GENI Component Reference Design

GDD-06-13

GENI: Global Environment for Network Innovations

November 3, 2006

Status: Draft (Version 0.76)

Note to the reader: this document is a work in progress and continues to evolve rapidly. Certain aspects of the GENI architecture are not yet addressed at all, and, for those aspects that are addressed here, a number of unresolved issues are identified in the text. Further, due to the active development and editing process, some portions of the document may be logically inconsistent with others.

This document was prepared by the Facility Architecture Working Group.Editors:

Andy Bavier, Princeton University

Jack Brassil, HP Laboratories

Marc E. Fiuczynski, Princeton University

Contributing workgroup members:

Larry Peterson, Princeton University

John Wroclawski, USC-ISI

Paul Barford, University of Wisconsin

Robert Ricci, University of Utah

Ted Faber, USC-ISI

Jay Lepreau, University of Utah

Steve Schwab, Sparta Inc.

The work is supported in part by NSF grants CNS-0540815 and CNS-0631422.

Table of Contents

Sidebars

This document makes use of sidebars to call out related and unresolved issues. They include:

Unresolved Issue: Identifies an unresolved issue or a topic that requires further development.

Engineering Issue: Addresses an engineering issue, such as performance, reliability, and availability.

Technology Decision: Identifies a concrete technology—protocol, format, language—that implements a function of a GENI component.

Technology Illustration: Identifies a concrete technology or solution that illustrates how a common function might be implemented on a specified GENI component.

Executive Summary

GENI is a geographically distributed facility designed to support the deployment and evaluation of global-scale network architectures and related services that will be qualitatively better than today's Internet. This document describes a reference model for a canonical GENI-compliant component.

A GENI component is a shared, commonly managed facility computing or networking resource. Examples of components include a

- programmable edge node, or PEN (i.e., a conventional compute server),
- programmable core node, or PCN (e.g., a backbone router), and a
- programmable access point, or PAP (e.g., for wireless connectivity).

Though these individual components provide distinct services to researchers or experimenters, they share a set of mandatory functions in support of the GENI Management Core [GDD-06-11]. For example, each component is customizable by authenticated GENI researchers and is remotely manageable by GENI operations staff. As a consequence, a component implementation must support a common set of functions providing for administration, management, and security. This document describes these common component functions.

1 Introduction

The GENI facility is a geographically distributed set of physical computing and network elements (e.g., processors, storage, communication links). A GENI component is a physical element (or set of elements) that is connected to and managed by the GENI facility. Each component is assigned a unique name called a distinguished identifier.

A GENI-compliant component implements a set of mandatory functions in support of the GENI Management Core [GDD-06-11]. The mandatory functions let authenticated GENI researchers access and deploy services to a component to meet their experimental system requirements. A compliant component provides multiplexing functions, which lets GENI researchers use the component in a time-shared manner. The mandatory functions also support isolation, security and privacy of hosted experiments. The mandatory functions also permit GENI operations staff to remotely manage and retain operational control of the component.

This document describes the common, mandatory local functions of a GENI-compliant Programmable Edge Node (i.e., edge compute server), which we present as a reference GENI component. We focus on the set of functions that must be present that let authenticated researchers access and share the component, and the set of functions required to remotely manage the component. Section 2 provides an overview of the component requirements, and Section 3 introduces a component software architecture supporting those requirements. Section 4 describes an illustrative implementation of the component software architecture, identifying specific, existing technologies that could support such an implementation. Section 5 presents a concrete example of a complex GENI site; examining a site helps to identify how component requirements complement GENI site requirements, and also illustrates how the required component functionality can be spread across multiple computing and network elements at a single site. Finally, Section 6 discusses some issues related to component requirements.

2 Requirements

The GMC defines components as the primary building block of GENI. For example, a component might correspond to an edge computer (i.e., PEN), or a programmable access point (PAP). A component encapsulates a collection of resources, including both physical resources (e.g., CPU, memory, disk, bandwidth) and logical resources (e.g., file descriptors, port numbers). These resources can be contained in a single physical device or distributed across a set of devices, depending on the nature of the component. A given resource can belong to at most one component.

Each component exports a well-defined Component Manager (CM) interface through which users create and control slivers on that component. The GENI architecture [PW06] identifies the software and hardware functions that a component must provide to support this interface; these functions include:

1. creating and destroying slivers, binding a set of resources to a sliver, and reclaiming those resources;
2. resource isolation between slivers, so that the resources consumed by one sliver do not unduly affect the performance of another sliver;

3. preventing one sliver from eavesdropping on network traffic to or from another sliver without permission;
4. preventing one sliver from accessing objects (e.g., files, ports, processes) belonging to another sliver;
5. allowing users to install software packages in their sliver without consideration for the packages installed in other slivers running on the same component;
6. allowing users to securely log into a sliver that has been created on their behalf;
7. delivering signals to slivers, including a “reboot” signal that runs whenever the sliver starts up;
8. granting privileged operations to select slivers, including the ability of one sliver to access private state associated with another sliver (thereby supporting sliver interposition);
9. powering the component on and off, and monitoring its hardware and operating system for errors;
10. securely booting the component into an initial configuration;
11. disconnecting the component from the network and bringing it into a safe state;
12. rate-limiting the network traffic generated by a sliver, as well as by all slivers running on the component;
13. for components with access to the Internet, limiting (filtering) how a sliver interacts with (exchanges packets with) the Internet;
14. for components with access to the Internet, providing a mechanism to audit all packet flows transmitted by slivers to the Internet, and determining what sliver (slice) is responsible for a given packet.

Functions 1-5 govern how the resources of the machine are multiplexed among different slivers; each sliver can be regarded as a "container" for a set of resources, whether physical (e.g., CPU, memory) or logical (e.g., TCP ports, files). Functions 6-8 affect how individual slivers are accessed by experimenters, and what extra capabilities a sliver may be granted. Functions 9-11 relates to the management of a physical component, especially in the event of a security incident or complaint; note that functions 9 and 10 supplement requirements found in [GDD-06-11]. Functions 12-14 provide for containment of the traffic generated by a sliver, both to allow for network Quality of Service assurances and to prevent parties outside of GENI from receiving unwanted experimental traffic.

This document describes a combination of software and hardware components for a GENI-compliant PEN that implements the functions outlined above on a single physical computer. Note that, though the PEN satisfies all functional requirements in a single "box", this is not mandatory. For example, while functions 12-14 might be implemented by the OS running on a

component, they might alternatively be realized in a separate device that effectively polices how the component (and the slivers it hosts) interacts with the rest of the network. If such a device is positioned at a choke point for an entire network (or administrative domain), it can enforce appropriate externally visible behavior for a set of components. This issue will be revisited in Section 5, which illustrates how one might implement a GENI Programmable Edge Cluster (PEC).

3 Design of a Programmable Edge Node

A PEN serves as a canonical GENI component, in that it offers a simple and straightforward realization of all the functional requirements of Section 2. This section describes a possible architecture for a PEN capable of realizing the GENI-compliant component requirements specified in the previous section.

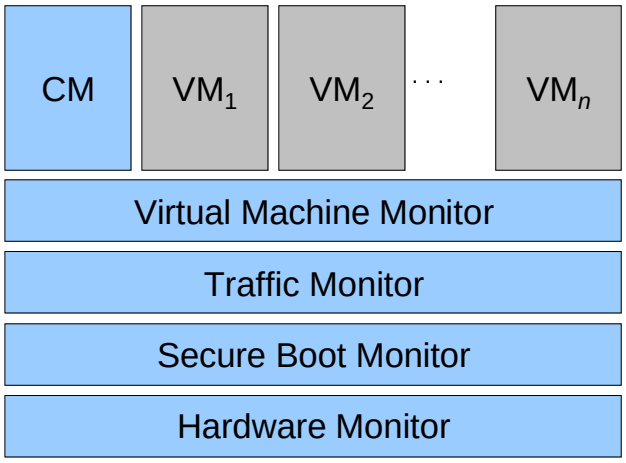

Figure 1: A Component Software Stack

Each GENI-compliant component has an associated Component Manager. The CM learns about changes in the component's state from the GENI Management Core (GMC), as described in [GDD-06-11], and then implements these changes by interacting with various monitoring and management subsystems in the component's software stack. Figure 1 illustrates the component software stack, which consists of the Hardware, Secure Boot, Traffic, and Virtual Machine Monitor subsystems, each controlled by a Component Manager. The figure also shows a number of virtual machines running on the component, each corresponding to an active sliver.

Subsystem	Requirements	Mandatory per component?
Component Manager	1 - 8	No (see Section 3.1)
Virtual Machine Monitor	1 - 8	No (see Section 3.2.6)

Traffic Monitor	12-14	No (see Section 4.2.2)
Boot Monitor	10, 11	Yes
Hardware Monitor	9	Yes

Table 1: Mapping Requirements to Component Subsystems

Table 1 provides a mapping of the component requirements listed in Section 2 to the various subsystems in the component's software stack. Every GENI-compliant component must at least implement the Hardware and Boot Monitor subsystems and Component Manager, so that it can be securely powered on, booted and initialized. A typical component will also support the Traffic Monitor and Virtual Machine Monitor subsystems, however we will show that these subsystems are not mandatory in certain special cases. The subsequent sections provide a description of each subsystem.

3.1 Component Manager

The Component Manager is the GENI Management Core's primary point-of-presence on the component. The job of the Component Manager is to manage slivers. It receives updates from the GMC and executes operations on the component. For instance, the GMC might invoke the CM to use the Virtual Machine Monitor (VMM) to create a sliver and load it with the credentials a researcher instantiating an experiment. The CM itself might execute in a virtual machine with sufficient privileges to execute the necessary VMM operations.

Though anticipated to be a rare use case, certain components can operate in a non-virtualized mode. In such a circumstance the component is not required to run a VMM (see Section 3.2.6), and hence may not have a CM either. Instead, researchers booting a custom image on a component are responsible for administering all aspects of its operation after boot. For example, the boot image needs to supply a secure authentication mechanism and the necessary credentials so that the researchers can log into the component after it comes up.

3.2 Virtual Machine Monitor

GENI components are composed of physical elements such as processors, memory, storage, network interfaces, wired and wireless communication links, sensors, etc. A primary goal of GENI is to share components between multiple experiments simultaneously. Therefore a GENI-compliant component must be able to partition/contextualize, allocate and schedule physical resources (cycles, bandwidth, memory, and storage) to prevent one sliver from interfering with another or gaining access to information in another sliver. Moreover, slivers must be able to request resource reservations to receive soft real-time performance guarantees. The Virtual Machine Monitor (VMM) is the "operating system" that lets one or more slivers run concurrently on a component. The VMM supports the multiplexing, isolation and security and privacy, and component-specific performance requirements of a GENI-compliant component.

> Unresolved Issue: It is unclear what the 'operating system' is for specialized components, whose hardware is architecturally different from

> conventional compute servers, such as network processors and FPGAs used in routers, switches, and access points.

A PEN could run any of several candidate operating systems: hypervisors, container-based operating systems, and user-mode virtualization. There is no requirement to specify a unique GENI edge server VMM; other existing experimental testbeds including both Emulab and PlanetLab can currently bootstrap multiple operating systems. Moreover, supporting multiple technologies is not mutually exclusive, and can provide complementary Virtual Machine Monitor services. One experimenter could choose to use a hypervisor-based system where one virtual machine would run a fully GENI-compliant container-based operating system with its own CM, while another experimenter could run a container-based operating system with an associated CM running in a separate container (e.g., a Gateway).

A key responsibility of the VMM is to allocate resources among multiple slivers. For resources that support it, the VMM implements resource *reservations* and proportional resource *shares*. A reservation entitles the sliver to the reserved amount of the resource but no more. A share provides a means of "over-subscribing" a resource among experiments while still providing some degree of fairness and isolation; a sliver's share entitles it to best-effort access to the a resource in proportion to the sliver's share divided by the sum of shares of slivers contending for that resource. An experiment will subscribe for a reservation if it requires consistent access to that resource (e.g., for repeatability), whereas it will request a share if it can tolerate variability.

3.2.1 Sharing the CPU

The VMM is responsible for scheduling multiple slivers on the component's processors. The CM can assign each sliver a CPU *reservation*, a CPU *share*, or both. A CPU reservation entitles the sliver to a fraction of the CPU's capacity. Reservations are *soft real-time* performance guarantees, meaning that the VMM makes a reasonable effort to deliver the reserved capacity to the sliver at a fine-grained timescale, but occasionally may not do so. A CPU reservation also provides an upper limit on the amount of CPU cycles that a sliver can receive; for example, a reservation of 10% means that the sliver will receive up to, but not more than, 10% of the CPU cycles on the machine.

> Unresolved Issue: What sort of soft real-time performance guarantees do GENI experiments require? How should the capabilities of different alternatives be quantified and compared?

A CPU share entitles the sliver to a share of the unused CPU capacity (i.e., unreserved CPU + reserved CPU not currently in use) in proportion to its numerical share. A single sliver can possess both a CPU reservation and a share. In this case, the sliver will receive its reserved CPU percentage plus a proportion of the unused capacity.

> Unresolved Issue: What are the semantics of CPU reservations and shares on a multi-processor? We expect that many GENI components will have more than one processor or core.

3.2.2 Sharing the Network

Incoming packets must be de-multiplexed to the correct sliver. If possible, the VMM associates a unique Layer 3 address with each sliver on the machine (e.g., an IPv6 address). In cases where this is impractical, the VMM may use a single Layer 3 address for multiple slivers, and de-multiplex packets based on Layer 4 information (e.g., use a single IPv4 address and de-multiplex on TCP/UDP port number).

> Unresolved Issue: How does a component implement or participate in per-experiment virtual topologies?

The Virtual Machine Monitor is responsible for tagging packets sent by a sliver with the sliver's ID. This information is used by the Traffic Monitor to authorize, shape, and audit the sliver's outgoing traffic as described in Section 3.3.

The CM can specify the scheduling and shaping of a sliver's network traffic; this is discussed in more detail in Section 3.3.2.

3.2.3 Sharing the File System

The VMM isolates slivers into different file systems, or different sub-trees of the same file system. A process in one sliver should not be able to read or write the files in another sliver without permission.

The CM can allocate a disk *reservation* to a sliver that both allocates a fixed amount of disk space to the sliver and limits it from using more than this amount.

> Unresolved Issue: What is the meaning of giving a disk *share* to a sliver? What do we give to slivers that don't require a reservation?

3.2.4 Sharing Physical Memory

The VMM is responsible for allocating the available physical memory to slivers. Memory consumption by one sliver should not overly affect the performance of other slivers. The CM can assign each sliver a memory *reservation* or a *share*. A memory reservation binds a fixed amount of physical memory to that sliver. A memory *share* entitles the sliver to a proportional share of the unreserved physical memory on the machine.

> Unresolved Issue: What are the semantics of a share of physical memory?

3.2.5 Interaction with Slivers

The CM loads a new sliver with the researchers' credentials, and runs the appropriate software to provide a secure access method.

- Granting special privileges (e.g., Proper)
- Signals (e.g., reboot)

3.2.6 Bare Hardware

The demands of an experiment may be cause the simultaneous sharing of a component with other GENI experiments to be undesirable or impractical. For example, the underlying VMM may not be able to provide the hard real-time resource guarantees required by an experiment, or it may not be possible to virtualize a particular piece of hardware (such as an FPGA). In these cases, the Boot Manager could be used to bootstrap an operating system directly onto the physical hardware that gives exclusive control to a single experiment. For example, researchers who require a hard real-time operating system could boot the one of their choice on the machine with no virtualization layer underneath. However, the component must still provide a Hardware Monitor that supports, at minimum, powering the component up and down.

> Unresolved Issue: Though the assignment of bare hardware is feasible, scaling GENI to large numbers of experiments suggests that this practice should be strictly limited. How will this resource control be accomplished?

3.3 Traffic Monitor

The Traffic Monitor is responsible for authorizing, shaping, and auditing network traffic that is sent and received by a component. While it is possible to engineer the Traffic Monitor to support authorization, shaping, and auditing in any order, this specific order for sent or received traffic is expected to impose: 1) lower overhead and 2) better time correlation for auditing. Lower overhead is achieved because authorization will filter erroneous traffic (e.g., spoofed packets, etc.) before it reaches the other two layers. Moreover, one can achieve better time correlation, as the auditing time stamp associated with a packet will be correlated with the actual time the packet leaves the component.

> Unresolved Issue: It there a need for a Traffic Monitor to shape and audit traffic received by a component?

While traffic monitoring for IP-based traffic is well understood, the GENI facility will support non-IP protocols and addressing schemes that will need to be supported by the Traffic Monitor. The remainder of this section describes the Traffic Monitor services for IP traffic, and addressees certain issues related to non-IP based traffic.

3.3.1 Authorization

The Authorization subsystem of the Traffic Monitor basically is an inverted firewall. Its goal is to protect the global communication infrastructure from a GENI component, rather than the other way around. For IP-based traffic it has two basic responsibilities: 1) prevent GENI components from sending packets to blacklisted networks, sites, or nodes, and 2) prevent source address / port spoofing such that the auditing subsystem can correctly associate outgoing traffic with the originating sliver. This can be implemented using widely available mechanisms available in most modern operating systems and routers.

One goal of GENI is to let researchers experiment with new layer-3 protocols and addressing schemes. The Traffic Monitor will not filter unrecognizable traffic that appears to be contained to the GENI facility networks.

> Unresolved Issue: The inverted firewall could support new layer-3 protocols if it were designed with a template based record format that lets it recognize new packet types. This functionality will also be required if new layer-3 and IP-based layer-4 protocols need to be shaped and audited.

3.3.2 Shaping

The Traffic Monitor's shaping subsystem is responsible for fair sharing of bandwidth and enforcing bandwidth reservations. For best effort communication, offered traffic must be forcibly shaped or deferred only when the network link utilization is moderate to heavy (approximately > 80%). If the aggregate traffic offered by slivers is always less than the capacity of the communications facility and less than the aggregate of the requested bandwidth reservations, then all traffic is carried without invoking bandwidth-shaping mechanisms. However, when a communications resource becomes oversubscribed, some traffic must be shaped through the deferral or discarding of some part of the offered traffic. Bandwidth shaping is fairly well understood and widely available in modern operating systems, switches, and routers.

> Unresolved Issue: Is this bandwidth shaping approach sufficient?

3.3.3 Auditing

Auditing outbound network traffic is an essential requirement for GENI, since any traffic that is sent by a component is a possible source of problems. An audit trail (or log) must let the GENI operations staff identify the sliver that generated it. For this reason, an auditing system must have the ability to bind network traffic generated by a component to a sliver (and thereby to its principal owner).

The auditing subsystem will need to recognize new protocols and addressing schemes to properly bind traffic sent from a component to the originating sliver. In essence, to future-proof the auditing system, an extensible packet recognition system is needed so researchers can define and register their protocols and addressing schemes.

3.4 Secure Boot Monitor

The GMC exports a Management Authority (MA) interface that is contacted by the Component Manager's Boot Monitor subsystem whenever a component reboots. Each component bootstraps from immutable media that instantiates a component-specific Boot Manager onto the component.

The Boot Manager, in turn, contacts the MA's Boot Server to download all necessary code and configuration information needed to fully boot the component. The immutable media also includes a set of certificate authority (CA) certificates corresponding to well-known root public keys. The Boot Monitor uses this set of CA certificates to authenticate the Boot Server. Note

that the bootstrapping code and certificates can be openly distributed on a CD-ROM, for example, and used to securely bootstrap any component. The certificates are GMC MA-specific, but not component-specific.

3.5 Hardware Monitor

Hardware sometimes fails, and there may be bugs in the component's software stack. The purpose of the Hardware Monitor is to make it as easy as possible to remotely manage and monitor a component given these realities. The Hardware Monitor provides a secure method for the Management Authority staff to:

- power the component up and down,
- monitor the component for software and hardware errors (e.g., remote console), and
- monitor the physical state of the component (e.g., temperature sensors).

> Unresolved Issue: What other features are required or likely to be useful for low-level remote management?

4 An Illustrative Programmable Edge Node Component

This section describes a concrete example of one component type – a PEN. Several existing experimental systems (e.g., Emulab, PlanetLab) have comparable elements that inform our discussion. The primary purpose of this section is to illustrate how mandatory GENI-compliant component functions could be realized in this familiar setting. It begins with the hardware configuration for an Edge Server and then proceeds with a bottom-up description of a GENI-compliant software implementation.

4.1 PEN Hardware

A *typical* edge node is a modern rack-mountable server consisting of the following hardware:

- modern processor (likely with multiple cores)
- 8 GB of RAM
- 1 TB of disk storage
- 2 NICs each with GigE Ethernet support
- BIOS capable of booting from USB flash media
- USB 2.0 port(s)
- System management hardware (providing remote KVM, virtual CD-Rom, temperature and fan speed sensors, powercycle / reboot / shutdown).

- Technology Illustration: Many server vendors integrate remote management support with their systems (e.g., Intel Intelligent Platform Management Interface, HP Integrated Lights Out, etc.) .

- Reliable-system support

  - Technology Illustration: Seamless process migration from one physical server to another is supported by virtual machine monitors such as Vmware and Xen, and container-based operating systems such as OpenVZ (a feature expected to be added to Linux Vservers).

The system management hardware described above provides the Hardware Monitor subsystem for the PEN component. Additionally, the edge node should provide cryptographically protected storage to hold the GMC MA generated share key and serve as the immutable media for the Secure Boot Monitor. A straightforward method is to use a tamperproof USB key, which currently integrate 256-bit AES encryption directly and 256MB variants cost $20 or less. Alternatively, this could be accomplished via an integrated TPM (trusted platform module) chip.

The approximate FY2006 price for such a server appropriately equipped is less than $4K (where the cost of dense 2G RAM sticks make up the bulk of the cost).

4.2 Component Software

4.2.1 Boot Monitor

The GMC exports a Management Authority (MA) interface that is contacted by the component's Boot Monitor subsystem whenever a component reboots.

When a component is registered with the MA, a secret key is generated for that component, stored in an MA database, and then exported to the component as part of the component configuration file. The component key can subsequently be used as a shared secret by the Boot Monitor to authenticate the component to the MA. We assume that the configuration file is cryptographically protected in a manner that only the MA-provided Boot Manager can decrypt.

> Engineering Issue: For components operated in physically insecure environments, it will be necessary to use additional mechanisms such as a secure BIOS [aegis] and TPM [tpm] to ensure secure bootstrap to the Boot Monitor. Without such a mechanism it would be possible for an unauthorized user with physical access to the component to gain access to the shared secret key after it has been decrypted.

Figure 3 illustrates the overall component bootstrap procedure. After a component boots (steps 1-3), the Boot Monitor contacts the MA securely to load the Boot Manager (steps 4-6). The bootstrap through step 6 is secure, because the Boot Monitor uses the set of CA certificates to authenticate the MA's Boot Server, downloads the Boot Manager code (e.g., using SSL), and validates that the cryptographic signature applied to the Boot Manager code matches the CA certificates before transferring control to it. In steps 7-9, the Boot Manager reads the secret shared key from the immutable media, which it will use to authenticate itself to the MA's Boot

Server (e.g., via HMAC) and learn what operational state it should put the component (step 10). The operational states include install (load all necessary code and configuration information onto the component's storage media), boot (chain boot into the currently installed operating system), and debug (open a secure channel to let GENI operations staff debug the hardware or perform forensic analysis). Assuming the component is in the install state, after installation completes, the Boot Manager will update the components state with the Boot Server and chain-boot to the install operating system (steps 11-13). Otherwise, if the node already is in boot state, the Boot Manager skips steps 11 and 12. Finally, the node is fully booted in step 14 with the component specific production Machine Monitor, which will be described later in section 3.2.

4.2.2 Traffic Monitor

A component's Traffic Monitor is responsible for authorizing, shaping, and auditing network traffic that is sent and received by a component.

> Technology Illustration: Slivers on PlanetLab nodes manage shaping and auditing of received traffic at the application-level, while transmitted traffic is shaped an audited at the kernel-level. Regardless of the decision on this issue, system supported shaping (e.g., Linux HTB [htb], etc.) and auditing (e.g., netlink-based ulogd [netlink,ulogd]) must be made available to slivers by the component's VMM .

4.2.2.1 Authorization

The Authorization subsystem of the Traffic Monitor can be implemented using widely available mechanisms available in most modern operating systems and routers.

> Technology Illustration: IPtables can restrict destination traffic for a sliver on a Linux-based edge server. Modern routers (e.g, Cisco) and switches (Foundry with layer-3/4 packet inspection enabled) support traffic ACLs that can restrict destination traffic from a given source address/port or network interface port.

This inverted firewall should by default drop all unauthorized layer-2 traffic. Most conventional components will predominately use IEEE 802.x based protocols (i.e., Ethernet) for layer-2 connectivity (exceptions being new ideas in wireless and direct lambda access). On untagged 802.x networks, unauthorized traffic consists of packets with spoofed source MAC addresses. This can be addressed by implementing an 802.1x authentication scheme, likely leveraging the shared secret key that is generated when a component is registered with the GMC Management Authority and read in by the Boot Monitor on system boot (viz. section 3.4).

On 802.1q tagged (VLAN-based) networks, unauthorized traffic consists of packets sent by components with VLAN tags that are outside the membership list associated to the port on the switch/router to which the component is connected.

> Unresolved Issue: VLANs labels are scarce resources due to their 12-bit address space. Letting slivers dynamically join/leave VLANs will need to be carefully managed and limited. An alternative approach is to define

all layer-2 networks that slivers can address using Multi-Protocol Label Switching (MPLS), which sits between layer-2 and layer-3 (in that it is independent of the layer-2 definition). MPLS assigns each packet a stack of labels, with each stack entry supporting a 32-bit label (of which 20 bits are available for defining the label space), which should suffice for managing a large number of virtual networks.

4.2.2.2 Traffic Control and Shaping

The Traffic Monitor's shaping subsystem is responsible for fair sharing of bandwidth and enforcing bandwidth reservations. For best effort communication, offered traffic must be forcibly shaped or deferred only when the network link utilization is moderate to heavy (approximately > 80%). If the aggregate traffic offered by slivers is always less than the capacity of the communications facility and less than the aggregate of the requested bandwidth reservations, then all traffic is carried without invoking bandwidth-shaping mechanisms. However, when a communications resource becomes oversubscribed, some traffic must be shaped through the deferral or discarding of some part of the offered traffic. Bandwidth shaping is fairly well understood and widely available in modern operating systems, switches, and routers.

Technology Illustration: The Linux kernel incorporates a hierarchical token bucket system that can be used to shape and prioritize layer-2 to layer-4 traffic. For example, PlanetLab nodes use this mechanism to both ensure fair sharing and guarantee reservations.

Technology Illustration: An Ethernet switch supporting VLAN technology can be configured to impose bandwidth limits on a per interface basis.

Of course, only recognizable traffic can be shaped. For new layer-3+ protocols / addressing schemes that are not recognized by the shaping subsystem, we assume that the traffic is contained within a VLAN (and enforced by the inverted firewall) and fair sharing / reservations is applied at layer-2.

Unresolved Issue: The degree of support for real-time traffic has not been determined. Hard and soft real-time are marked by the specification of deterministic or statistical service guarantees. A mechanism must be developed to create virtual links that are appropriately over provisioned with a guarantee that aggregate bandwidth going over such a link avoids the need for bandwidth shaping. It is unclear what (if anything) must be done to properly support soft real-time communication.

4.2.2.3 Auditing

As discussed earlier, auditing network traffic is an essential requirement for GENI. Any traffic that is sent by a component is a possible source of problems. An audit trail (or log) must let the GENI operations staff identify the sliver that generated it. For this reason, an auditing system

must have the ability to bind network traffic generated by a component to a sliver (and thereby to its principal owner).

For IP-based traffic, auditing is well understood. For example:

> Technology Illustration: PlanetFlow [OSR] and Cisco's NetFlow log the header of each IP-based packet sent by a component and classifies the packets into flows, using a 5-tuple of <source IP, source port, destination IP, destination port, protocol>. These tuples are augmented with start and end times, total number of packets, and total number of bytes. PlanetFlow augments the start and end times of such tuples with a sliver identifier, which thereby lets one attribute traffic to a specific sliver on hosts where multiple slivers share a single IP address. PlanetFlow does not keep track of the precise timing of each packet, other than the first and last packets within a session---please see [OSR] for details.

The auditing subsystem will need to recognize new protocols and addressing schemes to properly bind traffic sent from a component to the originating sliver. In essence, to future-proof the auditing system, an extensible packet recognition system is needed so researchers can define and register their protocols and addressing schemes.

> Technology Illustration: Cisco's NetFlow traffic logging system utilizes a template-based record format that lets it recognize new layer-3+ packet types.

4.2.3 Virtual Machine Monitor

The Virtual Machine Monitor supports the multiplexing, isolation and security and privacy, and component-specific performance requirements of a GENI-compliant component.

A hypervisor such as Xen provides maximal flexibility in that they provide the abstraction of the raw hardware to each experiment. It also provides strong isolation between experiments, as dedicated hardware resources can be assigned to each experiment. A drawback is that these systems come with a very limited programming environment; in theory experimenters would essentially be given little more than virtualized bare hardware, though in practice most experimenters will likely go with a variant of XenoLinux slightly modified for their experiment. Another drawback is that each sliver in a Xen VM would need to obtain a globally unique IPv4 address until some solution is developed to share such an IPv4 address between VMs, as was done for PlanetLab's container-based operating system (Linux Vserver) using VNET [VNET].

Another possible solution for the Virtual Machine Monitor on an edge server is a container-based operating system (e.g., Linux Vservers, Sun's OpenSolaris 10, or OpenVZ), which provides each experimenter the illusion of a virtual dedicated Unix operating system with root permission. Software inside these container-based systems prevents each container (i.e., a sliver) from exceeding its privileges. Soltesz et al. [Soltesz] demonstrate that Linux Vservers provides strong isolation, better scalability, and better performance compared to Xen, especially when it comes to networking and disk i/o related benchmarks. Moreover, Bavier et al. [VINI] demonstrate that soft real-time scheduling on Linux Vservers can be used to achieve low latency network packet forwarding on shared systems that are under moderate to heavy load.

But performance isolation is potentially limited as Vservers makes an intentional design decision to share the buffer cache (unlike Xen), which opens it up to cache poising attacks. The main drawback of using a containerized approach is that researchers cannot insert kernel modules.

> Engineering Issue: Approaches like Vservers and Xen each provide their own levels of resource isolation, e.g., though resource scheduling and partitioning. We must be able to measure the provided isolation along different dimensions, with the goal of enabling as much of an "apples-to-apples" comparison as possible.

4.2.3.1 Bare Edge Node Hardware

Rather than running an arbitrary operating system within a virtual machine, the Boot Manager could be used to bootstrap such an OS directly onto the physical hardware. This would be useful to researchers who want to experiment with hard real-time operating systems, as they would obtain complete control of the physical hardware. This approach requires the association of a second resource to serve as the physical machine's CM, implementing traffic management, auditing, remote management, etc services in support of the bare edge server. Since additional resources are required in support of bare edge node hardware, we anticipate that the availability of bare hardware will be initially limited to compute elements (e.g., blades) within Programmable Edge Clusters.

> Technology Illustration: Cluster management systems (e.g., Rocks) by default isolate their compute nodes from the public network, such that all of their traffic must go through "head" node. This head node acts as router, NAT, and firewall. A similar concept can be used in GENI to authorize, shape, and audit traffic from a bare edge server.

4.2.4 Component Manager

The Component Manager is a daemon that runs in a privileged virtual machine on top of the VMM.

4.2.5 Instrumentation and Measurement Support

The facility supports the creation of GENI-compliant measurement components that provide component-specific measurement services to experimenters. In addition, self-instrumentation of non-measurement components (e.g., programmable edge router) is both supported and encouraged.

As an example, the GENI edge server components can incorporate an optional network instrumentation element (e.g., Endace DAG 1GigE card). This type of hardware -- already in use by the ETOMIC subproject of the EU's EVERGROW Integrated project -- would let researchers carry out network measurements between geographically distributed sites with high temporal resolution (~10 nanoseconds) that is globally synchronized. It would provide GENI with a high

resolution, spatially extended dynamic picture of fast changes in network traffic, thereby open up new kinds of network tomography.

> Unresolved Issue: The cost of universal deployment of these instrumentation cards might be prohibitive, though deployment on a per-site basis might be feasible. A resource scheduling mechanism should permit researchers conducting timing experiments to get higher priority to share components equipped with optional measurement elements.

5 An Illustrative GENI Site

5.1 Example of a Complex GENI Site

In this section we present a concrete example of a complex GENI site. A GENI site has one or more co-located GENI components with common locally administration. Though this section serves to illustrate how some of the mandatory GENI-compliant component functions could be realized, its primary purpose is to expose system-level design issues and highlight how mandatory functions address these challenges.

We expect the GENI facility to grow to 100-200 edge sites, each of which will host a programmable edge cluster (PEC) component. These PECs will vary in complexity from two or three processors connected by a low-end commodity switch to a large cluster potentially including hundreds of processors and a high-performance network interconnect. Each site will be connected to the backbone by at least one tail circuit to a backbone PoP. These tail circuits will exploit a wide range of technologies, including dedicated fiber, leased layer-2 circuits, and tunnels through the commodity Internet. Each PEC will also be connected to the commodity Internet via whatever connectivity is provided by the hosting site.

Figure 3 depicts a site containing N+2 GENI-compliant components: a programmable edge cluster (in this case a single bladed chassis supporting N edge servers), a standalone edge server, and a programmable access point. Each GENI-compliant component has an associated CM, which is either implemented entirely locally or distributed across physical resources. Each edge server on a PEC blade runs a CM, as does the standalone edge server. The access point supports a distributed CM; the AP relies on a dedicated PEC blade to run certain CM functions that are not supported locally.

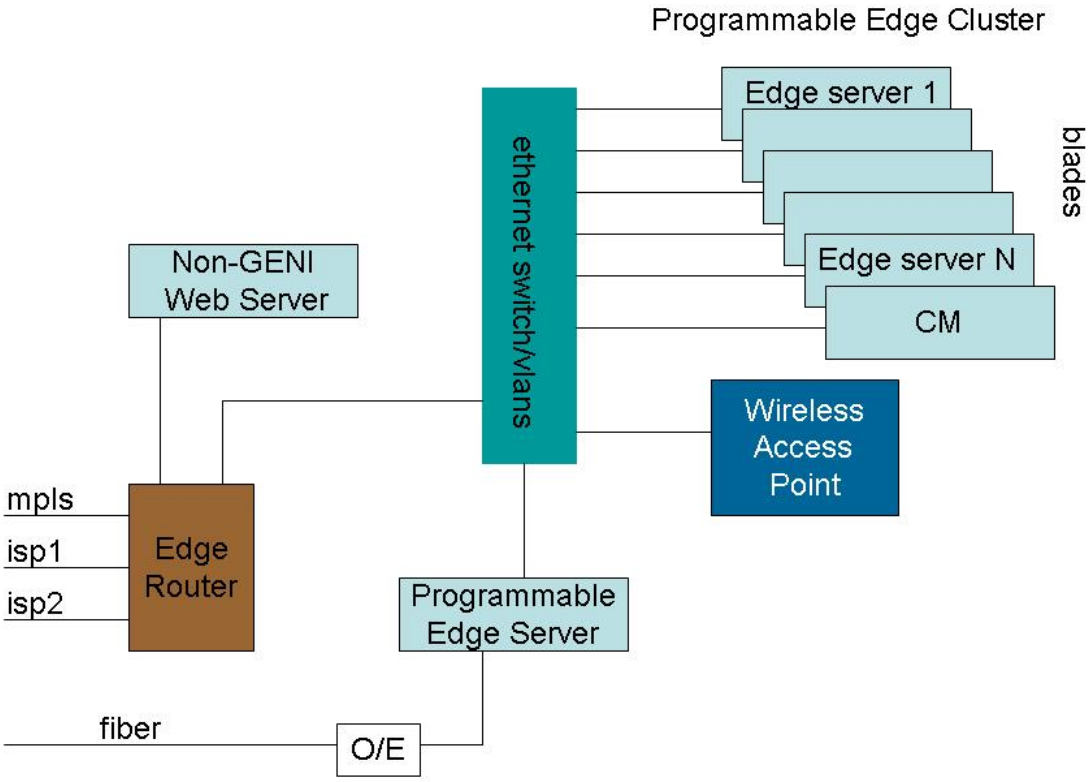

Figure 3: A representation of a complex GENI site.

The depicted GENI site also contains additional computing and networking resources including

- VLAN switch – This device serves as the principal local interconnect for the site's GENI components. It is controlled, in part, by the CMs of locally attached components. For example, a CM might issue a command to the switch to impose a bandwidth limitation on the amount of egress traffic on the switch's port to the Edge Router.
- Edge Router – This device is a conventional COTS router used to connect the GENI site to access networks. This is not a GENI-component, and hence not programmable by experimenters. The ER is administered by local site administration.
- Web Server – This device is representative of non-GENI computing or networking equipment that might be co-located at the site. This device shares at least one access network through the ER with the GENI components on site. Site administration is responsible for ensuring isolation of these resources from GENI components.

Multiple communication links connect the site to external access networks. In general, a CM audits and controls access to external networks on a per link basis.

The basic hardware building blocks for GENI sites include edge servers, Ethernet switches for intra-site connectivity, and routers for inter-site connectivity. Switches and routers are constructive examples of site elements that can be exploited to implement certain CM functions such as Traffic Management. The next section provides some high-level requirements on Ethernet switches that can facilitate site and component design; a similar set of requirements can be identified for COTS routers.

5.1.1 Ethernet Switch

A Gigabit Ethernet switch should have the following capabilities:

- >= 48 port copper 1GigE switch. Note that a canonical GENI cluster will likely have 20 servers with two NICs each.
- two (2) 10GigE uplink ports. For a 20-server cluster, provide each with full 1GigE up / down bandwidth to the backbone.
- VLAN (auto-configuration and QoS bandwidth support): a switch supporting VLAN technology with QoS bandwidth management will enable one to attach a non-GENI compliant component in a manner that it cannot see or affect traffic belonging to GENI-compliant components, and more importantly can be forced to send traffic thru at least one local GENI-compliant component (e.g., a sliver running on a compliant edge server) in order to communicate with the rest of GENI.
- secure remote management interface (e.g., https, ssl): remote management support will let operations staff configure the switch remotely. More importantly, a Component Manager's Traffic Monitor could securely program the switch to enforce bandwidth limits for VLANs associated to slivers.

Examples of switches that have these capabilities are the Foundry FastIron WorkGroup Switch X448 and the HP ProCurve 3400cl-24G + cl 10-GbE CX4 module. The general retail price range for such a switch is less than $10K, which includes the (expensive) two-port 10GigE uplink module.

6 Related Topics

6.1 GENI Gateway

A component's Traffic Monitor is responsible for authorizing, shaping, and auditing the component's network traffic (Section 4.2.2).

In some settings it is advantageous to have a single Traffic Monitor perform traffic management functions for multiple components. For example, it might be desirable for a single TM to consolidate and perform traffic management services for all of the components at a small GENI site. In such a case, a desirable location for the consolidated TM function would be at a 'choke point' where the site connects to access networks (e.g., a GENI tail circuit, or the internet).

A GENI Gateway refers to the logical set of functions that implement cross-component TM services at a choke point.

As a simple example, consider the GENI site depicted in Figure 3. In one possible implementation, each node in the programmable edge cluster could itself operate as a separate component (i.e., forming N PENs) and run a separate, local Traffic Monitor function. This implementation is similar to that deployed by Planetlab nodes, where each of multiple nodes at a site separately performed traffic management functions.

However, an alternative implementation could consolidate these functions. For example, TM functions for the entire set of components could be implemented primarily within the ethernet switch, with the nodes' CMs controlling the creation of VLANs and native switch-based traffic control services. The 'choke point' in this setting is the switch port connecting the site to the internet.

In this example the GENI gateway is a logical set of traffic management functions distributed across CMs and the Ethernet switch. This example highlights that a gateway 1) is not a mandatory site element, and 2) is not a separate, physical device.

6.2 Aggregate Components

An aggregate component is a set of GENI-compliant components that can be referenced with a single, unique identifier. An aggregate facilitates interactions with multiple components. A possible use of an aggregate is to jointly manage or use a set of components that share a common property; examples might include all the edge servers in a single PEC (e.g., blades in a single chassis), or all components at a specified site. An Aggregate Manager (AM) plays a role comparable to a CM in managing the aggregate.

6.3 Making a Component GENI-Compliant

We anticipate that the research community will develop novel components that would benefit the GENI community if integrated with GENI facility. The decision to add such a component to the facility is granted by a GENI management authority and is beyond the scope of this document. Here, however, we sketch a technical approach to enable an arbitrary component to be enhanced to compliance.

The initial step to GENI-compliance is to satisfy the mandatory isolation, multiplexing, and network traffic management requirements of Section 2. Satisfaction of these requirements will depend, in general, on the component type and component performance limitations. For example, suppose a developer would like to enhance a programmable wireless access point to achieve GENI compliance. Three approaches may be considered:

1. The desired component's core function may be added to an existing GENI-compliant component.

    i. Technology Illustration: A core function implemented as an expansion card can be integrated within a PEN and made visible from a sliver.

2. The component may be modified to incorporate the CM software stack to realize GENI-compliance.

3. The component may be associated with other resources running services on its behalf to realize compliance. For example, the component might rely on a CM running as a VM in another component.

Acknowledgements:

This document draws heavily from experience gained in developing, operating and using experimental systems including PlanetLab [NSDI04,OSDI06]. We also acknowledge comments and suggestions from Tom Anderson and Distributed Services Working Group membership.

References

[1] [GDD-06-11] Larry Peterson, John Wroclawski (Eds), "Overview of the GENI Architecture," GDD-06-11, Facility Architecture Working Group, September 2006.

[2] [GDD-06-24] Thomas Anderson (Ed), "GENI Distributed Services," GENI Design Document 06-24, Distributed Services Working Group, September 2006.

[3] [AKA] Akamai Inc., www.akamai.com

[4] [GDD-06-07] Larry Peterson (Ed), "GENI: Conceptual Design, Project Execution Plan," GENI Design Document 06-07, GENI Planning Group, January 2006.

[5] [GDD-06-08] Larry Peterson (Ed), "GENI Design Principles," GENI Planning Group, March 2006 (updated August 2006).

[6] [GDD-06-09] Jonathan Turner, "A Proposed Architecture for the GENI Backbone Platform," March 2006.

[7] [GDD-06-10] Jim Basney, Roy Campbell, Himanshu Khurana, Von Welch, "Towards Operational Security for GENI," GDD-06-10, July 2006.

[8] [GDD-06-12] Paul Barford (Ed), "GENI Instrumentation and Measurement Systems (GIMS) Specification," GENI Design Document 06-12, Facility Architecture Working Group, September 2006.

[9] [GDD-06-15] Dipankar Raychaudhuri, Sanjoy Paul, "Requirements Document for Management and Control of GENI Wireless Networks," GENI Design Document 06-15, Wireless Working Group, September 2006.

[10] [GDD-06-22] Chip Elliott, "System Engineering Document for GENI Suburban Wireless Network," GENI Design Document 06-22, Wireless Working Group, September 2006.

[11] [GDD-06-23] Thomas Anderson and Michael Reiter, "GENI Facility Security," GENI Design Document 06-23, September 2006.

[12] [GDD-06-26] Dan Blumenthal and Nick McKeown, "Optical Node Architecture," GENI Design Document 06-26, Backbone Working Group, September 2006.

[13] [GMC Spec] Ted Faber (Ed), "GMC Specifications," http://www.geni.net/wsdl.php, Facility Architecture Working Group, September 2006.