The Integration of a Programmable Edge Node to ProtoGENI Control Framework

University of Massachusetts Lowell
September 2009

Abstract

A Programmable Edge Node is a multicore based virtualization platform for GENI users to deploy experiments. It consists of general purpose processors on the host and network processors at the NIC level, isolating the experiments and measurements. We have developed such a prototype and integrated with one of the GENI control frameworks, ProtoGENI. ProtoGENI is primarily built on top of Emulab and being augmented to support GENI federation. In this document, we introduce the schemes how our PEN prototype is integrated to ProtoGENI control framework. We present the hardware architecture of our ProtoGENI aggregate established at UMass Lowell. We describe the software modules, including the component manager, developed for integrating PENs into ProtoGENI.

1. Introduction

GENI, a national-scale network infrastructure, is being designed and prototyped to support virtualized network experimentation. The users of GENI request resources from resources providers to deploy their experiments. As the experiments scale in size and diversity, GENI needs to rely on control frameworks to facilitate resource management, discovery and optimization. ProtoGENI is one of the control frameworks proposed at GENI Spiral One stage, and it is being actively improved to address the challenging issues in the federation process.

As one of the GENI Spiral One teams, we have developed a Programmable Edge Node prototype to support virtual experiments and provide modest isolation of experiments and measurement. Our PEN platform consists of both general-purpose and application-specific multicore processors for different workloads in the virtual network environment. Our goal is to integrate the PEN into ProtoGENI control framework such that users can discover the resources available at the PEN and request them for the users' experiments through a common interface.

ProtoGENI control framework provides a common interface for users to request resources. The current version of the interface is based on Flash technology. The graphical interface displays the available resources at all ProtoGENI aggregates. A user can drag and drop the node icons (e.g. computers and switches) to build a virtual network to be studied. Then the user can request the allocation of these nodes, and once granted, they can access the nodes and deploy their experiments.

The resource discovery and allocation is done by the ProtoGENI control framework through databases and RSpec of resources. Since our PEN is a new type of resource that supports dynamic virtual containers and virtual NICs, it needs a new RSpec definition and an unconventional resource management scheme. This is where we devote our effort to integrate PENs to ProtoGENI.

This document describes how the PEN prototype is integrated with ProtoGENI control framework. In Section 2, we describe the hardware architecture of the ProtoGENI aggregate

established at UMass Lowell. Our aggregate has currently one PEN prototype available for users' experiments. We also present the details of the software components we develop for this integration in Section 3. Among the software modules, we provide details of the "component manager" which communicates with the ProtoGENI clearinghouse, interacts with the Flash Interface, and most importantly performs the actual resource allocation and release. We conclude the document in Section 4 and point users to further readings in Section 5.

2. ProtoGENI Aggregate Architecture

In the ProtoGENI environment there are two major elements which comprise the architecture – the Clearinghouse and the Component Manager (CM). The Clearinghouse is a central server which keeps track of each CM and its associated credentials. Every site in the ProtoGENI project runs a CM which is responsible for managing local resources. When a user creates an experiment, the Clearinghouse is responsible for gathering resource information from each CM and reporting the resources which are available to the user. When the user creates a topology using the retrieved resources, the Clearinghouse sends the relevant topological information to each CM. The CMs, in turn, allocate their resources to match the requested layout.

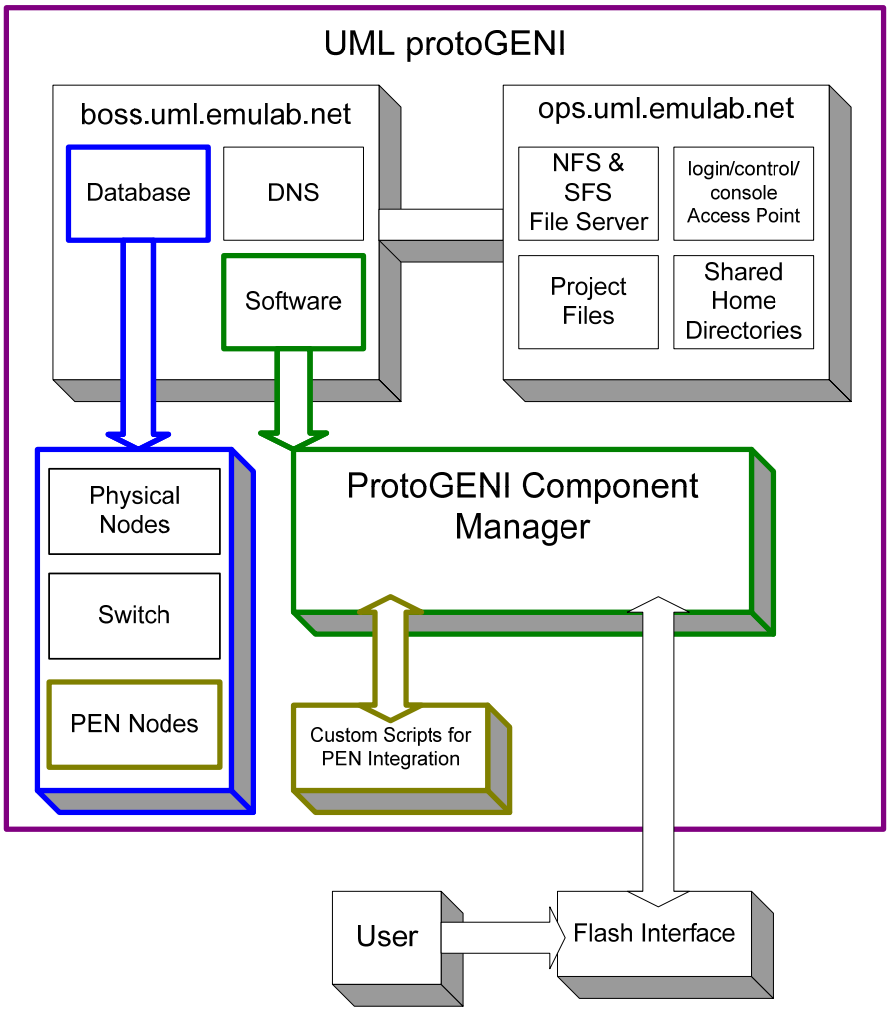

Figure 1: Architecture of ProtoGENI Aggregate at UMass Lowell

Figure 1 illustrates the hardware architecture of the ProtoGENI aggregate established in the CANS lab at UMass Lowell. The UML ProtoGENI site posses a boss node and an ops node. The ops node serves as the file manager. The boss node is responsible for the database and the CM. Within the UML CM modifications have been made to incorporate virtual nodes hosted on the PEN node. With the exception of the PEN node, the UML ProtoGENI follows the architectural specifications of the ProtoGENI project. Overall, such hardware setup is mandated by Emulab configuration. More detailed information is available at www.protogeni.net and www.emulab.net .

3. Software Modules for Integration

We have developed software modules to integrate the PEN to ProtoGENI control framework. ProtoGENI extensively use databases and scripts to track the resources and perform resource management operations. Our PEN is a new, special resource in terms of the virtual container creation and virtual NIC instantiation. Therefore, we devote our effort in developing RSpec and scripts customized for PEN-type resources. We then plug in our scripts into the ProtoGENI control software.

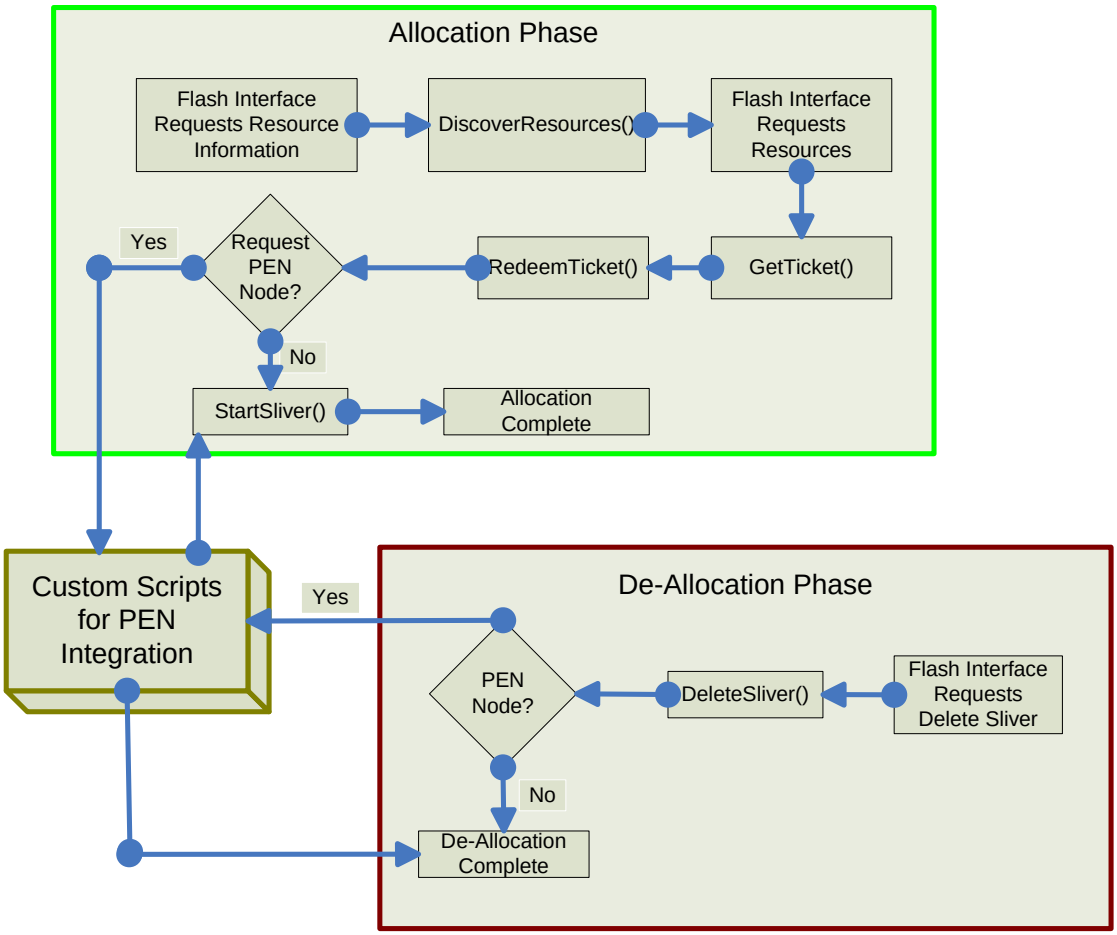

Figure 2: Software Flow Chart

Figure 2 shows the flow chart of the ProtoGENI software. The software is executed from the Flash Interface upon users' selection and commanding. The software consists of two major phases: allocation and de-allocation of resources. Our customization happens in both phases and

we will explain it in Section 3.2. For resource management, ProtoGENI leverages databases to define resources and track their usage. We next describe the database entries we add for PEN-type nodes.

3.1 Database:

A virtual node hosted on PEN is described in the Emulab database, tbdb, very similar to a regular node. A new node type was created, pcpen, which is used to describe each virtual node. The class of pcpen is pc and shares all of the same characteristics. The node type pcpen is used for differentiating PEN nodes from regular nodes in the custom PEN scripts. For a node to be available, it is manually inserted into the nodes table with the following features:

- node_id: pen*
- type: pcpen
- phys_node_id: pen*
- def_boot_osid: 10000
- priority: 1
- eventstate: PXEWAIT

All other values in the nodes table will be NULL, a timestamp, or the same as a regular PC. The interfaces of the virtual node are described in the interfaces table in the same manner as a regular network interface. One entry must be made for every desired port on the virtual machine – so if a machine can support a maximum of five ports, five ports must be described and associated with the virtual machine within the interfaces table.

These database entries will allow for a virtual node to be assigned by the Flash Interface. These modifications only trick the component manager into passing the information. Now that the component manager can acknowledge and pass the information regarding the virtual node, modifications are made to the component manager so that it accesses custom scripts in the case of a request for a virtual node on PEN.

3.2 Customized Scripts:

When the user accesses the Flash Interface, it sends a resource request to the UML component manager. The component manager then calls the function DiscoverResources and returns the results. The Flash Interface now has a list of the resources available at the UML site. The user now selects a virtual node hosted on PEN and clicks ‘Create Slivers’. The request is sent to the UML component manager which enters the GetTicket function, followed by the RedeemTicket function.

Within the RedeemTicket function, the first custom script, pcpen_setup, is called. Pcpen_setup is a PERL script which collects information about the node to be created (number of NICs, PID, EID) and passes it on to the PEN host. The pcpen_setup script uses the SSH protocol to login to the PEN host and call the second custom script, ve_setup. This script is responsible for creating the machine as described by pcpen_setup. It is responsible for issuing the OpenVZ commands that activate the container. Once the container is live, ve_setup will use the NFM to create the appropriate number of virtual NICs and allocate them to physical ports. If every step completes successfully, ve_setup returns success, pcpen_setup in turn returns success, and the allocation phase completes.

Once the slivers have been created, the user will click the 'Boot Slivers' button on the Flash Interface in order to activate the allocated machines. In the instance of the virtual PEN node, the machine is already live, so a condition has been added to the StartSliver function in the component manager. This condition allows for the StartSliver function to bypass the need to activate the already active virtual PEN node. The experiment is now live and available to the user.

Upon completing the desired experiment, the user clicks the 'Delete Slivers' button on the Flash Interface. This command will activate the DeleteSliver function in the UML component manager. The DeleteSliver function will cycle through all of the allocated nodes, commanding each to reboot, temporarily moving it to the 'reserved' table in the database. In the instance of the PEN node, rebooting and moving to the 'reserved' table is unnecessary. Instead, the DeleteSliver function will call the script pcpen_cancel to stop the virtual node.

Much like pcpen_setup, pcpen_cancel uses the SSH protocol to login to the PEN host to activate OpenVZ commands. Pcpen_cancel collects which virtual nodes must be stopped and passes the information to the ve_cancel script. This script first uses the NFM to close all virtual ports and de-allocate them from the physical ports. Once the virtual ports have been removed, the OpenVZ commands 'stop' and 'destroy' are called respectively. If each step completes successfully, ve_cancel returns success and then pcpen_cancel returns success. Once these steps complete successfully the script pcpen_delete is called, which frees the virtual node entry from the 'reserved' table. After this script completes, the component manager completes the natural sequence of DeleteSliver, and the physical nodes reboot.

4. Conclusion

We have successfully integrated a PEN prototype to ProtoGENI control framework through the customized software modules. The prototype is in operation in the CANS lab at UMass Lowell. We plan to perform addition testing of the prototype and frequently update our software with new ProtoGENI software releases. We welcome trial usage and comments on our PEN-type nodes, and will provide support to third party development.

5. Further Readings

For information about the UMLPEN project, please visit
http://cans.uml.edu/index.php?Research.PEN

For information about ProtoGENI, please visit http://www.protogeni.net/ .

For information about the design of the PEN prototype, please refer to the design document titled "*The Design of a Programmable Edge Node for GENI*" available at the project website http://cans.uml.edu/index.php?n=Research.PEN .

For information about the usage of the PEN prototype, please refer to the usage document titled "*The Usage of a PEN in ProtoGENI*" available at the project website http://cans.uml.edu/index.php?n=Research.PEN .