Facility Design

GDD-07-44

GENI: Global Environment for Network Innovations

March 9, 2007

Status: Conceptual Design (Version 1.0)

This document is prepared by the GENI Planning Group

Larry Peterson, Princeton (Editor)

Tom Anderson, Washington

Dan Blumenthal, UCSB

Dean Casey, Ngenet Research

David Clark, MIT

Deborah Estrin, UCLA

Joe Evans, Kansas

Nick McKeown, Stanford

Dipankar Raychaudhuri, Rutgers

Mike Reiter, CMU

Jennifer Rexford, Princeton

Scott Shenker, Berkeley

Amin Vahdat, UCSD

John Wroclawski, USC/ISI

We'd also like to acknowledge contributions, comments and suggestions from all the GENI working groups, with a special thanks to

Ted Faber, USC/ISI

Chip Elliot, BBN

Mic Bowman, Intel

Sanjoy Paul, Rutgers

Robert Ricci, Utah

Arvind Krishnamurthy, Washington

Sampath Rangarajan, Lucent

Stephen Schwab, Sparta

Alefyia Hussain, Sparta

Paul Barford, Wisconsin

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

Table of Contents

1 Introduction

GENI—a Global Environment for Network Innovations—is an experimental facility intended to enable fundamental innovations in networking and distributed systems. The essence of the GENI facility is its ability to rapidly and effectively embed *within* itself a broad range of experimental networks, interconnect these experimental networks as appropriate with other experiments and the existing Internet, provide these networks with users and an operating environment, and rigorously observe, measure, and record the resulting experimental outcomes. The proposed facility will span a range of extant and emerging technologies (e.g., wireless sensors, mobile wireless, high function optical), layers of network architecture (e.g., physical to network to network services), geographic reach (e.g., wearable personal-area networks to wide-area networks), and application domains (e.g., high bandwidth, compute intensive e-science to low bandwidth, low duty-cycle sensor applications to large scale information dissemination).

GENI provides these capabilities through an innovative combination of techniques: *virtualization*, *programmability, controlled interconnection,* and *modularity.* Virtualizing the physical hardware allows multiple network architectures and services to run simultaneously. This includes long-running services and applications that attract real users, which results in realistic evaluations and drives adoption and deployment. Programmability allows clean-slate designs to run side-by-side with incremental experiments. Controlled interconnection allows a wide variety of experiments to build on and interoperate with each other, while providing for each experiment an appropriate controlled-risk environment. GENI's modular design structure explicitly accommodates the evolution of new building block technologies over time. This same structure supports federation, which allows other countries and research communities to "plug into" GENI.

This document, building on the research plan and facility requirements presented elsewhere [GDD-06-28], describes the design of the GENI facility. It introduces the key hardware and software elements that make up GENI, along with the system architecture that combines these elements into a coherent facility. The document also sketches a reference design for the various pieces that define the facility, which forms the basis for a concrete budget and construction plan, which are also presented elsewhere [GDD-07-45].

1.1 System Architecture

The overall facility architecture can be divided into three levels, as illustrated in Figure 1.1. At the bottom level, GENI provides a set of physical facilities (e.g., routers, processors, links, wireless devices), which we refer to as the *physical substrate*. The design of this substrate is concerned with ensuring that physical resources, layout, and interconnection topology are sufficient to support GENI's research objectives. As technology evolves, it is expected that the set of network technologies included in the substrate will evolve as well. A key goal of the overall GENI architecture is to enable and accommodate the evolution of these technologies throughout the lifetime of the facility.

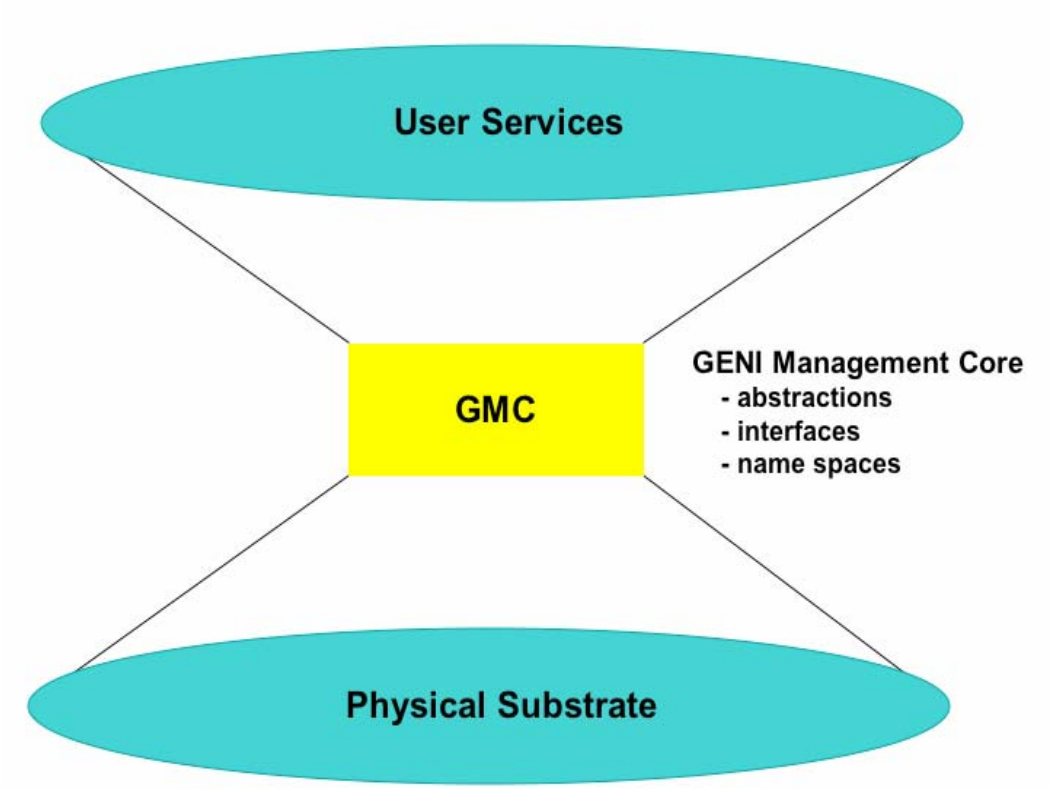

Figure 1.1: Overview of the GENI Architecture

At the top level, GENI's *user services* provide a rich array of user-visible support services intended to make the facility accessible and effective in meeting its research goals. As researcher requirements change and GENI's sophistication advances, it is expected that the set of available user services will change and advance as well. A key goal of the overall GENI architecture is to foster the evolution of these services throughout the lifetime of the facility.

Sitting between the physical substrate and the user services is the *GENI Management Core*, or GMC. The purpose of the GMC is to define a stable, predictable, long-lived framework—a set of abstractions, interfaces, and name spaces—to bind together the GENI architecture. Because GENI's physical substrate and user services will develop and evolve rapidly as the facility is constructed and used, the GMC is designed to provide a narrowly defined set of mechanisms that both *support and foster* this development and *isolate* developmental change in one part of the system from that in other parts, so that independent progress may be made.

By analogy with today's Internet architecture, the GENI architecture conforms with the hour-glass model: the GMC corresponds to the IP layer of the Internet stack with its attendant addressing routing and service model (i.e., it defines the "narrow waist" of the GENI hourglass); the set of high-level user services corresponds to the additional functionality needed to make the Internet a complete system (e.g., HTTP, WWW, BGP); and the GENI substrate corresponds to the collection of computing and networking devices that make up the physical Internet.

For GENI to function, the GMC must be reduced to a specific implementation. We refer to this implementation as the *GENI Management Core Implementation*, or GMCI. The GMCI provides two key elements of the overall GENI system: (1) the small set of core services that are necessary for the system to operate, and (2) an underlying messaging and remote operation invocation framework needed for elements of the GENI system to communicate with each other. An important aspect of the GMCI is its ability to allow different implementations and versions of non-core user services to be easily added to the system as appropriate.

1.2 Physical Substrate

The physical network substrate consists of an expandable collection of building block *components*. The set of components chosen for inclusion within GENI at any given time are intended to allow the creation of virtual networks covering the full range needed by GENI's constituent research communities.

We expect the set of building block components to evolve over time as technology and research requirements advance, but we define an initial set of components to be deployed:

- **Programmable Edge Clusters (PEC)** intended to provide the computational resources needed to build wide-area services and applications, as well as initial implementations of new network elements.
- **Programmable Core Nodes (PCN)** intended to implement core network data processing functions for high-speed, high volume traffic flows.
- **Programmable Edge Nodes (PEN)** intended to implement data forwarding functionality at the boundary between access networks and a high-speed backbone.
- **Programmable Wireless Nodes (PWN)** intended to implement proxies and other forwarding functionality within a wireless network.
- **Client Devices** intended to run applications that give end-users access to experimental services available on the combined wired/wireless substrate.
- A **National Fiber Facility** intended to provide 10-40Gbps light path interconnection between GENI core nodes, forming a nationwide backbone network.
- A large number of **tail circuits** of varying technologies, intended to connect GENI edge sites to the GENI core, and with appropriate security mechanisms, to connect the GENI core to the current commodity Internet.
- Multiple **Internet Exchange Points** connecting the nationwide backbone to the commodity Internet.
- One or more **Urban 802.11-based Mesh Wireless Subnets** intended to provide real-world experimental support for ad-hoc and mesh network research based on an emerging generation of short-range radios.
- One or more **Wide-Area Suburban 3G/WiMax-based Wireless Subnets** intended to provide open-access 3G/WiMax radios for wide area coverage, along with short-range 802.11 class radios for hotspot and hybrid service models.

- One or more **Cognitive Radio Subnets** intended to support experimental development and validation of emerging spectrum allocation, access, and negotiation models.
- One or more **Application-Specific Sensor Subnets** capable of supporting research on both underlying protocols and specific applications of sensor networks.
- One or more **Emulation Grids** that allow researchers to introduce and utilize controlled traffic and network conditions within an experimental framework.

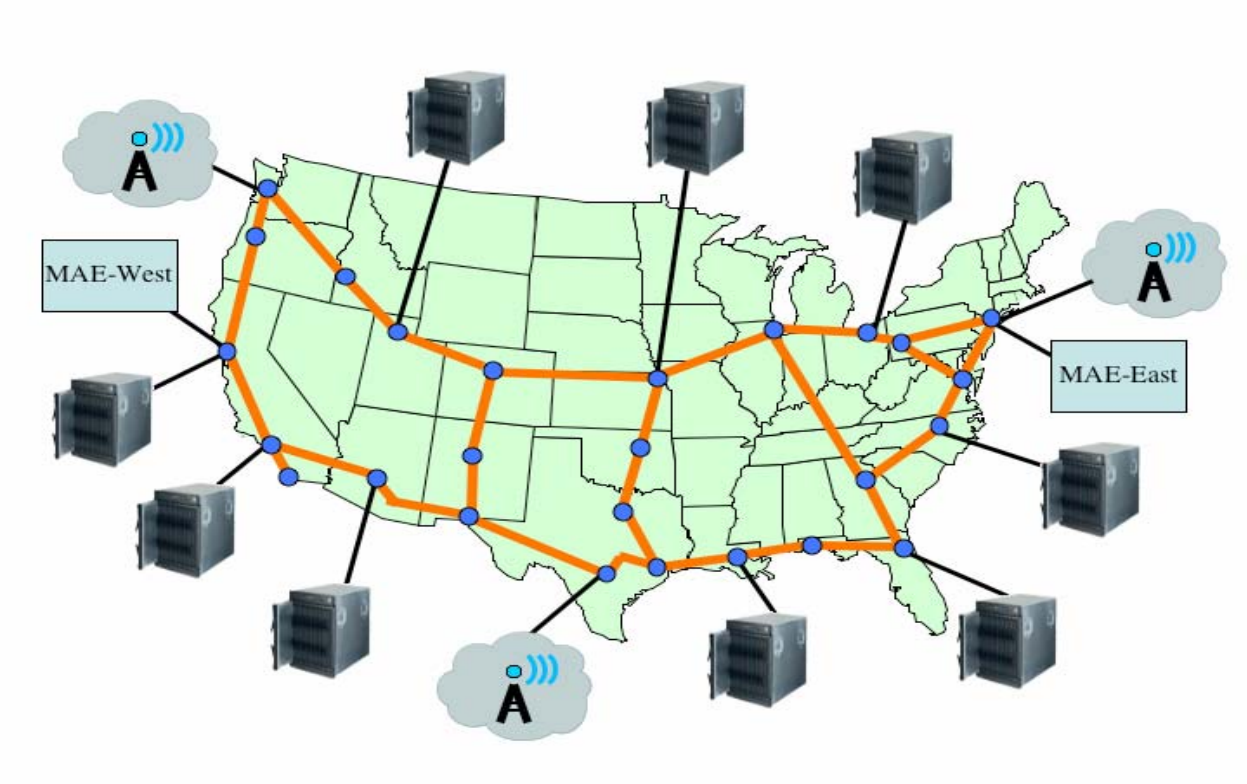

Figure 1.2: Global perspective of GENI physical substrate.

Figure 1.2 gives a high-level depiction of the physical substrate, showing a national fiber facility connecting a set of backbone sites, each of which is connected by tail circuits to edge sites that host clusters, wireless subnets, and sensor networks. Some of the backbone sites are also connected, via Internet Exchange Points, to the commodity Internet. Figure 1.3 provides an alternative view of a given backbone site, illustrating how the different components are connected into GENI. Note that each subnet (site) is connected to both the GENI backbone and the commodity Internet.

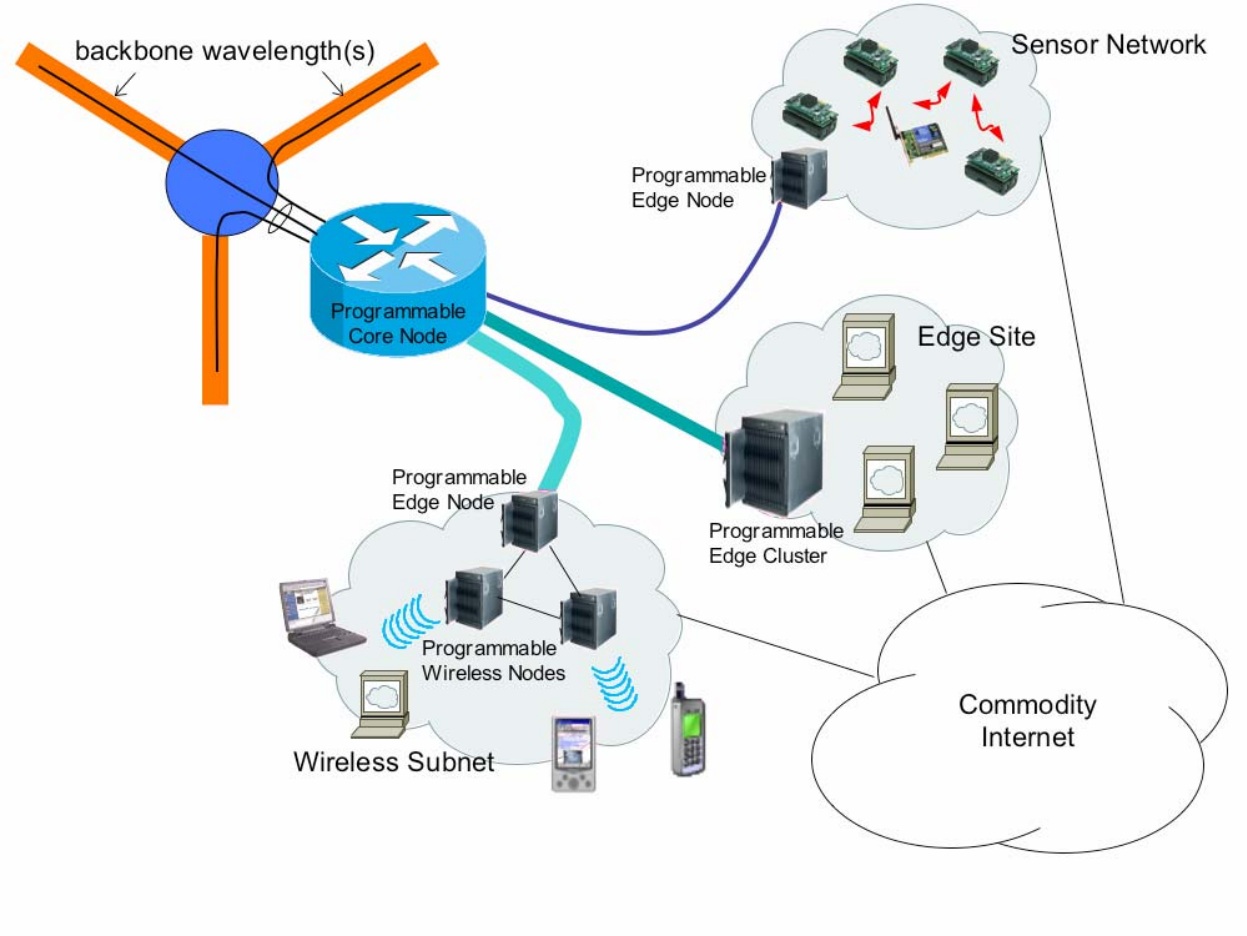

Figure 1.3: Backbone site perspective of GENI components that make up the physical substrate. The PCN at the backbone Point-of-Presence is connected to edge sites via some tail circuit technology. Edge sites are connected to both the GENI backbone and the commodity Internet.

1.3 User Services

The user services collectively knit the building blocks that comprise the physical substrate together into a coherent scientific instrument—a single distributed facility that is capable of supporting the research agenda. These services must support several distinct user communities, including:

- *Owners* of parts of the substrate, who are therefore responsible for the externally visible behavior of their equipment, and who establish the high-level policies for how their portion of the substrate is utilized.
- *Administrators* of parts of GENI, often working for owners or under contract to the GENI organization, whose job it is to keep the platform running, provide a service to researchers, and prevent malicious or otherwise damaging activity exploiting the platform.
- *Developers* of user services who build on the GMC interfaces to implement services of general use to the GENI community.

- *Researchers* employing GENI in their work, for running experiments, deploying experimental services, measuring aspects of the platform, and so on.
- *End users* not affiliated with GENI, but who access services provided by research projects that run over GENI.
- *Third parties* that may be confused or concerned about the effect that GENI-hosted experiments and services are having on their own enterprises.

Based on this list of players, we identify the following activities that the user services must provide (and the GMC must mediate):

- Allow owners to declare resource allocation and usage policies for substrate facilities under their control, and to provide mechanisms for enforcing those policies. The assumption is that there will be multiple owners and it will be a *federation* of these facilities that will form the entirety of the facility.
- Allow administrators to manage the GENI substrate, which includes installing new physical plant and retiring old or faulty plant, installing and updating system software, and monitoring GENI for performance, functionality, and security. Management is likely to be decentralized: there will be more than one organization administering disjoint collections of GENI sites. A broad spectrum of management styles is possible, ranging from individual owners managing their own machines, to a small number of large organizations that federate at a coarse grain.
- Allow researchers to create and populate experiments, allocate resources to them, and run experiment-specific software. Some of this functionality, such as convenient installation of software, including libraries or language runtimes, may be provided by higher-level services; the GMC aims to support the deployment and configuration of such software (see next point). The GMC must also expose an execution environment to researcher's applications, experiments, and services. These execution environments must be flexible (i.e., support a wide-range of program behavior) and perform satisfactorily (i.e., not unduly interfere with or distort the measurements or results).
- Expose low-level information about the state of the GENI substrate to developers, so they can implement high-level monitoring, measurement, auditing, and resource discovery services. In some sense, GMC can be regarded as analogous to the "kernel" of GENI as a distributed system, and consequently should expose (in a controlled manner) information to services interested in managing the system, using it effectively, and scientifically observing its operation.

2 System Architecture

This section describes, at a high level, the architecture of the GENI facility. The focus is on the *GENI Management Core* (GMC), which defines a stable, predictable, and long-lived framework—a set of abstractions, interfaces, and name spaces—that binds together the GENI architecture. Because GENI's physical substrate and user services will develop and evolve rapidly as the facility is constructed and used, the GMC is designed to provide a narrowly defined set of

mechanisms that both *support and foster* this development and *isolate* developmental change in one part of the system from that in other parts, so that independent progress may be made.

2.1 Abstractions

The GMC defines three key abstractions: *components*, *slices*, and *aggregates*. This section introduces the abstractions; the following section describes the interfaces they support.

2.1.1 Components

The GMC defines *components* as the primary building block of GENI. For example, a component might correspond to an edge computer, a customizable router, or a programmable access point.

A component encapsulates a collection of *resources*, including physical resources (e.g., CPU, memory, disk, bandwidth) logical resources (e.g., file descriptors, port numbers), and synthetic resources (e.g., packet forwarding fast paths). 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 is controlled via a *component manager* (CM), which exports a well-defined, remotely accessible interface. The component manager defines the operations available to user-level services to manage the allocation of component resources to different users and their experiments.

A component *owner* is a principal that establishes policies about how the component's resources are assigned to users; that is, owners define the resource allocation and acceptable use policies for their components.

Multiplexing

It must be possible to multiplex (slice) component resources among multiple users. This can be done by a combination of virtualizing the component (where each user acquires a virtual copy of the component's resources), or by partitioning the component into distinct resource sets (where each user acquires a distinct partition of the component's resources). In both cases, we say the user is granted a *sliver* of the component. Each component must include hardware or software mechanisms that isolate slivers from each other, making it appropriate to view a sliver as a "resource container."

Although every component must respond to the slice creation and control operations, each component may establish limits on the number of simultaneous users it can support. In particular, a component that is incapable of being shared may limit itself to supporting one active user at any given time.

A sliver that includes resources capable of loading and executing user-provided programs can also be viewed as supporting an *execution environment*. Slivers that support such execution environments are said to be *active slivers*.

Containment

It must also be possible to restrict the behavior of a component—and the slivers it hosts—in the surrounding network. This is particularly important with respect to traffic that flows between the component and the legacy Internet. This containment includes the rate at which a component can send traffic into the network, the other components and slivers with which it can communicate, and the rate at which a component can initiate communications with new components. It may also restrict packet-formats and the ability to spoof source addresses, as well as place protocol-specific rate limits on slivers. The component must also audit packets that are transmitted to the Internet so that responsibility for any unwanted or disruptive traffic can be traced back to the responsible sliver.

In the worst-case scenario, it must be possible to rapidly disconnect the component from the network, and bring it into a safe state, from which problems (including potential security compromises) can be diagnosed.

Virtual Interfaces

The precise execution environment provided by a sliver can be defined along several dimensions, and hence, any type system for slivers could become arbitrarily complex. We therefore define four base sliver types, and give users the ability to customize their environments by installing additional software packages. The four base sliver types correspond to the interface by which slivers access the underlying network substrate; they include:

- **Socket Interface**: Slivers access the network through the standard socket interface. This interface allows inter-component communication is via TCP or UDP connections, or IP tunnels. The socket interface does not imply a fixed topology—all components participating in the experiment are reachable (addressable) to the extent they are reachable via today's Internet. The experiment is free to choose whatever peering arrangement it wishes.
- **Virtual Link Interface**: Experiments access the network through a virtualized link interface. This interface exposes transmission queues and link failures, and the virtual links can be configured to provide best-effort service or CBR guarantees. The virtual link interface implies a fixed topology—the set of links (peers) available at each component is statically defined when the sliver is created, and is **not** under control of the experiment running in the sliver.[1] There are two sub-cases of this interface: (1) the virtual link is point-to-point between a pair of components; and (2) the virtual link is multi-point, corresponding to a VLAN. Details of this interface can be found elsewhere [GDD-06-25].
- **Virtual Radio Interface**: Experiments access radio spectrum through a virtualized radio interface. This interface exposes the characteristics of radio devices at the PHY/MAC layers. Specifically, the virtual radio can be configured to select a specific physical transmission rate, adjust the transmit power within a range, set threshold values of certain radio

[1] The experiment running in the slice is free to logically "inject failures" so as to logically bring one or more of its links down, for example, to evaluate a new routing algorithm.

parameters, and so on, depending on the type of radio device. Details of this interface can be found elsewhere [GDD-06-21].

- **Virtual Wire Interface**: Experiments access the network through a virtualized control interface that permits the dynamic creation, termination, and provisioning of end-to-end circuits. Some amount of physical link capacity (e.g., an optical wavelength) is allocated to a slice when it is created, with the set of end-to-end circuits multiplexed onto that capacity under the control of the experiment running in the slice. There are three sub-cases of this interface: (1) the underlying capacity corresponds to a framed TDM electrical circuit; (2) the underlying capacity corresponds to an unframed TDM electrical circuit; and (3) the underlying capacity corresponds to an unframed optical circuit (e.g., a wavelength). Details of this interface can be found elsewhere [GDD-06-26].

Management Authorities

A hierarchy of *management authorities* (MA) is responsible for the substrate components. GENI assigns a unique identifier to each component (as described in Section 2.2.1), but in addition, a component has a human-readable name corresponding to this hierarchy. For example,

geni.us.backbone.nyc

might name a component at the NYC PoP of GENI's US backbone. In this case, the geni.us.backbone management authority is responsible for the operational stability of the component.

2.1.2 Slices

A *slice* corresponds to a set of slivers spanning a set of GENI components, plus an associated set of users (researchers) that are allowed to access those slivers for the purpose of running an experiment on GENI. Users run their experiments in a slice of the GENI substrate.

A slice has a name, which is bound to the set of users associated with the slice, as described below. There exists a (possibly empty) set of slivers that participate in the slice, but the GMC does not define a concrete representation for slices. Instead, the representation of a slice is defined by whatever service creates the set of slivers on behalf of some user.

Slice Authorities

A hierarchy of *slice authorities* (SA) is responsible for the behavior of slice. GENI assigns a unique identifier to each slice (as described in Section 2.2.1), but in addition, a slice has a human-readable name corresponding to this hierarchy. For example,

geni.us.princeton.codeen

might name a slice created by the GENI slice authority, which has delegated to the US, and then to Princeton, the right to approve slices for individual projects (experiments), such as CoDeeN [WPPP04]. GENI defines a set of expectations for all slices it approves, and directly or indirectly vets the users assigned to those slices. Note that the GENI slice authority is expected to support slice creation on behalf of network and distributed systems researchers as part of the NSF GENI project. Because it is possible that other related facilities will federate with GENI, and there will

be other uses of the greater GENI ecosystem, we allow for the possibility that there will be other top-level slice authorities.

Note that slice names are used for many purposes:

- they identify slices to human users;
- they represent a hierarchy of slice authorities, where we assume a slice name is not both a slice and a slice authority;
- they represent authorization to create slices and manipulate slices, both in the sense that a user may invoke a slice create operation on a slice name in the slice authority hierarchy and in the sense that a user bound to this slice is explicitly authorized to perform all operations on the slice;
- they identify slices within an audit trail of responsibility; and
- they index a database describing all resources found in all slices (e.g., authorization keys).

Experiments

An *experiment* is a researcher-defined use of a slice; we say an experiment runs in a slice. Although often conflated, it is important to distinguish between the two concepts. The GMC defines an interface that is used to create and control slices. This includes operations for embedding the slice in a set of GENI components. Once a slice exists and resources have been bound to it, a researcher is free to further configure (program) the slice as part of an experiment, either directly, or by utilizing available experiment management services (Section 4.2). This includes loading and executing code in the slice's constituent slivers, and setting any parameters exposed by the constituent execution environments. Thus, slice creation is distinct from experiment configuration, and in fact, multiple experiments may run in a single slice over time, each parameterized in a different way but all utilizing the same set of component resources.

Many experiments are expected to be short-lived, for example, they might run for an hour at a time. Some experiments will be longer-lived, running continuously and processing real network traffic on behalf of a user community. Because such experiments are likely to provide a useful service to some user community, we often call them *experimental services*. In many respects, these experimental services are similar to the *user services* described in Section 4, the only difference being the former are likely targeted at end users not affiliated with GENI, while the latter are specifically designed on behalf of GENI researchers.

Services

We expect the set of distributed services that users employ to help manage their slices, and leverage to help implement the experiments running in those slices—collectively called *user services*—will themselves often run in slices of GENI. For example,

geni.monitor

geni.filesystem

geni.repository

might denote GENI-wide monitoring, file system, and repository services, respectively. All three of these services run in their own slice of GENI and export an interface to other GENI users. Note that similar services can be provided by any SA in the system; e.g.,

geni.us.utah.monitor

might correspond to an alternative monitoring service. Because it is often the case that slice names and service names are equivalent from a user's point of view, we sometimes use the terms "slice" and "service" interchangeably.

When a slice name is used to also denote a service, the slice name is sometimes bound to a URI at which an interface for the service can be called. Note that there are likely to exist services that do not run in a slice, for example, by supporting the composition of other services, where we sometimes call a user interface constructed from the composition of sub-services a *portal*.

2.1.3 Aggregates

While the architecture does not mandate any particular relationship among components—users are free to create a set of slivers on any set of components for which they can acquire the necessary rights—for a variety of reasons it is useful to define functionality relative to a related set of components. We introduce an *aggregation* construct for this purpose.

An *aggregate* is a GENI object that represents an unordered collection of components. It has an identity and supports the natural operations of a collection (e.g., addition, deletion, and enumeration). Aggregates provide a way for users, developers, or administrators to view a collection of GENI components together with some software-defined behavior as a single identifiable unit.

Aggregates can be hierarchical, that is, they can contain other aggregates as well as components. A given aggregate or component can be a member of zero, one, or many aggregates.

Example aggregates might correspond to a physical location (components co-located at the same site), a cluster (components that share a physical interconnect), an authority (a group of components managed by a single authority), a configuration (a group of components that share configuration information), or a network (a group of components that collectively implement a backbone network or a wireless subnet). There also might be a "root" aggregate that corresponds to all GENI components.

An aggregate is an active object, able to accept and respond to operations invoked on it. These operations are implemented by an *aggregate manager* (AM). Similar to a component and its component manager, an aggregate has a unique identifier that is used by the GMCI to invoke operations on the aggregate, and it has a human-readable name drawn from the same name space as components.

To the extent that an aggregate exports a component interface, it can be treated as a component, and there is a good argument for naming it as such. However, it is clear that aggregates will also export other interfaces that components will not; e.g., the aggregation interface. Our

approach is to name aggregates in the component space and provide a reflection interface to ask about other interfaces that the aggregate exports.

A more detailed discussion of how the component and aggregate abstractions can be used in practice can be found in a companion document [GDD-07-42].

2.1.4 Users

GENI users create and manipulate slices. Each user is identified by a certificate and key pair issued by one of the GENI authorities, where the ability to credential users is delegated. Rights to operate on slices are associated with user credentials. User credentials are associated with slices as described above.

2.2 Interfaces

This section briefly introduces the set of interfaces by which principals manipulate components, slices, and aggregates. It also describes how objects are identified, and defines the key data types that complete the interface picture. The discussion is purposely high-level. A companion set of XSD and WSDL specifications give precise definitions of the GMC interfaces [GMC Spec].

2.2.1 Names & Identifiers

The GMC defines unambiguous identifiers—called *GENI Global Identifiers* (GGID)—for the set of objects that make up GENI. GGIDs form the basis for a correct and secure system, such that an entity that possesses a GGID is able to confirm that the GGID was issued in accordance with the GMC and has not been forged, and to authenticate that the object claiming to correspond to the GGID is the one to which the GGID was actually issued.

Specifically, a GGID is represented as an X.509 certificate [X509, RFC-3280] that binds a Universally Unique Identifier (UUID) [X667] to a public key. The object identified by the GGID holds the private key, thereby forming the basis for authentication. Each GGID (X.509 certificate) is signed by the *authority* that created and controls the corresponding object; this authority must be identified by its own GGID. There may be one or many authorities that each implement the GMC, where every GGID is issued by an authority with the power and rights to sign GGIDs. Any entity may verify GGIDs via cryptographic keys that lead back, possibly in a chain, to a well-known root or roots.

A *name registry* maps strings to GGIDs, as well as to other domain-specific information about the corresponding object (e.g., the URI at which the object's manager can be reached, an IP or hardware address for the machine on which the object is implemented, the name and postal address of the organization that hosts the object, and so on). The GMCI provides a default registry that defines a hierarchical name space for objects, corresponding to the hierarchy of authorities that have been delegated the right to create and name objects. This default registry assumes a top-level naming authority trusted by all GENI entities, resulting in names of the form:

```
top-level_authority.sub_authority.sub_authority.name
```

For the purpose of this document, we assume "geni" is the top-level authority, but allow for the possibility that other similar authorities might federate in accordance with the GMC.

Note that the name assignment trust relationship represented by the default registry is distinct from other trust relationships in GENI. Rights to manipulate GENI objects may stem from different trust relations than the rights to name objects. For example, the right to name components might belong to a systems administrator, while the right to allow access to them might be allocated by the researcher in charge of a laboratory or by the GENI Science Council. There is no requirement that the name assignment hierarchy and these other hierarchies are distinct, but the flexibility of separating them is intended to allow different services to be implemented and to cleanly allocate responsibilities.

The GMCI includes two core repositories, as described in Section 2.4.2, but the architecture does not constrain the implementation of repositories. Any registry must be able to support retrieval of GGIDs and contact URIs, in response to whatever query format they export via WSDL. Repositories must scale appropriately to their domain, which implies that the default repositories must scale significantly. Their implementations and additional interfaces are unconstrained by this specification. Future repositories need not be hierarchical—e.g., they may be attribute-based—although GGID validation will continue to be constrained by the X.509 certificate structure.

The separation of identity (GGID) from the various present and future system name spaces (repositories) allows for simplicity of the GMC implementation, while encouraging new ways to organize system data. The GMC itself operates on GGIDs—no matter what name a caller uses to retrieve a GGID from a registry, the object the caller contacts will be asked to authenticate itself as the GGID, not the name. New services may create new name spaces without disrupting GMC operations. New interfaces can become visible to the system by using a registry to bind them to a GGID/URI pair.

2.2.2 Data Types

The GMC defines two key data types, which we briefly describe here. A more complete discussion can be found elsewhere [GDD-06-11].

RSpec

A *resource specification*, or RSpec, is the data structure used to describe resources in GENI. As such, it contains data about the resources possessed by components, such as their processing capabilities (such as processor architecture and speed), their network interfaces (including bandwidth and the like), and the instrumentation available on them (such as packet logging capabilities). The purpose of the RSpec standard is to give component managers, user services, and end users a common resource vocabulary.

RSpec describes a component in terms of the resources it possesses. Each resource (such as a processor or a network interface) is identified by an identifier (RID), which is assigned by the component manager and must be unique within the component. Thus, a combination of a component name and an RID is sufficient to unambiguously identify any resource in GENI.

RSpec is also used to describe the relationships between components, most significantly, the network links connecting components. This will require the ability to reference resources in other components' RSpecs, which, as noted above, can be done unambiguously. At this time, the component description role of RSpec is firmer than the relationship specification role of RSpec. Thus, the latter is still subject to change.

Most users are not expected to use RSpecs directly. In this sense, it can be thought of as a "machine language," used below the user-visible level of GENI. It should concentrate on clarity of definition and expressiveness, with lower priority given to user-friendliness. As with any assembly or machine language, some users may choose to use this language directly, but we expect that most will use some high-level interface. Such high-level specifications will be "compiled" to an RSpec by the service or aggregate that offers it. The GMC will support a wide variety of such higher-level embedding services (Section 4.2.2). The ability for higher-level services to provide a user-friendly interface relies on RSpec syntax and semantics having the desirable properties of an output language for machine-compiled higher level specifications, so that it will be a suitable target for a range of front-ends.

Much of the complexity of the GMC is embedded in resource specifications. RSpecs are an extensible part of the GMC interface. As new resources and capabilities are added, these specifications will inevitably need to be extended. While we strongly support a GENI-wide standardization process for extending RSpecs, we also expect these extensions will leverage a hierarchical name space. This will allow new communities that federate with GENI to extend RSpecs within their own partition of the name space, and components that offer specialized resources will similarly extend the RSpec in their own name space.

Ticket

A component owner signs an RSpec—one that includes the right to allocate the corresponding resources—to produce a *ticket.* Such tickets are "granted" by a component owner, and later "redeemed" to acquire resources on the component; these operations are defined more fully in Section 2.2.4. The GMC defines the format for tickets, and the set of rules governing their delegation. A ticket includes at least the following information:

1. an RSpec, describing the resources for which rights are being granted by the component owner;

2. a Slice Name, indicating the slice, or slice authority, to which rights to allocate the resources are being granted;

3. a timestamp indicating for how long the ticket is valid (the ticket must be "redeemed" before this time);

4. a timestamp indicating the period of time during which the resources specified in the ticket are available to be consumed by the slice; and

5. an indication of specific rights or privileges associated with the ticket.

2.2.3 Slice Operations

Slices are instantiated and represented by slice names, that is, the act of creating a slice name is indistinguishable from the act of creating a slice. Thus, slice operations are essentially operations performed by the GMC's slice naming service. The GMC slice registry supports three operations:

```
GGID = CreateSliceName(Name, UserInfo)
FreeSliceName(Name)
UserInfo = ResolveSliceName(Name)
```

The first operation creates a Name for a slice and binds it to the associated UserInfo, and the second operation deletes the given Name. The third operation returns the UserInfo bound to the named slice.

Once a slice name has been registered, the slice exists "in name only." The slice must also be instantiated as a set of slivers before it can be used by a researcher to conduct an experiment on GENI. This instantiation is performed by invoking operations on components on behalf of the slice, as described next.

2.2.4 Component Operations

Once a slice name has been registered with a trusted slice authority, the user can instantiate and provision the slice with the following operations. Although a particular user invokes these operations, it is reasonable to think of the slice as an "actor" that acquires, redistributes, and uses component resources; the users affiliated with the slice are effectively making requests under the name of the slice. We sometimes take the liberty of letting slices be the actors (representing user principals) in the following description.

Note that a single component is able to create only local slivers, meaning that the following operations must be invoked on each component that the slice is expected to span. We return to the issue of how a high-level service might assist the user in creating a slice that spans multiple components in Section 4.

Obtaining Rights to Component Resources

A slice-as-principal invokes the following operations on a component:

```
Ticket = GetTicket(Slice, Request)
```

to acquire rights to component resources. The Slice argument is a registered name of a slice. The Request argument indicates (in the form of an RSpec) what resources the slice wants. The returned Ticket effectively binds the slice to the right to allocate on the component the resources/rights indicated by the Request. Whether or not the call succeeds depends on any access control implemented by the component, the local resources available on the component, and the allocation policy implemented by the component.

Two timestamps are associated with each ticket. One indicates how long the ticket is valid. The ticket must be "redeemed" (using the CreateSlice operation described below) within that

lifetime. The second indicates a period of time during which the resources specified in the ticket are available to be consumed by the slice. Although not described further here, the GMC also defines a GrantTicket operation that allows a component (owner) to proactively delegate the right to a set of resources to some principal, such as a user or a slice.

Once a principal possesses a Ticket, it can create a sliver on the component by invoking:

```
Sliver = CreateSlice(Ticket)
```

The sliver effectively becomes part of the slice identified by the Ticket. In this case, the Ticket must include the create privilege. The returned Sliver is a certificate that grants the holder the right to invoke other operations (see below) on the sliver. Note that while this call creates a sliver, we call the operation CreateSlice because it effectively instantiates the slice on that component.

Resources can be bound to a slice—and the duration of their binding extended—via the

```
ModifySlice(Ticket)
```

operation with a new Ticket. The consequence of such a call is to augment the slice identified in the ticket with the additional resources specified in the ticket's RSpec; the Ticket must include the bind privilege. Similarly, resources can be taken away from a slice. In both cases, a signal (upcall) indicating how the resource allocation has changed is delivered to the sliver.

A component (or other principal) that holds a ticket may need to split off a portion of the corresponding resources to respond to a GetTicket call. The holder of the ticket may have to go back to the original source to do this. This is really just another way to acquire a new ticket, hence

```
(Ticket1, Ticket2) = SplitTicket(Ticket, Request)
```

where Request says how much of the original Ticket to split into Ticket1; Ticket2 corresponds to the remainder. Other rights are preserved.

Controlling a Slice

Component managers also support the following operations on slivers:

```
StopSlice(Sliver)
StartSlice(Sliver)
DestroySlice(Sliver)
```

The first two operations stop and start the execution of an existing slice. The slice retains any acquired resources on the component, although a component that uses work-conserving schedulers is free to utilize those resources for the duration of the suspension. These two operations are often used in tandem to "reboot" a slice, but are offered as a pair to give any management service assisting the slice an opportunity to "clean up state" before restarting. The final operation removes the slice from the component and releases all of its resources.

The Sliver argument for all three operations corresponds to the value returned by CreateSlice. However, any principal with a certificate granting it the right to invoke the operation on a given slice may do so; these operations are not limited to the user that created the slice.

Slice Information

Finally, any principal may invoke the following operations:

```
Names[ ] = ListSlices( )
StatusSpec = GetStatus( )
Name = GetResponsibleSlice(PacketSignature)
```

on a component to learn the names of the set of slices instantiated on that component, the status of the component, and the name of a slice responsible for sending a packet with a given signature, respectively. A PacketSignature is given by a timestamp and a subset of fields in the packet header (e.g., protocol number, source IP, source port, destinationIP, destination-port). The GetResponsibleSlice call can only be invoked relative to a packet to/from a GENI component and the legacy Internet.

2.2.5 Aggregate Operations

All aggregates support the following operations:

```
Components[ ] = ListComponents( )
AddComponent(Component)
DeleteComponent(Component)
```

Aggregates typically extend this interface with a set of aggregate-specific operations, some examples of which are given in the next section. Most commonly, aggregates support the component interface, thereby serving as a proxy for the set of components in the aggregate.

2.3 Security

The principal role of the GMC is to define how users and high-level services access and control low-level GENI resources. This section highlights the security-related aspects of the GMC. A more expansive discussion can be found elsewhere [GDD-06-23].

2.3.1 Requirements

Security considerations place several requirements on the GENI architecture:

- **Explicit Trust**: Privileges in a distributed system should be managed explicitly and formally. Enforcing security in GENI will be something of a moving target, as the facility will be used during its construction, and will progress from a single, centralized management entity to a federated, decentralized model. Thus we need a security model that can evolve along with GENI. We need to define access control approaches that provide the required flexibility, rather than hard-coding trust relationships. Without explicit trust, it is likely that trust will be unintentionally misplaced, leading to system-wide vulnerabilities that can be exploited by a determined attacker.

- **Least Privilege**: The principle of least privilege is a tenet of computer security that requires each component of a system be given exactly the authority it needs to perform its tasks and no more. Failures to implement this principle are ubiquitous, and we face the consequences frequently. For example, most web servers do not need to be able to open connections to arbitrary addresses in order to perform their tasks. Yet this is permitted, and exactly this ability has been used numerous times in the epidemic spread of worms. While achieving least privilege in an absolute sense is arguably not feasible, it is our belief that the GENI facility should embrace least privilege as far as is practicable. Least privilege can secure the GENI facility from malicious software, accidental violations, or just simple resource exhaustions—in general, it can mitigate the risks caused by runaway experiments. It is also equally useful in securing the experimenter's environment against attacks from other experiments or faulty system software.
- **Revocation**: Despite our best efforts, it is inevitable that keys, slices, and systems will be compromised in GENI. Thus a critical requirement for GENI is to be able to quickly revoke and replace keys, suspend all permissions (e.g., slices) derived from a compromised key, and reset each node to a known secure state.
- **Auditability**: The possibility of compromise also requires us to be able to trace why a problem occurred so that it can be prevented from recurring. GENI needs to include mechanisms that identify which slice is responsible for each packet, and also be able to determine the entire chain of responsibility (from central administrator to local administrator to local user) that gave the user a specific capability that was misused.
- **Scalability**: With large-scale distributed systems such as GENI, simple schemes such as using a small set of authentication servers and/or replicating information required by authentication and authorization tasks are not feasible. GENI adopts a scalable authorization architecture, as described below.
- **Autonomy**: A key requirement for GENI is the ability to federate autonomous facilities. A GENI site should be able to authenticate and authorize requests from users in other sites, support delegation of rights, and it should be able to do so without requiring centralized trust.
- **Usability**: The user must be explicitly modeled as part of the security architecture. Any system that is hard to use will be evaded and ignored. The implication is that GENI needs to develop intuitive and easy interfaces for users to create roles, restrict rights, and so on. GENI also needs to make it easy for users to protect their private keys. In essence, secure system and user behavior must happen naturally, in the course of operating or using the system.
- **Performance**: As with usability, the performance overhead of providing security needs to be modest, or users will have an incentive to disable or evade the system. In practice, this means managing security information (such as certificates delegating rights to a specific set of users) locally as far as possible, as cache-coherent, distributed state. Caching means that lookups can be fast in the common case, without compromising system semantics.

2.3.2 Authorization and Access Control

A key aspect of security is ensuring that appropriate authorization and security structures exist to protect the physical substrate and user services. This section gives a brief overview of the authorization model supported by the GMC.

Two primary classes of resources are protected by GMC-based authorization mechanisms. The first of these is slices, the containers for user-level experiments. The second is the collection of physical and logical resources implemented as, or made available by, components.

Slices are represented in the GMC by names; the creation of a slice name—bound to a GGID—creates the slice. Thus, the authorization model for slice creation is based on controlling access to the slice name space. The slice name space consists of a naming hierarchy rooted at a top-level Slice Authority (SA). Below the top level SA, intermediate nodes in the naming tree represent intermediate slice authorities, while the leaf nodes represent the slices themselves.

To support authorization, each SA is expected to implement an access control policy on its name creation function. Thus, before creating a sub-node of its portion of the name space, a slice authority must check the credentials of the calling user, and any other information as required by its local policy. It is assumed that each SA has been delegated authorization authority over its portion of the slice name space, and need not refer creation requests to a higher level authority.

GENI's physical resources are encapsulated as components. Each component has an associated component owner. The owner is responsible for defining and implementing the authorization policy governing use of the component. Tickets are used to implement this model, where a ticket represents a principal's right to (1) create a sliver (perhaps with some initial resources bound to it) on a component, and (2) bind component resources to an existing slice. Both rights can be delegated. A ticket is essentially an RSpec signed by a component owner, granting rights to allocate resources on one or more components.

We decompose the act of gaining access to a resource into two distinct steps: constructing a chain (or more generally, a set) of certificates that proves the access complies with access-control policy, and checking that the *proof chain* of certificates is valid, e.g. that it indeed entails the access.

The task of constructing the proof chain (hereafter, shortened to proof) can be accomplished by any of the following means. In the simplest case, the proof can be obtained using the very mechanism that was used for finding the resource. For instance, if the principal becomes aware of the resource through a resource allocator (such as Emulab's assign [RAL03]), a resource discovery tool (such as SWORD [OCCP06]), or a resource broker (such as SHARP [FCCS03]), the same underlying tool can be extended to generate the necessary certificates and provide it to the principal. In the more general case, one could use a general-purpose rights manager that would perform distributed queries to discover and assemble a set of credentials that would constitute the proof for authorized access. However, as the certificates are stored in distributed repositories, the certificate discovery process might require multiple remote accesses, potentially causing performance bottlenecks. Various implementation strategies for caching

credentials or otherwise mitigating these performance bottlenecks without compromising security are available.

The task of checking the validity of the chain of certificates is performed by a *resource monitor*: when provided with a formal security policy, and a chain of digitally signed credentials that claims a request satisfies this security policy, the resource monitor verifies the digital signatures on all certificates to ensure their validity, and then verify that the request is indeed allowed by the security policy.

The resource monitor thus embodies significant design decisions, not the least of which is with respect to what security policy language the resource monitor verifies the credential proof chains. There appears to be a tradeoff between the expressiveness of the security policy language, or underlying logic, and the ease of generating credential chains granting access. A careful study of the tradeoffs is still needed before deciding on the appropriate security policy language and underlying logic.

Within the above framework, we plan to employ a rich set of certificates or declarations to implement various forms of authorizations. Following previous efforts, we envisage the use of following abstractions:

- Identity certificates (i.e., GGIDs) that bind principals to keys. These certificates are useful for authenticating principals.
- Authorization certificates that attest to privileges associated with principals. When combined with the identity certificates, these certificates enable authorized access.
- Delegation certificates that pass on privileges (or a subset of privileges) from one principal to another. As part of the delegation certificate, the delegator can state whether the delegatee can further delegate the privileges to yet another principal in the system.
- Group membership certificates that allow an authorizing agent to group together principals and manage their privileges in a scalable and efficient manner.
- Roles that allow a principal to voluntarily restrict its privileges, thereby limiting the dangers posed by security violations.
- Revocation of previously issued certificates, as no authorization service would be complete without mechanisms for revoking privileges associated with misbehaving principals or compromised keys.

While it is beyond the scope of this document to define the full set of actions or resources for which authority should be checked in GENI, we believe the above framework is flexible enough to be used in the following contexts:

- The authorization service should enable fine-grained controls on resource usage of user-level experiments. It should enable the execution of user-level programs in sandboxed contexts that enforce least privilege.
- The authorization service should support the access checks performed during system administrative tasks for creating, removing, or modifying information regarding sites and

nodes. It should also support the process of granting, revoking, or checking roles, and authorize site-management tasks such as rebooting nodes or setting bandwidth limits.

- The authorization service should also be useful for implementing user tasks pertaining to updating user information, initiating and controlling experiments, including operations such as adding or removing slivers from a slice.

2.4 Implementation

The GMC is first-and-foremost a specification—a set of interfaces, data types, and name spaces that govern how the various sub-systems of GENI (user-level services and substrate devices) collectively form a coherent whole. Most of GENI's implementation is embodied in these sub-systems. However, there is a small set of core functionality that we consider part of the GMC implementation, the so-called GMCI. This set includes the glue code that comprises the underlying messaging and remote operation invocation framework used by the sub-systems to communicate with each other, and a small set of bootstrap services that must be present before the rest of GENI's sub-systems can function. This section describes this core functionality.

This section also sketches a reference design for two critical abstractions of GENI: components and aggregates. We call these reference designs a *canonical component* and a *canonical aggregate*, respectively; the description of the physical substrate presented throughout Section 3 then leverages these two reference designs by describing how they can be *specialized* on a sub-system by sub-system basis. The canonical objects are important to GENI's design for two reasons:

1. They simplify the construction process, exploiting the fact that components and aggregates are more alike than they are different, meaning that they are most easily built by specializing a common code base rather than starting from scratch.

2. They reduce unnecessary heterogeneity, giving users a more homogeneous programming environment across the set of components that populate the physical substrate.

Alternative implementations are of course permitted—as long as they adhere to the published set of interfaces—but canonical objects have enough value to be explicitly noted in GENI's design.

2.4.1 Invocation Framework

The various objects of the GMCI communicate using standard service invocation mechanisms from Web Services. Specifically, service interactions are described in the Web Service Description Language version 1.1 [WSDL], where requests and responses are encoded in XML Schema 1.1 [XSD1, XSD2] and encoded via SOAP [SOAP] or XML-RPC [XML-RPC].

Initially, this implementation provides a simple remote procedure call interface that closely follows the specification in the GENI architecture [GDD-06-11]. However, using the Web Services framework allows the GMCI to make use of new tools for the analysis and expansion of Web Services as those tools become available.

Although many systems provide the basic function of remote invocation across heterogeneous hardware, the GMCI is based on Web Services because that technology is designed to be

interoperable, is extensible and being extended, and is widely implemented. This combination allows the system to be developed and deployed quickly, to be enhanced incrementally, and to be open to outside developers by design.

Note that GENI's design presents a bootstrapping dilemma. On the one hand, GENI must provide an environment for experimenting with networking concepts that are not constrained by today's Internet. On the other hand, we must implement GENI in the context of today's Internet since tomorrow's Internet is yet to be discovered using GENI. The resolution is that GENI enables clean-slate experiments, but we leverage today's Internet to access and control GENI. The GMC is designed so that no assumptions about today's Internet are "architected in", which means alternative control planes can be substituted for today's protocols as they become available [GDD-06-37], but today's protocols are used to get GENI up-and-running quickly.

One important consequence of this bootstrapping strategy is that GENI's control plane suffers from the security vulnerabilities of today's Internet, and in particular, is may be possible to launch a DoS attack against GENI. We are willing to except this tradeoff, since the alternative strategy of having GENI depend on yet-to-be-demonstrated DoS-proof network architectures is a far greater risk to its success.

2.4.2 Name Registries

A name registry is implemented as a service in GENI, and hence, is identified with its own GGID. The protocols used to register with or query a repository are described in WSDL and are extensible and exportable. The GMCI includes two default name repositories: a *component registry* and a *slice registry*.

The component registry binds each component name and GGID to a record of information that includes (1) the URI at which the component's manager can be accessed and (2) other attributes and identifiers that might commonly be associated with a component (e.g., hardware addresses, IP addresses, DNS names). Entities needing this information ask the registry to translate either the component's name or its GGID into this data record. Note that the component registry also maintains information about the management authorities that are responsible for a set of components. This information may include the GGID and URI for an aggregate object that offers an interface to a set of components.

The slice registry binds each slice name and GGID to a record of information that includes (1) email addresses for the set of users associated with the slice, (2) a set of public keys for each of those users,[2] and (3) other contact information for each of those users. Entities needing this information ask the registry to translate either the slice's name or its GGID into this data record. Note that the slice registry also maintains information about slice authorities that are responsible for a set of slices.

[2] These keys are uploaded into the user's slivers so that he or she can securely access it. These are not the same as the public keys in the GGID's X.509 certificates.

2.4.3 Security Modules

The GENI Security implementation includes several software modules, which collectively support the secure interaction among the central abstractions of slice, component, aggregate, and user. The security mechanisms implemented in the modules outlined below enable the GMCI to implement fine-grained resource and access control on behalf of large numbers of users without compromising usability and performance.

- **Resource Monitor**: used to check that a request is valid, by determining if the set of rights conveyed in the tickets associated with the request are indeed bound to the user invoking the operation, and are compliant with the current security policy. The resource monitor is employed by individual components or component managers, as well as services resident in a slice, aggregate managers, and other GENI elements.
- **Cryptographic Support Libraries**: used to create keys and certificates, and bind them to GGIDs for use by components, slices, aggregates and users. This module also includes routines for signing and verifying messages, used to support tickets and other higher-level security functions.
- **Security Policy Manager**: used to create, manage, modify and distribute security policies. Different front-end interfaces for component and aggregate managers, slices/services, and slice and management authorities tailor the user-interaction, encapsulating the low-level details of security policy representation. The security policy manager serves as a common software module for manipulating the low-level concrete representation of security policies used throughout the GENI system. These security policies are then be consulted, both locally and across the distributed GENI system, by enforcement modules.
- **Ticket Generator**: used by component owners, component and aggregate managers, and management authorities to create, split and delegate tickets. The ticket generator assembles both primitive tickets granting rights to use a single component and more complex ensembles of tickets granting coordinated rights to use multiple components or aggregates as required to support user experiments.
- **Ticket Agent**: used by GENI entities and users as a container in which to hold and select tickets for use in subsequent GENI service invocations. In addition to merely acting as a key ring for privileges, the ticket agent also facilitates retrieval and management of complex sets of tickets. For example, a user or component might query the ticket agent to determine whether a specific set of tickets were sufficient to perform an operation relative to a known security policy, and if not, to perform the required steps to acquire additional tickets. The ticket agent supports management of fine-grained privileges, as well as ephemeral resources, such as the right to use a certain number of programmable blades or to allocate an aggregate number of hours of exclusive use of a component.

2.4.4 Canonical Component

A *canonical component* is a reference implementation of the component abstraction that can be specialized for the various sub-systems that make up the physical substrate (Section 3). A more detailed discussion of implementation options is presented elsewhere [GDD-06-13]. Here, we elaborate on the Component Manager (CM), which exports a well-defined interface through which users create and control slivers on that component. This interface implies that the

component must provide software and hardware mechanisms that implement a defined set of functions, including the ability to:

1. create and destroy slivers, bind a set of resources to a sliver, and reclaim those resources;

2. isolate slivers from each other, such that

a) the resources consumed by one sliver do not unduly affect the performance of another sliver,

b) one sliver cannot, without permission, eavesdrop on network traffic to or from another sliver,

c) one sliver cannot, without permission, access objects (e.g., files, ports, processes) belonging to another sliver, and

d) users are allowed to install software packages in their sliver without consideration for the packages installed in other slivers running on the same component;

3. allow users to securely remotely access a sliver that has been created on their behalf;

4. deliver signals to slivers, including a "reboot" signal that is delivered whenever the sliver starts up;

5. grant privileged operations to select slivers, including the ability of one sliver to access private state associated with another sliver (thereby supporting sliver interposition);

6. rate-limit the network traffic generated by a sliver, as well as by all slivers running on the component;

7. limit (filter) how a sliver interacts with (exchanges packets with) the Internet;

8. support a mechanism to audit all packet flows transmitted by slivers to the Internet, and determine what sliver (slice) is responsible for a given packet;

9. disconnect the component from the network and bring it into a safe state, and subsequently reboot the component into a correct configuration;

10. power the component on and off, and monitor its hardware and other software systems for errors;

Layered Implementation

The layers of software required to implement a canonical component are depicted in Figure 2.1, where the CM implements the interface through which these layers are controlled. Functions 1-5 from above address intra-component requirements, that is, how slivers interact with the component and each other. These functions are implemented by a *Virtual Machine Monitor*. Functions 6-8 correspond to inter-component behavior, that is, how the component and its slivers interact with the rest of the network. These functions are implemented by a *Traffic Monitor*. Functions 9 and 10 have to do with remotely controlling a component, that is, bringing

it into a state that is able to support the other functions. These functions are implemented by a *Secure Boot Monitor* and a *Hardware Monitor*, respectively.

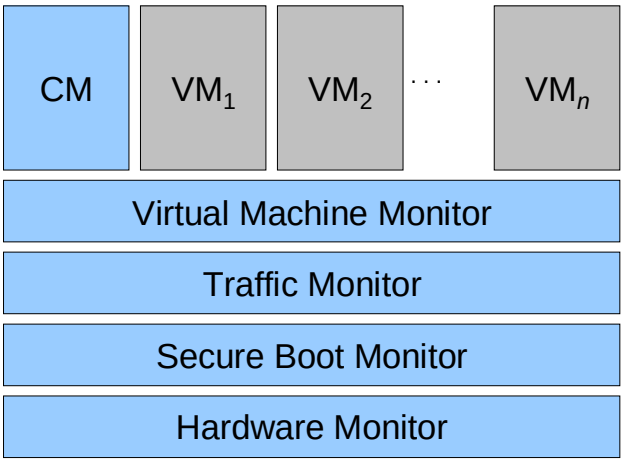

Figure 2.1. Layers implementing a canonical component. Each virtual machine (VM) corresponds to a sliver.

As discussed more thoroughly in [GDD-06-13], the implementation options for building a canonical component are well understood. The hardware consists of a *cluster of commodity processors* (possibly configured as a blade server) and includes substantial storage capacity. The Virtual Machine Monitor (VMM) that isolates slivers is a well-developed technology. GENI will mostly likely exploit a combination of paravirtualization (e.g., Xen [BDFH03]) and container-based virtualization (e.g., Vserver [VSERVER]), with thee CM running in a privileged VM.

The Traffic Monitor is responsible for authorizing, shaping, and auditing network traffic that is sent and received by a component. Modules implementing this functionality are widely used in modern operating systems. The Secure Boot Monitor interacts with a remote aggregate (Section 2.4.5) to install and boot a node with the correct software. Such software is commonly employed by hosting centers, including PlanetLab [BBCC04, PBFM06]. Finally, the Hardware Monitor provides a reliable method of remotely accessing and controlling a component. Such mechanisms (e.g., HP's iLO2 feature) are widely available on state-of-the-art hardware, and can be configured to use either a separate control channel or share a single data channel.

Performance Isolation

The requirement of performance isolation (2a) is purposely underspecified. Ideally, it would be possible to completely and perfectly isolate a sliver from the resources consumed by other slivers—i.e., offer absolute resource guarantees to individual slivers—as well as to provide for slivers that do not require deterministic guarantees. However, two factors make this difficult in practice.

First, the overhead cost of perfect isolation—which at the limit might include providing separate physical resources to the isolated slice—may be very high. In a virtualized system, the response to this problem is to provide isolation to an acceptable, rather than perfect, level of

fidelity. This in turn triggers the second limitation, which is that it is not clear how to best define "acceptable," balancing user requirements with the implementation pragmatics and overheads of isolation. There are many scheduler and virtualization options, each of which potentially offer a different level of fidelity to users. Which do we chose?

Rather than attempt to specify a single, global answer to this question, our response is to allow GENI's users to guide us. We provide an architectural *framework*, allowing different levels of isolation to be implemented by different component designs, and allowing users to select components based in part on their isolation capabilities and costs. We believe that over time usage and implementation experience will lead a small but useful set of design points to emerge. To be clear, it is the intent that GENI support both best-effort and reservation-based usage. What the architecture cannot (and should not) say is with what precision reservation-based guarantees can be made; the answer to that question depends on implementation choices.

2.4.5 Canonical Aggregate

GENI's base components are generally organized into aggregates, for example, corresponding to the components that comprise a backbone network or a wireless subnet. An Aggregate Manager (AM) provides an interface to the set of components, and in turn, implements certain functionality across the aggregate. We characterize this functionality as having two parts: *operations and maintenance (O&M) control,* and *slice coordination*. This section sketches the design of a *canonical aggregate* by describing these two parts. A more thorough description of how one might implement this functionality, drawing on the design of PlanetLab, is presented elsewhere [GDD-06-39].

While a component's owner might provide the functionality described in this section on a per-component basis, we envision this role being played on behalf of owners by a set of *management authorities* (MA), where a given authority is responsible for the components in some aggregate. Thus, a management authority is the *principal* responsible for a set of components and an aggregate is the *object* that encapsulates a set of components. We sometimes use the terms "management authority" and "aggregate" interchangeably in the following discussion.

Operations & Maintenance Control

O&M control corresponds to functionality that one might associate with a network operations center. It includes:

- booting components into the right state,
- detecting, debugging, and recovering failed components,
- responding to user queries (diagnosing user-perceived problems),
- responding to security complaints from aggrieved parties, and
- monitoring the system for AUP compliance.

A management authority likely provides an operations center—staffed by a set of administrators—but the MA also manifests itself as an aggregate object that exports an interface that can be invoked by other objects in the GENI system. For example, a backbone aggregate

might provide O&M control for all the components that make up a backbone network, and another aggregate might do the same for the components distributed across a particular wireless subnet.

We expect such aggregates to support the standard aggregate interface (e.g., ListComponents returns a reference to all components managed by this MA), extended with aggregate-wide variants of the component-level informational operations defined in Section 2.2.5: ListSlices, GetStatus, GetResponsibleSlice. These operations serve as underpinnings for higher level management services provided by the Management Authority.

While each aggregate exports a public interface to the rest of GENI—e.g., ListComponents, GetResponsibleSlice—each MA is free to establish a private interface that it uses to manage its components.[3] Although a private interface, we expect any MA that is part of GENI proper to make this interface open (i.e., to publish it); MAs in the larger GENI ecosystem are not required to do so. Note that this private interface need not be XML-RPC based. In addition to the O&M design presented in [GDD-06-39], other examples of O&M control tailored for specific networks can be found in a set of companion documents [GDD-06-13, GDD-06-15, GDD-06-25].

Slice Coordination

It is often the case that slivers are created in a coordinated fashion across an aggregate of components. For example, a backbone aggregate might coordinate link bandwidth allocation across a set of backbone components. Similarly, a wireless subnet aggregate might oversee spectrum allocation among a set of wireless components. In both examples, the aggregate effectively provides a common interface for a related set of components, treating the set as a unit for the purpose of allocating resources, instantiating slices, and starting and stopping slices. We sometimes say that the aggregate implements a *slice coordination* function.

It is likely that the aggregate exports exactly the same interface as the underlying components. For example, to create a slice that spans a backbone, a user might invoke CreateSlice on the backbone aggregate, which in turn calls CreateSlice on the individual backbone components. In this case, the aggregate maintains state for the slice—the set of components it runs on and the quantity of resources that have been bound to it.

A hierarchy of aggregates can be formed through the invocation of GetTicket and GrantTicket operations. For example, a set of programmable core nodes might explicitly grant control over 100% of their resources to a backbone aggregate. Alternatively, an aggregate might be created implicitly by sharing a code base across a set of constituent components; such an aggregate would simply be programmed (hardwired) to "know" what resources are available on the set of components.

[3] By private, we mean specific to the aggregate (and possibly different from aggregate to aggregate), as opposed to GENI-wide.

3 Physical Substrate

GENI's physical substrate consists of an expandable collection of building block components. Although no single component could do so by itself, the set of components chosen for inclusion within GENI at any given time are intended to allow the creation of virtual networks covering the full range needed by the constituent research communities. This section describes these components. The discussion is structured in accordance with the way we expect to organize the substrate components into aggregates.

3.1 Edge Sites

A *GENI site* corresponds to a geographic location that hosts one or more GENI components. Most edge sites will host clusters of commodity processors that serve as the workhorse components in GENI. These clusters, which we refer to as *Programmable Edge Clusters* (PEC), both provide the computational and storage resources needed to build wide-area distributed services and applications, and serve as "ingress routers" for new network architectures. We focus on the PEC component in 3.1, but generally, any of the components described in Section 3 may reside at a GENI site.

We expect to deploy components at 200 sites, and connect them to the backbone via a variety of tail circuit technologies. Moreover, since these components will be embedded in existing access networks (e.g., university campuses and research labs) they will be able to exploit existing connectivity to the commodity Internet. This will provide a path by which end-clients not affiliated with GENI can access experimental applications, services, and architectures deployed on GENI.

The set of PECs deployed across GENI sites will be managed as a single aggregate. This means these components will be remotely managed through a common O&M control point (Section 4.1), and slice embedding services will indirectly include edge sites through a shared slice coordinator (Section 4.2).

3.1.1 Programmable Edge Cluster

Programmable Edge Clusters require minimal specialization beyond the canonical component described in Section 2.4.4. They consist of a rack or chassis populated with commodity processors, interconnected by a switch, augmented with significant storage capacity, and attached to the local network infrastructure as described in the next subsection. The PEC runs well-understood virtualization software that implements a sliver as a *virtual machine* (VM), each of which can be bound to some amount of processor, memory, disk, and link capacity under the control of the Component Manager (CM). We expect two different virtualization technologies to be employed: (1) paravirtualization (e.g., Xen [BDFH03]), which gives slivers access to low-level hardware resources, and (2) container-based virtualization (e.g., Vserver [VSERVER]), which gives slivers access to a virtualized system call interface.

While PECs resemble computational clusters, they will serve many different purposes in GENI. In general, they may fill any role for which commodity computers are used today: they may act as clients, individual servers, server farms, or peers in a peer-to-peer network.

The capacity of PECs will differ significantly from site to site. At the low-end, a PEC will include 8-12 processors, and hence, be indistinguishable from a Programmable Edge Node (Section 3.3.1). At the high-end, a PEC might include 512-1024 processors, thereby supporting scalable services. In this latter case, a given slice can be allocated multiple VMs on a single PEC, as opposed to the standard practice of limiting a slice to a single VM/sliver per component.

3.1.2 Tail Circuits & Access Networks

The edge sites that host GENI components will largely correspond to research organizations participating in GENI. These components will be connected to the GENI backbone (Section 3.2) by dedicated tail circuits, and at the same time, they will be connected to the Internet by the host's existing commodity connectivity.

The successful realization of this design will require a significant level of planning and cooperation among the hosting organizations. Specifically, there will be a need for close and ongoing coordination among campus information technology organizations, regional networks, the network research community, and the GENI Project Office. The degree of coordination in this instance calls for a significant degree of advanced planning and the development of good working relationships and communications channels among all the players.

The timing of GENI deployment is fortuitous as GENI will be in a position to leverage two recent developments in available infrastructure. First, many research universities have recently made, or are currently making, significant investments in new on-campus data centers to support both centralized requirements and the growing Cyberinfrastructure needs of computational science groups. New models for co-location of non-centrally managed servers (e.g., GENI components) are emerging around these facilities. Second, the combination of growing research and education community bandwidth requirements and the availability of optical fiber assets has led to the creation of *regional optical networks* (RONs) around the country. In most cases, these RONs are facilities based with the management of the optical networking elements—including circuit provisioning—under the control of a higher education entity. For the GENI tail circuit capability, the existence of the RONs make it possible to envision multiple circuits (up to a bandwidth of 10-Gbps and higher) between the GENI backbone and the GENI sites.

The following focuses on two aspects of edge sites: (1) where components are placed within the hosting organization's network, and (2) the tail circuit technology by which the components are connected to the backbone. The description introduces a new network element—a *GENI Gateway* (GGW)—which is a device that connects the resident component(s) to both the GENI tail circuit and to the hosting site's existing network. This GGW is not programmable (and is not a stand-alone component of GENI), but rather, it is managed as part of the adjacent component(s). Specifically, the GGW implements the network-related functions normally supported by the OS running on each component (i.e., functions 6-8 in Section 2.4.4), but by placing the GGW at a choke point within the site, it does so on behalf of a set of components. The GGW can also be engineered to disconnect a failing or misbehaving component from the network. See [GDD-06-11] for more information about the GGW.

Campus Co-location Models

Each GENI site will be co-located within the campus network infrastructure of some hosting organization (e.g., a University or industrial research lab). While a GENI site can attach to a campus infrastructure at a variety of locations, we have identified three specific campus co-location models:

- **Peering Model:** Components are located on shared physical infrastructure, and considered directly connected to the GENI backbone as well as a peer of the campus networks (i.e., the GGW is outside both the border gateway and firewall);
- **DMZ Model:** Components are located outside the campus firewall but within the campus border router or gateways, on a boundary network between the campus and the Internet (i.e., the GGW is between the border router and firewall); and
- **Embedded Model:** Components are embedded within the campus network itself, such within a departmental network (i.e., the GGW is behind both the border router and firewall).

The following discusses each model, in turn.

First, in the *peering* (or direct connect) model, the GENI site is co-located on a campus, a local exchange point or a regional optical network. The GENI site leverages the host's network infrastructure, such as the fiber plant or Layer 2 network, but is otherwise separate from the campus network infrastructure. This model is the only one that will permit an optical GENI tail circuit from the site to the GENI backbone. As illustrated in Figure 3.1, the GENI tail circuit connects directly to the GGW, as does each GENI component.

In this model, the GENI site is independent, and is thus a peer of the host network infrastructure. The GENI site, through the GGW, establishes its own peering relationships, both with the campus itself as well as with the outside world. The GGW can then connect to the campus network infrastructure allowing it to leverage the campus's wide area, metropolitan area and local area infrastructure. Traffic between components within the site does not traverse the internal campus network.

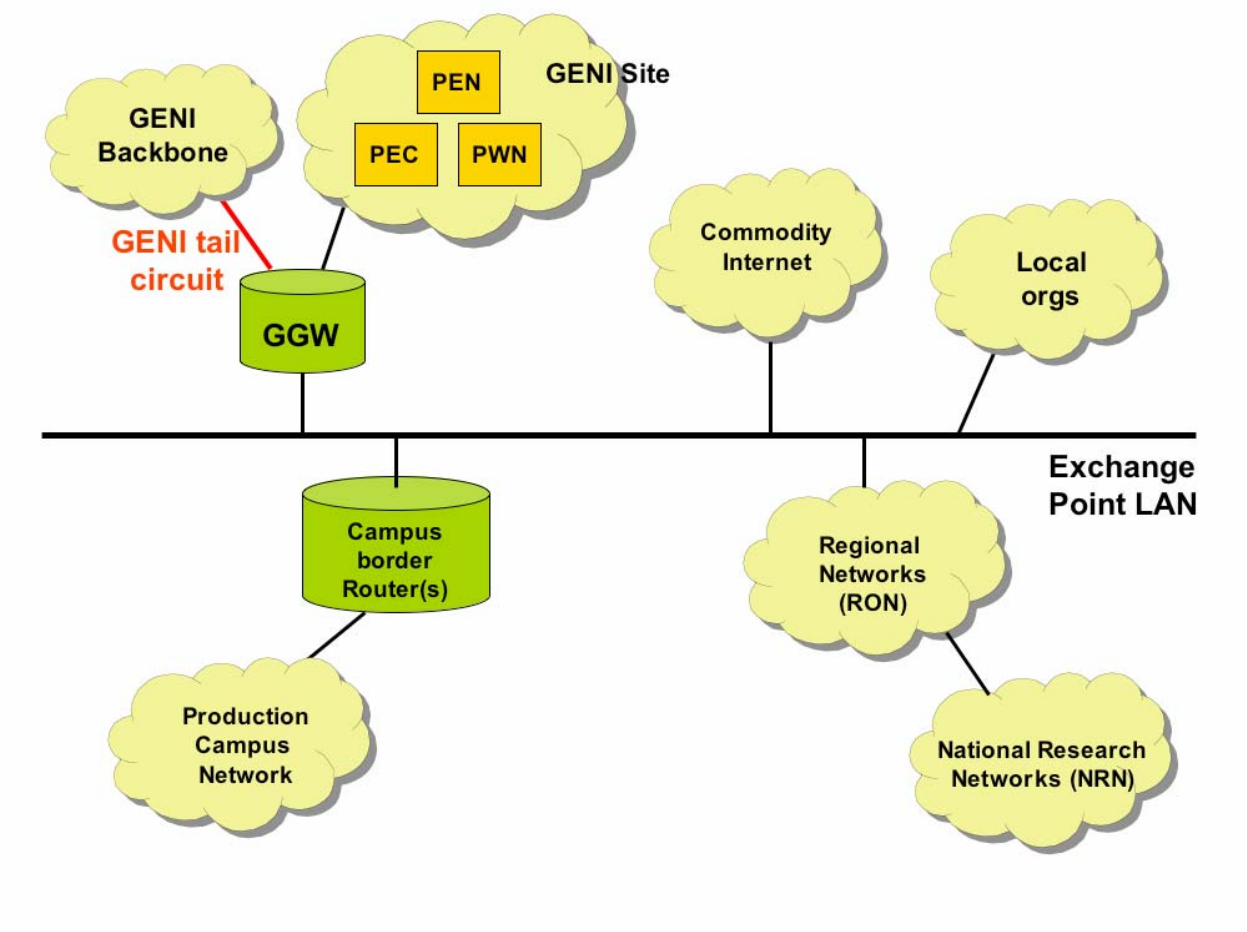

Figure 3.1: Peering Model for GENI site deployments.

Second, many campus networks include a physical network infrastructure that is behind the campus external gateways and in front of any internal campus firewall, routers or gateways. This network infrastructure is commonly referred to as the campus DMZ network, and we can use this structure to support the *DMZ model,* as illustrated in Figure 3.2.

In this model, the GGW is connected to the campus DMZ. The GENI site includes one or more components, connecting to the GGW. It is expected that the typical DMZ connection technology will be a dedicated Layer 2 technology, such as Ethernet. However, other technologies, including shared best effort TCP/IP connectivity, may be used as well. Regardless of the connection technology, the bandwidth allocated to connect the GGW must be sufficient to support the research requirements associated with the specific GENI site.

The GENI tail circuit connects to the campus border router or gateway. Therefore, the GENI site is dependent on campus resources for its external network connectivity. However, standard connection technologies, such as VLANs, MPLS, or IP tunnels can be used to provide independence from the campus routing policies, when needed.

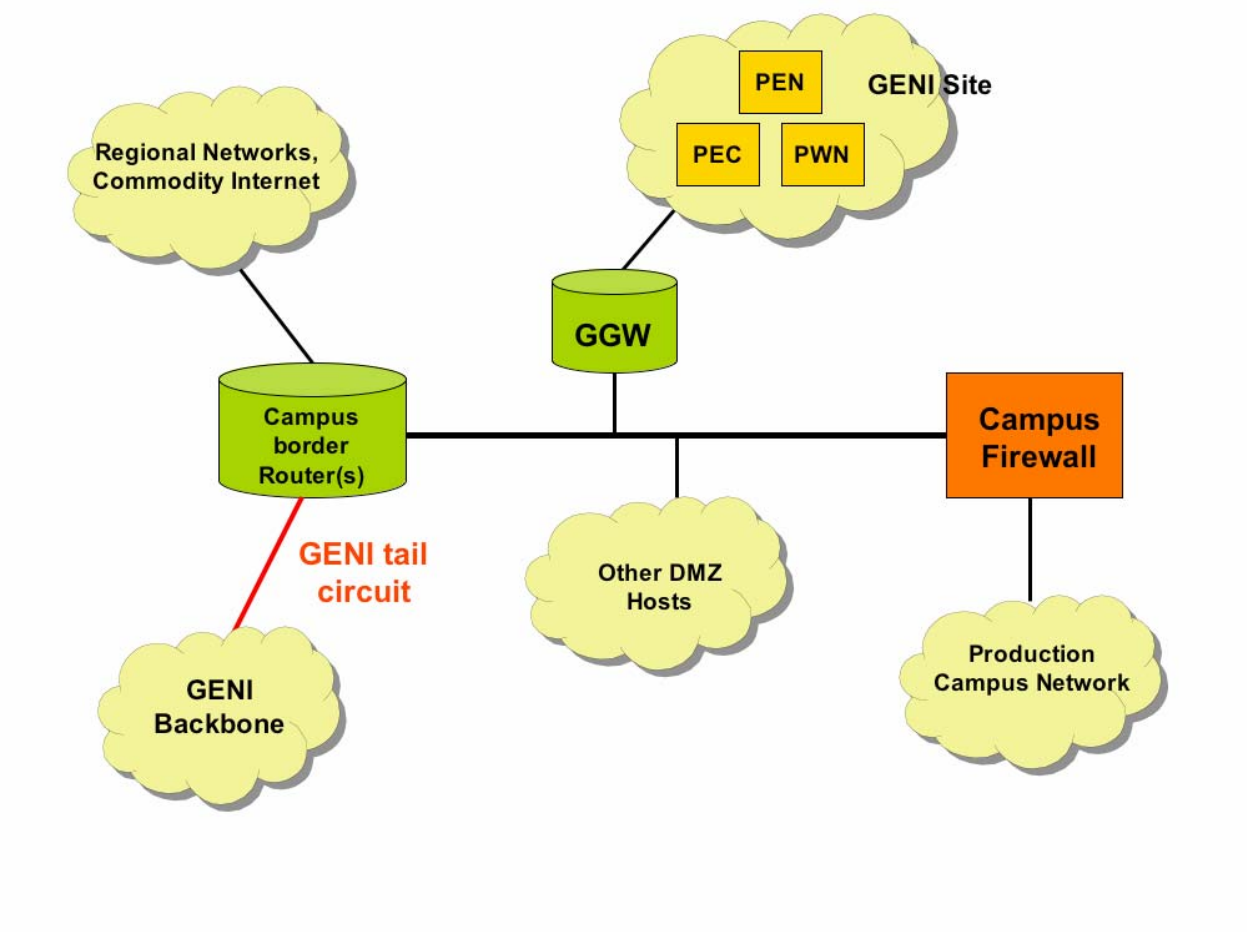

Figure 3.2: DMZ Model for GENI site deployments.

Third, for the *embedded model*, the GENI components reside within the campus network itself, behind both the campus firewall(s) and the campus border gateway. Figure 3.3 illustrates campus embedded model for a GENI site. As with the other models, the components would connect to the GGW, which then connects to the Campus Border Router. The components are completely dependent on the campus infrastructure for all network connectivity, including general Internet as well as connectivity to the GENI backbone. The GENI site would need to abide by the campus security and firewall policies. In some instances, the GENI site may be located within a department subject to additional departmental firewall or policy restrictions as well.

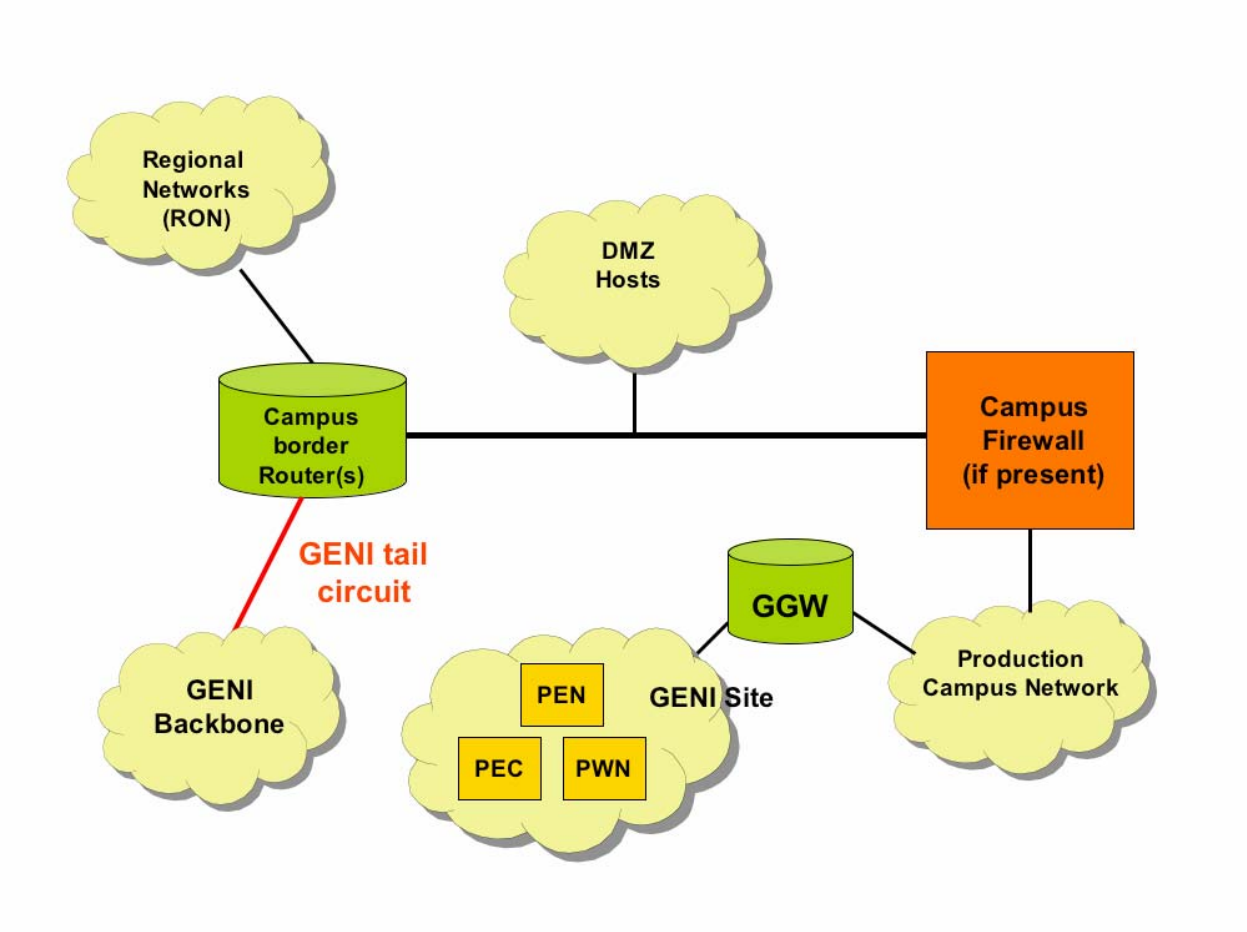

Figure 3.3: Embedded Model for GENI site deployments.

Tail Circuits

The connection between the GENI Backbone node and the GGW for a GENI site is referred to as the GENI tail circuit and can be provisioned in multiple ways. However, in all cases, it must be provisioned to support the services, characteristics and requirements for the GENI research projects hosted at the site. The GGW and its associated GENI site may have additional external or WAN network connections. These may include a commodity Internet connection, regional network connection as well as connections to the research and education networks. Typically these additional connections are all at Layer 3 (Network Layer) and are shared with the general campus community. With roughly 200 sites connecting to 25 backbone nodes, opportunities for sites to leverage existing regional network infrastructure, in particular infrastructure associated with the Regional Optical Networks (RONs) for their GENI tail circuit exists and is strongly encouraged.

Standard GENI tail circuit types are described below. They can best be described in terms of where they fit within the network stack, with Layer 1 (Physical), Layer 2 (Data Link) and Layer 3 (Network) the most common types of connections. For all three types, the tail circuit can either be dedicated to GENI or shared with the campus infrastructure as well as support any of

the campus co-location models described above. All three types of tail circuits can support a wide range of bandwidths.

First, GENI tail circuit can be implemented at the *physical level* (Layer 1) ranging from dedicated dark fiber to "lit services" such as SONET. Specific Layer 1 tail circuit types include:

- **Dark fiber**, dedicated dark fiber between the backbone and the GGW. This is the most flexible, but also most expensive of all tail circuit options. It can be lit using technology specific to the GENI site and can be changed or upgraded to meet requirements as needed. Typically, tail circuit management would be provided by the GENI site administration itself. The minimum bandwidth for a dark fiber connection would be 1-Gbps, with a maximum bandwidth of multiple 10-Gbps lambdas.
- **Shared Dark Fiber**, between the location of the backbone and the GGW. The fiber is lit and managed by a single entity, but specific channels, or entire wavelengths, can be dedicated to the GENI site and other projects. While providing less flexibility than the dedicated dark fiber option, it is also less expensive, supports a broader range of layer 1 services and can, within system limits, be upgraded to support new services. Available bandwidths will typically be in increments of 1-Gbps, 2.5-Gbps, or 10-Gbps assuming DWDM technology is used to light the dark fiber.
- **Leased service**, such as SONET, point-to-point Ethernet or even DSL. The service is typically procured from a standard telecommunications provide, terminates directly into the location hosting the PCN and GGW and is dedicated to the GENI site. A wide range of bandwidths exist using this technology, from 1Mb for DSL or other broadband technology up to multiple 10-Gbps SONET OC-192 service.

Second, tail circuits can be provided at the *data link layer* (Layer 2). The most common Layer 2 connection would an Ethernet environment where the network connections are terminated on a shared Ethernet switch. Both shared and dedicated services are available using the configuration options associated with the technology. For example, a dedicated bandwidth VLAN can be provisioned between the backbone node and the GGW. Less common Layer 2 technologies that could be used include Infiniband and Fiber Channel. As with Layer 1, the Layer 2 services can be purchased directly from a standard telecommunications provider or provisioned and purchased through the regional optical network in the area. A wide range of bandwidths exist using Layer 2 technology, with 1-Gbps or 10-Gbps being the most common. However, sub-gigabit and multi-gigabit data rates are available when portioning the Layer 2 infrastructure using VLANs or other similar technology.

Third, tail circuits at the *network layer* (Layer 3) can be either a shared best effort routed IP service or a dedicated service using a specific Layer 3 service such as MPLS. For the shared best-effort service, the overall amount of bandwidth available to the GENI site will depend on the shared campus connection type, typically ranging from 155-Mbps OC-3 to 10-Gbps Ethernet, as well as the average utilization of the link. When a dedicated service such as MPLS is used, a specific amount of bandwidth can be allocated to the GENI site.

We conclude this discussion by noting that in many instances, a GENI site will be located within an area served by one of the Regional Optical Networks (RON). In most cases, RONs are network aggregation points with the responsibility for managing the optical networking

elements associated with the network infrastructure under their control. Network management includes circuit provisioning, typically at the three network layers described above. In some instances, RONs have access to multiple pairs of fiber between the GENI backbone node and the GENI site. This often allows for site-managed optical network infrastructure between the GENI backbone node and the GENI site. Therefore, for GENI tail circuits, the existence of the RONs make it possible to provision multiple circuits, each with bandwidth up to 10-Gbpsbetween the GENI backbone nodes and the GENI sites and campuses. (This is not intended to preclude the use of even higher bandwidths in the future.)

For example, the Quilt [QUILT] is a consortium of over twenty leading research and education networking organizations in the United States whose mission is to promote consistent, reliable, interoperable and efficient advanced networking services that extend to the broadest possible community, and to represent common interests in the development and delivery of advanced network services. Through an ongoing series of Fiber Workshops, the Quilt has been instrumental in identifying standards, policies, and practices for developing, implementing, managing and provisioning fiber based network services. These activities can provide a mechanism for GENI sites, particularly those aggregating services through the RONs or leveraging RON resources, to define standards and practices for maintaining and managing GENI related services and resources.

3.2 Backbone Network

The GENI backbone consists of a collection of *Programmable Core Nodes* (PCNs) connected via an underlying fiber plant, with tail circuits to edge sites and connections to the legacy Internet at Internet eXchange Points (IXPs). The backbone plays several important roles in GENI:

- **Connectivity between GENI edge sites**: The backbone provides high-bandwidth, low-latency connectivity between GENI edge sites for experiments with distributed services and wireless/sensor subnets. The edge sites connect to the backbone over tail circuits that may have one of three types of link interfaces: packet switching over a constant-bit-rate or statistically-multiplexed connection, a time-division-multiplexed circuit (either optical or electrical), or a raw optical connection consisting of a whole wavelength.
- **Backbone network experiments**: The backbone enables experimentation with novel routing, forwarding, signaling, and addressing schemes, running on the PCNs. Programmable hardware (such as network processors and FPGAs) and flexible optical components enable realistic experiments that operate at high speed.
- **Connectivity to Internet hosts**: The backbone enables GENI experiments to communicate with hosts running in the legacy Internet, including end users and external services (such as Web sites). Direct connections to IXPs at several PCN sites, coupled with gateway software, provide the necessary bandwidth and flexibility.

Although some experiments may use the GENI backbone in just one of these three ways, we envision that many experiments will combine elements of all three. For example, consider an experiment evaluating a new content-delivery service. The servers storing and sending the content could run on the Programmable Edge Clusters. These servers would use the GENI backbone to communicate with each other, and with end users connected via remote GENI sites or the legacy Internet. The experiment may also evaluate a custom backbone design that

applies novel features such as multicast, quality of service, anycast, or caching, for better content delivery.

The set of PCNs deployed across the GENI backbone will be managed as a single aggregate. This means these components will be remotely managed through a common O&M control point (Section 4.1), and slice embedding services will indirectly include backbone nodse through a shared slice coordinator (Section 4.2). In the latter case, the aggregate will be responsible for ensuring that backbone-wide paths are allocated in a meaningful and consistent way.

3.2.1 Programmable Core Node (PCN)

Each site in the GENI backbone consists of a Programmable Core Node (PCN) that terminates links from GENI edge sites, Internet Service Providers, and other GENI backbone locations. The main sub-systems inside a PCN are:

- **Packet Processing System (PPS):** A collection of line cards, general-purpose processors, and programmable hardware (e.g., network processors and FPGAs) connected via a switch fabric. The PPS supports both the socket and virtual link interface defined in Section 2.1.1.
- **Circuit Processing System (CPS):** A combination of circuit-oriented elements, including a *programmable framer* that supports alternate framing and transport protocols; a *fast circuit switch* that enables the framer and PPS to access any of the wavelengths on the network; and a *reconfigurable optical add/drop multiplexer (ROADM)* that switches traffic at the wavelength level, enabling the GENI backbone to scale without requiring the bandwidth of the electronic components to match the total capacity of the network. The CPS supports the virtual wire interface defined in Section 2.1.1.

Links can connect to the PCN at several levels, from packet-switched connections to the Packet Processing System to circuit-switched connections via the ROADM. In addition, an experiment running on a subset of the PCNs can easily "cut through" intermediate PCNs at a variety of levels. For example, data packets may travel directly from one line card to another in the PPS, or a circuit may travel through the ROADM without any processing at the intermediate PCN. In the remainder of this subsection, we describe the PPS and the CPS, in turn.

Packet Processing System (PPS)

The Packet Processing System (PPS) leverages the canonical component to create a flexible, high-speed programmable device that supports multiple *virtual routers* (belonging to different slices) within a shared platform. The term virtual router is used here to mean any network element with multiple interfaces that forwards information through a network, while possibly processing it as it passes through. A virtual router can provide functionality similar to that of an IP router, a device that switches TDM circuits, or a firewall or media gateway. Another virtual router, in a different slice, might implement entirely different functionality, such as forwarding packets based on geographic coordinates or XML records, caching popular content, or performing source routing. The functionality of a given virtual router depends on the experiment. As such, the PPS has two main goals: (i) to provide the resources needed for researchers to build their own virtual routers that can operate at high speed and (ii) to ensure that virtual routers in different slices can run independently, free from unwanted interference.

A more complete discussion of the PPS, including a concrete reference design, is presented elsewhere [GDD-06-09].

The design of the PPS is distinctly different from the design of conventional routers and switches in that it must allow bandwidth to be allocated flexibly among multiple virtual routers and must provide third-party access to generic processing resources that can be allocated flexibly to different virtual routers. The basic PPS requirements include: open hardware and software components, scalable performance, stability and reliability, ease of use, technology diversity and adaptability, flexible allocation of link bandwidth and compute resources to slices, and strong isolation between virtual routers. For a more concrete discussion, we present a reference design of the PPS that attempts to meet these objectives, as shown in Figure 3.4. Wherever possible, we identify specific components and subsystems that could be used to implement various parts of the system, to illustrate that the PPS can be assembled largely from devices that have been (or are being) developed for commercial use. While integrating these devices is not a trivial effort, very little new hardware must be developed, significantly reducing the risks associated with the PPS development. The design has several key ingredients:

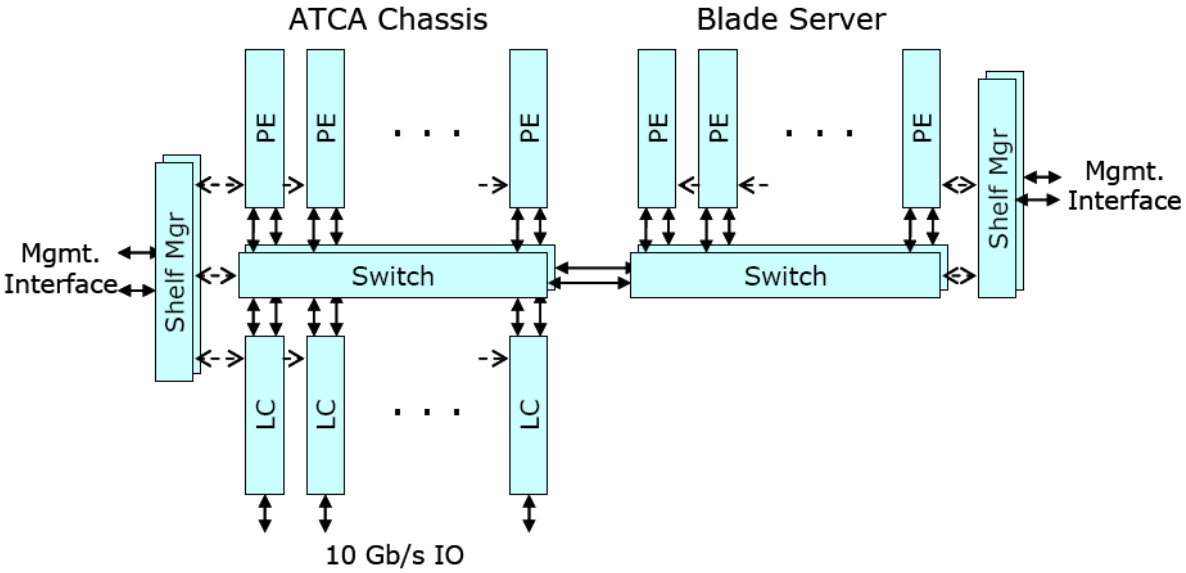

Figure 3.4. Organization of PPS Reference Platform

- **Advanced Telecommunications Computing Architecture (ATCA):** ATCA is a rapidly developing set of standards designed to facilitate the development of carrier-class communications and computing systems [PCMG] that defines standard physical components and some standard patterns for how to use those components to construct high performance systems. The standard governs key aspects of the boards that are used with the chassis, including physical size, connector type and placement and the use of certain of the connector signals. ATCA has attracted broad industry support and is expected to lead to the development of a range of interoperable subsystems that will allow more cost effective and

flexible development of new communication systems. Figure 3.4 shows a single ATCA chassis plus a commercial blade server. In the ATCA chassis, an Ethernet switch connects the Shelf Manager (that monitors the operation of the system and controls power and cooling), Line Cards (that terminate the external IO links), and Processing Engines (that provide generic processing resources for use by different virtual routers).

- **Network Processor (NP) and FPGA Blades:** Programmable hardware can support experiments that need high speed packet processing. We envision at least two ways these processing elements would be used. First, multiple experiments may share a single NP or FPGA for conventional packet-handling functions (such as forwarding and filtering) where each experiment has its own share of memory for storing tables (e.g., forwarding tables and access-control lists), accessed through a standard software interface. Second, an experiment may have its own dedicated blade for maximum flexibility and throughput, where the researchers can run their own software or logic on NP or FPGA. The NP blades can be implemented using the Radisys ATCA 7010. These blades each contain two Intel IXP 2800 network processors [RADISYS]. Each NP has sixteen internal processor cores for high throughput data processing, plus an Xscale processor (running Linux) for control. Each NP has three banks of RDRAM providing 750 MB of storage and four banks of QDR SRAM. The two NPs also share access to a dual-port TCAM that can be used for packet classification and other applications requiring associative lookup. In addition, an FPGA-based, configurable logic blade can be used to implement hardware-based PEs.

- **Switch Blades:** The switch blade enables communication between the other blades in the system, and plays a crucial role in providing isolation between virtual routers. A suitable switch blade has been developed by Radisys (ATCA 2210). This blade includes a 20-port 10 Gigabit Ethernet switch that provides one port to each of the 12 slots designated for PEs and LCs, plus two ports for connection to a redundant switch blade. It also provides four ports that can be connected either to the front panel, or the RTM connector. The 10 GE switch includes VLAN support, making it possible to constrain the routing of traffic from different virtual routers. This is useful for providing the traffic isolation needed to keep virtual routers from interfering with one another. The board also includes a 24-port 1 Gigabit Ethernet switch intended for carrying control traffic and a Control Processor that configures the two switch components through an on-board PCI interface.

- **Line Cards:** The line cards provide the functionality needed to allow multiple virtual links to share the external physical links, including fine-grain resource allocation and (de)multiplexing of packets to/from the appropriate processing element and slice. The LCs can be implemented either using an NP blade or a configurable logic blade. The LC can be configured to terminate IP and/or MPLS tunnels to facilitate reception of packets from remote sites that have no dedicated connections to the PPS. It must include a header-mapping function to map arriving packets to a virtual router number, a virtual interface number, and a physical destination within the PPS. Packets are labeled with their virtual router number and virtual interface number by the LC and forwarded through the switch to the specified destination. Packets going to the switch are sent through queues with a configured maximum rate, in order to prevent switch congestion. On the egress side, packets are received from the switch, already labeled with their virtual router and meta-interface numbers. The LC uses these to map the packets to the proper outgoing queue, which is configured to provide the appropriate encapsulation (if necessary). The egress-side

software also monitors the rate at which packets are received on each virtual interface, and raises an exception to the PPS control software, if the received rate exceeds the allowed rate for a given virtual interface; the PPS control software then decide what action to take, if any.

- **General-Purpose Processing Blades:** The general-purpose processing blades behave very much like the processors in a canonical component. They host slices that do not need specialized hardware for the sake of performance; in addition, some experiments may use a general-purpose blade to run control plane software (such as routing and signaling protocols), and use the NP and FPGA blades only for data plane functions (such as packet forwarding). To reduce overall system cost, the reference design uses a commercial blade server to host the general purpose Processing Elements (PEs), rather than using ATCA blades. An IBM Blade Center system is typical of the class of systems that can be used to provide general-purpose processing in the PPS. A suitable IBM Blade Center system would include 14 processor blades, each with two 3.6 Ghz Xeon processors with two on-board 80 GB disks and up to 8 GB of memory. These are interconnected through redundant switch cards that plug into the rear side of the chassis and support redundant 1 Gbps Ethernet connections to each slot. Each switch card has six 1 Gbps up links that can be used to connect to the ATCA chassis. It is likely that switch cards with 10 Gbps up links will be available soon.

Virtual router functionality will be implemented in PEs, thus the PPS requires software and interfaces to support both control and data plane virtual network components. A great strength of the hardware reference design lies in the minimal set of constraints placed on virtual router developers—the design philosophy provides resources to developers with few assumptions as to how those resources will be marshaled to implement innovative networks. To illustrate how the control and programming of the PCN will be staged, the PEs can be used in one of two modes. In *raw mode*, the entire PE blade is under the complete control of its user. Users may run their own operating system on a PE in raw mode and are fully responsible for its operation. There is no software to implement substrate functionality on a PE in raw mode, making it necessary for the system to use the switch fabric and LCs to ensure the necessary isolation. In *cooked mode*, a PE blade runs a standard GBP OS that provides substrate functionality and allows the blade to be shared by multiple virtual routers. In this mode, users may reserve a portion of the blade's resources for their exclusive use, and the system will attempt to accommodate such requests by mapping virtual PEs to physical PEs where the needed resources are available.

To further lower the barrier to experimentation, we plan to provide APIs for slices to install forwarding state in the (shared) NP and FPGA blades. In particular, we envision that an experiment may employ one of these three APIs:

- **Forwarding and Control Element Separation (ForCES) standard**: ForCES is an emerging IETF standard [RFC-3746] that consists of three key components: (i) a standard communication protocol between the control element (CE) and the forwarding element (FE), (ii) a standard and very simple operation code set for the CE to send control commands to the FE and, (iii) a standard numbering scheme for identifying data (or operand) at the FE which is very similar to the well known SNMP MIB numbering scheme. Several ForCES implementations already exist in the commercial world.

- **Forwarding Element Abstraction (FEA) in Click**: Many researchers building prototypes of networked systems use the Click modular router [KMJK00] as a building block. Click installs forwarding state based on commands received via a control socket interface. The forwarding element abstraction (FEA) used by Click is an appealing API for researchers to use in installing forwarding state in the NPs and FPGAs, due the familiarity of the API.
- **Kernel forwarding table (e.g., in Linux):** An experiment running on the general-purpose processor may update the forwarding table in the underlying operating system, such as Linux. In fact, some experiments may initially implement packet forwarding by having all data packets pass through the general-purpose processor for forwarding decisions, taking advantage of the dedicated NP and FPGA hardware at a later stage of deployment. We envision providing a library that would transparently copy the kernel forwarding tables to the shared NP or FPGA blade.

This software is described in more detail in [GDD-06-36].

Circuit Processing System (CPS)

The Circuit Processing System (CPS) consists of a layered collection of elements, where researchers can access GENI at whatever layer provides the right level of abstraction for their work. The strategy is to provide a set of concurrent substrates, so that researchers can experiment with new architectures and protocols at one level without interfering with the stability of the system being used by researchers investigating designs at another level.

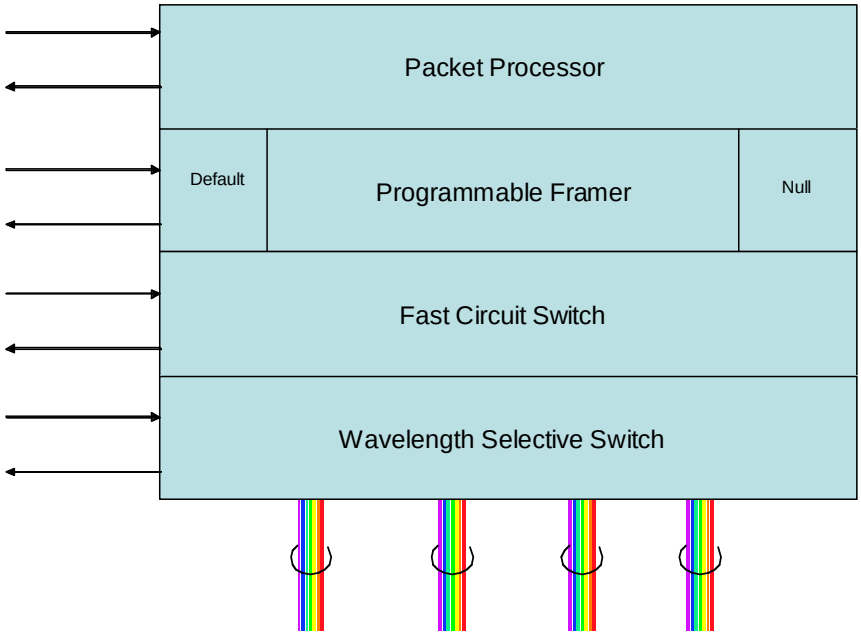

Figure 3.5: Layered design for the Circuit Processing System (CPS).

The design is depicted by the block diagram given in Figure 3.5, where the top-most layer (Packet Processor) corresponds to the PPS described above. The following briefly introduces the lower layers of the system. The design is described in more detail in [GDD-06-26].

- **Wavelength Selective Switch (WSS)**: Data on one 10-Gbps wavelength can be switched to another wavelength, or delivered to the Fast Circuit Switch. Data carried on a wavelength is not interpreted in any way by the WSS. The WSS will connect to the FS via optical fiber. User research equipment can connect directly to the WSS (shown on the left hand side of Figure 3.5) using spare optical ports on the WSS.
- **Fast Circuit Switch (FCS)**: Circuits are time-division-multiplexed onto a 10-Gbps wavelength. The minimum granularity will be 1-Mbps, and the circuit switch should be capable of establishing a new circuit in less than 1ms. Initially, the FCS will be electronic. Over time, optical FCS's are possible. Virtual circuits of any bandwidth with granularity of 1-Mbps can be established. Individual circuits can be assembled from multiple basic slots within and across wavelengths. The FCS will connect to the WSS via optical fiber. User research equipment can connect directly to the FCS (shown on the left hand side of Figure 3.5) using spare electrical ports on the FCS.
- **Programmable Framer (PF):** The framer will frame packets inside circuits. SONET is one obvious choice for a default framing format. The framer should have a null framing format in cases where the packets themselves carry sufficient information for recovery at the destination. The PF will connect to the FCS via electrical links or short-reach optics. User research equipment can connect directly to the PF (shown on the left hand side of Figure 3.5) using spare ports on the PF.

This design can be realized using commercially available hardware, as sketched in Figure 3.6. A Fiber Switch (FS) interconnects raw fibers from network input to network output without any modification of wavelengths or data within the fibers. Fibers in the plant can be physically allocated to the primary GENI nodes, next generation GENI nodes scheduled for hitless rollover, and fibers used for fiber optical physical layer research. The FS provides complete isolation at all levels between the running GENI infrastructure and future GENI nodes and technologies.

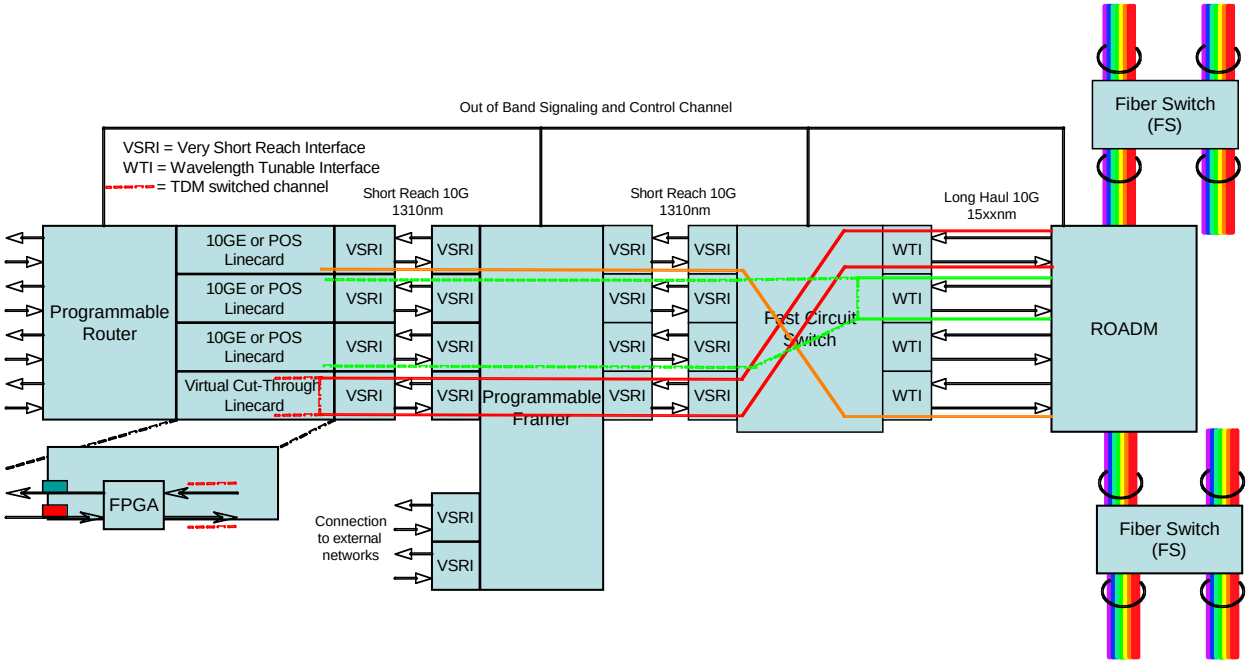

Figure 3.6: Reference design for the Circuit Processing System (CPS).

The Fiber Switch connects to a ROADM via four pairs of optical fiber (four ingress and four egress). The ROADM—which implements the WSS functionality—interconnects different wavelengths, adding/dropping some at this PCN, while converting others to a different wavelength and passing them along to the next node. The ROADM enables arbitrary topologies of wavelengths to be built for redundancy, and in some cases will be under the control of individual experiments. The ROADM also allows a wavelength to be added or dropped from a new non-GENI platform, perhaps one that is being tested for deployment in GENI. The ROADM will initially support up to four fibers, each carrying eight wavelengths.

The Fast Circuit Switch (FCS) sits above the ROADM and terminates each optical wavelength. The ROADM connects to the FCS via four pairs of optical fibers, each carrying up to eight wavelengths. The FCS allows multiple TDM channels to be multiplexed onto a single wavelength, and so performs the equivalent job as the ROADM in the time domain. It is equivalent to the SONET cross-connect switches used to terminate TDM circuits today, exact that the FCS will allow different framing structures to be used in each TDM channel. By default, SONET will be used at the normal SDH hierarchy rates, down to a minimum of 1.5-Mbps. However, experiments can install new framing structures, or bypass it completely and terminate the circuit in the PPS line card. The PPS will connect to the FCS using 32 10-Gbps channels, probably using short-reach 10-Gbps optics.

3.2.2 National Fiber Facility

Subjecting experimental network architectures to realistic traffic and network conditions is one of the main goals of the GENI facility. Certainly, researchers may start evaluating their new architectures via controlled experiments with synthetic traffic, emulated network topologies, and artificial network events. Yet, the purpose of the GENI facility is to allow them to gradually subject their architectures to increasing realism, such as carrying real user traffic, handling

unexpected network events, and operating at scale. This enables the experiments run in GENI to uncover unanticipated interactions that would not arise in controlled experiments in a simulator or testbed. In fact, some GENI experiments may evolve into long-running deployment studies that offer new services to end users. Running realistic experiments and attracting real users requires a facility that can offer the kind of performance the users would expect on the Internet.

The desire for realism is major factor in the design and sizing of the GENI facility, though this goal must be tempered by the need to limit cost. As an illustrative example, consider a straw-man design that places all GENI components (such as backbone nodes, wireless/sensor subnets, and compute clusters) at a single location, with connections to end users and legacy services via dedicated tail circuits and upstream connections to the Internet. In this model, artificial delay could be added to emulate the propagation delays in a realistic backbone network. The appeal of this approach is the reduction in deployment and management costs by placing all of the equipment in a single location. However, a centralized facility would have several serious shortcomings, including high latency (due to propagation delay to/from the central site), high backhaul costs (i.e., for the tail circuits), and poor robustness (e.g., where a single failure could disconnect the central site from the rest of the Internet). These issues lead us to design a distributed facility. To reduce propagation delay and backhaul cost, we envision having a distributed collection of PCNs with tail circuits to nearby GENI edge sites and upstream connections to the legacy Internet.

In selecting a backbone topology for the facility, we look to the rules of thumb that drive the design of commercial Internet Service Provider backbones, and the large research and education networks like the National Lambda Rail (NLR) [NLR] and the Abilene Internet2 backbone [Internet2], including [GDD-06-27]:

- **Path diversity**: Many experiments with new network architectures capitalize on the presence of multiple paths between a pair of sites; some architectures even need multiple link-disjoint or node-disjoint paths. For example, some architectures perform load balancing by splitting traffic over multiple paths, whereas others switch from one path to another in response to congestion or equipment failures. In addition, the ability of the GENI facility itself to survive node and link failures depends on the underlying diversity of the backbone network.
- **Underlying fiber paths**: The existing fiber-optic map in the United States imposes limits on the specific backbone sites that can have a direct fiber-optic connection between them. Placing backbone nodes in the key cities where multiple fiber-optic connections are available is extremely important to reduce the cost and deployment time of GENI. In addition, though it is possible to provide the illusion of dedicated links between any pair of backbone sites, providing links that match the underlying fiber map reduces cost and offers a more realistic deployment scenario.
- **Major interconnection points**: Deploying GENI backbone elements at existing IXPs where other ISPs have their backbone sites would allow GENI to amortize the costs of space, power, and "hands and eyes" support. Locating GENI backbone nodes at major exchange points would be useful for acquiring upstream connectivity to the legacy Internet; similarly,

having GENI backbone nodes at major aggregation points (such as the GigaPoPs) would facilitate efficient, low-cost connectivity to certain edge sites, such as university campuses.

- **Tail circuit cost**: The tail circuits connect to edge sites that house compute clusters, wireless/sensor subnets, and end users. The cost of these tail circuits depends, in large part, on the length of the circuit and the presence of existing fiber. Most major campus and enterprise sites already have tail circuits to the legacy Internet and perhaps also to research/education networks like the Abilene Internet2 backbone and the National Lambda Rail (NLR). Many of the university campuses connect via 15-20 GigaPoPs that provide connectivity, making it attractive to locate GENI backbone sites at or near these locations.
- **Limiting propagation delay**: Backbone networks typically have a sufficiently rich topology such that the end-to-end propagation delay between each pair of sites is small enough to support interactive applications, such as telephone calls and video conferencing. This limits end-to-end paths to around 100 msec of delay. Commercial ISPs typically try to limit the end-to-end propagation delay for each pair of backbone sites to some small multiplier (e.g., 2x) of the "air miles" between the sites. Otherwise, a competing ISP with a direct link between the same two locations could offer much lower latency.

All of these issues point to having a backbone design that is similar to existing commercial ISPs, which have around 25 major sites spread throughout the country, to provide low latency, path diversity, a good match with the underlying fiber infrastructure, sufficient peering points with other providers, and economical tail circuits. In addition, some GENI experiments may want the illusion of a less "backbone centric" kind of network architecture, with direct links between edge sites. Although in practice the fiber map may not have direct connectivity between pairs of edge sites, a 25-node GENI backbone can easily support the embedding of virtual links that connect pairs of edge sites. Having a rich topology that closely matches the fiber map, with short tail circuits to edge sites, would make that illusion as close to a reality as possible. Given the current plan of having around 200 edge sites, each backbone site would terminate an average of eight tail circuits.

One of the lessons learned from PlanetLab is that a useful platform for experimental research will attract lots of people who want to use it. This makes it difficult (and probably fatal to try) to estimate the number of experiments that will run simultaneously, and the amount of capacity they will need. It would be a disaster if GENI were built successfully, only to find that it then sinks under the weight of its own success and becomes overwhelmed by users and experiments, and is unable to react quickly enough to deploy new link and processing capacity.

On the other hand, financial constraints dictate that we don't overbuild the network. The trick will be to deploy enough capacity up-front in all components of the backbone network so as to allow experiments to start at reasonable cost; and architect the backbone network so as to allow (perhaps rapid) deployment of additional link, switching and processing capacity where needed. In other words, the architecture of the network should be somewhat decoupled from the particular data-rates of links and processing speeds of programmable elements. This suggests a phased approach in which a specific set of processing capacities and link rates are supported in each phase, with the set of processing capacities and link rates being expanded in each successive phase. For example, an early phase could support data-rates of 10 Gbps per wavelength, with each fiber carrying eight wavelengths. Over time, as needed, the facility can

be enhanced to support additional wavelengths, but won't support link rates above 10 Gbps. Later phases could then support higher link rates, up to 40 Gbps or 100 Gbps, and would be backwardly compatible with links at 10 Gbps. It is too early to decide between 40 Gbps and 100 Gbps as this should be dictated by the availability of commercial components.

At this time, it is not yet known whether GENI will operate over an existing fiber infrastructure (e.g., National Lambda Rail [NLR] or Internet2 Abilene) or over a new dedicated infrastructure. But either way, the GENI equipment will connect directly to the long-haul optical fibers that connect two adjacent backbone nodes [GDD-06-26]. At the lowest level, the Fiber Switch (FS) will connect to the incoming fiber. Typical backbone nodes have a degree no greater than four; in other words, they connect to no more than four adjacent backbone nodes (but could connect with multiple fibers to each neighbor). Likewise, GENI backbone nodes will connect to up to four adjacent backbone nodes, with one or more fibers to each. Initially, we will use one fiber per link. This means that the Packet Processing System (PPS) will hold up to 32 10 Gbps linecards (4 full duplex fibers, each carrying up to 8 wavelengths), although they might not be fully populated until all the wavelengths are used.

3.2.3 Internet Exchange Points

The GENI backbone will have connections to Internet Service Providers (ISPs) to reach hosts in the legacy Internet, most likely at Internet eXchange Points (IXP) in the major metropolitan areas where today's ISPs peer (e.g., Atlanta, Chicago, Denver, Los Angeles, New York, San Francisco, and Washington D.C.). These connections will terminate directly at the backbone PCNs, either at the packet level via line cards that connect to the Packet Processing System (PPS), or at the circuit level to the Circuit Processing System (CPS), depending on the arrangement with the ISP.

The IXP sits between experimental network architectures running within GENI, and the widely deployed Internet architecture. This means several "boundary" functions will need to be implemented at this point. This set includes the same traffic shaping, filtering, and auditing functionality implemented by the GGW that connects edge sites to the Internet (Section 3.1.2).

It also includes software modules that allow experimental services running in GENI slices to interact meaningfully with the Internet. For example, the IXP stands at the right place to terminate tunnels through the Internet and either translate addresses into a GENI format, or establish a tunnel into a slice entry point. As another example, slices may need to interact with infrastructure protocols, such as the Domain Name System (DNS) [RFC-1034, RFC-1035] and the Border Gateway Protocol (BGP) [RFC-4271]. This requires software running at the IXP to act as a "virtual player" with which the Internet negotiates. Several of these functions, which generally allow GENI to interoperate with the legacy Internet, are described in Section 4.3.3. These functions are also described in greater detail in [GDD-06-36, GDD-06-31].

3.3 Wireless Subnets

The physical substrate includes a range of wireless networking capabilities, providing researchers with access to emerging radio technologies that are becoming increasingly important at the network's edge [GDD-06-14]. As illustrated in Figure 3.7, specific technologies include an 802.11-based mesh network, an open API "4G" cellular/WiMax system, a cognitive

radio technology demonstrator, two large-scale sensor net deployments, and two emulation grids.

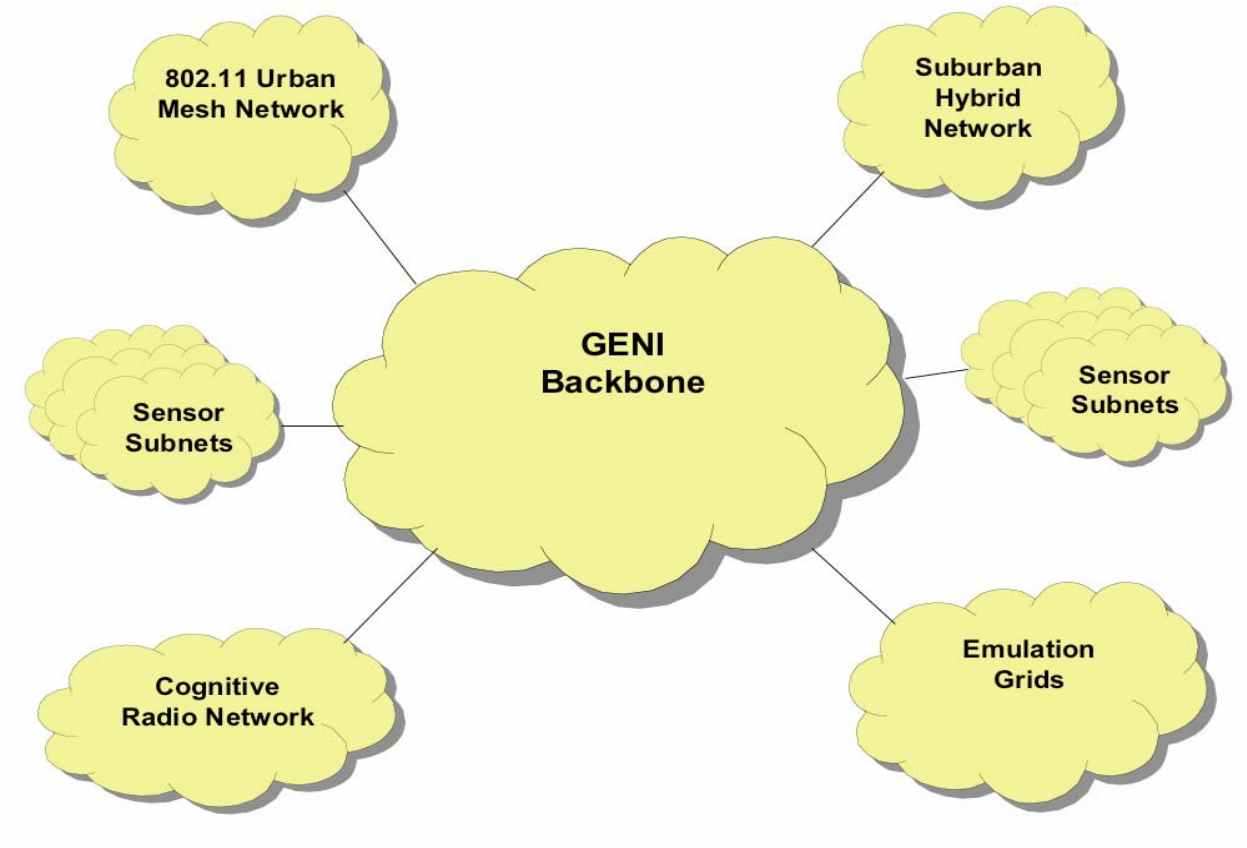

Figure 3.7: A collection of wireless technologies are included in GENI's physical substrate.

The wireless subnets of GENI are built around two classes of components: *Programmable Edge Nodes* (PEN) that connect the subnet (edge site) to the GENI backbone (as well as to any commodity Internet service that might be available), and *Programmable Wireless Nodes* (PWN) that are deployed throughout the subnet. This section describes these two components and the various subnets in which they are deployed.

3.3.1 Programmable Edge Node

A Programmable Edge Node (PEN) connects a wireless subnet, including a set of Programmable Wireless Nodes (PWN) and client devices, to the larger GENI substrate, as shown in Figure 3.8. Because a PEN connects to both the backbone and a wireless subnet, slices instantiated on a PEN likely host experiments and services that span the wired/wireless domains.

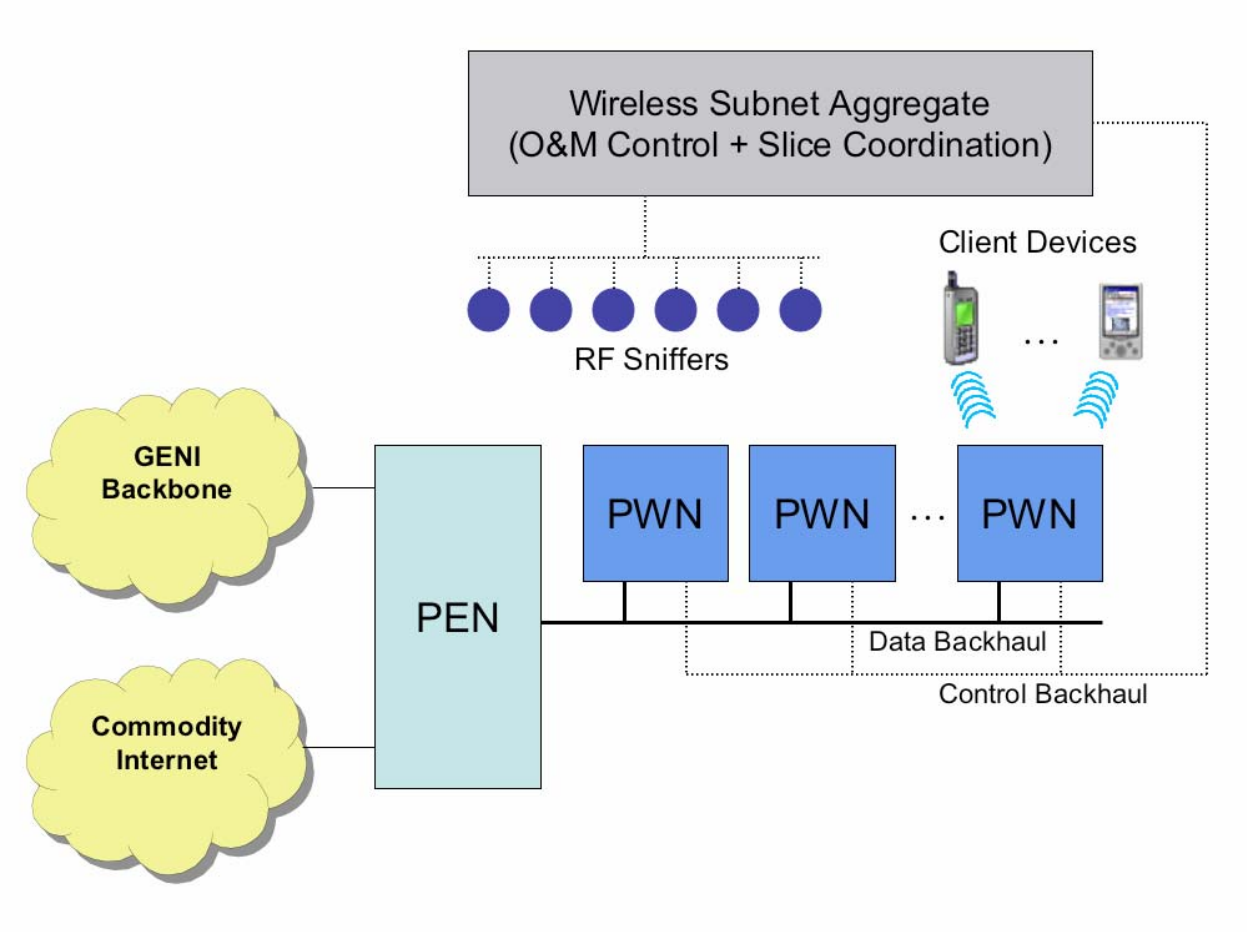

Figure 3.8: Programmable Edge Node (PEN) in a wireless subnet deployment, connecting a set of Programmable Wireless Nodes (PWN) to the GENI backbone and the commodity Internet.

The PEN is similar to the canonical component described in Section 2.4.4. It consists of a set of commodity processors, and runs virtualization software that allocates a VM to each slice instantiated on the node. A Component Manager establishes and controls these VMs according to the interface described in Section 2.2.3. A PEN includes network cards that connect it to both the GENI backbone and to the Programmable Wireless Nodes (PWNs) deployed throughout the site. It supports the Socket, Virtual Link, and Virtual Radio interfaces described in Section 2.1.1 [GDD-06-17].

The PEN's main job is to act as a mux/demux between the shared tail circuit connecting the site to the backbone and the many PWNs within the subnet. As such, its exact configuration depends on the environment in which it is deployed. For example, if each PWN in the subnet is able to act as source or sink of a 10 Mbps flow and there are 1,000 nodes in the subnet (and the PEN is connected to the backbone by a 10 Gbps tail circuit), the PEN needs to be configured with the same packet processing cards as the PCN described in Section 3.2.1. Sites with lesser tail circuits and fewer PWNs can be built from commodity processors and standard line cards.

3.3.2 Programmable Wireless Node

Programmable Wireless Nodes (PWNs) form the core of the physical substrate for each wireless subnet. They are designed to support both single-hop and multi-hop experimentation. They are also designed to be flexible—able to be configured with any of a number of radio technologies, able to handle high-bandwidth technologies, and able to perform processing tasks associated with cognitive radios.

Each PWN includes a set of one or more *radio cards*, depending on the subnet in which they are deployed. As described in subsequent subsections, these include both commercial transceivers (e.g. Cellular, WiFi or WiMax), software defined radios (SDR), and cognitive radios. The PWN supports the Virtual Radio Interface (Section 2.1.1), which defines the interface between a sliver running in the main processor and a given radio card. New types of radios may be introduced into existing PWNs, provided the new card supports this interface. Each PWN also supports the Socket interface.

A given PWN can also be configured to support some number of *sensor devices*, including a Global Positioning System (GPS). GPS devices are of interest in mobile (vehicular) nodes used in field trials; they may also be useful in stationary nodes since they provide an accurate, synchronized time reference.

Each PWN is either *tethered* or *untethered.* Tethered PWNs include a high-speed backhaul channel to the site's Programmable Edge Node (PEN). This channel may be implemented using either wired technology (e.g., DSL and PON) or dedicated wireless channels (e.g., WiMax). Backhaul channel will be used primarily for management and control. In addition to the backhaul channel, tethered PWNs will also have wireless data channels (e.g., WiFi) that will be used mostly for multi-hop transport experiments. Untethered PWNs do not include a high-speed backhaul channel; they communicate with the rest of GENI through their radio channel(s) only. In this case, both management/control and data channels are implemented using wide area Cellular/WiMax links. However, to leverage intermittent high-capacity connectivity channels, the untethered PWNs will also be equipped with WiFi/UWB interfaces that can be used when an otherwise mobile node (e.g., car or bus) is in a garage or otherwise near a hot-spot. Untethered nodes are intended for experimentation in vehicles or in hand-held devices in the field.

The proposed hardware baseline for a PWN is shown in Figure 3.9. It is organized around a PCI bus with a general-purpose processor, making it a degenerate (single processor) configuration of the cluster-based canonical component. Multiple radio cards of various types are plugged in to the PCI bus, along with sensors and GPS units.

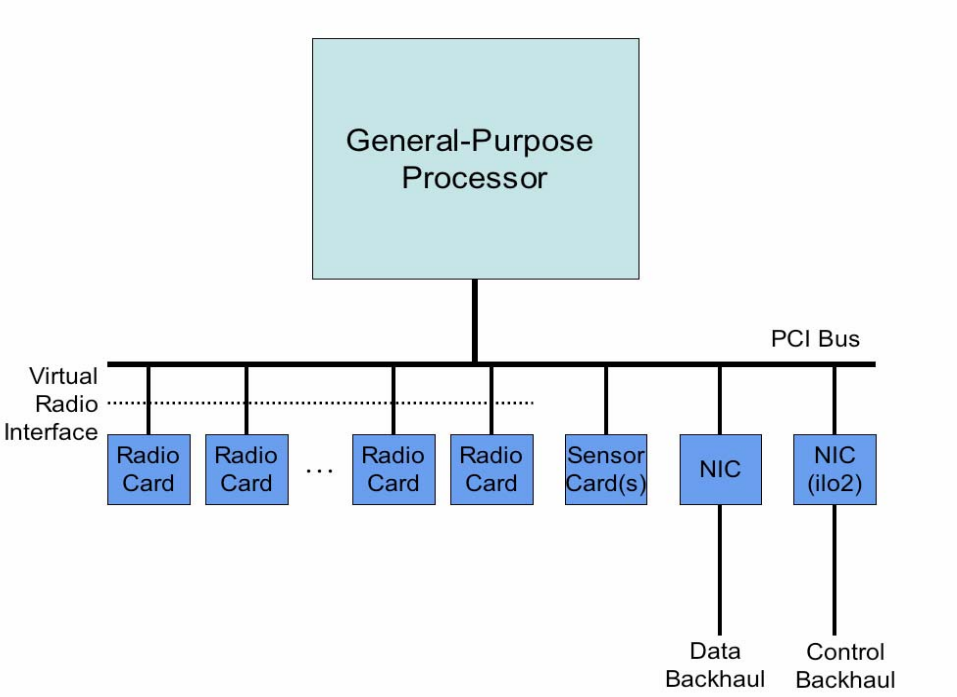

Figure 3.9: Proposed PWN hardware with distinct backhaul control and data channels.

Finally, as shown in Figure 3.9, a PWN can be configured with two backhaul channels: a data channel supporting GENI experiments (corresponding to the tethered case described above), and a control channel used to invoke slice operations and retrieve experimental statistics. This separation is especially important in those deployments that utilize low-bandwidth data backhaul channels. Note that even an untethered PWN can support a dedicated control channel, for example, using a low-speed cellular channel (e.g., GPRS, EVDO). Also note that in some cases, the control channel can be implemented in-band as a reserved sub-channel on the data channel.

3.3.3 Subnet Deployments

We now describe five wireless subnet deployments that are to be included in the GENI facility. In addition to these deployments, we also plan to produce *wireless kits* that allow others to deploy their own subnets, and connect them to GENI. These kits include the software and hardware that collectively define a set of PENs, PWNs, but specifically configured to operate in deployments similar to those described below.

Each subnet is treated as a distinct aggregate, and hence, includes an aggregate manager that implements the O&M Control and Slice Coordination functionality described in Section 2.4.5. This aggregate manager is co-located with the subnet, and connected to the constituent components through a control channel. This control channel is implemented in one of two ways: (1) as a logical sub-channel of the backhaul data channel that connects the components to the rest of GENI, or (2) as a dedicated control channel using a separate backhaul link.

Unique to the wireless subnets, the aggregate manager oversees the RF spectrum for the entire deployment. Essentially, the aggregate checks for compatible allocation of frequencies, power,

and so on, across nodes and slices in the subnet. This function is supported by an independent monitoring infrastructure that includes RF sniffers that measure ambient RF spectra. This data is used to reliably profile of RF activity during experimental activities, whether generated by the experiment itself or externally induced.

802.11 Urban Mesh Network

Mesh networks represent an important area of current research and technology development activity, and have the promise of providing lower-cost solutions for broadband access particularly in medium- and high-density urban areas (Figure 3.10). While protocols for ad-hoc mesh networks have been maturing, the research community has limited field experience with large-scale systems and application development. Research topics to be addressed using the GENI system include ad-hoc network discovery and self-organization, integration of ad-hoc routing with core network routing, cross-layer protocol implementations, MAC layer enhancements for ad-hoc-supporting broadband media QoS, impact of mobility on ad-hoc network performance and real-world, location-aware application studies.

Figure 3.10: 802.11 Urban Mesh Network Deployment.

An 802.11 Urban Mesh Subnet allows experimentation with wireless mesh architectures, as well as with multi-hop radio networks and mobile ad hoc networks (MANETs). It also provides access to GENI-wide experiments and services from moving vehicles and other mobile devices [GDD-06-18].

A field deployment of urban mesh subnet in GENI would consist of ~1000 PWNs densely deployed in one or more urban areas or campus settings with coverage area ~10 Sq-Km. A typical PWN, housed in packaging suitable for urban environments, would be installed at ~8m

height using available lighting poles or other utilities, and would have electrical power and a wired or wireless (i.e., cellular) remote management interface. Approximately 25% of the nodes would be designated as access points with wired interfaces (typically VDSL or fiber) to PENs, supporting experiments that want to send traffic between PWN and PEN without using multi-hop RF. These channels can be ignored for experiments wishing to experiment with multi-hop RF technology. Also, this channel can double as the control channel, eliminating the need for a separate backhaul link. The urban deployment will also include support for ~100 vehicular mobile nodes associated with a suitable bus or taxi fleet operating in that environment. End-user mobile devices can also opt into the GENI urban mesh as described in Section 3.3.3. Note that wireless control and management interfaces such as GPRS or EVD0 will be required at vehicular or mobile nodes, limiting the degree of programmability and remote data collection.

Suburban Cellular/WiMax-WiFi Hybrid Network

GENI will also include a wide-area suburban wireless network with open-access Cellular/ WiMax radios for wide-area coverage, along with short-range 802.11 radios for hotspot and hybrid service models (Figure 3.11). This wireless scenario is of particular importance for the future Internet as cellular phone and data devices are expected to migrate from vertical protocol stacks such as GSM, CDMA and UMTS towards an open Internet protocol model. Experimental research on future cellular networks and their integration into the Internet is currently restricted by the lack of open systems that can support new types of protocols and applications. Research topics to be studied using the proposed experimental network include transport-layer protocols for cellular, mobility support in the future Internet, 3G/WLAN handover, multicasting and broadcasting, security in "4G" networks, information caching/media delivery, and location-aware services.

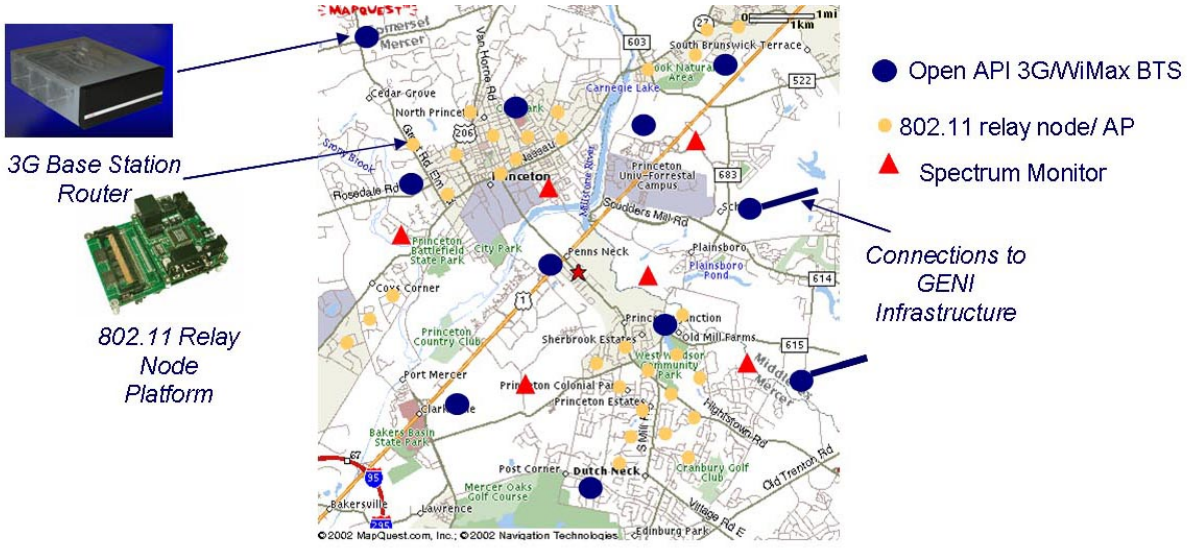

Figure 3.11: Suburban Cellular/WiMax-WiFi Hybrid Network Deployment.

A wide-area Suburban Access Subnet allows experimentation with Cellular and WiMax/WiBro (802.16 d/e) network architectures, as well as with hybrid WiFi/WiMax and hybrid

WiFi/Cellular networks. It also provides access to GENI-wide experiments and services from moving vehicles and other mobile devices.

A field deployment of a wide-area suburban wireless network includes a set of PWNs configured with open-access cellular/WiMax radios for wide-area coverage, and short-range 802.11 radios for hotspot and hybrid service models. Specifically, we expext approximately 10 Cellular (UMTS/CDMA) or WiMax-based nodes and 100 802.11-based nodes covering a suburban area of about 50 Sq-km. The Cellular/WiMax nodes will need to be mounted at heights of ~30m or higher on buildings or towers, while 802.11 nodes can be installed at ~8m on utility poles. A deployment will also include 25-50 mobile nodes with UMTS/CDMA/WiMax mounted on mobile vehicles (e.g., buses or taxis) that move throughout an area covered with 802.11 and/or UMTS/CDMA/802.16 radios.

Each PWN is connected to the PEN by a high-speed backhaul channel (e.g., Ethernet), supporting experiments that want to send traffic between PWN and PEN without using multi-hop RF. These channels can be ignored by experiments wishing to experiment with multi-hop RF technology. Also, this channel can double as the control channel, eliminating the need for a separate backhaul link. The Component Manger (CM) interface will use Cellular data (GPRS, EV-DO) channels for accessing the mobile components.

Cognitive Radio Access Network

A cognitive radio (CR) network deployment is planned as an advanced technology demonstrator, with a focus on building adaptive, spectrum-efficient systems with emerging programmable radios (Figure 3.12) [GDD-06-20]. The emerging cognitive radio scenario is of current interest to both policy makers and technologists because of the potential for order-of-magnitude gains in spectral efficiency and network performance. NSF and industry funded R&D projects aimed at developing cognitive radio platforms are currently in progress and are expected to lead to equipment that can be used for GENI in the 2008-09 timeframe. Research teams are interested in investigating radio spectrum architectures, hardware platform (SDR) integration, cognitive networking adaptation algorithms, mobile/wireless network control and management functions, developing new protocols supporting new network services, and are further interested in testing and evaluating their work in both simulated/emulated and realistic environments.

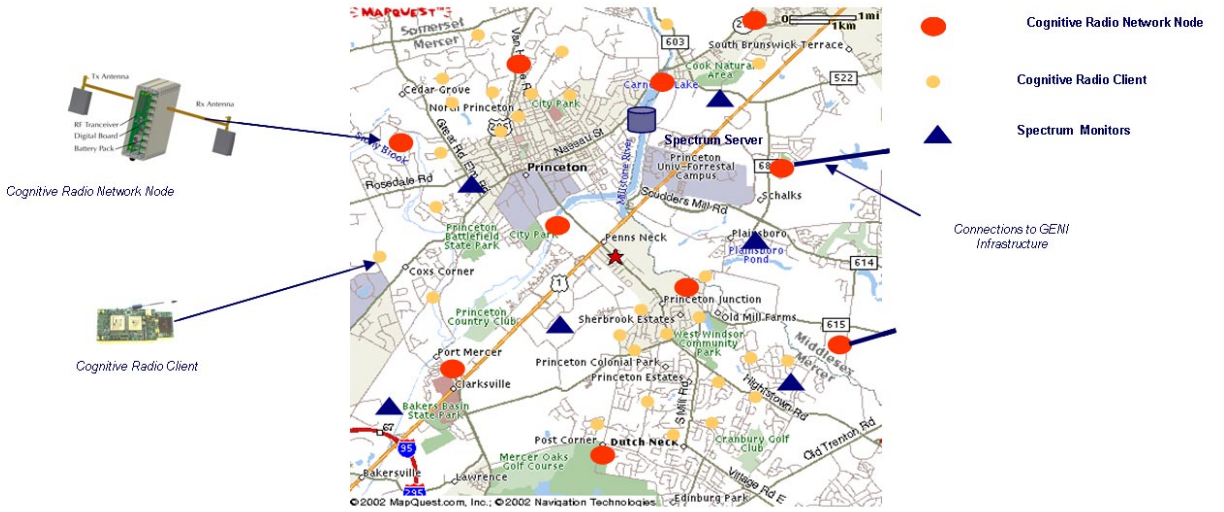

Figure 3.12: Deployment of Cognitive Radio Demonstrator Network

In order to perform experimentation on architectures that inherently build upon larger network services, a CR field deployment will be integrated into and interoperate with the larger GENI facility. This deployment will span a suburban/medium-density coverage area ~50 Sq-Km with the objective of demonstrating and evaluating this technology as an alternative to available cellular and hybrid cellular/WLAN solutions. Deploying this system also involves constructing a distributed spectrum measurement infrastructure along with centralized spectrum coordination resources (such as spectrum broker, spectrum server).

As with all wireless subnet deployments, we will produce a CR Kit that can be deployed in other environments. For example, this will make it possible to augment the Urban Mesh installation described earlier in this section by including these kits as "plug in" upgrades to the PWNs installed in that setting. This will allow us to leverage an existing field for cognitive radio trials without the expense of obtaining permission from additional communities, adding new devices to pole tops, and so on.

Spectrum policy is likely to be a real issue for field installations of cognitive radios, since much of the operation of such radios is under the control of experimental software. It would be desirable to obtain an experimental spectrum allocation from the FCC for cognitive radio field installations; additionally, the hardware should probably be designed so that compliance with important FCC policies can be readily ascertained no matter what software runs on the platform.

Application-Specific Sensor Subnets

GENI will include sensor networks capable of supporting research on both protocols and applications (Figure 3.13) [GDD-06-19]. Since the design of a sensor network tends to be application specific, GENI will provide experimental access to a small number of selected sensor net deployments that leverage the urban and suburban wireless infrastructures described earlier. There will also be a sensor deployment kit consisting of network proxies (from sensor radios to 802.11 or cellular), sensor modules and related platform software to

enable users to build additional sensor networks for specific research or application objectives. Research topics to be studied using experimental sensor net systems include general-purpose sensor network protocol stacks, data aggregation, power efficiency, scaling and hierarchies, information processing, platform hardware/software optimization, real-time, closed-loop sensor control applications, vehicular, smart space and other applications.

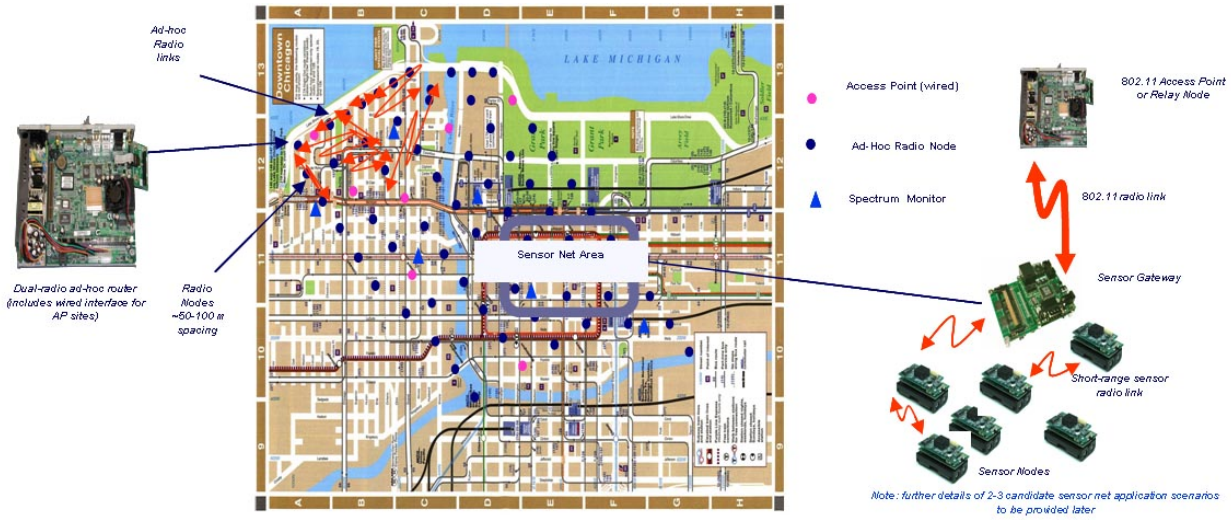

Figure 3:13: Deployment Concept for Sensor Networks

Sensor Subnets provide a means by which networks of sensors and actuators can be brought into the overall GENI facility. These devices are generally microcontrollers, often powered by batteries, and typically communicating over an RF channel.

There will be four kinds of sensor subnet facilities connected to the GENI facility:

- **Local sensor subnets**: These will consist of up to 100 nodes each, and will be designed with a particular application in mind. These subnets will be realized by making kits available to GENI edge sites that have a local sensor application they are willing to deploy.
- **Outdoor deployment**: These will be deployed on a campus or metropolitan scale, and will consist of several sensor "sub"-networks, each designed with a particular application in mind. For example, specific sensor deployments in areas such as environmental monitoring, security, traffic control, vehicular safety or smart spaces. Each such sensor net application is expected to involve up to 1000 sensors and 10's of network gateways. The low-tier sensor nodes interface with a gateway typically at distances ~10m or less, while gateways would in turn connect to either 802.11 or cellular nodes with commensurate coverage areas.
- **Indoor deployment**: This deployment contains a large number of sensor nodes (roughly 1000), and allows sensor network researchers to experiment at scale with a greater variety of traffic patterns and network configurations than possible with the deployed regional backbone, and in a repeatable and controlled manner.

- **Existing sensor networks**: There are several existing sensor networks that might be integrated into GENI via a "GENI sensor proxy", which can be thought of as a standard proxy provided for sensor networks.

Note that sensor nodes can be incorporated into GENI using the methods outlined in Section 4.3.4: they may be fully-compliant GENI components (e.g., a PWN configured with a sensor card or Stargate devices that have been modified to support a Component Manager and slices); slice-supporting devices managed by an external CM using a private CM-device interface; or a "GENI unaware" motes (or equivalent) that connect to GENI via a GENI sensor proxy running on a GENI-compliant component. We expect all three types to be deployed in each subnet.

Emulation Grids

Emulation grids provide a means by which experiments with wireless subsystems can be run in a controlled, repeatable way. They complement the large-scale field deployments that support experimentation "in the wild." The emulation grids are intended to be compatible with the field deployments, so that the same user experiments can run unchanged in both environment, or any mixture of the two. We envision two types of grid facilities: (1) GENI Wireless Simulation Clusters and (2) GENI Controlled Experimental Facility.

The first, a *wireless simulation cluster*, provide a means by which wireless network simulations may be linked into the end-to-end GENI system. These simulations run exactly the same researcher code as a PWN, but provide simulated radio interconnectivity rather than actual radio channels between the nodes. By selecting the appropriate radio channel simulation software, researchers can simulate networks of PWNs, including ones configured with sensors and cognitive radio cards.

A Simulation Cluster utilizes the same hardware as a Programmable Edge Cluster, sized to support a wireless network with up to 500 nodes (i.e., with 50-100 processors). It includes software to simulate radio channel behavior, along with instrumentation and data archiving. We expect simulations running on these clusters to be controlled by the same O&M Control as other GENI components.

The second, a *controlled experimental facility*, consist of RF-shielded, anechoic rooms in which repeatable experiments can be performed with real, "over the air" radio signals. These rooms will be suitable for experimentation with Programmable Wireless Nodes, including those configured with cognitive radio cards and sensor devices. As for Simulation Clusters, the Controlled Experimentation Facilities will be compatible with field-deployed wireless subnets, allowing researchers to try move their experiments between the grid and field deployments, as necessary.

3.4 Discussion

Throughout this section, we describe a set of seemingly distinct components: Programmable Edge Clusters (PEC), Programmable Core Nodes (PCN), Programmable Edge Nodes (PEN), and Programmable Wireless Nodes (PWN). One might ask exactly what distinguishes one component from another. At their core, they share a common design: they are constructed from a set of general-purpose processors running virtualization software, the so-called canonical

component. Each is then specialized or configured for a particular purpose (e.g., the PEC scales to a large number of processors, the PEN and PWN include RF line cards, and the PCN is configured to drive high-speed link).

However, these distinctions are not always so crisp, with hybrid configurations, or sets of components, a possibility at any given site. For example, a site might require a configuration that includes a large cluster of processors, outfitted with wireless cards to support a local wireless subnet, and configured with high-speed line cards to support a high-speed tail circuit to the backbone. The configuration might even include the Circuit Processing Subsystem of the PCN if the site is connected to the backbone by a dedicated fiber. At the same time, a small-scale PEC is essentially identical to a tethered PEN.

4 User Services

The physical substrate provides the raw material from which researchers construct experiments. A set of *user services* collectively knit these building blocks together into a coherent scientific instrument, leveraging the slice, component, and aggregate interfaces defined in Section 2. The goal of the services is to give researchers the means to create, manage, and harvest scientific measurements from experiments running in GENI. Simply stated, these services make GENI *usable*, lowering the barrier-to-entry for researchers to effectively take advantage of the unique capabilities of the physical substrate. None of these are part of the GMC, but instead, they are built on top of the interfaces defined by the GMC.

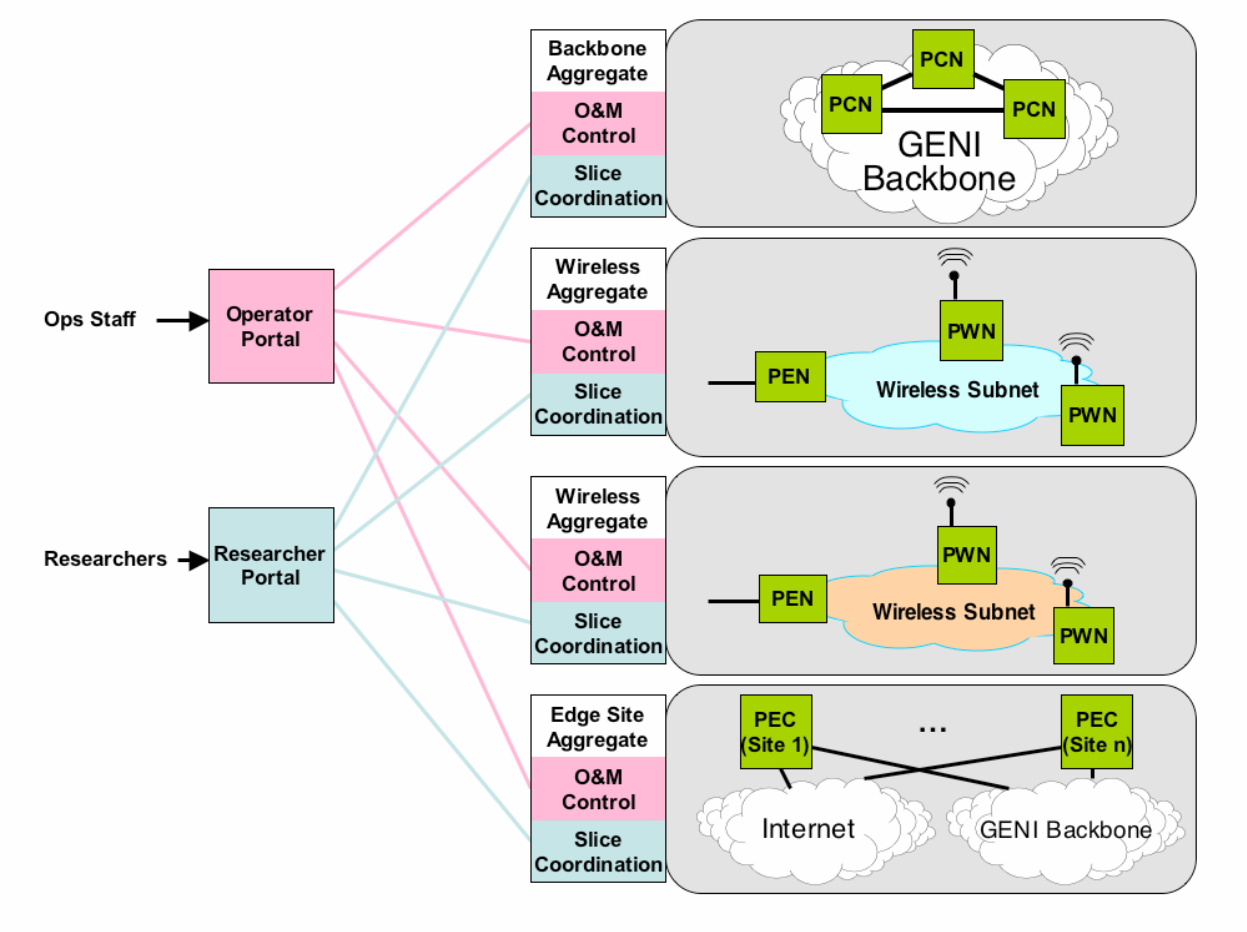

Figure 4.1: Operator and Researcher Portals interact with subnet aggregates (combined O&M Control and Slice Coordination functions) to provide a front-end to operators and researchers, respectively. Although not shown, the services encapsulated by each portal may also run in slices that span GENI's components.

This section describes these services, organized around two main user populations: operators and researchers. As introduced in Section 2, we use the term *portal* to denote the interface (front-end) to the set of services tailored for a given user community. Figure 4.1 depicts the general relationship between the portals and the set of components and aggregates that make up the substrate. This section also described common sub-services upon which researchers and operators will depend, and outlines tools used to collect and analyze data.

4.1 Operator Portal

For GENI to adequately serve the research community at the envisioned scale, it must provide a stable and reliable platform for running experiments. Problems with GENI hardware and software must be detected and resolved quickly to avoid impacting experiments in progress. Experiences with smaller-scale network testbeds, supercomputer research centers, and commercial communications networks, have demonstrated the importance of a comprehensive approach to network and system operations. These experiences have also demonstrated the

need to design operations functions early in the overall design process rather than retrofitting operations functions into an existing system.

The *operator portal* provides a front-end to this functionality, tailored for use by network operators. We assume that GENI will have a relatively small full-time operations staff, and as a consequence, the tools used to manage GENI must be able to detect and resolve most problems with minimal human intervention. That is, GENI operations must be automated to the greatest extent possible.

To ensure complete coverage of the problem space, we classify the major GENI operations functions in terms of the FCAPS model developed by the International Telecommunications Union. FCAPS is an acronym, where each letter represents a different management function:

- **F**ault Management
- **C**onfiguration Management
- **A**ccounting Management
- **P**erformance Management
- **S**ecurity Management

While each FCAPS function captures one aspect of operational support for GENI, they often rely on information gathered by (and functionality implemented by) other parts of the system. The rest of this section considers each of these five functions, both describing the function and showing how it is integrated with some part of GENI's design. To re-emphasize, the goal of the operator portal is to define a convenient interface (front-end) that the GENI operations staff can use to manage GENI—most of the "management systems" defined in this section are really "interfaces" to functionality implemented by components, aggregates, registries, resource monitors, user services, and so on.

In addition to the FCAPS model, we further divide operations functions into *online functions* and *offline functions*. Online functions directly interact with GENI components (e.g., restarting a faulty node) while offline functions are used to manage less critical operations tasks (e.g., tracking the repair of known bugs in GENI infrastructure). The FCAPS operations functions are all online functions.

4.1.1 Fault Management

The *Fault Management System* detects problems with GENI hardware and software, and initiates the process of repairing these faults. Not every fault that is detected must be acted upon. Experience building fault management systems for the communications industry has demonstrated the need to perform accurate *root-cause analysis* of failures. For example, if many GENI slices are using a node and that node fails, the operations staff does not need to know that the slices have failed (symptoms of the problem), but instead should know that the node has failed and why it has failed (the root cause of the problem).

To perform accurate root-cause analysis, the GENI Fault Management System needs several kinds of information: (1) fault data from individual GENI nodes and links, (2) topology of all GENI nodes and links, and (3) rules that characterize when a GENI component may be faulty.

Monitoring services running on each component will collect data about the health of that component; aggregate managers will coalesce this data on behalf of the subnets they manage. This data represents the primitive events used by the Fault Management System to perform root-cause analysis, and in many cases, it corresponds to data collected to discover available resources (Section 4.2.2) and on behalf of experiments (Section 4.4). Examples include information about hardware failures, processes performing unauthorized actions, and slices consuming too many resources. To avoid excessive use of GENI resources, the GENI component and aggregate managers likely perform low-level analysis of the primitive events.

Topology information is critical for understanding the relationships between various GENI components and how problems with one component affect the rest of GENI. Understanding the relationship between components is essential for performing root-cause analysis. The GENI topology data is maintained by the GENI Configuration Management System (Section 4.1.2) and made available to all GENI management systems.

The Fault Management Rules describe the mapping between an observed set of faults with the root-cause problem requiring corrective action. Experience with similar systems has shown that system failures often exhibit observable and repeatable behavior. By codifying these behaviors as rules, it is possible to accurately identify problems. For example, an occasional dropped packet in a communications network is not unusual. When the number of dropped packets per unit time exceeds some threshold, however, there may be a more serious problem. As experience is gained with GENI, we believe it will be possible to define a set of rules that can identify many kinds of failures automatically.

Once the root cause of a problem has been identified, it must be repaired. Restarting failed software modules repairs most failures. Some problems, however, can only be repaired with a hard reset of the hardware. This requires the hardware monitor functionality described in Section 2.4.4. In the most extreme cases, human intervention is required. The Fault Management System must therefore have the ability to alert a person that a problem exists and provide a mechanism for the person correcting the problem to notify the Fault Management System when the problem has been resolved. If the problem is not resolved within a specified time interval, the Fault Management System must also have a mechanism to escalate the problem.

Because GENI is a highly distributed system, many GENI resources will not be under the direct physical control of the GENI Operations Staff. Consequently, the GENI Fault Management System must be able to contact administrators at remote sites when a problem is occurring at their site. The Accounting Management System (Section 4.1.3) keeps track of the people and institutions associated with GENI resources, so this data can also be used by the Fault Management System to notify the appropriate people of problems at their site.

Examples of fault management systems in use today that might be considered starting points for the GENI Fault Management System include the NAGIOS open source network and system monitoring application, HP OpenView, and Micromuse (now part of IBM) NetCool.

4.1.2 Configuration Management

The *Configuration Management System* facilitates the orderly introduction of new nodes, links, and sensors into GENI and tracks where hardware components are located, what versions of software they're running, which patches have been installed, and whether they're in a valid configuration.

The Configuration Management System leverages the component registry described in Section 2.4.2. When a new GENI component is physically connected to the rest of GENI, it announces itself (and its capabilities) in this registry, resulting in an inventory and topology database. This database (registry) keeps track of all pertinent details associated with every GENI component, which can be used by the Fault Management System (Section 4.1.1) to perform root-cause fault analysis, as well as by the Accounting Management System (Section 4.1.3) to map GENI resources to the person or organization responsible for maintaining them.

The Configuration Management System provisions, configures, and validates new components. This is also done in a decentralized way, involving the aggregate managers (AM) running throughout GENI (Section 2.4.5). Each such AM is responsible for validating and configuring the components in the subnet it manages. This process involves an interaction between the AM and individual component, initiated for example, by the boot program running on each component.

Examples of configuration management systems in use today that might be considered starting points for the GENI Configuration Management System include the PlanetLab management authority, Telcordia Granite Inventory Manager, Amdocs Cramer Inventory, and MetaSolv (recently acquired by Oracle) Inventory Management software.

4.1.3 Accounting Management

The *Accounting Management System* is used to map GENI users to real people and institutions, and to control access to GENI resources. In doing so, the Accounting Management System can ensure that only authorized users and experiments are permitted to consume GENI resources, and express policies about how many resources they may consume. If an experiment causes problems for other users, the Accounting Management System can help the GENI Operations Staff determine the person responsible for the faulty experiment and their institution.

Accounting in GENI is managed hierarchically: a hierarchy of slice authorities take responsibility for the behavior of slices and the users that create and use them, and a hierarchy of management authorities are responsible for the correct operation of components. This information is recorded in the component and slice registries, respectively (Section 2.4.2). The Accounting Management System will simply provide an interface through which resource allocation policies can be set. These policies will be set by the GENI Science Council, and defined in terms of both aggregates of components and slices.

Examples of accounting management systems in use today that might be considered starting points for the GENI Accounting Management System include accounting management tools used in PlanetLab and Emulab, and the Grid Account Management Architecture (GAMA) developed at the San Diego Supercomputer Center.

4.1.4 Performance Management

The *Performance Management System* supports fine-grained tracking of resource usage. The data collected by this system is accessible to the GENI Operations Staff, but it is also made available to experimental services that adapt their behavior in response to current network conditions. In many cases, this is the same data collected to discover available resources (Section 4.2.2) and on behalf of experiments (Section 4.4).

There is significant overlap between the Performance Management System and the Fault Management System described in Section 4.1.1. While the Fault Management System's primary function is to detect when GENI resources fail completely, the Performance Management System looks at the utilization of various GENI resources to determine when they are overloaded or trending toward becoming overloaded.

In addition to component and aggregate oriented data, the Performance Management System also monitors the resources consumed by each slice to determine when a slice is operating outside of its resource profile. Ideally, automated "anomaly detection software" can be used to flag the early stages, giving researchers an opportunity to correct their experiments before it is necessary to escalate. Ultimately, misbehaving slices will need to be suspended or terminated.

When building the Performance Management System, commercial the same network-monitoring tools available for fault management could be used as the starting point. Additionally, the CoMon system built used in PlanetLab also has much of the functionality necessary for a Performance Management System.

4.1.5 Security Management

The *Security Management System* logs all security-related events in an auditable trail, thereby allowing the GENI Operations Staff to determine when GENI is being attacked, or an experiment it hosts is violating the GENI Acceptable Use Policy. Given the scale and level of programmability of GENI, and its potential for being used to attack other users and networks, it's critical that potentially bad behavior be detected and stopped as quickly as possible. The Security Management System primarily leverages the audit trail generated by the resource monitors that enforce access on GENI components, but also works closely with the Fault Management System (Section 4.1.1) and the Performance Management System (Section 4.1.4) to detect security-related problems and alert GENI Operations Staff before serious damage is done.

Sometimes a security alert or Acceptable Use Policy violation is not the result of a malicious attack, but rather the result of a buggy experiment, or merely a user trying to create a slice without supplying tickets granting rights to the required resources. Making security-related data for a specific experiment or slice available to the appropriate researcher may help them determine why their experiment did not run correctly. The traffic auditing system running on each component (Section 2.4.4) is able to map packet signatures back to the responsible slice, which the slice registry is then able to map into human users, including both researchers and the PIs that have vouched for them.

The Security Management System must also monitor the ongoing setup and configuration of user rights in the Accounting Management System (Section 4.1.3). Repeated failed attempts to create new users, change the user's privileges, or associate users with pre-existing slices may indicate a potential security violation. When these problems are detected, the GENI Operations Staff should be alerted.

Tools like intrusion detection systems, policy-based network management systems, and logging and auditing mechanisms available in modern operating systems can also serve as building blocks of the Security Management System. Components can leverage these facilities in conjunction with the GENI-specific resource monitor enforcement functions, as well as send low-level security events to audit and alerting software dedicated to analyzing the large number of security events and presenting higher-level interfaces. The Operations Staff notification mechanisms built into the Fault Management System should also be reused here.

4.1.6 Offline Management Functions

As stated at the beginning of this section, the FCAPS functions required for operations support are on-line functions. They directly manipulate and collect data from the GENI infrastructure to keep it running smoothly. In addition to these on-line functions, GENI should also provide offline tools to aid the Operations Staff in doing their job.

The most important offline function for GENI in the short-term is *problem tracking*. When GENI is fully operational, a large user base will rely on it to conduct their research. This user base will undoubtedly discover problems with GENI and have suggestions for improving GENI. Users may also have questions of the form "Why is this traffic affecting my experiment?" Consequently, the GENI Operations Staff must have a mechanism for tracking GENI bugs, user questions, maintenance requests, and other problems.

Once the problems and questions are logged, there will likely be more requests for changes to be made to GENI than resources to make them. The GENI Science Council must be able to view these requests and prioritize the relative importance of them to the research community. After the change requests have been prioritized, the GENI Operations Staff can use the problem tracking system to log when the changes have been made and use standard source-code configuration management tools to make sure that new GENI software releases are built correctly.

Tracking problems with software is by no means unique to GENI. Many tools already exist that can be readily applied to GENI. Examples of such tools include Best Practical Solutions LLP Request Tracker (RT), Bugzilla, and the IBM/Rational ClearQuest tool suite. RT and Bugzilla are both open-source tools and ClearQuest is a commercial product.

To facilitate collaboration, cooperation, and communication amongst the various members of the GENI research community, offline tools that allow researchers to exchange "best practices", share code, and discuss experiments should be provided. These tools should also allow GENI Operations Staff to communicate with GENI users to report outages, upgrades, user meetings, etc. By also providing a discussion forum for GENI users to ask and answer each others' questions, the support burden on the GENI Operations Staff can be reduced. Conventional websites, email, and Wiki technology can be used to build this functionality. The discussion

forums should be modeled after the kinds of support forums provided by many computer-technology manufacturers on their websites.

4.2 Researcher Portal

A *researcher portal* allows researchers and developers to acquire GENI resources, specify their service requirements, and manage their running experiments. An easy-to-use yet comprehensive researcher portal is critical to GENI's success. This portal is really a front-end to a set of services that collectively give researchers the ability to create and steer their experiments. We group these services into three levels. At the bottom-most level, *resource allocation* defines how component resources are shared (acquired, scheduled, and released) among a set of experiments. On top of this foundation, *slice embedding* governs the process of instantiating a researcher's slice on some set of components. Finally, at the upper-most level, an *experimenter workbench* provides a set of tools for researchers to create, configure, and control their experiments. We expect most researchers will interact with GENI through the workbench, but the lower-level services are also available, both to foster innovation at higher levels and to support advanced users that wish to directly access GENI.

This section describes these three levels from the bottom-up. A more detailed description can be found elsewhere [GDD-06-24].

4.2.1 Resource Allocation

Resource allocation is key to any shared computing and communication infrastructure. In the case of GENI, the underlying facility will potentially be shared by thousands of simultaneous users. While there will be significant resources available system-wide, GENI will suffer from a ``tragedy of the commons"—whereby the system loses value for all participants—if all users are allowed to greedily consume as many resources as possible. One or more *brokerage services* address this problem.

Existing shared testbeds either perform strict space sharing, where individual jobs control a subset of nodes through completion (e.g., Emulab and DETER), or best effort scheduling, where individual jobs receive resources inversely proportional to the number of currently competing jobs (e.g., PlanetLab). However, to achieve both high system utilization and to allow at least some applications to run with dedicated resources, some fraction of GENI's resources will be subject to admission control at a fine granularity; i.e., dedicated access to portions of multiple machine's CPU, memory, network, and storage resources. The global GENI resource allocation mechanism will interface with individual Component Managers to effect appropriate resource isolation. By default, most slices will run in *best-effort* mode (perhaps with some priority relative to other slices) to ensure a high-level of resource utilization and to reduce the barrier-to-entry for running experiments on GENI. The amount of resources that can be reserved by an application, the relative priority of best-effort applications, and the proportion of global resources subject to admission control are all matters of policy.

An important task of resource allocation is to encourage efficient resource usage, that is, to ensure that users acquire only as much resources as they truly need, and for only the period of time that they really need them. This second point is particularly important. The duration of a particular resource binding should be coarse-grained to reduce the overhead associated with

performing resource allocation (i.e., we wish to avoid requiring an application to renew its resource privileges every few seconds). On the other hand, we believe that resource re-negotiation must take place more frequently for experiments as they acquire increasing amounts of resources. For instance, a slice with dedicated use of a substantial fraction of global GENI resources may be required to renew its resource privileges every few minutes (to allow the system to be agile to changing resource demands), while a best-effort slice running with low priority may only be required to re-negotiate on the granularity of days or even weeks. In general, GENI's resource allocation mechanism should be general enough to support a variety of policies for the duration of resource bindings, but guaranteed resource allocations must always be time-limited.

The resources allocated to individual slices will often be for shorter durations than the lifetime of those slices. Thus, many GENI slices should be prepared for dynamically changing levels of available resources independent of whether they are running with resource guarantees or in best-effort mode. To allow experiments to cope with a constantly changing resource landscape, the resource allocation mechanism will support callbacks to the slice (or designated agents of the slice) to inform it of current or upcoming resource availability changes.

GENI's resource allocation mechanism is agnostic to specific policies. The underlying principle is that individual component owners maintain local autonomy over their local resources. While many components will be owned by and under the control of GENI itself, this principle aims to encourage other organizations to contribute resources, with the assurance that they will maintain some control over how their components are used. This control begins with individual Component Managers, which decide to grant *tickets* giving control over a subset of locally available resources to one or more *Resource Brokers*, using the interfaces described in Section 2.2.2. Initially, there will be a single logical Resource Broker, which we describe in this section. We expect that as GENI evolves, other Resource Brokers will be developed. Some may specialize in certain types of resources, while some may use new marketplace or economic models to determine the value of resources.

In exchange for granting tickets to a Resource Broker, each participating organization receives a broker-signed capability granting access to some portion of global system resources. While the resources granted by the Component Manager are specific (e.g., individual machines for some period of time), the resources received in return are abstract. We call them *tokens*. The value of these tokens strongly depends upon the specific resource allocation policy that GENI wishes to enforce.

Tokens are cryptographically signed statements of privilege over some portion of global GENI resources. They may be specific, for instance, dedicated access to a wireless frequency for some period of time as determined by the GENI Science Board, or more general, for instance shared access to computation, storage, and bandwidth across the entire GENI substrate. The encoding and interpretation of tokens is left to the requirements of the GENI Science Board and the particular implementation of Resource Brokers. Relative to the *tickets* described earlier, tokens are a broker-level concept and present a higher-level abstraction of resource privilege. Users and organizations present tokens (describing global system resource privileges) and a description of their compute requirements to Resource Brokers. The brokers then in turn grant tickets for specific resources to individual users. Said another way, a token can be viewed as an

under-bound or under-specified form of a ticket, where a brokerage service binds a token to concrete resources by converting it into a ticket that can be used to acquire specific resources.

Once an organization has offered resources to a Resource Broker and the Resource Broker has returned one or more tokens, the organization can subdivide those tokens among its users. In general, users can receive tokens from a Resource Broker, another user, or the GENI Science Board. The latter allocates tokens according to merit-based evaluations of experiments. Users and organization may subdivide tokens for their own use (e.g., dividing a token of value 100 into 100 separate tokens each of value 1 to enable proportional-share access to 100 separate nodes) or to share with other users or other organizations. Whether such token pooling is allowed is a question of policy. For instance, it is easy for GENI to disallow the use of any tokens that have been transferred from one user to another, because the software verifying the validity of a given token verifies the entire chain of transfers.

4.2.2 Slice Embedding

In contrast to the component-level resource allocation mechanism that is sufficient to allocate single or a small number of related/co-located resources, the slice embedding service enables users to issue commands to find a collection of system-wide GENI resources that match some criteria, such as constraints on network capacity or processor load. Results of such queries can be used to inform the placement of individual slivers that make up a slice, a critical step before an experiment is deployed. Thus, the slice embedding service sits on top of the more foundational resource allocation mechanism and, in turn, exports a service that allows the user or an experimenter's workbench to initiate system-wide experiments.

As with other GENI services, we expect that, as GENI evolves, multiple slice embedding services will emerge. GENI will start by providing a generic embedding service, suitable to the needs of most users. However, specialized domains may call for specialized embedding services that take into account unique features and needs. A complex slice that utilizes resources of many different types may submit sub-sets of its request to several specialized embedding services.

At the most basic level, slice embedding comprises the following three steps:

1. **Resource discovery**: Requires monitoring the state of substrate components using sensors and other measurement processes. The monitored state characteristics include processor loads, memory usage, network performance data between pairs of components, and so on. One important issue is the frequency that various sensors gather the required performance data. To the extent possible, the information should be collected by a single logical entity (rather than repeatedly re-measured by multiple entities) and made available to a variety of applications and services that may benefit from it, such as multiple resource discovery services and adaptive applications. Resource discovery likely leverages the monitoring capabilities involved in both fault diagnosis (Section 4.1) and instrumentation and data analysis (Section 4.4).

2. **Query processing (matching):** This step matches researchers' requests with the available resources and selects some set of components to host the slice. This task is by far the most complex step in slice embedding. Its design depends on a number of considerations relating

to researcher criteria. These include node-oriented constraints (e.g., desired physical location, processor speeds and storage capacity, and type of network connectivity), as well as inter-node constraints (e.g., network topology and combinations of network technologies).

3. **Slice-wide resource allocation:** This step involves assigning the resources that match the user query to the appropriate slice. As multiple resources are typically required for an individual slice, the slice embedding service needs to invoke the appropriate resource allocation methods on individual components. If the resource allocation step fails, the matching step has to be re-invoked. In addition, this step can be repeated periodically to acquire additional resources (or new resources for a particular period of time) for a slice that has already been embedded on some set of components.

Constraints

The slice embedding service has to respect two potentially conflicting concerns. On one hand, decentralization of the service eases the integration of new types of resources, as each aggregate of components (e.g., a subnet) knows best about its resources and how to allocate them. On the other hand, a fully decentralized solution increases communication and implementation complexity as it needs to perform coordination among many distributed components.

It is also worth noting that if resources are to be used in best effort mode, as opposed to exclusive access achieved by reservations, then the embedding problem is easier. In the case of reservations, individual operations may fail requiring reservation operations to be reinitiated. Alternately, if hard reservations are required, the lower-level resource allocation mechanisms could export a form of two-phase commit operation; the resource allocation mechanism could support a tentative reservation primitive, which could be invoked first as soon as the slice embedding service has identified a set of target components, followed by a hard reservation operation invoked once the slice embedding service has contacted all of the target components.

Slice embedding is also complicated by inter-node constraints. Resources matching node-specific constraints, such as CPU capabilities, are easier to identify. Once the user desires inter-node constraints slice embedding becomes difficult. For instance, the user may specify inter-node latency requirements expressed using upper/lower bounds, and similar constraints on bandwidth and loss-rates. The most general form of such constraints is an actual topology with desired connectivity requirements. These types of constraints are difficult to satisfy in the general case, with the topology embedding degenerating into an NP-complete problem. Such inter-node constraints are computationally hard to realize even within the context of a single subnet resource allocation system. Thus, slice embedding services that handle large slices with inter-node constraints will have to rely heavily on heuristics.

Finally, inter-node constraints that express requirements across node types are also more difficult to satisfy than homogeneous constraints. For example, the experimenter could request resources at two different sites with x wired nodes and y wireless nodes per site. These constraints are hard to realize if resource allocation is performed by different subnets. A partial solution computed by the wired subnet might not lead to a complete solution if the wireless subnet cannot find an appropriate set of wireless nodes co-located with the partial solution.

Approach

Our approach is guided by the following principle: The slice embedding service should support the simplest forms of allocation in its initial rollout, but should provide enough flexibility to allow users to hand-pick their resources, if necessary, by running specialized matching code. To accomplish this, each aggregate exports an embedding interface with the following features:

- A resource discovery method that can be queried to find the resources available within the aggregate.
- A query matching method that returns the set of resources that match user queries, but does not allocate them to the user.
- A query matching and resource allocation method that does both matching and resource allocation in a single step.
- A resource allocation method that allocates a set of resources explicitly named by the user.

Given such an interface, the embedding service operates as follows.

- **Policy enforcement:** If allocation policies are stored only at a single researcher portal, then all queries have to be routed through it. In particular, if allocation policies are stored at each aggregate and if the aggregate account for them while performing resource allocations within its domain, then one can eliminate the need for a single point of entry. Also, note that the task of stitching results from different aggregates is not a privileged operation: it does not have to be performed by the portal, but can instead be orchestrated by a slice embedding service running on any node in the system. This also allows for multiple slice embedding services in the long run.
- **Slice stitching:** Resource allocation across aggregates is performed in the following manner:
  - Slice embedding service breaks a request into different pieces based on the aggregates that can perform the resource allocation requests.
  - It picks the most difficult to satisfy request (using some heuristic) and routes the request to the appropriate aggregate.
  - It attempts to extend the result by iteratively contacting other aggregates.
  - In most simple cases, especially if only best-effort allocations are required for resource allocation in the subsequent aggregates, extending the results should be easy to accomplish. For instance, if a wireless frequency has to be reserved and then a best-effort allocation has to be made on a wired node at the same site, it should be easy to accomplish.
  - If such a stitching fails, then the slice embedding service simply throws an exception and the burden falls on the user to explicitly choose the resources that are appropriate.
  - The aggregates can perform allocation of resources explicitly named in a request initiated by the user.

– If such an operation fails, the user can either retry with a different choice, or accept partial success.

4.2.3 Experimenter Workbench

For GENI to enable experimental network and distributed systems research, it must be accessible to the broadest set of researchers, including those with a single PI working on a project with a single graduate student in a school with little prior institutional experience in building distributed systems. Such users should find it feasible to map their experiment onto GENI in a straightforward and predictable fashion, and to manage the results they collect in a way that accelerates the discovery process. Desired attributes of the experiment management toolkit include:

- **Support gradual refinement**: We believe that a primary requirement for ease of facility use is a smooth implementation path from simulation to deployment, with a single, integrated toolset.
- **Flexible programming environment**: The workbench should export experimentation functionality in various forms to support the requirements of a variety of users. A beginner might desire a single monolithic tool to address all aspects of experimentation, such as resource discovery, experiment setup, experiment monitoring, and the gathering of results. But with additional experience, the programmer might desire greater flexibility and not be constrained by tools that prescribe a specific sequence of operations in setting up. It should support the use of existing tools to automate all aspects of the experiment.
- **Sophisticated fault handling**: It is likely that experiments will face a variety of failures on a frequent basis. The experiment support toolkit should enable the gradual "hardening" of distributed services by first allowing service execution in controlled settings, where faults can be deterministically and gradually administered and programs can be spared from having to face the full brunt of faults.
- **Instrumentation, data collection and archiving**: The workbench should provide convenient interfaces for users to manage instrumentation and data collection (Section 4.4). These tools should also enable published work to reference the relevant data collected, resources used, and code executed as part of an experiment. It should also enable other researchers to repeat experiments and compare against published results.
- **Multi-experiment workflow:** Users will not conduct just a single experiment, but many. The workbench will need to support workflows over groups of related experiments, as well as tools to help researchers navigate through a history of experimental results. Encapsulation—packaging the specifications, code, and configuration parameters that define an experiment—and sophisticated search facility are important tools in this process.

Some of these requirements have been translated into workbench efforts on smaller testbeds in the past. For example, the DETER testbed has a security workbench that provides a set of tools that experimenters can draw upon to automate a series of repeatable tests using existing software and analysis tools [DETER]. It also provides methodological guidance that aids the experimenter to design security related experiments and supports both a graphical user interface and a perl-based scripting language to allow even a novice user to rapidly experiment with different configurations. Similarly, the Emulab experiment management system provides

support for realistic, large-scale, replayable networking research [ESL07]. It implements a scientific workflow-based experimentation model that allows people to move forward and backward through their experimentation processes. Integrated tools in the workbench enable researchers to manage their activities (both planned and unplanned), software artifacts, data, and analyses.

Given the above requirements and the past workbench development experience, we make the following observations about the overall design for experiment support and how it might be broken down into quantized components and introduced over time.

- Develop and implement an API for experiment instantiation and system-wide job control using some parallel execution mechanism. Develop a shell program to allow interactive invocation of the tasks provided by the API. Demonstrate that a new user can quickly and easily deploy experiments on Emulab, PlanetLab and GENI using the shell.
- Develop support for using the API with scripting languages, such as Python and Perl.
- Develop more advanced support for parallel process management, including suspend/resume/kill, and automated I/O management. Also develop support for issuing system-wide commands in asynchronous mode, querying the status of previously issued asynchronous operations, and to synchronize subsequent operations with those initiated earlier.
- Provide support for simple forms of node selection, e.g., based on processor load or memory availability. The goal is to provide the abstraction of "run this experiment on some set of nodes." Also, for experimenters that require specific topologies, provide algorithms/heuristics that facilitate mapping user requirements to available topologies.
- Develop support for global synchronization: Following along the lines of Plush, the toolkit should provide various forms of synchronization primitives for use during program execution. Synchronization primitives should span the entire range from traditional barriers to weak partial barriers and point-to-point synchronization.
- Develop support for permanent services. Develop mechanisms for continuous monitoring of program state and reporting faults to the user. Develop support for deploying experiments on heterogeneous GENI resources. This would include support for specifying network configuration, controlling (or injecting) network events, exporting information regarding network conditions, and providing more control over resource allocation for a diversity of resources. The goal is to provide the abstractions of "keep this service running" for service developers and "run this experiment on a matching set of nodes/topologies" to network experimenters desiring a specific system configuration.
- Interface the experiment support toolkit with other services described in this document, such as performance monitoring and data archiving.
- Develop support for fault-tolerant execution: In particular, develop support for commonly used design patterns such as transactional operations, two-phase commits, or operations with “executed exactly once” semantics.

- Develop intrusive and non-intrusive techniques for monitoring program state, detecting abnormal behavior, and debugging support such as single-stepping.
- Address scalability issues so that the control infrastructure can scale to hundreds and thousands of nodes without developing hotspots. Control signals, monitoring, and data gathering needs to be performed over an overlay tree or a mesh to achieve scalability.
- Address network reliability issues by having the monitoring and control infrastructure use a resilient communication layer that routes control messages around network faults and hides transient connectivity problems (as in MIT-RON).

Additionally, we plan to consider developing workbench tools that provide ontology-based guidance to novice experimenters when defining experiments, tools for program and protocol analysis, experiment validation and verification..

4.3 Common Sub-Services

This section outlines a shared set of building block services that contribute to making GENI easy to use. In some cases, these services provide a set of common abstractions upon which experimenters can build rather than forcing them to reinvent from scratch. In other cases, abstracting these services into separate modules will also make the facility itself easier to construct, by avoiding duplication. We focus on three separate areas: basic communication services, basic file storage services, and access to the legacy Internet.

4.3.1 Communication Services

A set of communication services provides common abstractions upon which higher level functionality can be built. Generally speaking, these services support typical communication patterns used to distribute data to, collect data from, or to control GENI systems.

The first is a *bulk transfer service* that can be used, for example, to load experiment code onto a distributed set of components. Such services typically break large files into chunks, distribute those chunks across a distributed set of caches, and deliver the chunks in parallel to client machines. The combination of caching and parallel transfers allow such services to scale in terms of the number of clients that can be simultaneously serviced. Without such a service, the server hosting the large file can be easily overloaded by too many client requests. Example bulk transfer services include CoBlitz [PP06], Bullet [KRAV03], and SplitStream [CDKN03].

The second is an *event dissemination service* that efficiently delivers small messages (events) to a widely distributed set of consumer machines. An event system can be used for distributed control and co-ordination across the slivers in a slice. Event services are often designed around a publish-subscribe (pub-sub) paradigm, in which clients register to receive a subset of the events published by some source. Such systems sometimes leverage multicast trees [CRZ00] or distributed hash tables [RGKK05] to direct the dissemination, and must be able to scale to thousands of subscribers. Example event dissemination services include Corona [RPS06].

A third service is an *information plane* that provides topology, failure, and load information for the underlying network. Applications and higher-level services, such as slice embedding services, often use this information to adapt to current network conditions. Such systems collect

data from sensors distributed throughout the network—this data includes network probes, traffic statistics, and local node status—aggregates this data for analysis, and publishes the results to a set of consumers. Example information planes include ScriptRoute [SWA04], PlanetSeer [ZZPP04] and iPlane [MIPD06].

4.3.2 Storage Services

GENI requires several forms of storage to facilitate both management and experimentation. Experimenters will need local storage on the components they use (e.g., to store results, logs, or the data their experiments operate on); they will benefit from convenient remote access to those storage resources, to debug their experiments and examine results; and they will need high-performance storage resources for the data that their experiments operate on (e.g., cached web pages, network measurement data, and so on). At the same time, the management of GENI will require significant quantities of reliable storage for audit trails, resource use accounting, and tracking GENI resources such as equipment and network connectivity.

To support these usage scenarios, GENI will provide a suite of storage services with varying interfaces, including: (1) direct access by experimenters running code on an individual component; (2) access through databases for more convenient access to structured data storage; and (3) convenient remote access for experiment and system management. Specifically:

Researchers expect a convenient file system on each component that can be used to read and write the data accessed by their slice, service, or networked application. In addition, many experiments will benefit from a more structured, database-like access mechanism to, for example, store and query data from experiments. To this end, GENI will provide both a local file system and an SQL database.

Both researchers and operators need to remotely access data stored on GENI components. Researchers must distribute code, gather analysis results, and may need to interactively access data while developing and debugging networking software. Operators need remote access to analyze problems, failures, and monitor the performance of the system. To support these needs, GENI will provide a convenient mechanism for remote data access. This access has two aspects. The first is a remote file system-like access mechanism that provides easy, human-friendly remote access, perhaps at the cost of some performance. The second is a high-performance data interface that can be a building block for the development of other services.

Researchers designing experimental services that must store and access large data sets require a high-performance clustered file system for use within a single, geographically co-located collection of components. GENI will include such a cluster-based file service. These high-performance file systems may also be reused as sources or sinks for traffic exercising or testing new network architectures.

Researchers that want to experiment with new storage services, as well as leverage advanced functionality offered by other experimental services require special consideration. This includes access to a block-level storage interface, so that any new storage service need not presume a full-featured file system, and a robust loopback file system interface that allows researchers to build on (be composed with) each others' services. GENI will provide both.

These storage services are all well understood, so the main engineering effort involves integrating existing off-the-shelf solutions with the GENI resource accounting and authorization model.

One of the main challenges in providing these services is dealing with the threat of disk failures. In this context, it is helpful to make a clear distinction between hard-state and soft-state data. Hard-state data must be durable, perhaps through replication and archiving. Soft-state data can be reclaimed at anytime, perhaps, if possible, after warning the owner of the data. We envision that GENI will enable hard-state and soft-state services, but GENI itself will provide at a low level only "best-effort" durability.

Best-effort storage is defined by three characteristics: First, GENI will allocate long-term storage to projects when necessary, and not reclaim the storage until that time has expired. Second, the GENI operations staff must create and follow guidelines about the frequency of intentional storage erasures (e.g., due to upgrades or routine maintenance). Finally, services must use replication or other strategies to ensure that they can survive accidents or erasures within these parameters.

Hard-state services are more challenging to develop and maintain than soft-state services. We do not assume that the GENI facility will be directly responsible for archiving and replicating hard-state data. Instead, we assume that GENI allocates disk storage to a service for a long time period with the promise not to reclaim that storage. This should be suitable for most experimental slice needs. Alternately, the service may use the assigned storage to provide durable hard-state data. (If such a service becomes popular, the GENI operations staff may choose to take on the responsibility of running the service.)

4.3.3 Legacy Internet Services

A goal of GENI is to foster the use of experimental services by end users. Connecting the GENI substrate to the commodity Internet at both edge sites (Section 3.1.2) and Internet eXchange Points (Section 3.2.3) is only part of the story. The rest of the story is to provide the "software glue" that allows the commodity Internet and GENI experiments to interoperate.

Since end users today rely on a suite of applications—services and information sources provided on top of the Internet—we believe a key step is to make it easy for these legacy applications and services to take advantage of new functionality provided by new architectures on GENI. The term "legacy applications" refers to existing applications such as web browsers, databases, chat clients, remote file access applications, and so on. Similarly, the term "legacy service" refers to existing services that are invoked by legacy applications through a standard interface. Examples of such services include DNS, email, and IP packet delivery.

Our main goal is to develop "compatibility frameworks" that allow the researchers to easily extend the functionality of existing services or deploy new functionality altogether without worrying about supporting legacy applications. Such frameworks will significantly lower the development cost and ensure meaningful comparisons between different implementations of the same functionality/service. Furthermore, these frameworks will lower the barrier of acceptance of new functionality by real-world end-users, and allow the researchers to evaluate their designs in the presence of real applications and real users.

Today's examples of such frameworks are Click [KMJK00], which allows developers to easily modify and add data-path functionality (such as better buffer management, packet scheduling, and packet inspection), and XORP [HKGH05], which allows developers to experiment with new routing protocols. Specifically, we propose to build as part of GENI:

1. **Virtualized BGP**: This would allow multiple experimental architectures to coexist, while simultaneously interoperating with the rest of the legacy Internet.

2. **Virtualized Data Plane:** In the common case, experimental architectures will need fast packet processing and forwarding implemented on various GENI hardware bases.

3. **Virtualized HTTP:** We expect that many experiments will attempt to define HTTP compatible interfaces to their services, to make it easy for users to subscribe.

4. **Virtualized DNS:** A similar argument applies to experiments that redefine DNS; at least three such experiments are running on PlanetLab at the present time, and we expect much more interest in this given GENI's increased scope.

5. **Distributed Dynamic NAT:** A connection setup packet leaving GENI destined for the legacy Internet will need to be manipulated to allow return packets to be delivered back to the originating slice; combined with the virtualized BGP specified elsewhere, the return packets may be delivered to any of a number of different entry points in GENI, but must be forwarded to the slice regardless of the entry point.

4.3.4 Client Devices / Opt-In

In addition to various components deployed throughout the GENI substrate, we allow for a wide range of client devices—desktop workstations, PDAs, laptops, cell phones, sensor motes, and so on—that can participate in GENI. Such devices are integrated into GENI in one of four ways.

First, they may be full-fledged GENI components, differing from the components described elsewhere in this document only in that they are contributed by someone for the sake of participating in some experimental service on GENI (i.e., they federate with the set of components described in this document). Such systems run their own Component Manager and instantiate slivers on behalf of researchers.

Second, rather than co-locate a Component Manager on each component, a client device might leverage an implementation that decouples the CM and the device, with the CM implemented on a separate control computer. In many such cases, it may be appropriate for a single Component Manager to support and manage more than one component.

In this scenario, a single Component Manager, implemented on a general-purpose computer, implements and exports (registers) a separate Component interface for two or more devices. The result is that each device is visible to the system as a separate component, and can be configured and allocated independently by users and higher-level services. However, the hardware and operating system of the device does not need to support a CM, and does not need to directly concern itself with registering or managing the component. The form of the

connection between the CM and the individual devices it manages is private to the CM and its managed devices.

Third, a client device can be treated as an "unmanaged'' GENI component. This means the device is under control of some user—e.g., running whatever OS and applications the device would run independent of GENI—but that the device exports the CM interface, and hence, is willing to create slivers on behalf of GENI users. The device owner would be free to decide what slices she is willing and not willing to host, independent whatever policies govern the rest of the GENI facility.

Fourth, a client device may be completely independent of GENI, but configured to "opt into" an experiment running on GENI. That is to say, the device does not run a CM or support slices that can be configured to support various experiments, but rather, an application running on the device can be configured to send or receive data via an experimental service running on one or more GENI components. There are at least three ways users can opt into a GENI slice/service:

1. existing applications can be directly configured to forward traffic through a GENI slice (e.g., a web browser configured to use a proxy running in GENI);

2. the host operating system can run a generic proxy (gateway module) that forwards traffic through a GENI slice on behalf of unaware legacy applications [JKKL06, FLM06]; or

3. a native GENI application can be installed to explicitly to take advantage of some capability provided by an experimental service running in GENI.

The extent to which legacy (GENI-unware) applications can be redirected to use GENI, versus new GENI-aware applications have to be written, is a matter of researcher ingenuity.

Finally, note that while this discussion focuses on what has to happen on the client device to opt into GENI, researchers that want to build experimental services that take advantage of this workload will have to program their slices to handle this traffic. Said another way, the "ingress" sliver will effectively act as a proxy between the slice and the external world. In some case, this proxy code will be generic enough for use by many different slices. For example, there may be a generic "GENI sensor proxy" that shuffles data between an "external" sensor mote and a GENI-wide sensor experiment (Section 3.3.3). This proxy (again, running in a sliver on a GENI component) might aggregate data from multiple motes, and translate between header formats used in GENI versus a non-GENI sensor network. The goal is to enable the "external" devices to share data with GENI slices without changing the way they transport data in the sensor network itself.

4.4 Instrumentation & Data Repository

As a research instrument, being able to collect, archive, and analyze measurements of experimental services and architectures is central to GENI's mission. To this end, this section describes the GENI Instrumentation and Measurement System (GIMS). A more thorough discussion can be found in [GDD-06-12].

Requirements

The primary objective of the GIMS is to support a broad spectrum of empirical network research. The general requirements include:

- Ubiquitous deployment;
- No (or at least measurable) impact on experiments;
- Extensibility (i.e., the ability to add new instrumentation and/or new measurement synthesis capability);
- High availability (at least as available as GENI systems on which experiments are conducted);
- Large capacity (i.e., the ability to support a diverse set of simultaneous activities from a large number of experiments);
- The ability to measure detailed activity with high accuracy and precision from physical layer through application layer (includes the ability to calibrate measurements);
- The ability to specify required measurements for an experiment in a slice (using either standard measurement types from a library or defining user specific measurements) and then having these measurements initialized in the infrastructure when an experiment is activated;
- Access control (i.e., the ability to specify what data is available from a particular device or collection of devices, to whom, and for how long);
- A secure central repository in which collected data can be anonymized and made available to researchers; and
- A "data analysis warehouse" where tools for visualizing, interpreting and reporting measurement data can be developed and made openly available. These tools will be available separately, as well as integrated within the GENI experimenter's workbench (Section 4.2.3).

Approach

GENI is designed to support disruptive research across all layers of the traditional protocol stack. The implication of this objective on the measurement capability is that both known data primitives (e.g., 802.3 Ethernet packets or BGP updates or queue occupancy or the routing table on a node) and yet-to-be-conceived data primitives need to be measured. There are likely to be a number of possible approaches for developing a specification for GIMS with these capabilities.

The GIMS takes a *resource centric* approach. It begins by specifying a general framework for describing GENI resources, which includes their basic data types and how the resources/data types will be accessed. An important aspect of this framework is to be as comprehensive as possible in definition of the basic data types, so as to accommodate higher level primitives that are yet to be determined. At the same time, the methods the GIMS defines for accessing data

types should be common (to the extent possible) to all resources. That is, there should be a standard but extensible way for (authorized) gathering of data types in the infrastructure.

Our resource-centric approach begins by defining two abstract resources, *links* and *nodes* and one special case resource, *time sensors*. From these we organize the GIMS specification around three general concepts, (1) instrumentation, (2) measurement synthesis and (3) analysis and archiving, with access control policies applied to each. Items (1) and (2) are the mechanisms for realizing instances in the resource framework, and can be thought of as a hierarchy with instrumentation at the lowest level, referring to the physical taps in GENI systems (e.g., on links or within programmable components); synthesis refers to transformations of the signals provided by the taps into meaningful data (e.g., providing layer 2 framing or flow export, or high level aggregates such as routing configuration information); and analysis and archival at the highest level refers to data evaluation and storage in a common repository for future use. These three components form the intrinsic measurement capability of the infrastructure, which is complemented by additional features and capabilities that will accommodate experimental research on instrumentation and measurement (i.e., deploying and testing *new measurement systems and protocols* in GENI).

Tradeoffs

Like many other aspects of GENI, the scope, capabilities and rollout of the measurement systems will depend on a variety of factors, including available budget, user requirements, management requirements, security requirements, and availability of hardware/software for the purpose of measurement, among other factors. An important design issue that must be considered is how/where measurement capabilities will be realized in the facility. This issue can be boiled down to the question of whether to deploy special-purpose measurement hardware or to enhance the general-purpose programmable hardware being developed for GENI to support required measurements.

It can be argued that special-purpose measurement systems (e.g., Endace DAG cards) offer several advantages. They are designed and built to operate at line rate, and many such systems are available for very high-speed (e.g., 10Gbps) links today. In this sense, they almost certainly offer a price/performance advantage over the general-purpose systems. By virtue of the fact that the specialized systems are "external", they do not pose the risk of introducing a load on the general-purpose systems being used in experiments. Special-purpose systems are also deployed and in use today, which means that there is already community knowledge on how to use them and an existing analysis tool suite for data they collect. Recreating high-speed measurement capability that exists today on commercial products will be a non-trivial enterprise that may not make sense in the short or medium term. Finally, some special purpose measurement systems have some ability to be programmed, which may enable them to fit more directly into GENI.

Enhancing the programmable general-purpose GENI systems with the required measurement capability has the distinct advantage of making measurement truly intrinsic to the infrastructure. This approach is a "cleaner" solution from a design, deployment and management perspective, and a better fit with the overall GENI mission. The risk, however, is that it may be determined that it is infeasible to support certain kinds of required measurements on the general-purpose hardware (e.g., the hardware/OS cannot do what is required), or that

certain common types of measurements consistently cause degraded performance on the general-purpose systems.

We expect the deployed solution will be a hybrid between the two different approaches. The advantages of special-purpose systems would seem to dominate (cost-performance, existing solution) especially in the short and medium term. However, it is anticipated the actual realization will evolve as methods for incorporating measurement capability into general-purpose systems are developed, implemented and deployed. The remainder of this section reflects a design that includes both special-purpose measurement systems and measurement built into general-purpose GENI components.

4.4.1 Instrumentation

The resource description framework (as described more fully in [GDD-06-12]) establishes the basis for describing the instrumentation deployed in the infrastructure. Instrumentation will be realized in three types of sensors: (1) link sensors, (2) node sensors, and (3) time sensors. The purpose of the sensors is to provide the basic signals that can be synthesized into a variety of specific, well-formed data that will available as measurements to experiments running in the infrastructure.

Link Sensors

Link sensors are deployed on all (or some portion) of the physical links in the infrastructure and enable signals transmitted on those links to be extracted. At this level, no assumptions are made as to what the signals mean—interpreting signals is part of the synthesis activity. Signals on the links can be either light or electrical impulses. In the case of **fiber/light** signals, the standard method for tapping a link is to install a passive 50/50 Y-splitter that makes a duplicate of the light signal on the primary path available on a secondary path that will terminate at a collection system (described below). Standard optical Test Access Port devices and others are also commonly used to mirror light signals. These devices typically do some kind of standard layer 2 framing such as 802.3 (i.e., some basic synthesis), which will make them applicable to a fairly broad range of measurement activities, especially in the short term. However, in the most general sense of the architecture, these taps are not ideal for GIMS unless they are programmable and enabled flexibility in how framing is defined.

In the case of signals on **copper/electrical** links, there are several standard methods for instrumentation. The first is to use the basic port mirroring capability available on many types of network nodes. This approach requires the network node to copy all signals on a given port to a monitoring port. For high-speed links, resource demands in this configuration can be considerable—to the extent that they could alter the performance or behavior of the network node. A second method is the "bump in the road" approach, which refers to physically breaking the link between two nodes through a mirroring device similar to the test access port devices mentioned above. In the case of copper/electrical links, this is likely to be the best approach in the short term although, in terms of off-the-shelf solutions, it suffers from the same problems of imposing a fixed type of layer 2 framing on basic signals.

Link sensors for **wireless** environments have different characteristics than wireline link sensors but basicly, the capabilities in many respects are similar [GDD-06-15]. There is a need to

monitor the physical environment with respect to the signals being transmitted between multiple nodes simultaneously. This means that separate RF sensing devices with (perhaps) higher quality antennas will have to be deployed in order to gather a complete picture of transmission and interference patterns in a given environment.

Node Sensors

Node sensors are deployed on all nodes interconnected by links in GENI. The essence of node sensors is that they provide basic utilization, state and configuration information about components and/or subsystems in a node. There are two canonical examples of node sensors. The first is the /proc file system that is available in Linux. Proc is a real time, pseudo file system that provides configuration information and measurements of both underlying hardware components and the processes that are running on the hardware. Examples include CPU speed, cache size, CPU utilization, memory utilization, packets sent, packets received, etc. The second example is the Simple Network Management Protocol (SNMP) Management Information Base (MIB) primitives provided by switches, routers, and other networking hardware. In both cases, the available data is limited by what is provided by the underlying hardware.

Time Sensors

Time sensors are deployed on all GENI node locations. Time sensors provide a high fidelity, synchronized time source for all GENI nodes that can be used for a variety of purposes. In the GIMS context, time sensors will be used as the basis for applying high precision timestamps to measurements.

The problem of distributed time synchronization has been studied for many years. The Network Time Protocol (NTP) offers the ability to generate timestamps that are synchronized on the order of milliseconds based on using a small number of highly accurate time sources (such as Cesium clocks). The risk with NTP is that it does not provide sufficient precision for a significant set of GENI experiments. In this case, higher fidelity time sources, such as those provided by a GPS, which enable timestamp accuracy on the order of micro-seconds will be required. GPS receivers can be installed in simple PC hosts that can be rack-mounted in GENI locations. These systems can then provide Pulse per Second (PPS) signals and other data (such as latitude and longitude) to any device in the location. The risk of GPS is that it typically requires an external antenna which can be difficult to deploy in some locations. In this case, software clocks [PV02] or CDMA-based GPS systems will be sufficient.

Other Issues

There is no specific data buffering or storage requirement for any of the three types of sensors. Ideally, sensors simply provide raw signals, and buffering/short term storage is part of the synthesis component described below. However, it is likely that certain solutions will include some amount of local synthesis and buffering. In this case the synthesis component will have to query the sensor using a well-defined protocol (e.g., Simple Common Sensor Interface for PlanetLab [RPKW03]).

4.4.2 Data Synthesis

The second element of the GIMS is the set of systems that collect and synthesize the signals provided by the sensors into meaningful data. Collection and synthesis systems will be physically connected to the sensors available in all GENI locations. While they are referred to in this section as "collection and synthesis systems" and their function is logically distinct from the sensors described above, their physical instantiation may be a single system or multiple systems with both sensor and synthesis capability. In the logically separate context, synthesis systems will require specific direction from users in order to interpret the arriving signals appropriately. An example of this context, which would be provided during the creation of a GENI experiment, is that light signals from a fiber link (a GENI resource/component available for inclusion in a slice) during a specified period of time, are Ethernet packets. In this case the first aspect of measurement synthesis that is required is layer 2 Ethernet framing.

It is expected that the ability to frame and capture packets will be one of the most important network measurement features in the infrastructure (although GIMS is by no means limited to interpreting signals as packet). There are two examples of collection/synthesis systems that are commonly used today that provide this functionality. The first is a standard PC running the standard tcpdump utility. These systems rely on standard network interface cards to provide framing. The second are Endace DAG daughter cards installed in a standard PC which generate packet traces using layer 1 signals gathered directly from a network link [DAG]. In each case, the systems can be configured to gather full header/payload traces (or some subset there of), and the host PC can be used to buffer a certain volume of this data before it must be streamed to a larger repository.

The synthesis function can also be more complex. For example, an experiment may require flow-export measurements for specified packet streams. In this case, layer 2 framing along with maintaining appropriate flow records of the observed traffic are required. Another example is merging measurements (including time stamps) from several locations in a coordinated fashion. There is also the possibility of even greater levels of measurement synthesis complexity. For example using collection systems for streaming queries or statistical anomaly detection. In each case, a transformation function (realized as a program) from the basic link signals must be provided by the user or selected from a library of functions that have already been developed. In each of these cases, the requirements of the collection/synthesis nodes is that are programmable thereby allowing users to define exactly what they want to extract from the low level sensors.

From the perspective of node sensors, collection systems require additional capabilities. In particular, a data gathering protocol will be required to request transfer of the MIB variables from the node sensors. This data gathering protocol must be light weight, extensible in terms of the kinds of things that can be requested, and include mechanisms for enforcing appropriate, per experiment, access control policies. The most obvious example of an existing protocol in this space is SNMP, although it's well documented deficiencies may preclude it from use in GIMS. Another possibility is the Simple Common Sensor Interface for PlanetLab that has a data query protocol based on HTTP.

The availability of resources (e.g., CPU, disk, memory) on the collection and synthesis systems will obviously limit the extent of these activities. In some cases, this may cause certain

experiments to be conducted at times when overall demand is sufficiently low. Regardless of their capacity, collection and synthesis systems are not meant to be anything other than short-term repositories for collected data. Therefore, collection systems must have a direct connection with the archival data repository via a secure, transaction oriented protocol. This protocol will be used to move potentially large amounts of data off of the collection systems and into the repository.

4.4.3 Data Archive and Analysis

The third element of the GIMS is the archival repository into which measurements from the collection systems are stored and then made available for analysis. An example of an existing repository similar to what is envisioned for GENI is CRAWDAD, which is a portal for storing and sharing wireless measurements [CRAWDAD]. The GIMS archive will have two primary sub-systems. The first is the repository itself, which resides on high capacity storage systems that will be located in several GENI sites. The second is a data catalog that indexes the data in the repository. The objectives for the archive are, (1) to provide a secure, reliable repository to GENI users for measurements taken in their experiments, and (2) to provide a standard interface for data extraction so that users can visualize and evaluate their data at their own sites. Both of these sub-systems leverage the experimenter workbench described in Section 4.2.3.

The archival system will interface with the collection and synthesis systems through a data transfer protocol such as the Simple Common Sensor Interface for PlanetLab. This data transfer protocol must be the same as the data transfer protocol used between sensors and collection/synthesis systems for consistency and to enable the possibility of direct streaming into the archival system.

The archive itself will be distributed across several GENI sites. This enables the archive to be resilient to site-specific failures and will spread the network load for both sensor-to-archive transfers and archive-to-user transfers. A data management system will be used to store and retrieve the measurement data on these systems. An example of a large-scale network data management system that has been used for some time is AT&T's Daytona system. Daytona is a file-oriented archive system that provides a SQL-like interface [DAYTONA].

Like the measurements themselves, users creating experiments will have to specify the details of how their data will be archived. This will include the type of data being stored, and the size and lifetime of the storage space required. It may certainly be the case that an individual experiment does not need to use the archive, *e.g.*, an application log. Access privileges will also have to be specified via policy (this can include access to anonymized data see [GDD-06-12] and [GDD-06-40] for details])—since data can range from projects in which data is considered private to data that is openly available and permanently archived (e.g., in the case of studies of GENI-wide activity).

The archival system will also include a separate but related component, which is a data index for all data in the archive (such as the Internet Measurement Data Catalog from CAIDA [DATCAT]). This index enables users to find data in the archive. It will also act as a portal pointing to data sets collected in GENI that reside on remote sites. Furthermore, the data catalog will form the foundation for developing visualization and analysis tools that can be made available for general use by the community.

Glossary

In the following, *italics* indicate a reference to another alphabetically listed headword. Each reference is only italicized once per entry. Headwords themselves are **boldface** and followed by a colon. Abbreviations are not italicized and all appear as headwords that expand the abbreviation.

Access Network: Site network connecting a PEN to a set of PWCs or PECs. This is usually a LAN dedicated to GENI, though it may be multiplexed with or virtualized from a more conventional campus or enterprise network.

Accounting management Service: *User service* that validates *users* to ensure that only authorized access is permitted.

Active Sliver: A *sliver* that supports user-installed code.

Aggregate: A group of *components*, where a given component can belong to zero, one, or more aggregates. Aggregates can be hierarchical, meaning that an aggregate can contain either components or other aggregates. Aggregates provide a way for users, developers, or administrators to view a collection of GENI nodes together with some software-defined behavior as a single identifiable unit. Generally aggregates export at least a component interface – i.e., they can be addressed as a component – though aggregates may export other interfaces as well.

Authorization Logic: A logic in which formulas represent credentials that define access-control policy, and with which one can express proofs that that a request satisfies access-control policy. Such a proof would typically be accompanied by credentials; formulas of the logic are instantiated form credentials to serve as premises of the proof.

Backbone Network: Nationwide network consisting of a set of PCNs connected by an underlying *national fiber facility*.

Brokerage Service: An alternative name for a *resource broker*.

Bulk Transfer Service: a communication service optimized for transferring large files. GENI uses this service to instantiate slices.

Canonical Aggregate: A reference implementation of the *aggregate* abstraction that can be specialized for various elements of the *physical substrate.* The canonical aggregate implements a *Node Control Aggregate* and a *Slice Coordination Aggregate.*

Canonical Component: A reference implementation of the *component* abstraction that can be specialized for various elements of the *physical substrate.*

Capture protection: A technique for rendering cryptographic keys more difficult to misuse if captured by an adversary. A capture-protected key can be used only with the consent of a remote server; the remote server typically gives this consent only if it can authenticate the person attempting to use the key.

Circuit Processing System (CPS): A combination of circuit-oriented elements that provides the *virtual wire interface* to the GENI backbone.

Client Devices: Devices that run applications giving end users access to experimental services. These are commonly, but not exclusively, wireless devices.

Cluster File System: A file system that aggregates storage nodes located on a high-performance, reliable local-area network.

Component: Building block object of GENI, corresponding to a physical device in the substrate. A component consists of collection of *resources*. Such physical resources belong to precisely one component. Each component runs a *component manager* that implements a well-defined interface for the component. In addition to describing physical devices, components may be defined that represent logical devices as well.

Component Manager: The entity responsible for allocating *resources* at a *component*. It exports a standard interface defined in the GENI architecture document. That interface is also exported by *aggregates*. Though it is intuitive to think of the component manager residing on the component it manages, this is not necessarily the case.

Component Registry: A registry that maps between a component name, e.g. geni.us.princeton.component and a *GENI Global Identifier. Component*s and *aggregates* share a name space, so a component registry maps aggregate names to *GGID*s as well.

Configuration Management Service: *User service* that coordinates other user services when new resources become available in GENI.

Cognitive Radio Subnet: A wireless subnet that uses cognitive radio nodes to carry traffic. Cognitive radios are experimental wide-spectrum software-based radios.

CPS: *Circuit Processing System*

Credential: A digitally signed statement that encodes access-control policy.

Data Push Service: A storage service that pushes data (e.g., computer programs) efficiently to many receiving nodes by, for example, serving the data cooperatively.

Emulation Grid: A wireless network designed to test wireless protocol implementations under controlled conditions. These either run the wireless protocol implementations using simulated radio links or in a strictly controlled radio environment.

Event Dissemination Service: a communication service optimized for the reliable and timely delivery of small messages, e.g., events and control operations.

Experiment: A *slice* primarily used to test a hypothesis about some aspect of a GENI user's research. Experiments tend to be shorter-lived than services and less accessible from the world outside GENI, but there are no hard requirements on those attributes. In addition to slice embedding, experiments may take advantage of the *experiment control plane* that performs *experiment instantiation*.

Experiment Control Plane: A toolkit and GENI service that manages the deployment and execution of user programs used in *experiments*. It supports system-wide job control, synchronization of control operations, control message dissemination, and *fault detection and recovery*.

Experiment Instantiation: The task of customizing *slice* nodes to prepare them to execute a particular experiment. This typically includes copying over user executables, setting up necessary software packages, and starting system services.

Experimental Services: *Services* exported by *experiments* especially those that are exported to users outside GENI through an *IXP*. These are logically similar to *user services*, but are usually shorter-lived and targeted to non-GENI users.

Experimenter's Workbench: A toolkit that simplifies embedding an experiment into GENI and collecting results. It can be accessed through the *researcher portal.*

Execution Environment: The environment a *component* provides for user interaction. This can range from an operating system and virtualized services on a workstation to a network management interface to routing tables or measurement equipment.

Fault Management Service: a GENI service for monitoring the execution of user experiments, identifying abnormal conditions, invoking appropriate user-specified handlers to handle faults, and potentially allocating new resources to enable continued execution of the experiment.

FCAPS Management Framework: An International telecommunications Union framework for organizing management functions into fault management, Configuration management, accounting management, performance management, and security management.

GENI Gateway (GGW): A device that connects a GENI *access network* to the Internet. Includes one or more of the following functions: routing (to GENI backbone and/or the legacy Internet) traffic shaping, packet filtering, and packet auditing. A GGW is not a component that can be sliced, and it is not required if these functions can be provided by the connected component itself.

GENI Global Identifier (GGID): Unambiguous, certified unique identifiers assigned to GENI objects. Specifically these are assigned to *users, components,* and *slices.* A GGID consists of an X.509 certificate that binds a UUID to a public key.

GENI Management Core (GMC): The framework for describing and implementing the fundamental GENI constructs that make up the GENI substrate. This includes the actors and objects in the GENI system as well as the interfaces between them and name spaces for describing them. It is intended as a small set of fixed abstractions around which GENI as a whole can evolve and expand, situated between the *physical substrate* and the *user services.*

GENI Management Core Implementation (GMCI): An implementation of the messaging and services that make up the *GMC.* An important aspect of the GMCI is its ability to allow different implementations and versions of non-core user services to be easily added to the system as appropriate.

GENI Site: A geographic location connected to the GENI backbone that supports one or more *components.*

GGID: *GENI Global Identifier*

GGW: *GENI Gateway*

GMC: *GENI Management Core*

GMCI: *GENI Management Core Implementation*

Hard-state Storage Service: A service that requires that the storage system stores data durably through node reboots, failures, and system upgrades.

Information Plane: A service that provides topology, failure, and load information about the underlying network. Used by other services that want to adapt to current network conditions.

Internet Exchange Point (IXP): Device that connects GENI to the commodity Internet. This allows GENI services to be accessed by Internet users via *user opt-in.* The somewhat counterintuitive abbreviation (IXP rather than IEP) is commonly used in Internet circles.

IXP: *Internet Exchange Point*

MA: *Management Authority*

Management Authority (MA): The real-world entity responsible for managing a set of GENI *components*. Management Authorities operate as an o*perations and management controller* for a set of components that share an owner or administrator. This allows administrators to set subnet-wide or installation-wide policies or configurations. Additionally the MA may be directly involved in slicing components under its control, including participating in the resource discovery, allocation, and configuration of components it controls. The operations and management tasks are carried out by a *Node Control Aggregate* and allocation tasks are carried out by a S*lice Coordination Aggregate*. These aggregates are the implementations of the polices created by the authority. The aggregates are GMC aggregates, the authority is a human being or set of humans.

National Fiber Facility: The shared fiber backbone that interconnects *PCNs* in GENI. Though not necessarily constructed of optical fiber, this network is high capacity and broadly connected.

Node Control Aggregate: An *aggregate* that implements the *operations and management control* policies decided upon by the Management Authority responsible for the *components* that comprise the aggregate. A node control aggregate it the GMC aggregate that configures individual components in ways not bound to an individual slice, e.g., the aggregate configures the operating system or power controllers of the components.

O&M Control: *Operations and Management Control.*

Operator Portal: *User service* that provides an integrated interface to the *FCAPS*-based *user services* responsible for providing operator information.

Operations and Management Control (O&M Control): The mechanism an owner uses to configure, operate, and manage components under the owner's control. This function is usually encapsulated as a GENI *Node Control Aggregate*.

Packet Processing System (PPS): A flexible, high speed programmable device capable of operating on packets. The PPS supports one or more *virtual routers,* and exports the *virtual link interface.*

PCN: *Programmable Core Node.*

PEC: *Programmable Edge Cluster.*

PEN: *Programmable Edge Node.*

Performance Management System: *User service* that tracks resource usage to provide data for operations and user performance evaluation.

Physical Substrate: The physical facilities into which *slices* are embedded, including routers, processors, links, and other support hardware.

Portal: An interface constructed from composition of *services*. It appears to users as a full *service*, but the implementation is a composition of existing services.

PPS: *Packet processing System*

Problem Tracking System: A system that collects GENI problems and trouble reports from users and administrators. The GENI operations staff will use the Problem Tracking System to prioritize requests for repairing problems or adding new features to GENI, and to track the progress of these changes.

Programmable Wireless Node (PWN): GENI *component* available throughout wireless *access networks*.

Programmable Core Node (PCN): GENI *component* available throughout the *backbone network*. Typically configured to support high-speed links (e.g., using FPGAs) and including programmable elements at the framing layer.

Programmable Edge Cluster (PEC): GENI *component* available throughout wired *access networks.*

Programmable Edge Node (PEN): GENI *component* available at edge sites, connecting the local *access network* to the GENI *backbone network*.

PWN: *Programmable Wireless Node.*

Reconfigurable optical add/drop multiplexer (ROADM): A *circuit processing system* component that can efficiently and dynamically switch wavelength division multiplexed circuits.

Regional Optical Network (RON): A medium or large scale optical network, often administered for or by a set of educational institutions.

Researcher Portal: *Portal* that consolidates access to the *user services* intended for use by researchers. These services include *slice embedding* and an *experimenter's workbench.*

Resource: Abstractions of the sliceable features of a *component* that are allocated by a *component manager* and described by an *rspec*. Resources are divided into computation, communication, measurement, and storage.

Resource Broker: A *user service* that assists in *slice embedding* by collecting and administering *component* and *aggregate resources.* Also called a *brokerage service.*

Resource Monitor: Conceptual entity that performs access control to component resources. It compares certified requests from *users* against security policies originated by a *management authority* and grants only those requests that conform to the policy. The resource monitor may be implemented directly in a *component manager* or may be a distinct element.

Resource Specification (RSpec): Represents all GENI *resources* that can be bound to a *sliver* within GENI. An rspec describes both the resources available, advertised or allocated at a component and the relationships between those resources, and perhaps other resources in GENI.

ROADM: *Reconfigurable Optical Add/Drop Multiplexer*

RON: *Regional Optical Network*

Rspec: *Resource Specification*

SA: *Slice Authority.*

Secure Control Plane: a communication service that offers authentication and high availability, to configure and control GENI itself. It uses only GENI resources, not the commodity internet to configure GENI.

Security Policy Manager: Tool used by a representative of the *management authority*to create, manage, and distribute security policies. These policies are used by *resource monitors* to control *component* access.

Segment Storage Service: a storage service that allows clients to read or write files to fixed-size or bounded-length chunks ("segments") that are named in a flat manner. Such a service may be used directly by clients, but will more commonly be used as the building block for more complex services such as a *cluster file system*.

Sensor: A device that produces a stream of data about its local environment. Sensors can be physical (e.g., temperature, audio sampling) or virtual (e.g., reporting CPU and memory load). GENI includes both, using virtual sensors to monitor its own internal behavior.

Service: A *slice* that provides some functionality to other running slices. The GMC name service and resource discovery service are examples of GMC services. Services tend to be longer lived than slices used for experiments.

Slice: A distributed, named collection of *slivers* which collectively provide the execution context for an experiment, service, or network architecture. A slice typically includes no more than one sliver per *component*, but this is not a requirement.

Slice Authority (SA): Entity responsible for the behavior of some set of *slices*. Participates in the slice naming hierarchy.

Slice Coordination Aggregate: An *aggregate* that implements the policies decided upon by the Management Authority responsible for the *components* that comprise the aggregate that pertain to how slices are embedded in the aggregates components. For example, this aggregate may advertise resources or schedule their use. The aggregate may also start or stop slices in a coordinated fashion across a group of components it controls.

Slice Embedding: The *user service* that identifies and allocates a set of resources a user needs to run a service or experiment. Experiments often make use of the *experiment control plane* to perform *experiment instantiation* after embedding is complete.

Slice Registry: A registry that maps between a slice name, e.g., geni.princeton.codeen.slice, and a *GENI Global Identifier*.

Sliver: A set of *resources* multiplexed between all slivers sharing a *component*. For example, over a PC server node, a sliver would be a virtual machine with associated resource container. An *active sliver* is a resource container that supports user-installed code. Such code is said to run in an *execution environment* provided by the sliver. Active slivers might be in "running" or "suspended" states; in the latter the sliver exists and with resources bound to it, but is prevented from executing any code.

Socket Interface: Sliver interface to the underlying network services that approximates the POSIX socket interface used by many network environments. Slivers can access the network through IP, UDP, or TCP stacks and can make use of common Internet protocols such as ARP and DNS.

Soft-state Storage Service: A service that needs to store data but doesn't care if the data is durably stored, either because the data is not important or because it can be reconstructed easily.

Specification-based Intrusion Detection: In security, a method for detecting intrusions by comparing observed behavior to a specification of proper behavior. This is distinct from signature-based intrusion detection, where observed behavior is compared to specifications of improper behavior, and anomaly detection, where observed behavior is compared to a typically learned model of normal behavior.

Storage Node: Any node in GENI that can store data, but more commonly used to refer to a node that has substantial storage resources. The node may a computer with one or more hard disks attached or a specialized node such as a network-attached storage device.

Storage Service: A *user service* that stores data for clients. The service may provide one of several interfaces for clients to access storage: a file system, a database, a segment store, or any other interface that makes storage available to clients.

Tail Circuit: Connects PENs and PECs to PCNs in the *backbone network*.

Ticket: Signed data structure issued by a *component* or *aggregate* that guarantees the availability of a given set of resources. The ticket may have time or other constraints. Tickets are redeemed to create *slivers*.

Ticket Agent: Application to manage and administer *tickets* possessed by a *user.* The ticket agent facilitates selection and combination of tickets to create a *slice.*

User: A researcher or operator who interacts with GENI.

User Opt-in: software running on user nodes, such as PC's, PDA's, and cell phones, to allow normal users to select which part of their network traffic is to be directed to/from GENI slices.

User Services: user visible support *services* that make the facility accessible to researchers. Examples include *storage service* and *slice embedding*.

Virtual Link Interface: Sliver interface to the underlying networking services that allows software in the sliver to access the virtual links between components. Links may be reconfigured, though the topology remains fixed. The link may be point-to-point or a multipoint link, simulating a local area network.

Virtual Radio Interface: A sliver interface to the underlying networking services that allows the sliver to control the use of radio spectrum and other virtual radio frequency parameters.

Virtual Router: A virtual packet processor running in a sliver on a *packet processing system.*

Virtual Wire Interface: A sliver interface to the underlying networking services that allows the sliver to reconfigure circuit-based services, such as provisioning and framing. This allows configuration of multiplexed electrical links or optical switches.

Virtualized BGP: A service for coordinating read/write access to BGP announcements to/from GENI from/to the rest of the Internet.

Virtualized DNS: A service for coordinating DNS responses to allow virtualized services running on GENI to be accessible, by name, to the outside Internet.

Wide-area File System: A file system that aggregates storage nodes located across a wide-area network.

References

[BBCC04] Operating System Support for Planetary-Scale Network Services. A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. (*NSDI '04),* May 2004.

[BDFH03] Xen and the Art of Virtualization. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. In Proc. 19th ACM Symposium on Operating Systems Principles (SOSP), pages 164--177, Bolton Landing, NY, Oct. 2003.

[CDKN03] SplitStream: High-bandwidth multicast in a cooperative environment. M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, (*SOSP '03*).

[CRAWDAD] CRAWDAD, A Community Resource for Archiving Wireless Data at Dartmouth, http://carwdad.cs.dartmouth.edu, 2006.

[CRZ00] A Case for End System Multicast. Y.-H. Chu, S. G. Rao, and H. Zhang. In ACM SIGMETRICS 2000.

[DATCAT] CAIDA, The Internet Measurement Data Catalog, http://imdc.datcat.org, 2006.

[DAG] Endace, Network Monitoring Cards, http://www.endace.com/, 2006.

[DAYTONA] AT&T Labs Research, The Daytona Data Management System, http://www.research.att.com/daytona, 2006.

[DETER] Deterlab: Network Security Testbed based on Emulab, http://www.deterlab.net/

[ESL07] Eric Eide, Leigh Stoller, and Jay Lepreau. *An Experimentation Workbench for Replayable Networking Research*. In Proceedings of the Fourth USENIX Symposium on Networked System Design and Implementation, April 2007.

[FCCS03] SHARP: An Architecture for Secure Resource Peering. Yun Fu, Jeffery Chase, Brent Chun, Stephen Schwab, Amin Vahdat. (*SOSP '03*)

[FLM06] OASIS: Anycast for Any Service. Michael J. Freedman, Karthik Lakshminarayanan, David Mazières. (*NSDI '06*).

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

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

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

[GDD-06-13] Andy Bavier, Jack Brassil, Marc E. Fiuczynski (Eds), "GENI Component: Reference Design," *GENI Design Document 06-13*, Facility Architecture Working Group, September 2006.

[GDD-06-14] Joe Evans, Dipankar Raychaudhuri, Sanjoy Paul, "Overview of Wireless, Mobile and Sensor Networks in GENI," *GENI Design Document 06-14*, Wireless Working Group, September 2006.

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

[GDD-06-17] Sanjoy Paul, Srini Seshan, "Virtualization and Slicing of Wireless Networks," *GENI Design Document 06-17*, Wireless Working Group, September 2006.

[GDD-06-18] Mario Gerla, "Urban Vehicular Networks in GENI," *GENI Design Document 06-18*, Wireless Working Group, September 2006.

[GDD-06-19] Ramesh Govindan, John Heidemann, Matt Welsh, Deborah Estrin, "Sensor Networks in GENI," *GENI Design Document 06-19*, Wireless Working Group, September 2006.

[GDD-06-20] Joe Evans, Gary Minden, Ed Knightly, "Cognitive Radio Networks in GENI," *GENI Design Document 06-20*, Wireless Working Group, September 2006.

[GDD-06-21] Chip Elliott, "System Engineering Document for Wireless Subnets," *GENI Design Document 06-21*, Wireless Working Group, September 2006.

[GDD-06-23] Thomas Anderson and Michael Reiter, "GENI Facility Security," *GENI Design Document 06-23*, Distributed Services Working Group, September 2006.

[GDD-06-24] Thomas Anderson, Amin Vahdat (Eds), "GENI Distributed Services," *GENI Design Document 06-24*, Distributed Services Working Group, September 2006.

[GDD-06-25] Sampath Rangarajan, Jennifer Rexford (Eds), "Backbone Software Architecture," *GENI Design Document 06-25*, Backbone Working Group, September 2006.

[GDD-06-26] Dan Blumenthal and Nick McKeown, "Backbone Node: Requirements and Architecture," *GENI Design Document 06-26*, Backbone Working Group, November 2006.

[GDD-06-27] Jennifer Rexford, "GENI Topology Design," *GENI Design Document 06-27*, Backbone Working Group, September 2006.

[GDD-06-28] David D. Clark, Scott Shenker (Chairs), Aaron Falk (Ed), "GENI Research Plan," *GENI Design Document 06-28*, Research Coordination Working Group, September 2006.

[GDD-06-31] Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, Jennifer Rexford, "In VINI Veritas: Realistic and Controlled Network Experimentation," *GENI Design Document 06-31*, September 2006.

[GDD-06-36] Jennifer Rexford (Ed), "GENI Backbone Run-Time Software for Experimenters," *GENI Design Document 06-36*, Backbone Working Group, November 2006.

[GDD-06-37] David A. Maltz, T. S. Eugene Ng, Hemant Gogineni, Hong Yan, Hui Zhang, "Meta-Management System for GENI," *GENI Design Document 06-37*, Backbone Working Group, December 2006.

[GDD-06-39] Larry Peterson, Aaron Klingaman, Steve Muir, "Management Authority: Reference Implementation," *GENI Design Document 06-39*, Facility Architecture Working Group, December 2006.

[GDD-06-40] KC Claffy, Mark Crovella, Timur Friedman, Colleen Shannon, Neil Spring, "Community-Oriented Network Measurement Infrastructure (CONMI) Workship Report," *GENI Design Document 06-40*, December 2005.

[GDD-06-42] John Wroclawski, "Using the Component and Aggregate Abstractions in the GENI Architecture," *GENI Design Document 06-42*, Facility Architecture Working Group, December 2006.

[GDD-07-45] John Wroclawski (Ed.), "GENI Construction Plan," *GENI Design Document 07-42*, Facility Architecture Working Group, February 2007.

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

[HKGH05] Designing Extensible Router Software. Mark Handley, Eddie Kohler, Atanu Ghosh, Orion Hodson, Pavlin Radoslavov. In the proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI) 2005.

[INTERNET2] Internet2. http://www.internet2.edu.

[JKKL06] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayananan, I. Stoica, and K. Wehrle, "OCALA: An Architecture for Supporting Legacy Applications over Overlays," *3rd Symposium on Network System Design and Implementation*, pp. 267-280, May 2006.

[KMJK00] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, vol. 18, pp. 263-297, August 2000.

[MIPD06] iPlane: An Information Plane for Distributed Services. Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy and Arun Venkataramani. *OSDI 2006*, November 2006.

[NLR] National LambdaRail. http://www.nlr.net.

[OCCP06] Service Placement in Shared Wide-Area Platforms. David Oppenheimer, Brent Chun, David Patterson, Alex C. Snoeren, and Amin Vahdat. (*To appear in USENIX '06*).

[PBFM06] Experiences Implementing PlanetLab. Larry Peterson, Andy Bavier, Marc Fiuczynski, and Steve Muir. *(OSDI '06),* November 2006.

[PCMG] PCI Industrial Computer Manufacturers Group. "AdvancedTCA Specifications for Next Generation Telecommunications Equipment ," available at http://www.picmg.org/newinitiative.stm.

[PP06] Scale and Performance in the CoBlitz Large-File Distribution Service KyoungSoo Park and Vivek S. Pai. *(NSDI '06).*

[PV02] PC Based Precision Timing without GPS. A. Pasztor and D. Veitch. In Proceedings of ACM SIGMETRICS '02, Marina del Rey, CA, June, 2002.

[QUILT] The Quilt. http://www.thequilt.net.

[KRAV03] Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. Dejan Kostic, Adolfo Rodriguez, Jeannie Albrecht, Amin Vahdat. (SOSP '03).

[RADISYS] Radisys Corporation. "Promentum™ ATCA-7010 Data Sheet," product brief, available at http://www.radisys.com/files/ATCA-7010_07-1283-01_0505_datasheet.pdf.

[RAL03] Robert Ricci, Chris Alfeld, and Jay Lepreau. A Solver For The Network Testbed Mapping Problem. *SIGCOMM Computer Communications Review*, April 2003

[RFC-1034] Domain Names – Concepts and Facilities. Paul Mockapetris, RFC 1034, November 1987.

[RFC-1035] Domain Names – Implementation and Specification. Paul Mockapetris, RFC 1035, November 1987.

[RFC-4271] A Border Gateway Protocol 4 (BGP-4). Yakov Rekhter (ed.), Tony Li (ed.), and Susan Hares (ed.), RFC 4271, January 2006.

[RGKK05] OpenDHT: A Public DHT Service and Its Uses. Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. *(SIGCOMM '05*).

[RPKW03] T. Roscoe, L. Peterson, S. Karlin, and M. Wawrzoniak. A Simple Common Sensor Interface for PlanetLab. http://www.planet-lab.org/PDN, March, 2003.

[RPS06] Corona: A High Performance Publish-Subscribe System for the World Wide Web. Venugopalan Ramasubramanian, Ryan Peterson and Emin Gun Sirer. (*NSDI '06*).

[SOAP] Nilo Mitra, SOAP Version 1.2 Part 0: Primer, June 2003, http://www.w3.org/TR/soap12-part0/

[SWA04] Scriptroute: A Public Internet Measurement Facility. Neil Spring, David Wetherall, and Tom Anderson. (*OSDI '04*).

[VSERVER] Linux Vserver Project. http://linux-vserver.org

[WPPP04] Reliability and Security in the CoDeeN Content Distribution Network. Limin Wang, KyoungSoo Park, Ruoming Pang, Vivek S. Pai, and Larry Peterson. (*USENIX '04).*

[WSDL] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, Web Services Description Language (WSDL) 1.1, March 2001, http://www.w3.org/TR/wsdl

[X509] ITU-T Recommendation X.509. Information Technology - Open Systems Interconnection - The Directory: Authentication Framework.

[X667] ITU-T Recommendation X.667. Information Technology - Open Systems Interconnection - The Directory: Authentication Framework.

[XML-RPC] Dave Winer, ,ÄúXML-RPC Specification, June 2003, http://www.xmlrpc.com/spec

[XSD1] Henry S. Thompson, C. M. Sperberg-McQueen, Shudi (Sandy) Gao, Noah Mendelsohn, David Beech, Murray Maloney, ,ÄúXML Schema 1.1 Part 1: Structures, http://www.w3.org/TR/xmlschema11-1/

[XSD2] David Peterson, Paul V. Biron, Ashok Malhotra, C. M. Sperberg-McQueen, XML Schema 1.1 Part 2: Datatypes,,Äù http://www.w3.org/TR/xmlschema11-2/

[ZZPP04] PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services. Ming Zhang, Chi Zhang, Vivek Pai, Larry Peterson, and Randolph Wang. (*OSDI '04*).