGT_DIR/sbin/connect-nimbus-vmm-to-head $host
    done
fi

exit 0
```

D.2.23 update-hosts

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

echo "updating hosts files..."
for host in $(cut -f1 $TB_HOSTS_FILE); do
    echo "updating $host's host file..."
    host_ip=$(get_tb_host_wired_ip $host)
    tmp_hosts=`mktemp /tmp/tbmuh.XXXXXXX`
    scp -q $TB_HOSTS_USERNAME@$host_ip:/etc/hosts $tmp_hosts
    sed -i "/testbed wired network/,/^[[:blank:]]*$/d" $tmp_hosts
    sed -i "/testbed olsr network/,/^[[:blank:]]*$/d" $tmp_hosts
    sed -i '/^[[:space:]]*$/{:a;$d;N;/\n[[:space:]]*$/ba}' $tmp_hosts
    echo >> $tmp_hosts
    sed -n "/testbed wired network/,/^[[:blank:]]*$/p" /etc/hosts >>
$tmp_hosts
    sed -n "/testbed olsr network/,/^[[:blank:]]*$/p" /etc/hosts >>
$tmp_hosts
    cat $tmp_hosts | ssh $TB_HOSTS_USERNAME@$host_ip "cat - | sudo tee
/etc/hosts > /dev/null"
```

```
    echo "cleaning up..."
    rm -f $tmp_hosts
done

echo "script complete"
exit 0
```

D.2.24 update-nimbus-client-node

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

if ! tb_host_is_client_node $HOST_NAME; then
    echo "Nimbus client not set up on host"
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

echo "updating cloudclient.sh..."
tmp_cc_dir=$(ssh $TB_HOSTS_USERNAME@$HOST_IP "mktemp -d
/tmp/tbmuncn.XXXXXXXX")
scp -q $TB_MGT_DIR/dist/nimbus/client/cloud-client.sh
$TB_HOSTS_USERNAME@$HOST_IP:$tmp_cc_dir/
scp -q $TB_MGT_DIR/dist/nimbus/client/*-host-entry
$TB_HOSTS_USERNAME@$HOST_IP:$tmp_cc_dir/
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sudo cp $tmp_cc_dir/cloud-client.sh
$TB_HOSTS_NIMBUS_CLIENT_DIR/bin/cloud-client.sh
     sudo cp $tmp_cc_dir/*-host-entry $TB_HOSTS_EXTRA_SCRIPTS_DIR"

echo "cleaning up..."
ssh $TB_HOSTS_USERNAME@$HOST_IP "rm -rf $tmp_cc_dir"
```

```
echo "script complete"
exit 0
```

D.2.25 update-nimbus-client-nodes

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

echo "updating client node settings..."
for host in $(get_testbed_client_nodes); do
    echo "updating host \"$host\"..."
    $BASH_EXE $TB_MGT_DIR/sbin/update-nimbus-client-node $host
done

exit 0
```

D.2.26 update-nimbus-head-node

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

if ! tb_host_head_node_exists; then
    echo "No head node found in testbed"
    exit 1
fi

HOST_NAME=$(get_testbed_head_node)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

echo "creating network pool files..."
echo "creating public..."
[ -d $TB_MGT_DIR/dist/nimbus/head/network ] || mkdir
$TB_MGT_DIR/dist/nimbus/head/network
echo "# DNS IP address (or 'none'):" >
$TB_MGT_DIR/dist/nimbus/head/network/public
```

```
echo $TB_NIMBUS_VMS_DNS_SERVER >>
$TB_MGT_DIR/dist/nimbus/head/network/public
echo -e "\n# hostname ipaddress gateway broadcast subnetmask [MAC]\n"
\
    >> $TB_MGT_DIR/dist/nimbus/head/network/public
vm_gw=$TB_NIMBUS_VMS_GATEWAY
if [[ $vm_gw == "default" ]]; then
    vm_gw=$(get_tb_host_olsr_ip $HOST_NAME)
fi
no_ips=$(get_no_ips_in_network $TB_NIMBUS_VMS_PUBLIC_IP_NETWORK)
vm_no=1
for vm_ip in $(get_ips_in_network $TB_NIMBUS_VMS_PUBLIC_IP_NETWORK);
do
    if [[ $vm_no == 1 ]] || [[ $vm_no -gt $(($no_ips-1)) ]]; then
      vm_no=$(($vm_no+1))
      continue
    fi
    echo "$TB_NIMBUS_VMS_PUBLIC_HOST_PREFIX$(printf %02d $vm_no)
$vm_ip $vm_gw none $TB_OLSR_NETMASK" \
      >> $TB_MGT_DIR/dist/nimbus/head/network/public
    vm_no=$(($vm_no+1))
    print_status_bar $vm_no $no_ips
done
echo "creating private..."
echo "# DNS IP address (or 'none'):" >
$TB_MGT_DIR/dist/nimbus/head/network/private
echo "none" >> $TB_MGT_DIR/dist/nimbus/head/network/private
echo -e "\n# hostname ipaddress gateway broadcast subnetmask [MAC]\n"
\
    >> $TB_MGT_DIR/dist/nimbus/head/network/private
no_ips=$(get_no_ips_in_network $TB_NIMBUS_VMS_PRIVATE_IP_NETWORK)
vm_no=1
for vm_ip in $(get_ips_in_network $TB_NIMBUS_VMS_PRIVATE_IP_NETWORK);
do
    if [[ $vm_no == 1 ]] || [[ $vm_no -gt $(($no_ips-1)) ]]; then
      vm_no=$(($vm_no+1))
      continue
    fi
    echo "$TB_NIMBUS_VMS_PRIVATE_HOST_PREFIX$(printf %03d $vm_no)
$vm_ip none none $TB_OLSR_NETMASK" \
      >> $TB_MGT_DIR/dist/nimbus/head/network/private
    vm_no=$(($vm_no+1))
    print_status_bar $vm_no $no_ips
done

echo "updating files..."
scp -q $TB_MGT_DIR/dist/nimbus/head/network/* \

$TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_NIMBUS_HEAD_DIR/services/etc/nim
bus/workspace-service/network-pools/
scp -q $TB_MGT_DIR/dist/nimbus/head/var/cloud.properties.in \
```

```
$TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_NIMBUS_HEAD_DIR/var/cloud.proper
ties.in
tmp_md_conf=`mktemp /tmp/tbmunhn.XXXXXXX`
cp $TB_MGT_DIR/dist/nimbus/head/conf/metadata.conf.template
$tmp_md_conf
sed -i "s/@HOST@/$HOST_IP/g" $tmp_md_conf
scp -q $tmp_md_conf \

$TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_NIMBUS_HEAD_DIR/services/etc/nim
bus/workspace-service/metadata.conf
scp -q $TB_MGT_DIR/dist/nimbus/head/conf/*.conf \

$TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_NIMBUS_HEAD_DIR/services/etc/nim
bus/workspace-service/
scp -q $TB_MGT_DIR/dist/nimbus/head/elastic/other/other-elastic.conf \

$TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_NIMBUS_HEAD_DIR/services/etc/nim
bus/elastic/other/other-elastic.conf
scp -q $TB_MGT_DIR/dist/nimbus/head/elastic/elastic.conf \

$TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_NIMBUS_HEAD_DIR/services/etc/nim
bus/elastic/elastic.conf
ssh $TB_HOSTS_USERNAME@$HOST_IP "sed -i
\"s/^details.hostname=.*$/details.hostname=true/\" \
     $TB_HOSTS_NIMBUS_HEAD_DIR/services/etc/nimbus/workspace-
service/other/common.conf"

echo "restarting nimbus..."
ssh $TB_HOSTS_USERNAME@$HOST_IP
"$TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl restart"

if [[ $TB_NIMBUS_VMS_DNS_SERVER == 'none' ]]; then
    echo "updating netsample file..."
    ssh $TB_HOSTS_USERNAME@$HOST_IP \
      "echo \"dns: 1.1.1.1\" >>
$TB_HOSTS_NIMBUS_HEAD_DIR/services/var/nimbus/control.netsample.txt"
fi

echo "cleaning up..."
rm -f $tmp_md_conf

echo "script complete"
exit 0
```

D.2.27 update-nimbus-vmm-node

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

if ! tb_host_is_vmm_node $HOST_NAME; then
    echo "Nimbus vmm not set up on host"
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

echo "updating workspace-control.sh, dhcp-config.sh, and
dhcpd.conf..."
tmp_wscs_dir=$(ssh $TB_HOSTS_USERNAME@$HOST_IP "mktemp -d
/tmp/tbmunvn.XXXXXXXX")
scp -q $TB_MGT_DIR/dist/nimbus/vmm/workspace-control.sh
$TB_HOSTS_USERNAME@$HOST_IP:$tmp_wscs_dir/
scp -q $TB_MGT_DIR/dist/nimbus/vmm/dhcp-config.sh
$TB_HOSTS_USERNAME@$HOST_IP:$tmp_wscs_dir/
scp -q $TB_MGT_DIR/dist/nimbus/vmm/dhcp/dhcpd.conf
$TB_HOSTS_USERNAME@$HOST_IP:$tmp_wscs_dir/
scp -q $TB_MGT_DIR/dist/nimbus/vmm/*-vm-network
$TB_HOSTS_USERNAME@$HOST_IP:$tmp_wscs_dir/
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sudo cp $tmp_wscs_dir/workspace-control.sh
/opt/nimbus/bin/workspace-control.sh
     sudo cp $tmp_wscs_dir/dhcp-config.sh
/opt/nimbus/libexec/workspace-control/dhcp-config.sh
     sudo cp $tmp_wscs_dir/dhcpd.conf /etc/dhcp/dhcpd.conf
     sed -i \"s/wlan0/$(get_tb_host_wifi_iface $HOST_NAME)/g\"
$tmp_wscs_dir/setup-vm-network
     sudo cp $tmp_wscs_dir/*-vm-network $TB_HOSTS_EXTRA_SCRIPTS_DIR"

echo "copying nimbus vmm configuration files..."
scp -q $TB_MGT_DIR/dist/nimbus/vmm/conf/* \
    $TB_HOSTS_USERNAME@$HOST_IP:/opt/nimbus/etc/workspace-control/

echo "updating and starting resource information collector script..."
tmp_orss_dir=$(ssh $TB_HOSTS_USERNAME@$HOST_IP "mktemp -d
/tmp/tbmunvn.XXXXXXXXX")
scp -q $TB_MGT_DIR/dist/nimbus/vmm/output-resource-status.py \
```

```
    $TB_HOSTS_USERNAME@$HOST_IP:$tmp_orss_dir/
scp -q $TB_MGT_DIR/dist/nimbus/vmm/ors-startup-script
$TB_HOSTS_USERNAME@$HOST_IP:$tmp_orss_dir/
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sudo cp $tmp_orss_dir/output-resource-status.py /usr/bin/ors
     sudo cp $tmp_orss_dir/ors-startup-script /etc/init.d/ors
     sudo chmod +x /etc/init.d/ors
     sudo update-rc.d ors defaults 99
     sudo service ors restart" > /dev/null

echo "cleaning up..."
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "rm -rf $tmp_wscs_dir
     rm -rf $tmp_orss_dir"

echo "script complete"
exit 0
```

D.2.28 update-nimbus-vmm-nodes

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

echo "updating vmm node settings..."
for host in $(get_testbed_vmm_nodes); do
    echo "updating host \"$host\"..."
    $BASH_EXE $TB_MGT_DIR/sbin/update-nimbus-vmm-node $host
done

exit 0
```

D.2.29 update-olsrd-host

```
#!/bin/bash
set -e

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE
```

```
HOST_NAME=$1

if ! is_testbed_host $HOST_NAME; then
    echo "Host not found in testbed"
    exit 1
fi

if ! tb_host_has_olsr_ip $HOST_NAME; then
    echo "OLSRd not set up on the host"
    exit 1
fi

HOST_NAME=$(get_tb_host_name $HOST_NAME)
HOST_IP=$(get_tb_host_wired_ip $HOST_NAME)

echo "setting up network..."
scp -q $TB_MGT_DIR/dist/olsrd/olsrd-adhoc-setup \
    $TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_OLSRD_DIR/files/olsrd-adhoc-
setup
tmp_network_info=`mktemp /tmp/tbmuoh.XXXXXXX`
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sudo $TB_HOSTS_OLSRD_DIR/files/olsrd-adhoc-setup \"\"
$TB_OLSR_CHANNEL $TB_OLSR_ESSID $TB_OLSR_BSSID " \
    &> $tmp_network_info
HOST_OLSR_IP=`cat $tmp_network_info | grep "with IP" | sed 's/.*with
IP //'`
HOST_WIFI_IFACE=`cat $tmp_network_info | grep "with IP" | sed
's/.*setup on //' | awk '{print $1;}'`

echo "adding olsr data to hosts lists..."
set_tb_host_olsr_ip $HOST_NAME $HOST_OLSR_IP
set_tb_host_wifi_iface $HOST_NAME $HOST_WIFI_IFACE
grep -Fqw "$HOST_OLSR_IP" /etc/hosts || \
    sudo sed -i "/testbed olsr network/a $HOST_OLSR_IP\t$HOST_NAME"
/etc/hosts

echo "killing olsrd..."
ssh $TB_HOSTS_USERNAME@$HOST_IP "pgrep olsrd > /dev/null && sudo
killall olsrd" || true

echo "updating configuration files..."
scp -q $TB_MGT_DIR/dist/olsrd/$TB_OLSRD_CONFIG_FILE_NAME \
    $TB_HOSTS_USERNAME@$HOST_IP:$TB_HOSTS_OLSRD_DIR/
ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "sudo mkdir -p /etc/olsrd
     sudo cp $TB_HOSTS_OLSRD_DIR/$TB_OLSRD_CONFIG_FILE_NAME
/etc/olsrd/olsrd.conf"

echo "starting olsrd..."
ssh $TB_HOSTS_USERNAME@$HOST_IP "sudo $TB_HOSTS_OLSRD_DIR/olsrd -i
$HOST_WIFI_IFACE" &> /dev/null
```

```
echo "updating network interfaces file..."
tmp_network_ifaces=`mktemp /tmp/tbmuoh.XXXXXXXX`
scp -q $TB_HOSTS_USERNAME@$HOST_IP:/etc/network/interfaces
$tmp_network_ifaces
sed -i "/auto $HOST_WIFI_IFACE/,/^[[:blank:]]*$/d" $tmp_network_ifaces
sed -i "/olsr network interface/,/^[[:blank:]]*$/d"
$tmp_network_ifaces
sed -i '/^[[:space:]]*$/{:a;$d;N;/\n[[:space:]]*$/ba}'
$tmp_network_ifaces
echo -e "\n# The olsr network interface" >> $tmp_network_ifaces
echo -e "auto $HOST_WIFI_IFACE\niface $HOST_WIFI_IFACE inet static" >>
$tmp_network_ifaces
echo -e "\taddress $HOST_OLSR_IP" >> $tmp_network_ifaces
echo -e "\tnetmask $TB_OLSR_NETMASK" >> $tmp_network_ifaces
echo -e "\twireless-channel $TB_OLSR_CHANNEL" >> $tmp_network_ifaces
echo -e "\twireless-essid $TB_OLSR_ESSID" >> $tmp_network_ifaces
echo -e "\twireless-mode ad-hoc" >> $tmp_network_ifaces
cat $tmp_network_ifaces | ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "cat - | sudo tee /etc/network/interfaces > /dev/null"

echo "updating startup script..."
tmp_olsrd_startup=`mktemp /tmp/tbmuoh.XXXXXXXXX`
cp $TB_MGT_DIR/dist/olsrd/olsrd-startup-script.template
$tmp_olsrd_startup
sed -i "s/@IFACE@/$HOST_WIFI_IFACE/g" $tmp_olsrd_startup
cat $tmp_olsrd_startup | ssh $TB_HOSTS_USERNAME@$HOST_IP \
    "cat - | sudo tee /etc/init.d/olsrd > /dev/null
     sudo chmod +x /etc/init.d/olsrd
     sudo update-rc.d olsrd defaults 99" > /dev/null

echo "cleaning up..."
rm -f $tmp_network_info
rm -f $tmp_network_ifaces
rm -f $tmp_olsrd_startup

echo "script complete"
exit 0
```

D.2.30 update-olsrd-hosts

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
source $TESTBED_MGT_DIR/etc/main.conf
source $TB_BASH_FUNCTIONS_LIB_FILE

echo "updating olsrd hosts settings..."
```

```
for host in $(get_olsr_testbed_hosts); do
    echo "updating host \"$host\"..."
    $BASH_EXE $TB_MGT_DIR/sbin/update-olsrd-host $host
done

echo "updating hosts..."
$BASH_EXE $TB_MGT_DIR/sbin/update-hosts

exit 0
```

D.3 Other Scripts

D.3.1 setup

This script is contained in `antrum/management/setup/`.

```
#!/bin/bash

TESTBED_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
if $(source $TESTBED_MGT_DIR/etc/main.conf &> /dev/null); then
    source $TESTBED_MGT_DIR/etc/main.conf
else
    source $TESTBED_MGT_DIR/etc/main.conf.example
fi

if [ -f $TB_HOSTS_FILE ]; then
    echo "Setup already run"
    exit 1;
fi

echo "setting up passwordless sudo..."
sudo $TB_MGT_DIR/dist/set-pwdless-sudo $(whoami)

echo "installing necessary software..."
$TB_PKG_MGR_UPDATE_CMD
$TB_PKG_MGR_INSTALL_CMD python-dev python-netaddr

if [ ! -f $HOME/.ssh/id_rsa ]; then
    echo "setting up rsa keys..."
    ssh-keygen -f $HOME/.ssh/id_rsa -N ''
fi

if [ ! -f $TESTBED_MGT_DIR/etc/main.conf ]; then
    echo "creating config file..."
    cp $TESTBED_MGT_DIR/etc/main.conf.example
$TESTBED_MGT_DIR/etc/main.conf
fi
```

```
if ! $(grep -Fqw "# testbed wired network" /etc/hosts); then
    echo "adding heading to /etc/hosts file..."
    echo -e "\n# testbed wired network\n\n# testbed olsr network" |
sudo tee -a /etc/hosts > /dev/null
fi

echo "creating testbed hosts file..."
mkdir $TESTBED_MGT_DIR/var
touch $TB_HOSTS_FILE

echo "downloading OLSRd setup files..."
cd $TB_MGT_DIR/dist/olsrd/
curl -O http://www.olsr.org/releases/0.6/olsrd-
$TB_OLSRD_VERSION.tar.gz
echo "downloading Nimbus setup files..."
cd $TB_MGT_DIR/dist/nimbus/
curl -O http://www.nimbusproject.org/downloads/nimbus-iaas-
$TB_NIMBUS_IAAS_VERSION-src.tar.gz
curl -O http://www.nimbusproject.org/downloads/nimbus-iaas-controls-
$TB_NIMBUS_IAAS_VERSION.tar.gz
curl -O http://www.nimbusproject.org/downloads/nimbus-cloud-client-
$TB_NIMBUS_CLIENT_VERSION.tar.gz
echo "downloading test VMM image..."
cd $TB_MGT_DIR/dist/nimbus/vmm/
curl -O http://www.nimbusproject.org/downloads/ubuntu10.10.gz

echo "script complete"
exit 0
```

D.4 Configuration Files

These configuration files are `contained in antrum/management/sbin/`.

D.4.1 main.conf.example

This is an example configuration file for the test bed management scripts.

```
##########################################
### Master configuration file for aNTRuM ###
##########################################
#
#################################################################

# Username for all nodes of the testbed
TB_HOSTS_USERNAME="nimbus"
```

```
# This string will be appended to the non-adhoc IP address of each
testbed
# host
TB_HOSTS_WIRED_IP_APPEND="x"

# For the scripts that require rebooting a host, this is the port that
will
# be used to listen for the host to signal that the reboot is complete
TB_REBOOT_WAIT_PORT="9876"

# File and directory paths on the master node
#
# The path for antrum
TB_MGT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )/.." && pwd )"
# The function libraries paths
TB_BASH_FUNCTIONS_LIB_FILE="$TB_MGT_DIR/lib/functions.sh"
TB_PYTHON_FUNCTIONS_LIB_FILE="$TB_MGT_DIR/lib/functions.py"
# The path to the file that contains the testbed host information
TB_HOSTS_FILE="$TB_MGT_DIR/var/testbed_hosts"

# The testbed nodes home directory
TB_HOSTS_HOME_DIR="/home/$TB_HOSTS_USERNAME"

# The path to bash
BASH_EXE="/bin/bash"

# Package manager commands
# (Changing these may break the scripts, as the names of the sofware
may
# be different for different package managers)
TB_PACKAGE_MANAGER="apt-get"
TB_PKG_MGR_UPDATE_CMD="sudo $TB_PACKAGE_MANAGER -q -y update"
TB_PKG_MGR_INSTALL_CMD="sudo $TB_PACKAGE_MANAGER -q -y install"
TB_PKG_MGR_REMOVE_CMD="sudo $TB_PACKAGE_MANAGER -q -y remove"

# The path for the auxillary scripts running on the testbed nodes
# If changed, the path specified in the scripts that call the auxilary
# scripts must be changed
TB_HOSTS_EXTRA_SCRIPTS_DIR="/usr/local/bin"


##############################################################
#
# OLSRd Scripts Configurations
#

# Version of OLSRd to use
TB_OLSRD_VERSION="0.6.7.1"

# The name of the OLSRd configuration file to use (found in
dist/olsrd/)
TB_OLSRD_CONFIG_FILE_NAME="tb_olsrd.conf"
```

```
# OLSRd network configuration
#
# The ad hoc network on which OLSRd will run. IP addresses will be
# assigned to the wireless cards from this network.
# Do not change this value, as it is hardcoded into other scripts
TB_OLSR_NETWORK="172.29.0.0/16"
# The ad hoc network netmask, set to include the network for the VMs
TB_OLSR_NETMASK="255.252.0.0"
# The ad hoc network wirless channel
TB_OLSR_CHANNEL="1"
# The ad hoc network ESSID
TB_OLSR_ESSID="nimbusnodes"
# The ad hoc network BSSID
TB_OLSR_BSSID="02:ca:ff:ee:ba:be"

# The path to the OLSRd software on the master node
TB_OLSRD_TAR_FILE="$TB_MGT_DIR/dist/olsrd/olsrd-
$TB_OLSRD_VERSION.tar.gz"

# Paths to the OLSR and OLSRd directories on the testbed nodes
TB_HOSTS_OLSR_DIR="$TB_HOSTS_HOME_DIR/olsr"
TB_HOSTS_OLSRD_DIR="$TB_HOSTS_HOME_DIR/olsr/olsrd-$TB_OLSRD_VERSION"

#####################################################################
#
# Nimbus Scripts Configurations
#

# Versions of Nimbus software to use
TB_NIMBUS_IAAS_VERSION="2.10.1"
TB_NIMBUS_CLIENT_VERSION="022"

# Domain name to use for Nimbus client setup
TB_NIMBUS_CLIENT_DOMAIN="nimbustb"

# Nimbus VM network configuration
#
# The hostname prefixes to use for for the public and private VM
# network pools
TB_NIMBUS_VMS_PUBLIC_HOST_PREFIX="pub"
TB_NIMBUS_VMS_PRIVATE_HOST_PREFIX="priv"
# The public and private networks from which IP addresses will be
# assinged to the VMs. The public network is set so as to be
# accessible from the testbed nodes ad hoc network
# Mask bits value (/x) should be between 16 and 32
TB_NIMBUS_VMS_PUBLIC_IP_NETWORK="172.30.0.0/23"
TB_NIMBUS_VMS_PRIVATE_IP_NETWORK="10.30.0.0/23"
# DNS server to use for access outisde the ad hoc network
# If set to "none", VMs will have not internet connectivity
# If this value is changed after the Nimbus head node is set up,
# the nimbus head node update script will need to be run
```

```
TB_NIMBUS_VMS_DNS_SERVER="none"
# Testbed node to use as gateway for access outside the ad hoc
# network
# "default" is Nimbus head node. If changed, use an IP address
# (e.g. 172.29.99.99)
TB_NIMBUS_VMS_GATEWAY="default"

# The paths to the Nimbus Head, VMM, and Client node softwares on
# the master node
TB_NIMBUS_HEAD_TAR_FILE="$TB_MGT_DIR/dist/nimbus/nimbus-iaas-
$TB_NIMBUS_IAAS_VERSION-src.tar.gz"
TB_NIMBUS_VMM_TAR_FILE="$TB_MGT_DIR/dist/nimbus/nimbus-iaas-controls-
$TB_NIMBUS_IAAS_VERSION.tar.gz"
TB_NIMBUS_CLIENT_TAR_FILE="$TB_MGT_DIR/dist/nimbus/nimbus-cloud-
client-$TB_NIMBUS_CLIENT_VERSION.tar.gz"

# Nimbus directory paths on the testbed nodes
# (Changing these may break the scripts, as some are interdependent)
#
# Nimbus directory
TB_HOSTS_NIMBUS_DIR="$TB_HOSTS_HOME_DIR/nimbus"
# Nimbus software source files directory
TB_HOSTS_NIMBUS_SRC_DIR="$TB_HOSTS_HOME_DIR/nimbus-src"
# Nimbus head node software directory
TB_HOSTS_NIMBUS_HEAD_DIR="$TB_HOSTS_NIMBUS_DIR"
# Nimbus head node software source files directory
TB_HOSTS_NIMBUS_HEAD_SRC_DIR="$TB_HOSTS_NIMBUS_SRC_DIR/nimbus-iaas-
$TB_NIMBUS_IAAS_VERSION-src"
# Nimbus VMM node software directory
TB_HOSTS_NIMBUS_VMM_DIR="$TB_HOSTS_NIMBUS_SRC_DIR/nimbus-iaas-
controls-$TB_NIMBUS_IAAS_VERSION"
# Nimbus Client software directory
TB_HOSTS_NIMBUS_CLIENT_DIR="$TB_HOSTS_NIMBUS_DIR/nimbus-cloud-client-
$TB_NIMBUS_CLIENT_VERSION"

#################################################################
#
# Phantom Scripts Configurations
#

# Path to the VM image file to be used to set up phantom
TB_PHANTOM_VM_IMAGE_FILE="$TB_MGT_DIR/dist/phantom/phantom-ubuntu.gz"

# Phantom file and directory paths on the testbed nodes
#
# Phantom directory
TB_HOSTS_PHANTOM_DIR="$TB_HOSTS_HOME_DIR/phantom"
# Configuration file to be used to set up Phantom
TB_HOSTS_PHANTOM_CONF_FILE="$TB_HOSTS_PHANTOM_DIR/Phantom/plan/test.co
nf"
```

D.5 Libraries

These functions libraries are contained in `antrum/management/lib/`.

D.5.1 functions.py

```
from netaddr import IPNetwork, IPAddress
import socket

def inNetwork(addr, network):
    if IPAddress(addr) in IPNetwork(network):
        return True
    return False

def isValidIP(addr):
    try:
        socket.inet_aton(addr)
        return True
    except socket.error:
        return False

def getIPsInNetwork(iprange):
    for ipaddr in IPNetwork(iprange):
        print ipaddr

def getNoIPsInNetwork(iprange):
    return len(IPNetwork(iprange))
```

D.5.2 functions.sh

```
### PYTHON FUNCTIONS ###

# calls a python function $1 with params $2,...,$n (passed as strings)
_call_python_function()
{
    local params=${@:2}
    python -c "import imp; functions = imp.load_source(\"functions\",
\"$TB_PYTHON_FUNCTIONS_LIB_FILE\"); print functions.$1($(echo
\'${params// /\', \'}\'));" | sed '${/None/d;}'
}

# checks if $1 is in the network $2
in_network()
{
    [[ $(_call_python_function inNetwork $1 $2) == "True" ]]
```

```
}

# checks if $1 is a valid ip address
is_valid_ip()
{
    [[ $(_call_python_function isValidIP $1) == "True" ]]
}

# returns a list of ip addresses in the network (separated by a
newline)
get_ips_in_network()
{
    _call_python_function getIPsInNetwork $1
}

# returns the number of ip addresses in the network
get_no_ips_in_network()
{
    _call_python_function getNoIPsInNetwork $1
}

### TESTBED HOSTS LIST FUNTIONS ###

# adds new host $1 to the testbed hosts list (no values set)
add_testbed_host()
{
    echo -e "$1\tNONE\tNONE\tNONE\tNONE" >> $TB_HOSTS_FILE
}

# removes host $1 from the testbed hosts list
remove_testbed_host()
{
    sed -i "/^$1\t.*$/d" $TB_HOSTS_FILE
}

# checks if $1 belongs to the testbed
is_testbed_host()
{
    [ -n $1 ] || return 1

    if is_valid_ip $1; then
      grep -Fqw "$1" $TB_HOSTS_FILE
    else
      cut -f1 $TB_HOSTS_FILE | grep -Fqw "$1"
    fi
}

# returns all testbed host names
get_testbed_hosts()
{
    for host in $(cut -f1 $TB_HOSTS_FILE); do
      echo $host
```

```
    done
}

# returns all testbed host names with olsr IP addresses
get_olsr_testbed_hosts()
{
    for host in $(cut -f1 $TB_HOSTS_FILE); do
      if tb_host_has_olsr_ip $host; then
          echo $host
      fi
    done
}

# returns the host name of the testbed head node
get_testbed_head_node()
{
    for host in $(cut -f1 $TB_HOSTS_FILE); do
      if tb_host_is_head_node $host; then
          echo $host
          return 0
      fi
    done
}

# returns all the host names of the testbed vmm nodes
get_testbed_vmm_nodes()
{
    for host in $(cut -f1 $TB_HOSTS_FILE); do
      if tb_host_is_vmm_node $host; then
          echo $host
      fi
    done
}

# returns the host name running the phantom vm
get_testbed_phantom_vm_node()
{
    for host in $(cut -f1 $TB_HOSTS_FILE); do
      if tb_host_is_phantom_vm_node $host; then
          echo $host
          return 0
      fi
    done
}

# returns all the host names of the testbed client nodes
get_testbed_client_nodes()
{
    for host in $(cut -f1 $TB_HOSTS_FILE); do
      if tb_host_is_client_node $host; then
          echo $host
      fi
```

```
    done
}

# gets the host name of host $1
get_tb_host_name()
{
    echo `awk '/'$1'/{ print $1 }' $TB_HOSTS_FILE`
}

# sets the wired IP address of host $1 to $2
set_tb_host_wired_ip()
{
    sed -i "/$1/s/[^\t]*[^\t]/$2/2" $TB_HOSTS_FILE
}

# gets the wired IP address of host $1
get_tb_host_wired_ip()
{
    echo `awk '/'$1'/{ print $2 }' $TB_HOSTS_FILE`
}

# sets the olsr IP address of host $1 to $2
set_tb_host_olsr_ip()
{
    sed -i "/$1/s/[^\t]*[^\t]/$2/3" $TB_HOSTS_FILE
}

# gets the olsr IP address of host $1
get_tb_host_olsr_ip()
{
    echo `awk '/'$1'/{ print $3 }' $TB_HOSTS_FILE`
}

# checks if host $1 has an olsr ip address
tb_host_has_olsr_ip()
{
    [[ $(awk '/'$1'/{ print $3 }' $TB_HOSTS_FILE) != "NONE" ]]
}

# sets the wireless interface name of host $1 to $2
set_tb_host_wifi_iface()
{
    sed -i "/$1/s/[^\t]*[^\t]/$2/4" $TB_HOSTS_FILE
}

# gets the wireless interface name of host $1
get_tb_host_wifi_iface()
{
    echo `awk '/'$1'/{ print $4 }' $TB_HOSTS_FILE`
}

# sets the nimbus node type of host $1 to $2
```

```
_set_tb_host_node_type()
{
    if [[ $(_get_tb_host_node_types $1) == "NONE" ]]; then
      sed -i "/$1/s/[^\t]*[^\t]/$2/5" $TB_HOSTS_FILE
    else
      sed -i "/$1/s/[^\t]*[^\t]/&,$2/5" $TB_HOSTS_FILE
    fi
}

# unsets the nimbus node type $2 of host $1
_unset_tb_host_node_type()
{
    if [[ $(_get_tb_host_node_types $1) == *,* ]]; then
      tmp_host_node_types=$(echo `_get_tb_host_node_types $1` | \
          sed -e "s/$2,\?//" -e "s/^,//" -e "s/,$//")
      sed -i "/$1/s/[^\t]*[^\t]/$tmp_host_node_types/5" $TB_HOSTS_FILE
    else
      sed -i "/$1/s/[^\t]*[^\t]/NONE/5" $TB_HOSTS_FILE
    fi
}

# gets the nimbus node types of host $1
_get_tb_host_node_types()
{
    echo `awk '/'$1'/{ print $5 }' $TB_HOSTS_FILE`
}

# checks if host $1 is the head node
tb_host_is_head_node()
{
    [[ $(_get_tb_host_node_types $1) == *HEAD* ]]
}

# sets host $1 as the head node
set_tb_host_as_head_node()
{
    _set_tb_host_node_type $1 "HEAD"
}

# unsets host $1 as the head node
unset_tb_host_as_head_node()
{
    _unset_tb_host_node_type $1 "HEAD"
}

# checks if head node exists
tb_host_head_node_exists()
{
    cut -f5 $TB_HOSTS_FILE | grep -Fqw "HEAD"
}

# checks if host $1 is a client node
```

```
tb_host_is_client_node()
{
    [[ $(_get_tb_host_node_types $1) == *CLIENT* ]]
}

# sets host $1 as a client node
set_tb_host_as_client_node()
{
    _set_tb_host_node_type $1 "CLIENT"
}

# unsets host $1 as a client node
unset_tb_host_as_client_node()
{
    _unset_tb_host_node_type $1 "CLIENT"
}

# checks if at least one client node exists
tb_host_client_node_exists()
{
    cut -f5 $TB_HOSTS_FILE | grep -Fqw "CLIENT"
}

# checks if host $1 is a vmm node
tb_host_is_vmm_node()
{
    [[ $(_get_tb_host_node_types $1) == *VMM* ]]
}

# sets host $1 as a vmm node
set_tb_host_as_vmm_node()
{
    _set_tb_host_node_type $1 "VMM"
}

# unsets host $1 as a vmm node
unset_tb_host_as_vmm_node()
{
    _unset_tb_host_node_type $1 "VMM"
}

# checks if at least one vmm node exists
tb_host_vmm_node_exists()
{
    cut -f5 $TB_HOSTS_FILE | grep -Fqw "VMM"
}

# checks if host $1 is running the phantom vm
tb_host_is_phantom_vm_node()
{
    [[ $(_get_tb_host_node_types $1) == *PHAN* ]]
}
```

```
# sets host $1 as running the phantom vm
set_tb_host_as_phantom_vm_node()
{
    _set_tb_host_node_type $1 "PHAN"
}

# unsets host $1 as running the phantom vm
unset_tb_host_as_phantom_vm_node()
{
    _unset_tb_host_node_type $1 "PHAN"
}

# checks if the phantom vm is running on a node
tb_host_phantom_vm_exists()
{
    cut -f5 $TB_HOSTS_FILE | grep -Fqw "PHAN"
}

### TESTBED HOSTS FUNTIONS ###

# checks if nimbus vmm node ($1) is connected to the nimbus head node
in the testbed
# a head node must exist in the testbed, or the function will not work
properly
is_vmm_connected_to_head()
{
    local vmm_node_name=$(get_tb_host_name $1)
    local head_node_name=$(get_testbed_head_node)
    local head_node_ip=$(get_tb_host_wired_ip $head_node_name)

    ssh $TB_HOSTS_USERNAME@$head_node_ip \
      "[[ \$($TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl services status |
\
              awk '{print \$3}') == \"running\" ]] || \
              $TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbusctl services start"
    ssh $TB_HOSTS_USERNAME@$head_node_ip \
      "$TB_HOSTS_NIMBUS_HEAD_DIR/bin/nimbus-nodes --list | \
              grep -qw \"hostname.*[[:space:]]$vmm_node_name\""
}

### MISC FUNCTIONS ###

# prompts for y/n answer to $1
prompt_accepted()
{
    while true; do
      read -p "$1 " yn
      case $yn in
            [Yy]* ) return 0; break;;
            [Nn]* ) return 1; break;;
            * ) echo "Please answer yes or no (y/n)";;
```

```
      esac
    done
}

# reboots host $1 and waits for it to start
reboot_and_wait()
{
    local host=$(get_tb_host_wired_ip $1)
    local local_ip=$(ip route get $host | awk '{ print $NF; exit }')

    echo "rebooting host..."
    ssh $TB_HOSTS_USERNAME@$host \
      "sudo sed -i \"/^$/{s/.*/echo 'done' | nc $local_ip
$TB_REBOOT_WAIT_PORT\n/;:a;n;ba}\" /etc/rc.local
          sudo reboot"
    nc -l $TB_REBOOT_WAIT_PORT -q 5
    ssh $TB_HOSTS_USERNAME@$host \
      "sudo sed -i \"/echo 'done' | nc $local_ip
$TB_REBOOT_WAIT_PORT/d\" /etc/rc.local"
}

# reboots host $1 and waits for it to start only if a reboot is
required
reboot_and_wait_if_needed()
{
    local host=$(get_tb_host_wired_ip $1)

    if ssh $TB_HOSTS_USERNAME@$host "[ -f /var/run/reboot-required ]";
then
      reboot_and_wait $host
    fi
}

# prints a status bar for a loop using $1 (iteration #) of $2 (total)
print_status_bar()
{
    p_complete=$(($1*100/$2))
    if [[ $p_complete == 100 ]]; then
      echo -ne "\r\033[0Kdone\n"
      return 0
    fi
    stat_bar=""
    for i in $(seq 1 $(($p_complete/2))); do
      stat_bar="$stat_bar#"
    done
    for i in $(seq $(($p_complete/2+1)) 50); do
      stat_bar="$stat_bar."
    done
    echo -ne "\r[$(printf %-50s $stat_bar)][$p_complete%]"
}
```

```
# prints an error for the following trap function with exit code $1,
line number $2,
# absolute path of scripts $3, and script arguments $4
# does not print error on exit code 1, as that is used to exit the
scripts on purpose
handle_error()
{
    if [[ $1 != 1 ]]; then
      echo "ERROR: An error occurred in $3 on line $2 (exit code $1)"
      echo "To debug, run 'bash -x $3 ${@:4}'"
    fi
}
trap 'handle_error $? $LINENO \
    "$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )/$(basename $0)"
$@' ERR
```

Appendix E: Test Bed Scripts and Software Modifications

This Appendix provides the scripts that run on the hosts that are a part of aNTRuM, as well as the software files that were modified to work with aNTRuM, found at https://github.com/hipersys/antrum/tree/master/management/dist.

E.1 Test Bed Management Helper Scripts

E.1.1 set-pwdless-sudo

This script is contained in `dist/`.

```
#!/bin/bash
set -e

if [[ $EUID -ne 0 ]]; then
    echo "$0 must be run as root"
    exit 1
fi

if [ -z "$1" ]; then
    echo "No user specified"
    exit 1
fi

USER=$1

# Set passwordless sudo for USER
echo -e "$USER\tALL=(ALL)NOPASSWD:ALL" > /tmp/$USER
chmod 0440 /tmp/$USER
visudo -c -f /tmp/$USER
cp /tmp/$USER /etc/sudoers.d/$USER
rm -f /tmp/$USER
```

E.2 Test Bed Scripts

E.2.1 output-resource-status.py

This is the script that collects and outputs system resource information. It can be found in `dist/nimbus/vmm/`.

```
#!/usr/bin/python
#
```

```
# Created by: Joshua McKee
#
# Outputs resource information to a file
#
# These are the resource metrics provided:
# For memory information:
#  - Total memory (in kB): memory.memtotal
#  - Free memory (in kB):  memory.memfree
# For device information:
#  - Device has battery(s) (0|1): device.hasbattery
# (The following will be provided only if the device has a battery)
#  - Device is plugged in (0|1): device.pluggedin
# For battery information:
#  - For each battery:
#     - Battery status ('full'|'charging'|'discharging'):
battery_[bat_no].status
#     - Battery capacity (in %): battery_[bat_no].capacity
#

import os
import re
import time
import glob

COLLECTION_INTERVAL = 30 # seconds
DESTINATION_FILE = "/tmp/resource_info"

if os.listdir("/sys/class/power_supply"):
    dev_hasbattery = 1
else:
    dev_hasbattery = 0

while True:
    f_output = open(DESTINATION_FILE,"w")
    f_meminfo = open("/proc/meminfo", "r")
    if dev_hasbattery:
        path_acinfo = glob.glob("/sys/class/power_supply/A*/online")
        with open(path_acinfo[0], "r") as f_acinfo:
            if f_acinfo.readline():
                dev_pluggedin = 1
            else:
                dev_pluggedin = 0

        f_batteryinfo = dict([])
        for bat in glob.glob("/sys/class/power_supply/BAT[0-
9]*/uevent"):
            m = re.match("/sys/class/power_supply/BAT([0-9]*)/uevent",
bat)
            if not m:
                continue
            bat_no = m.group(1)
```

```
            f_batteryinfo[bat_no] =
open("/sys/class/power_supply/BAT%s/uevent" % bat_no, "r")

    # memory information
    f_meminfo.seek(0)
    for line in f_meminfo:
        m = re.match("(\w+):\s+(\d+)\s+(\w+)", line)
        if m and (m.group(1).lower() == 'memtotal' or
m.group(1).lower() == 'memfree'):
            f_output.write("memory.%s %s\n" % (m.group(1).lower(),
m.group(2)))

    # device information
    f_output.write("device.hasbattery %s\n" % dev_hasbattery)
    if dev_hasbattery:
        f_output.write("device.pluggedin %s\n" % dev_pluggedin)

    # battery information (if available)
    if dev_hasbattery:
        for bat_no in f_batteryinfo.keys():
            f = f_batteryinfo[bat_no]
            f.seek(0)
            for line in f:
                m = re.match("POWER_SUPPLY_(\w+)=(\w+)", line)
                if m and (m.group(1).lower() == 'capacity' or
m.group(1).lower() == 'status'):
                    f_output.write("battery_%s.%s %s\n"
                                   % (bat_no, m.group(1).lower(),
m.group(2).lower()))

    f_output.close()
    time.sleep(COLLECTION_INTERVAL)
```

E.3 Modified Software Files

This section contains files that were modified to work with the aNTRuM test bed, as well as auxiliary scripts called by them.

E.3.1 cloud-client.sh

This script is from the Nimbus cloud client software package. It was modified to call `add-host-entry` and `remove-host-entry`. It can be found in `dist/nimbus/client/`.

```
#!/bin/bash

# Modified by Joshua McKee
```

```
BASEDIR_REL="`dirname $0`/.."
BASEDIR=`cd $BASEDIR_REL; pwd`
BASEDIR=${BASEDIR/ /\\ }

EMBEDDED_GL="$BASEDIR/lib/globus"
USER_PROPFILE="$BASEDIR/conf/cloud.properties"
HISTORY_DIR="$BASEDIR/history"
EMBEDDED_CADIR="$BASEDIR/lib/certs"

if [ -n "$NIMBUS_X509_TRUSTED_CERTS" ]; then
  X509_CERT_DIR="$NIMBUS_X509_TRUSTED_CERTS"
else
  X509_CERT_DIR="$EMBEDDED_CADIR"
fi
export X509_CERT_DIR

OLD_GLOBUS_LOCATION=""
if [ -n "$GLOBUS_LOCATION" ]; then
  OLD_GLOBUS_LOCATION="$GLOBUS_LOCATION"
fi

GLOBUS_LOCATION=$EMBEDDED_GL
export GLOBUS_LOCATION

if [ -n "$OLD_GLOBUS_LOCATION" ]; then
  if [ "$OLD_GLOBUS_LOCATION" != "$GLOBUS_LOCATION" ]; then
      echo "(Overriding old GLOBUS_LOCATION '$OLD_GLOBUS_LOCATION')"
      echo -e "(New GLOBUS_LOCATION: '$GLOBUS_LOCATION')"
  fi
fi

needsconf="y"
needshist="y"
for i in "$@"; do
  if [ "--conf" == "$i" ]; then
    needsconf="n"
  fi
  if [ "--history-dir" == "$i" ]; then
    needshist="n"
  fi
done

INCLUDED_COMMANDLINE_STRING=""
if [ "X$needsconf" == "Xy" ]; then
  INCLUDED_COMMANDLINE_STRING="$INCLUDED_COMMANDLINE_STRING --conf
$USER_PROPFILE"
fi
if [ "X$needshist" == "Xy" ]; then
  INCLUDED_COMMANDLINE_STRING="$INCLUDED_COMMANDLINE_STRING --history-
dir $HISTORY_DIR"
fi
```

```
####### JAVA CHECK ##########

if [ "X$JAVA_HOME" = "X" ] ; then
  _RUNJAVA=java
 else
  _RUNJAVA="$JAVA_HOME"/bin/java
fi

####### Generated globus client sh script follows.

DELIM="#"
EXEC="org.globus.bootstrap.Bootstrap
org.globus.workspace.cloud.client.CloudClient"

DEF_OPTIONS=""
DEF_CMD_OPTIONS="$INCLUDED_COMMANDLINE_STRING"
EGD_DEVICE="/dev/urandom"

updateOptions() {

  if [ "X$2" != "X" ] ; then
    GLOBUS_OPTIONS="$GLOBUS_OPTIONS -D$1=$2"
  fi

}

####### MAIN BODY ##########

if [ ! -d $GLOBUS_LOCATION ] ; then
  echo "Error: GLOBUS_LOCATION invalid or not set: $GLOBUS_LOCATION"
1>&2
  exit 1
fi

LOCALCLASSPATH=$GLOBUS_LOCATION/lib/bootstrap.jar:$GLOBUS_LOCATION/lib
/cog-url.jar:$GLOBUS_LOCATION/lib/axis-url.jar

### SETUP OTHER VARIABLES ####

updateOptions "GLOBUS_LOCATION" "$GLOBUS_LOCATION"
updateOptions "java.endorsed.dirs" "$GLOBUS_LOCATION/endorsed"
updateOptions "X509_USER_PROXY" "$X509_USER_PROXY"
updateOptions "X509_CERT_DIR" "$X509_CERT_DIR"
updateOptions "GLOBUS_HOSTNAME" "$GLOBUS_HOSTNAME"
updateOptions "GLOBUS_TCP_PORT_RANGE" "$GLOBUS_TCP_PORT_RANGE"
updateOptions "GLOBUS_TCP_SOURCE_PORT_RANGE"
"$GLOBUS_TCP_SOURCE_PORT_RANGE"
updateOptions "GLOBUS_UDP_SOURCE_PORT_RANGE"
"$GLOBUS_UDP_SOURCE_PORT_RANGE"

if [ -c "$EGD_DEVICE" -a  -r "$EGD_DEVICE" ]; then
    updateOptions "java.security.egd" "file://$EGD_DEVICE"
```

```
fi

if [ "X$IBM_JAVA_OPTIONS" = "X" ] ; then
  IBM_JAVA_OPTIONS=-Xquickstart
  export IBM_JAVA_OPTIONS
fi

if [ $# -gt 0 ]; then
  if [ "X${DEF_CMD_OPTIONS}" != "X" ]; then
    set - ${GLOBUS_OPTIONS} -classpath ${LOCALCLASSPATH} ${EXEC}
${DEF_CMD_OPTIONS} "$@"
  else
    set - ${GLOBUS_OPTIONS} -classpath ${LOCALCLASSPATH} ${EXEC} "$@"
  fi
else
  if [ "X${DEF_CMD_OPTIONS}" != "X" ]; then
    set - ${GLOBUS_OPTIONS} -classpath ${LOCALCLASSPATH} ${EXEC}
${DEF_CMD_OPTIONS}
  else
    set - ${GLOBUS_OPTIONS} -classpath ${LOCALCLASSPATH} ${EXEC}
  fi
fi

OLD_IFS=${IFS}
IFS=${DELIM}
for i in ${DEF_OPTIONS} ; do
  IFS=${OLD_IFS}
  DEFINE=`echo $i|cut -d'=' -f1`
  if [ "$DEFINE" != "$i" ]; then
    VALUE="`echo $i|cut -d'=' -f2-`"
    set - $DEFINE="$VALUE" "$@"
  else
    set - $DEFINE "$@"
  fi
  IFS=${DELIM}
done
IFS=${OLD_IFS}

### EXECUTE ############

##########################
### BEGIN MODIFICATION ###
##########################
#
# DESCRIPTION:
# Runs the appropriate script for adding/removing an entry for the new
VM to
# the hosts file

# the location of the host list addition/removal scripts
SCRIPTS_DIR="/usr/local/bin"
```

```
if $(echo "$@" | grep -qw "\-\-run") && ! $(echo "$@" | grep -qw "\-\-
cluster"); then
    tmp_vm_info=`mktemp /tmp/tbmcc.XXXXXXX`
    $SCRIPTS_DIR/add-host-entry $tmp_vm_info &
    exec $_RUNJAVA "$@" | tee $tmp_vm_info
    exit 0
elif $(echo "$@" | grep -qw "\-\-terminate") || $(echo "$@" | grep -qw
"\-\-save"); then
    vm_handle=$(echo "$@" | tr ' ' '\n' |  sed -n '/--handle/{n;p}')
    $SCRIPTS_DIR/remove-host-entry $vm_handle
fi

exec $_RUNJAVA "$@"

############################
###  END MODIFICATION  ###
############################
```

E.3.2 add-host-entry

This script is called by `cloud-client.sh`. It can be found in `dist/nimbus/client/`

```
#!/bin/bash

# Created by: Joshua McKee
#
# Adds an entry to the hosts file for a VM created by the cloud client
by
# reading a file generated by the cloud client script (cloud-
client.sh). Once
# the file exists, the entry is added.
# Meant to be used with cloud-client.sh
# Meant to be paired with remove-host-entry
# Arguments:
#  $1 - The file generated by the cloud client
#
# NOTE: Passwordless sudo required for script to run properly!

sourcefile=$1

# wait until file with vm info exists
while [ ! -f $sourcefile ]
do
    sleep 1
done
sleep 10 # wait for file to be written to
```

```
vm_handle=$(cat $sourcefile | grep "Creating workspace" | awk '{print
$3}' | \
    sed -e 's/^"//' -e 's/"...$//')
vm_ip=$(cat $sourcefile | grep "IP address" | awk '{print $3}')
vm_hostname=$(cat $sourcefile | grep "Hostname" | awk '{print $2}')

sudo sed -i "/^$vm_ip\t.*$/d" /etc/hosts
sudo sed -i "/testbed vms/a $vm_ip\t$vm_hostname\t# $vm_handle"
/etc/hosts

# clean up
rm -f $sourcefile

exit 0
```

E.3.3 remove-host-entry

This script is called by `cloud-client.sh`. It can be found in `dist/nimbus/client/`

```
#!/bin/bash

# Created by: Joshua McKee
#
# Removes an entry from the hosts file for a VM terminated by the
cloud client.
# Meant to be used with cloud-client.sh
# Meant to be paired with remove-host-entry
# Arguments:
#  $1 - The vm handle
#
# NOTE: Passwordless sudo required for script to run properly!

vm_handle=$1

sudo sed -i "/^.*\t# $vm_handle.*$/d" /etc/hosts

exit 0
```

E.3.4 workspace-control.sh

This script is from the Nimbus control agent software package. This script was modified to call `setup-vm-network` and `cleanup-vm-network`. It can be found in `dist/nimbus/vmm/`.

```
#!/bin/bash

# Modified by: Joshua McKee

PYTHON_EXE="/usr/bin/env python"

NIMBUS_CONTROL_DIR_REL="`dirname $0`/.."
NIMBUS_CONTROL_DIR=`cd $NIMBUS_CONTROL_DIR_REL; pwd`

NIMBUS_CONTROL_MAINCONF="$NIMBUS_CONTROL_DIR/etc/workspace-
control/main.conf"

if [ ! -f "$NIMBUS_CONTROL_MAINCONF" ]; then
    echo ""
    echo "Cannot find main conf file, exiting. (expected at
'$NIMBUS_CONTROL_MAINCONF')"
    exit 1
fi

NIMBUS_CONTROL_PYLIB="$NIMBUS_CONTROL_DIR/lib/python"
NIMBUS_CONTROL_PYSRC="$NIMBUS_CONTROL_DIR/src/python"
PYTHONPATH="$NIMBUS_CONTROL_PYSRC:$NIMBUS_CONTROL_PYLIB:$PYTHONPATH"
export PYTHONPATH

# ------------------------------------------------------------------
---------

##########################
### BEGIN MODIFICATION ###
##########################
#
# DESCRIPTION:
# Aquires a few values from the available paramters, and runs the vm
network
# setup/cleanup scripts as appropriate.

# the location of the vm network setup/cleanup scripts
SCRIPTS_DIR="/usr/local/bin"

action=`echo $@ | tr ' ' '\n' | sed -n '/--action/{n;p}'`
if [ -z "$action" ]; then
    action=`echo $@ | awk '{print $1}' | sed 's/--//'`
fi
vm_name=`echo $@ | tr ' ' '\n' | sed -n '/--name/{n;p}'`

if [[ $action == create ]]; then
    vm_ips=`echo $@ | tr ' ' '\n' | sed -n '/--network/{n;p}' | tr ';'
'\n' | sed -n '6~16p' | egrep
'[[:digit:]]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3
}'`
    i=0
    for vm_ip in $vm_ips; do
```

```
      $SCRIPTS_DIR/setup-vm-network $vm_name-$i $vm_ip
      let i++
    done
fi

#########################
### PAUSE MODIFICATION ###
#########################

$PYTHON_EXE $NIMBUS_CONTROL_PYSRC/workspacecontrol/main/wc_cmdline.py
-c $NIMBUS_CONTROL_MAINCONF "$@"

#########################
### CONT' MODIFICATION ###
#########################

if [[ $action == remove ]]; then
    nics=`/sbin/ifconfig -a | grep -w "$vm_name" | awk '{print $1}'`
    for nic in $nics; do
      $SCRIPTS_DIR/cleanup-vm-network $nic
    done
fi

#########################
###  END MODIFICATION  ###
#########################
```

E.3.5 setup-vm-network

This script is called by `workspace-control.sh`. It can be found in `dist/nimbus/vmm/`.

```
#!/bin/bash

# Created by: Joshua McKee
#
# Sets up a TAP interface for a VM to use to bridge a wireless
interface. The
# interface is created and assigned an IP address in the same network
as the
# VM's Ip address. IP forwarding is enabled, and proxy ARP is enabled
for the
# TAP and wifi interfaces. A route for the VM's IP address is created
for the
# TAP device. Needs to run
#  (1) before the VM is created and
#  (2) before Nimbus restarts the local dhcp server.
# Meant to be paired with cleanup-vm-network.
```

```
# Arguments:
#  $1 - The name of the TAP device
#  $2 - The IP address of the VM
#
# NOTE: Passwordless sudo required for script to run properly!

nic=$1
vm_ip=$2

sudo tunctl -u $EUID -t $nic

sudo sysctl -w net.ipv4.ip_forward=1
sudo sysctl -w net.ipv4.conf.$nic.proxy_arp=1
sudo sysctl -w net.ipv4.conf.wlan0.proxy_arp=1

#Use an IP address outside the range of available VM IPs
nic_ip=`echo $vm_ip | awk -F '.'
'{printf("%d.%d.%d.%d",$1,$2+1,$3,$4)}'`

sudo ip addr add $nic_ip dev $nic
sudo ip link set $nic up

sudo route add -host $vm_ip dev $nic

exit 0
```

E.3.6 cleanup-vm-network

This script is called by `workspace-control.sh`. It can be found in `dist/nimbus/vmm/`.

```
#!/bin/bash

# Created by: Joshua McKee
#
# Removes the TAP interface used by a VM for bridging a wireless
interface.
# Needs to run after the VM is destroyed.
# Meant to be paired with setup-vm-network.
# Arguments:
#  $1 - The name of the TAP device
#
# NOTE: Passwordless sudo required for script to run properly!

nic=$1

sudo ifconfig $nic down
sudo tunctl -u $EUID -d $nic
```

```
# Note that route and proxy ARP for interface are automatically
removed.
# Note that IP forwarding and proxy ARP are not disabled, since
another VM may
# be running.

exit 0
```

E.4 Other Files

The test bed management scripts propagate many modified software configuration files and additional scripts to allow aNTRuM to run properly. Because these are not original work and most come from the various software packages used in aNTRuM, and for the sake of brevity, only pointers to their locations are included here.

E.4.1 Nimbus Service Node Files

The directory `dist/nimbus/head/` contains modified configuration files from the Nimbus software that runs on the Nimbus service (head) node, specifically:

- `autoconfig-decisions.sh.template`, a template for a Nimbus configuration file sourced by `autoconfig-adjustments.sh`, from `$NIMBUS_INSTALL_DIR/services/share/nimbus-autoconfig/`.
- In the `conf` subdirectory, Nimbus configuration files from `$NIMBUS_INSTALL_DIR/services/etc/nimbus/workspace-service/`, including:
  - `accounting.conf`
  - `admin.conf`
  - `async.conf`
  - `global-policies.conf`
  - `logging.conf`
  - `metadata.conf.template` (serves as a template for the actual `metadata.conf` file)
  - `network.conf`
  - `pilot-authz.conf`
  - `pilot.conf`
  - `repository.conf`
  - `vmm.conf`
- In the `elastic` subdirectory, Nimbus configuration files from `$NIMBUS_INSTALL_DIR/services/etc/nimbus/elastic/`, including:
  - `elastic.conf`
  - `other/other-elastic.conf`

- In the `var` subdirectory, a Nimbus configuration file from `$NIMBUS_INSTALL_DIR/var/`, namely:
    - `cloud.properties.in`

E.4.2 Nimbus VMM Node Files

The directory `dist/nimbus/vmm/` contains modified configuration files and scripts from and for the software that runs on the Nimbus VMM nodes, specifically:

- In the `conf` subdirectory, Nimbus VMM configuration files from `$NIMBUS_INSTALL_DIR/services/etc/nimbus/workspace-service/`, including:
    - `dirs.conf`
    - `images.conf`
    - `internal.conf`
    - `kernels.conf`
    - `libvirt.conf`
    - `libvirt_template.xml`
    - `logging.conf`
    - `main.conf`
    - `mount.conf`
    - `networks.conf`
    - `propagation.conf`
    - `sudo.conf`
    - `xen.conf`
- `dhcp-config.sh`, a Nimbus VMM script for modifying DHCP for Nimbus, from `$NIMBUS_INSTALL_DIR/libexec/workspace-control/`.
- In the `dhcp` subdirectory, the DHCPd configuration file from `/etc/dhcp/`, namely:
    - `dhcpd.conf`
- In the `libvirt` subdirectory, libvirt and qemu configuration files from `/etc/libvirt/`, including:
    - `libvirtd.conf`
    - `qemu.conf`
- `ors-startup-script`, a script for `/etc/init.d/` to allow the system resource collector script in Section E.2.1 to run on startup.

E.4.3 OLSRd Software Files

The directory `dist/olsrd/` contains modified configuration files and scripts from and for the OLSRd software, specifically:

- `olsrd-adhoc-setup`, a script for setting up an ad hoc connection on a wireless card, from `$OLSRD_DIR/files/`.
- `olsrd-startup-script.template`, a template for a script for `/etc/init.d/` to allow OLSRd to run on startup.

- `tb_olsrd.conf`, the configuration file for OLSRd, to be placed in `/etc/olsrd/`.

E.4.4 Nimbus Phantom Files

The directory `dist/phantom/` contains modified files and scripts from and for setting up a VM running Phantom, specifically:

- `nimbus-register-keypair`, a script for registering a key with the Nimbus cloud, from https://gist.github.com/oldpatricka/3752775.
- `phantom-creds.template`, template for a file containing values for environment variables that need to be set to set up Phantom using cloudinit.d.
- `phantom-ubuntu.gz`, a VM image ready to be used to set up Phantom. Essentially, it is a Nimbus cloud and KVM compatible 64-bit Ubuntu Server 13.10 VM image with git and chef installed on it.
- In the `vm` subdirectory, modified scripts and a configuration file for use in finalizing the Phantom VM setup, from `/home/epu/phantom/sanbox/FG/` on the Phantom VM, including:
    - `add_sites.sh`
    - `add_users.py.template` (serves as a template for the actual `add_users.py` script)
    - `antrum.yml.template` (serves as a template for the actual `antrum.yml` file)
    - `test_add_user.py.template` (serves as a template for the actual `test_add_user.py` script)

Appendix F: Test Bed Scripts Directory Structure

This appendix describes the directory structure of the test bed management scripts found at https://github.com/hipersys/antrum. It also lists the directories of `antrum` in a tree like structure.

F.1 Directory Structure Description

In `antrum`, there is a directory called `management`, which contains the following directories:

- `bin`, a directory containing the primary scripts for setting up the test bed.
- `dist`, a directory containing the files distributed to the devices in the test bed by the management scripts. The directories in `dist` are:
    - `nimbus`, a directory containing files relating to the Nimbus software. The directories in `nimbus` are:
        - `client`, a directory containing the files relating to the Nimbus client software.
        - `head`, a directory containing the files relating to the Nimbus software that runs on the Nimbus service (head) node. The directories in `head` are:
            - `conf`, a directory containing the main configuration files for the Nimbus service node software.
            - `elastic`, a directory containing the configuration files related to Nimbus' EC2 interface. (`elastic` contains a directory called `other`, which also contains a configuration file for the EC2 interface).
            - `network`, a directory containing the VM network pool files for the Nimbus service node software.
            - `var`, a directory containing the configuration file used when creating a new Nimbus cloud user.
        - `vmm`, a directory containing the files relating to the Nimbus VMM software. The directories in `vmm` are:
            - `conf`, a directory containing the main configuration files for the Nimbus VMM software.
            - `dhcp`, a directory containing the configuration file for the DHCP server software that runs on Nimbus VMM nodes.
            - `libvirt`, a directory containing the configuration files for the libvirt and qemu software that runs on Nimbus VMM nodes.
    - `olsrd`, a directory containing files relating to the OLSRd software.
    - `phantom`, a directory containing files relating to the Phantom software. The directories in `phantom` are:
        - `vm`, a directory containing files used in the Phantom VM.
- `etc`, a directory containing the configuration file for the management scripts.
- `lib`, a directory containing functions used by the management scripts.
- `sbin`, a directory containing the core scripts called by the primary scripts in `bin`.

- `setup`, a directory containing the script for setting up the host that will run the management scripts.
- `var`, a directory containting the list of hosts in the test bed.

F.2 Directory Tree

```
antrum
└── management
    ├── bin
    ├── dist
    │   ├── nimbus
    │   │   ├── client
    │   │   ├── head
    │   │   │   ├── conf
    │   │   │   ├── elastic
    │   │   │   │   └── other
    │   │   │   ├── network
    │   │   │   └── var
    │   │   └── vmm
    │   │       ├── conf
    │   │       ├── dhcp
    │   │       └── libvirt
    │   ├── olsrd
    │   └── phantom
    │       └── vm
    ├── etc
    ├── lib
    ├── sbin
    ├── setup
    └── var
```
