GENI Backbone Software Architecture

GDD-06-25

GENI: Global Environment for Network Innovations

September 19, 2006

Status: Draft (Version 0.3)

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

This document is prepared by the Backbone Working Group.

Editors:

Jennifer Rexford, Princeton University

Sampath Rangarajan, Lucent Technologies Bell Laboratories

Contributing workgroup members:

T.V. Lakshman, Lucent Technologies Bell Laboratories

Sampath Rangarajan, Lucent Technologies Bell Laboratories

Jennifer Rexford, Princeton University

Hui Zhang, Carnegie Mellon University

Table of Contents

1 Introduction

Early in the preliminary design of the GENI backbone, our working group informally split into two sub-groups focusing on hardware and software issues, respectively. While the hardware group designs the network elements in each Point-of-Presence (PoP) in the GENI backbone, the software group is identifying the software that should run on these components to enable researchers to conduct experiments over the nodes and links. This document describes our current thinking about the software components.

1.1 Physical Infrastructure

To set the stage for the discussion, we start with a high-level description of the physical infrastructure. The GENI backbone will consist of around twenty-five PoPs distributed across the United States. A fiber facility with high-speed (e.g., 10 Gbps) links will interconnect these PoPs. In addition, GENI PoPs will connect to the legacy Internet to reach existing Internet services and end users. Tail circuits will connect GENI PoPs to edge sites that host wireless/sensor subnets or PC clusters, or provide end users with direct access to GENI. At each edge site, an edge device will multiplex and demultiplex traffic over the tail circuit, and perhaps provide direct connectivity from the edge site to the legacy Internet and local users.

Within the PoPs, we envision an incremental deployment of equipment with the goal of providing ever more bandwidth and flexibility to researchers, as well as improved isolation between experiments. The PoP deployment will start with a programmable hardware router that consists of processors (including both general-purpose compute blades and network processors or FPGAs), line cards, and a switching fabric. Next, the PoPs can include a cross-connect (such as a SONET cross-connect) that provides virtual circuits with dedicated bandwidth between PoPs, and the ability for traffic to "cut through" a PoP without imparting load on the programmable router. Then, the PoP can incorporate a ROADM (Reconfigurable Add-Drop Multiplexer) to provide researchers with more flexible and dynamic control over bandwidth allocation at the optical level.

1.2 Software Architecture

In addition to the hardware in the PoPs, the GENI backbone design must consider the software that runs on these network elements. The software architecture must address several key requirements:

- **Virtualization layer:** The hardware in the GENI backbone is meant to be shared across multiple experiments through the use of virtualization. For example, a general-purpose compute blade in the programmable router would need to run an operating system that supports multiple "slivers" (the portion of a slice running on one component), each with a portion of the processing resources. As another example, a slice may need its own IP address blocks that it can advertise via BGP sessions to the external Internet. In both cases, the virtualization layer must provide an experiment the illusion it is running on its

own dedicated infrastructure.

- **Component manager**: Each network element needs to run a component manager that can communicate with the GENI Management Core (GMC), either via the legacy Internet or over the GENI facility itself. The component managers are necessary to allow the GMC to instantiate the individual slivers in a slice, and for the network elements to provide measurement data to the GMC. For example, the programmable router would need to be able to communicate with GMC to instantiate a sliver that needs (say) 10% of the processing resources on a general-purpose compute blade and 10 Mbps of bandwidth on a line card connecting to a neighboring PoP.

- **Communicating with the legacy Internet:** The PoPs need to be able to send and receive IP packets with edge sites and the legacy Internet. The PoPs must also support other functionality such as address translation, firewalls, tunneling, and routing. For example, tunneling is necessary to allow an end host to connect to a GENI PoP via the legacy Internet, and routing is necessary to allow the GENI PoPs to learn how to reach external destinations and to advertise reachability to the addresses of GENI nodes and services (including the addresses corresponding to experimental services that researchers have deployed on GENI).

- **Libraries to support experiments:** Although researchers could conceivably write their own software from scratch, we envision that GENI should provide a number of software libraries to lower the barrier to constructing experiments. For example, researchers experimenting with a new control plane (e.g., a new routing or signaling protocol) may want to use a conventional IPv4 data plane for forwarding data packets. Rather than requiring researchers to write their own data-plane software (e.g., to run on a network processor), GENI could include a library that implements this function.

The remainder of this document describes the software architecture in more detail. First, we present an overview of three broad classes of network experiments that we use to identify requirements. Next, we enumerate a set of principles that drive our design, and discuss the software components that we are considering as a consequence of these principles. Then, we discuss some specific questions and issues we are exploring, with the goal of highlight areas that may require coordination across groups or broader discussion.

2 GENI Backbone Architecture

2.1 Physical Infrastructure

In this section we will describe the physical infrastructure of the backbone network and the software architecture in more detail. As described in the last section, the GENI physical backbone infrastructure would consist of GENI PoPs connected through a GENI optical backbone through a SONET cross-connect as well as a ROADM. We will refer to the optical backbone as the Core Backbone. Within each PoP, there will be one or more Programmable

Routers. We will refer to these routers as the Programmable Core Routers (PCR).

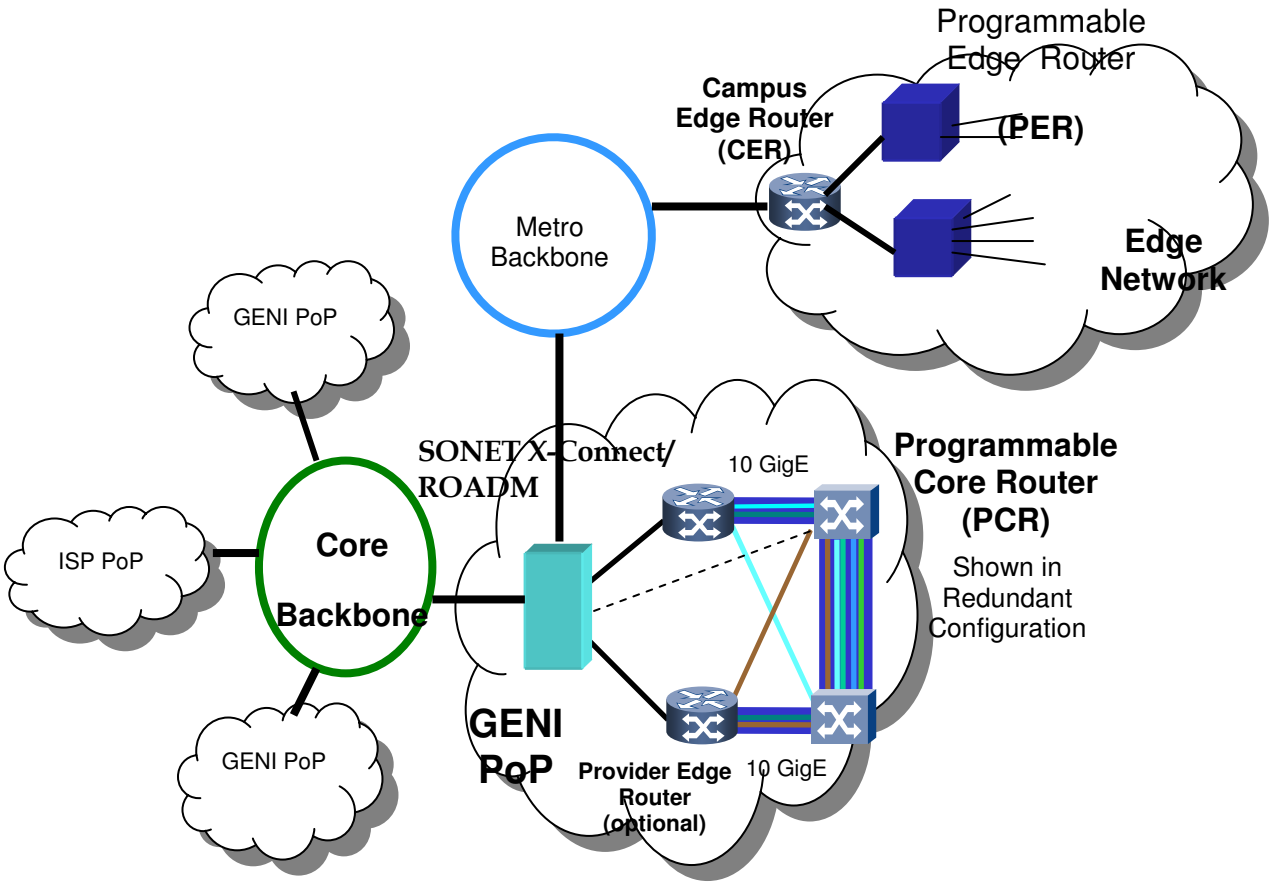

Figure 2.1: GENI PoP and Edge Network Architecture

In addition to being connected to the Core Backbone, each GENI PoP would be connected to **a)** the Internet by peering with a Internet Service Provider PoP at a NAP or at a private peering point**, b)** to one or more tail circuits which connect other wireless and wireline networks that may reside at edge sites (eg. university campuses) that are part of the GENI infrastructure. Standard agreements have to be put in place to connect the GENI POP to the Internet through a service provider PoP. In the rest of the section, we will focus on the second type of interconnection, which is the interconnection to the edge networks through tail circuits. These tail circuits could be provided through the use metro optical rings and we will refer to these as the Metro Backbone. The architecture from the perspective of a GENI PoP and an Edge Network is shown in Figure 2.1. The figure shows an edge network connecting to a GENI PoP

through a Metro Backbone and multiple GENI PoPs being connected through a Core Backbone.

2.1.1 Tail Circuit Termination - Edge Networks

We envision that there could be two types of edge networks: **a)** existing experimental networks such as PlanetLab, Emulab and ORBIT that are federated into the GENI infrastructure by making them GENI compliant, and **b)** purpose built edge networks that are specifically built as part of the GENI effort. For each of these, the characteristics of the tail circuits and how they provide connectivity to a GENI PoP could be different, as described below. There are two end-points to a tail circuit that traverses a metro backbone, one on the GENI PoP and the other on the edge site.

At the edge site, we envision a Campus Edge Router (CER) that provides connectivity to the metro backbone by terminating the access links such as T1, T3 etc. There could be two types of CERs. If a tail circuit connects an edge network that is federated into the GENI infrastructure, it appears that it is almost always the case that the CER would be the campus edge router which is now configured to provide a tail circuit into the GENI PoP (in addition to whatever PoP it might be connected to, to provide Internet connectivity to the campus). In this case, we do not anticipate that an experimenter would be able to get a sliver of this router and this router would only provide basic connectivity to the GENI PoP. If a tail circuit connects a purpose built GENI edge network to the GENI PoP, then the CER would be deployed as part of the GENI infrastructure. Even in this case we envision that the CER would be a commercial router on which an experimenter would be able to provision a network sliver (to the extent of receiving a bandwidth guaranteed MPLS tunnel to the GENI PoP) but nothing more in terms of provisioning a sliver with CPU and memory guarantees (Note: This requires further discussion; if the CER will be a custom built GENI device, then the CER and the PER, described below, can be combined into one device). Behind the CER would be an Edge Programmable Router (PER) that would be built as part of the GENI infrastructure and this is the router that is expected to provide GENI type programmability. This router is expected to be built out of off-the-shelf components (for eg. a PC running Linux) and would provide the capability for an experimenter to obtain slivers that could become part of an experiment slice.

It is envisioned that there will be a PER per edge network (be it federated or purpose built) and there are two ways in which a PER can connect to the GENI PoP. One way would be for the PER to connect to a CER which is connected to the GENI PoP through the metro backbone as described above. For this type of connectivity, we anticipate that the PER has to be close enough to the CER to connect directly using ethernet. The other way would be for the PER to connect to the CER through another PER that has direct connectivity to the CER. We envision that such type of connectivity would require a tunneling mechanism between the two PERs over the public Internet. For example, there will be a CER and one or more PERs at Princeton and an PER at NJIT would tunnel packets to the CER (to be sent to the GENI PoP) through a PER at Princeton. An example experiment that uses this type of connectivity would require the experimenter to get a sliver at the PERs at NJIT and Princeton as well as configure the CER to obtain bandwidth guarantees.

2.1.2 Tail Circuit Termination - GENI PoP

At the GENI POP, the tail circuit traversing the metro backbone would be terminated (after going over the SONET cross-connect or ROADM) at the PCR. We envision that before the PCR, there may be another commercial core router on which the tail circuit is terminated before traffic is forwarded to the PCR. A use case of this would be one where a federated edge network that connects to the GENI PoP through a campus edge router. Although, no Internet traffic is carried on this circuit out of the campus, it is possible that the campus network administrators feel comfortable terminating the circuit on a commercial router, as shown in Figure 2.1, rather than on an experimental router. Also, the presence of a commercial router would enable some of the standard functions (such as different types of tunnel termination capabilities) to be enabled at this router rather than requiring these functions to be implemented at the PCR. The figure also shows a dotted line from the SONET X-connect to the PCR indicating that such direct connectivity is possible. It is anticipated that the connection between the SONET X-connect and the PCR would be over multiple 10 GigE links.

The PCR would consist of one or more chassis with each chassis hosting multiple line cards, one or more control cards and a switch fabric. The control cards would host general purpose processors whereas the line cards may be built using network processors and/or general purpose processors. In addition to the router chasses, it is anticipated that there will be a blade-server chassis that consist of general purpose processors that is an adjunct to the PCR and will provide slow-path packet processing. The blade-server will be programmable and an experimenter would be able to obtain a sliver of the compute and communication capabilities. To make virtualization easier, we recommend that a CER consist of one chassis exclusively hosting control cards and multiple other chasses exclusively hosting line cards and a switch fabric card to connect the line cards within the chassis. A metarouter will be made up of a sliver on one control card from the "control chassis" and slivers on one or more line cards from a "line-card chassis". The communication between the control card and the line-cards would be over ethernet. A metarouter may consist of line cards from different line-card chassis although it is recommended that as a first step, a metarouter be limited to slivers on line-cards within the same chassis. As a first-step, it is also recommended that a sliver consist of whole line cards; that is, no sharing of line cards between different slivers.

More than one PCR at a PoP is envisioned mainly for redundancy reasons, though the level of redundancy that can be provided is unclear. For example, if one of the PCRs is not accessible, a checkpointed state of all slivers at that PCR may have to be moved to the backup PCR. We could require that if PCR fails, it is restarted and the experimenters need to restart their experiment again. If one of the line cards on the PCR fail, redundant line cards should be able to take over without the need for the experimenters to restart their experiment.

2.2 Edge Site Configuration

To reiterate, the tail circuit at the edge site can be terminated at either the CER or the PER (Programmable edge router). In addition, the edge site would consist of a few other components which could be connected in various configurations. In this section, we look at these components and there possible interconnections. The various components (including the CER

and PER are): Campus Edge Router (CER): The campus edge router connects the GENI Edge-Site (GES) to the GENI PoP using a tail circuit. It can also connect the GENI Edge-Site to the general Internet as it will anyway be providing connectivity to the Internet for other campus traffic. This element is not programmable in the typical GENI sense.

One of the issues to consider is the following. Any traffic destined from one GES to another would be routed by the CER through one or more GENI PoPs. What about traffic from a GES to the Internet? What if some experimenters require their traffic destined to the general Internet to first go through a GENI PoP (for example, the experimenter is using a Programmable Core Router (PCR)) and would like traffic to go through the PCR) and some others would like traffic to be directly routed to the Internet?

Programmable Edge Router (PER): These routers at the edge sites connect to the CER and are programmable. Through the CER, they connect to the GENI PoP via a tail circuit, or possibly an IP tunnel. They are also connected to one or more access networks within the edge site.

Flexible Edge Device (FED): These devices reside within the access networks and are programmable. They typically host distributed services that live at the edge of the network. (The architecture of the FEDs and the services they run are within the scope of the Distributed Services Working Group).

Programmable Access Point (PAP): In addition, depending on the type of access network, there could be programmable devices that are network specific. For example, if the access network is a 802.11 network, there could be Access Points which are programmable and if the network is a sensor network network, there could be sensor gateways that are programmable. We refer to these types of elements which may include RF interfaces as Programmable Access Points (PAP).

The above elements within the edge site could be combined and configured in different ways. The CER would be a separate entity, but the PER and FED could be separate entities or could be combined together. The PAP, when it is required, could be a separate element or could be one incarnation of an FED. We look at the different configurations and their implications, below.

2.2.1 Configuration 1

Figure 2.2 illustrates a 802.11 network access network which consists of a 802.11 access network which connects wireless devices to the PER together with another access network which hosts an FED cluster. The FED cluster is connected to the PER through a switch. In this configuration, an experimenter who proposes to run wireless experiments would program PAP and PER (if necessary, otherwise, the PER could be a pass through that forwards IP tra□c) whereas an experimenter who plans to run distributed services experiments would program the FEDs and the PER, if ncecessary.

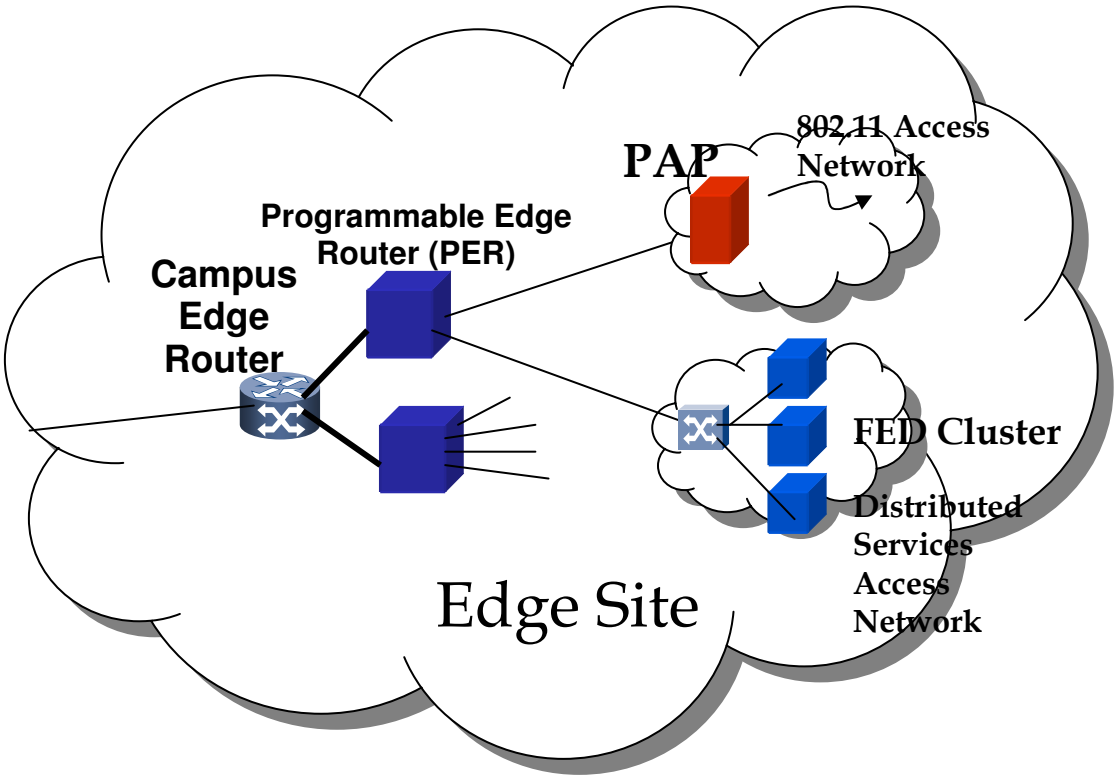

Figure 2.2: Edge Site Configuration 1

2.2.2 Configuration 2

Figure 2.3 illustrates a configuration where the PER and the FED are combined together in one device. In this configuration, a wireless experimenter would program the PAP as well as the PER+FED if necessary. A distributed services experimenter would program the PER+FED device to get a sliver of this device. This may be a preferred configuration in some cases but may not provide the capability to run experiments within distributed service device clusters. We believe that the FEDs and the PERs should be separate components to provide this added flexibility.

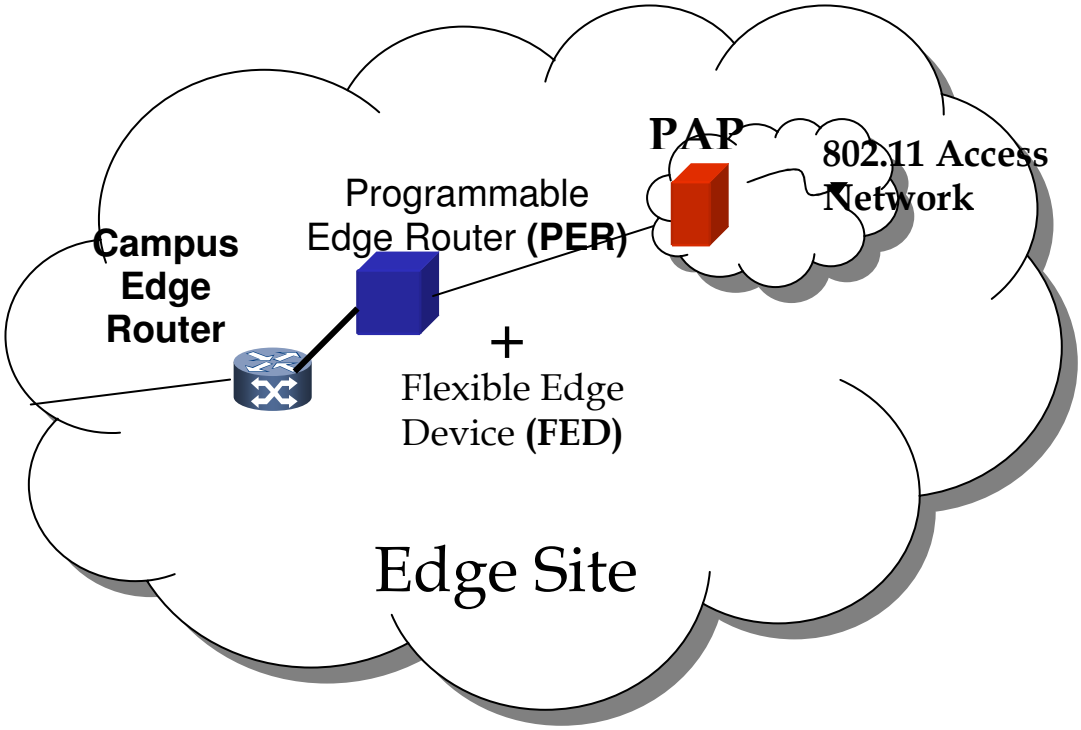

Figure 2.3: Edge Site Configuration 2

2.2.3 Configuration 3

In another configuration shown in Figure 2.4, the PAP is nothing but an incarnation of an FED with wireless interfaces. We expect a PAP to have similar software requirements to an FED and could be one and the same device except for the addition of RF interfaces. Typically, we expect a PAP to be used both to experiment with RF capabilities and to experiment with capabilities unrelated to RF (such as the capabilities provided by a WLAN switch) and an FED should serve that purpose.

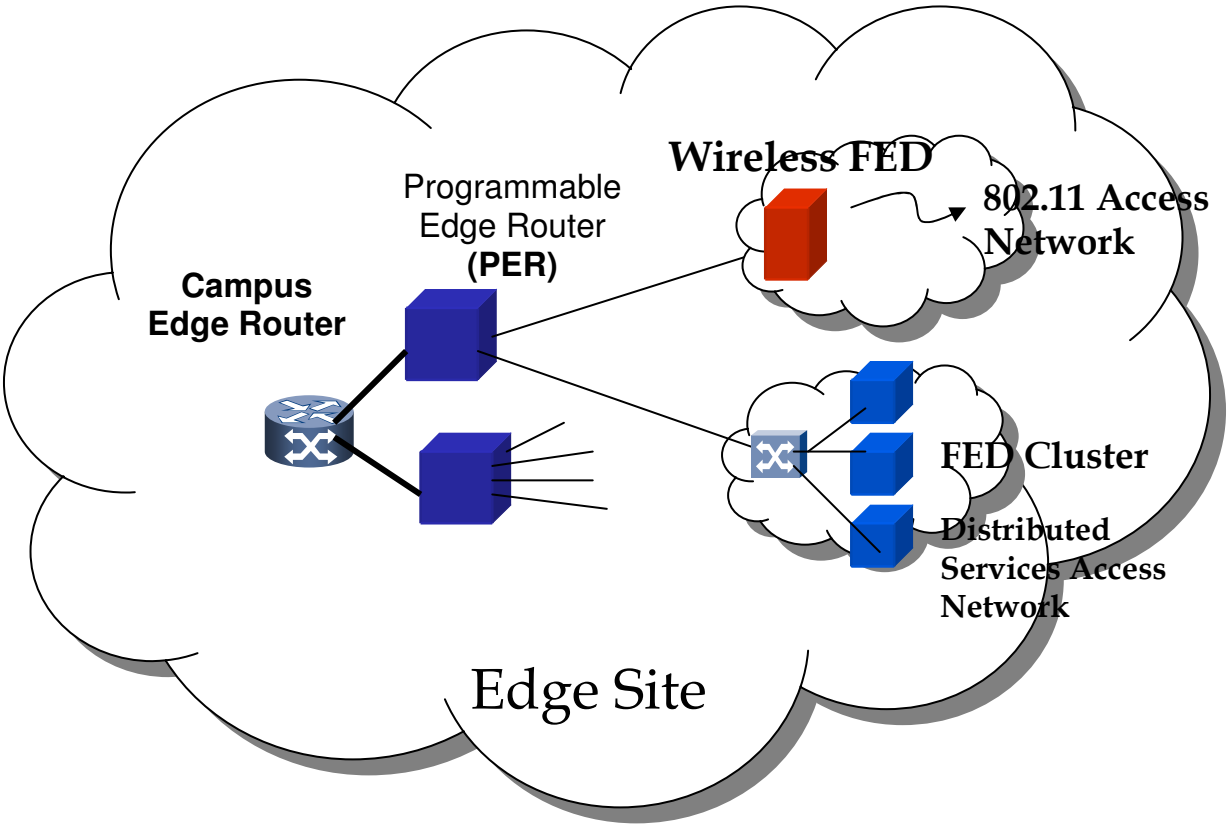

Figure 2.4: Edge Site Configuration 3

2.3 Software Architecture

In this section, we provide an overview of the basic software components that should be provided by the GENI software substrate on the two main programmable components, namely the PCR and the PER. Software required on the SONET cross-connect and the ROADM to provide virtualization at the optical circuit level is not covered.

2.3.1 Programmable Core Router

The PCR connects to **a)** other GENI PoPs**, b)** to the edge networks through the tail circuits, and **c)** to a service provider PoP for Internet connectivity. In order to provide this capability, the control plane on the PCR would require to support for the following routing protocols.

- BGP4 (E-BGP) to exchange routes with the edge router at the service provider PoP and with the CER at the other end of the tail circuits (this assumes that the edge network behind the CER is assigned IP addresses from the hosted site and is in a di□erent AS than the GENI AS).

- BGP4 (I-BGP) to exchange routes with other PCRs at other GENI PoPs).

- One of OSPFv2, IS-IS, RIPv2 (for IPv4) to perform routing between the GENI PoPs and within the PoP. It is possible that all these protocols may have to be implemented for experimentation purposes.

- Multicast routing protocols such as PIM-SM, PIM-DM, DVMRP, Single Source Multicast (SSM) etc. These protocols are purely for experimentation purposes and the need for these is up for discussion.

Other protocol and basic packet processing support to be provided by the GENI substrate at the PCR are listed below.

- A protocol for the control plane to communicate control information to the forwarding plane. We advocate this protocol be standards based such as ForCES which is currently being standardized within the IETF. There are other well-known but non-standard protocol such as the Forward Element Abstraction API proposed by the Click modular router.

- To provide bandwidth guarantees and QoS for both packets routed within the GENI AS and between the GENI PoP and a GENI edge network, we expect that the PCR support MPLS and this would require LDP and RSVP-TE to be supported.

- DiffServ support for labeling packets into di□erent equivalence classes and priority based forwarding.

- It is possible that the PERs connect to the PCR using layer-2 tunnels as well as encrypted layer-3 site-to-site tunnels. This would require support for L2TP, PPP and IPSec termination at the PCR.

- NAT, stateless and stateful firewall capabilities (these requirements need more discussion - it appears that functionality such as NAT and stateful firewall would be needed only for experimental purposes and may be left to the experimenter to implement).

- DHCP server, relay capability (as required) to provide IP addresses to end-points of layer-2 tunnels that may be terminated on the PCR.

- VRRP if two PCRs are used in redundant configuration.

In addition to this, basic operating system support and other platform software such as IPC, timers and memory management software needs to be provided on the control card general purpose processors, the line-card network processor host CPUs (for example, Embedded Linux on the Xscale processor on Intel IXP2800) and on the blade-server processors.

In the next section, we enumerate the reasons for our recommendation to support a to-be-standards based API such as ForCES by providing an overview and arguing for its applicability within the PCR.

2.3.2 The ForCES protocol

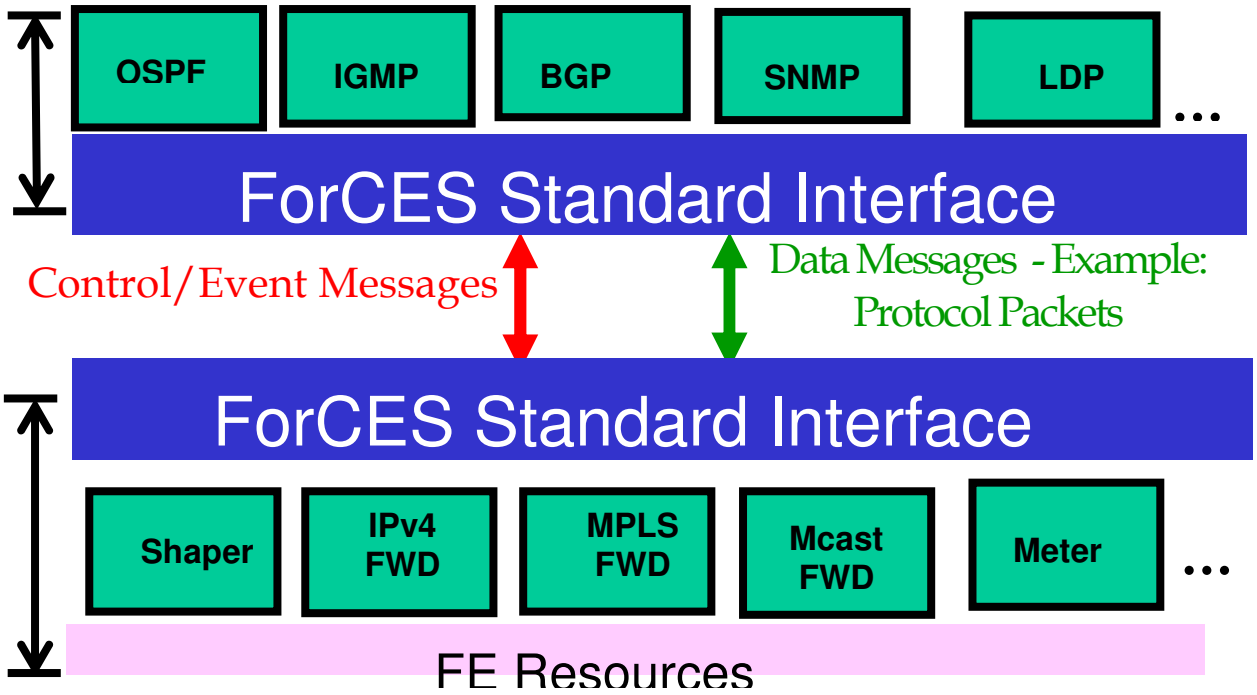

Figure 2.5: ForCES Protocol Architecture

ForCES stands for Forwarding and Control Element Separation and is a protocol that is being standardized within IETF. It would be an appropriate protocol within a PCR specifically since the PCR is supposed to provide the capability to experiment with different combinations of control plane protocols and forwarding functionalities.

ForCES defines three key components. These are a) a standard communication protocol between the control element (CE) and the forwarding element (FE), b) A standard and very simple operation code set for the CE to send control commands to the FE and, c) a standard numbering scheme for identifying data (or operand) at the FE which is very similar to the well known SNMP MIB numbering scheme.

In traditional commercial routers, a CE controls the FE(s) using proprietary APIs. ForCES aims to define a standard API for communication between a CE and an FE. In a sense, use of such a standard protocol within PCR would enable experimenters to a) implement experimental control protocols and experiment with them by interacting with standard FoRCES compliant FE functions implemented by the GENI substrate, and b) implement ForCES compliant FE functions and control them using the control plane provided by the GENI substrate. This flexibility is available to the experimenters independent of whether the FEs in the metarouter

they use is confined to one chassis or spans multiple chasses. A high-level ForCES architecture diagram is shown in Figure 2.5.

Conceptually, an FE consists of a group of Logical Function Blocks (or LFBs). Each LFB provides a specific function. For example, an LPM (longest-prefix-match) LFB takes an IP address as input and generates an index to the next-hop table. Each LFB has attributes that can be manipulated by a CE. For example, an LPM LFB may have a table of IP prefixes and next-hop indices. CE controls the behavior of an FE by manipulating the attributes of the LFBs in the FE. For example, CE adds or deletes a prefix table entry of the LPM LFB. We envision that within the context of the PCR, the GENI substrate would provide standard software CE functions such as BGP, MPLS etc. and standard LFBs such as IPv4 packet forwarding, NAT, stateful and stateless firewall (some of these are up for discussion as mentioned earlier). Both the CE functions and the LFBs provided by GENI would be ForCES compliant. Software support would be provided so that the experimenters could load their own CE functions on the GENI control cards and own LFBs. within the line cards and experiment with combinations of standard GENI CEs and LFBs with those provided by the experimenters.

There are only a small set of control op-codes that are specified by ForCES and the data naming convention is very similar to that of SNMP MIB. With ForCES, each API call to be performed on an FE can be modeled as one or more basic operations. Each basic operation can be specified as: <op-code, data-id, data> and the op-code is set is limited to create, add, delete, set/replace, get/query. ForCES categorizes all the configurable and accessible data into LFB classes. Each LFB class can have one or more instances. Each data within an LFB instance is assigned a path ID. So, a data-id can be viewed as the concatenation of an LFB instance ID and a path-ID: For example: <5.1.1.2.3> denotes data 1.2.3 of LFB instance 5.1. Using this example, <SET, 5.1.1.2.3, 192.168.1.1> would mean set data 1.2.3 of LFB 5.1 to IP address 192.1681.l.1 and <GET, 5.1.1.2.3, NULL> would read the value of data 1.2.3 of LFB 5.1. This illustrates the fact that the naming convention is very similar to SNMP which is well understood in general; we believe that this makes the addition of new LFB and naming and accessing the different parameters within the LFB very easy.

2.3.3 Programmable Edge Router

Compared to the PCR, we expect the PER to provide support for a more limited set of routing protocols. As the PER is not expected to connect directly to either the GENI PoP or to the PoP of another service provider (this is the function of the CER) BGP support would not be needed. To support routing within the edge network, PER is expected to support one or more of OSPFv2, RIP and IS-IS (depending on the requirements). We do not anticipate multicast support either (do we?).

We expect the CER to support MPLS/DiffServ but the PER should be able to support tagged VLANs, so that traffic can be differentiated at the PER with VLAN tags which will then be used by the PCR to use the appropriate MPLS tunnels and tag the packet with the appropriate DiffServ FEC bits.

We expect the PER to support IPSec, PPP and L2TP termination so that PERs that do not have

direct access to a CER can connect to the GENI PoP by tunneling packets through another PER, as described earlier. Of course, PPP, L2TP and IPSec client support would be needed as well so that site-to-site tunnels between a PER and a PCR can be established if required.

Platform support including timers and RPC as well as virtual machine support (as described in the previous section) would be needed as well. Redundant configuration of PERs is a question that has similar implications and challenges as we discussed earlier and is left for further discussion.

2.3.4 Flexible Edge Device

FEDs would enable experimenters to run distributed services on a sliver. Experimenters should also be able to cluster FEDs and provide load-balancing capabilities. For this to work, one of the FEDs could be used as a load-balancing switch which front-ends a cluster of other FEDs. The software support provided at the FEDs is to be determined by the Distributed Services Working Group.

2.3.5 Programmable Access Point

The PAP is wireless specific and should be programmable. The PAP should provide software support to tune radio parameters as well as support for other capabilities similar to that provided by a WLAN switch. These may include support for L2TP tunnel termination, user authentication capabilities, IAP (Inter-Access Point Protocol) support etc. In addition software support should be provided so that experimenters can run mobility experiments including micro-mobility and macro-mobility functionality (for example, by running a FA on the PAP). The software support on the PAP is to be determined by the Wireless Working Group.

2.3.6 Machine and Network Virtualization Support

Within the PCR and PER, we recommend that support for virtual machines (VM) be provided so that each experimenter receives a VM both on the control plane and on the forwarding plane. A VM will allow for resource isolation within the programmable routers. Experimenters can run their own versions of routing protocols on the control plane and implement their own LFBs without adversely affecting other slivers within the same processor/card. The support for VMs will also enable experimenters to modify/load their own kernel with their version of the network stack on the control plane. Similarly, we believe that the adjunct blade-server which will be used for higher layer processing (for example. XML based routing) should also provide VM support. In addition to resource isolation, network level isolation between simultaneous experiments needs to provided.

We envision an implementation similar to VINI implemented on the PlanetLab virtual infrastructure (PL-VINI). There are multiple design requirements that are specified and implemented within PL-VINI. Not all of the support provided by PL-VINI may be necessary but at the same time some additional support may be needed. The requirements put forth by VINI include a) ability to configure flexible network topologies, b) support for flexible forwarding and routing, c) support to connect to external networks (outside of the VINI nodes), and d) support for simultaneous experiments.

Within the context of being able to configure flexible network topologies, there are three main

issues. The first one is the provision of a large number of multiple virtual interfaces for an experiment although the physical infrastructure may support only a few. This may not be a concern within the PCRs at the GENI PoPs as the current thinking is to allow experimenters to configure a metarouter that may consist of multiple line cards with as many physical interfaces as the experiment needs. This would be a requirement within the PERs at the edge networks as the PERs would have very limited number of physical interfaces and may require a larger number of virtual interfaces to be provided for user experiments. The second issue is in providing virtual point-to-point connectivity between nodes. Within GENI, emulation of point-to-point connectivity may be required (depending on the experimental network topology) between the different physical interfaces on the PCRs on the GENI PoP and also between the physical interfaces on the GENI PoP and virtual interfaces on the PERs at the edge networks. Emulation of point-to-point connectivity between PCRs may be through MPLS tunnels between the PCRs to emulate multiple virtual links over the same physical links and provide bandwidth guarantees for each virtual link. Between a PCR and an PER, emulation of multiple virtual links could be through MPLS tunnels between the PCR and the nearest CER and the use of VLAN tags between the CER and the PER (this was discussed earlier and is based on the idea that CERs, which are anticipated to be commercial routers, would terminate MPLS tunnels). The third issue is with regard to manifesting the failure of physical links as failures in virtual links. This support would definitely be needed as part of the virtual link emulation within the GENI infrastructure. Note that virtual links may have to be emulated between PERs as well. We anticipate that this level of emulation could be supported through layer-2 tunnels as we expect PERs to support layer-2 tunneling mechanisms (as indicated earlier).

To provide flexible forwarding and routing VINI considers two issues. Firstly, experimenters should be allowed to load their own routing process (which could potentially run proprietary routing protocols) within their VM on the control plane. This support is definitely needed within GENI. These routing protocols need to able to find destination routes to machines within the GENI infrastructure and also to outside hosts. The issue could be complicated as the routing within GENI could be used on proprietary addressing whereas routing to external hosts would be based on IP. We envision that routing within GENI be handled by the experimenter using their routing protocols (which will then be carried over the virtual links that makeup the topology specified by the experimenter). The experimenters may experiment with standard routing protocols such as BGP and may want to establish adjacency with BGP peers on the Internet to route packets outside of GENI. This needs to be supported by the GENI substrate. Routing could be destination IP address based or any other scheme based on other parameters (including proprietary addressing mechanisms used by the experimenters). We envision that a Forwarding function would be only one of many other LFBs within the forwarding plane. Expanding on this concept, the experimenters should be able to load different types of LFBs within their own VMs on the line card processors to process their packets in di□erent ways before forwarding them over their virtual links. As previously indicated, ForCES seems to be an ideal protocol to provide the communication abstraction between the control plane and the forwarding plane to configure and monitor various LFBs of an experiment.

Another important design requirement as indicated by VINI and is important to GENI is the ability to forward packets to external hosts from GENI nodes. There are two issues that should be considered. Firstly, end hosts within GENI should be able to route their packets through the GENI infrastructure. This involves first sending packets from the end hosts through the edge

network to an PER. The mechanisms for this requires more discussion and will depend on the configuration. For example, if the experimenter uses IP forwarding and if the PER is one hop away, it could be configured as the default router. Otherwise, tunneling mechanisms may have to be supported between the end-hosts and the PER. Then, these packets need to be forwarded from the PER to a PCR over a virtual link (which, as indicated earlier, could be over MPLS tunnels). Finally, the PCR needs to forward the packets to the external hosts (which has to be IP communication). Note that the above discussion assumes that there is no direct exit point to the Internet from the PER other than going through the PCR. Secondly, return tra□c needs to be directed back to the PCR and this could be achieved through a NATing mechanism as in VINI.

The last design requirement posed by VINI and is applicable to GENI is isolation of multiple simultaneous experiments on the GENI nodes. VINI partly achieves this by using mechanisms provided in PlanetLab for resource isolation and also by adding PlanetLab extensions for VINI. Similar support is needed within GENI and again we stress that VM support should be provided for each experiment to support resource isolation and virtual link support to provide network isolation as discussed above. Within PL-VINI resource isolation is supported using UML (user mode Linux). It appears that there are other VM implementations such as Xen which claims better performance than the likes of UML, VMware etc. Further study is required to decide on a specific VM implementation for use within GENI.

In the next section, possible experimental network architectures are considered that would make use of the capabilities of the GENI software substrate.

3 Classes of Experiments

To help identify the key software components for the GENI backbone, our working group focused on three broad classes of experimental network architectures. First, we considered novel architectures that are not backwards compatible with the Internet. Second, we explored how GENI can support research in new control planes or distributed services, while exchanging conventional IP packets with Internet hosts. Third, we investigated architectures that dynamically establish virtual circuits at the optical layer. To make the discussion concrete, we focus on existing architectural proposals in each category, rather than trying to conceive of novel architectures.

3.1 End-to-End Architectures

A major goal of the GENI facility is to support experiments with "clean slate" network architectures that might not be backwards compatible with the legacy Internet. For example,

- **Wide-area Ethernet:** A network could consist entirely of "layer 2" devices such as Ethernet switches that forward frames based on MAC addresses.

- **Capability-based systems**: In a capability-based system, a data packet carries a capability that determines whether and where a router chooses to forward the traffic.

- **Internet indirection infrastructure:** In the i3 system, senders deposit packets at intermediate points in the network, and receivers install triggers that enable them to receive the traffic.

- **XML routers:** In an XML router, the data packets carry messages in XML format, with tags and attributes that determine how the packets are forwarded and processed at each hop.

Although these experimental architectures do not have an IPv4 data plane, they need to be able to reach end hosts via the Internet. As such, the GENI backbone needs to be able to maintain tunnels with end hosts to send and receive non-IP traffic over an Internet path, and use Network Address Translation (NAT) to ensure that return traffic goes through the GENI backbone. Inside the GENI backbone, these non-IP packets should be forwarded based on the experimental architecture. For high speed, packet forwarding may be done by software running on a network processor or FPGA. To support researchers in exploring new data planes, GENI could conceivably include software libraries for high-speed forwarding based on flat addresses (as specified by bit masks on the packet); in addition, a library could support experimenters in collecting packet traces that log certain sets of bits in each packet.

3.2 Alternate Control and Management Architectures

GENI is likely to be used to evaluate novel control and management planes, under conventional IPv4 packet forwarding. For example,

- **Routing control platform:** An RCP is a distributed service that monitors the network topology and exchanges reachability information with neighboring Autonomous Systems (ASes), and uses this information to compute forwarding-table entries on behalf of the routers.

- **Multicast overlay:** A multicast overlay supports on-demand streaming of audio and video content to many clients simultaneously. Servers running in the backbone can serve as relayant nodes in the multicast tree.

These experiments view the GENI nodes as a programmable ISP backbone that can run new control-plane software, while still forwarding IP data packets. Experiments involving routing may need to participate in the Border Gateway Protocol (BGP) with neighboring ASes in the legacy Internet, and advertise their own address blocks. Since many experiments may rely on IPv4 packet forwarding, the GENI backbone could include a software library for a high-speed IPv4 data plane running on network processors. To multiple experiments to share a single network processor, the NP may have separate forwarding tables (and other forwarding state, such as access-control lists) on a shared network processor.

3.3 Virtual Circuits

Another major goal of GENI is to enable experiments with network architectures that exploit the ability to establish virtual circuits. For example,

- **Valiant Load Balancing:** A VLB network forms a full mesh of virtual circuits between the routers. Data packets are forwarded through an intermediate node to distributed the load evenly over the backbone.

- **TCP flow switching:** Under TCP flow switching, a TCP SYN packet triggers the creation of a virtual circuit for carrying the data packets through the network. After an idle period, the network can tear down the circuit to free resources for future data transfers.

These experiments require researchers to have the ability to establish virtual circuits through the underlying network. In the VLB example, the circuits may be statically established as part of instantiating the experiment. However, an experiment with TCP flow switching requires a way for the experiment itself to trigger the set-up and tear down of virtual circuits on a relatively small time scale.

4 Principles for the GENI Backbone Design

This section discusses the guiding principles for the design of software for the GENI backbone. Some of these principles are motivated by the example experiments, and others translate general requirements for the GENI facility into specific guidelines for the backbone software. In addition to outlining the principles, we also discuss the kinds of software components necessary to realize these goals.

4.1 Robustness to Misbehaving Experiments

A cornerstone of the GENI facility is the ability to provide isolation between multiple independent experiments running on the same physical components. A buggy or malicious experiment should not be able to compromise the resource guarantees for other experiments. In some cases, existing techniques for providing isolation are sufficient. For example, a general-purpose compute blade could provide guaranteed CPU resources to each sliver by running an operating system with a CPU scheduler. Similarly, a SONET cross-connect can provide virtual circuits with guaranteed bandwidth. Components that are hard to share (such as network processors) can be allocated in their entirety to a single slice. However, the design of the GENI backbone may have points of contention that are difficult to prevent, forcing a careful balance of proactive enforcement (where possible) and reactive detection (where necessary).

For example, consider a slice that has been allocated a network processor and a certain fraction of the bandwidth on a particular line card. If the experimenter programs the network processor to send traffic in excess of the bandwidth allocation, the extra traffic may introduce contention in the switching fabric that affects other slices. GENI can reduce the likelihood of

contention by providing software libraries that shape traffic as it leaves the network process to enter the switch fabric. Still, an experimenter may accidentally or intentionally install its own software on the network process to send at a higher rate. We envision that GENI would need to monitor the network traffic to detect and report such violations, rather than relying solely on explicit enforcement of the sending rate. When violations occur, the researchers running experiments in other slices must be notified that isolation was compromised.

4.2 Multiplexing Access to Logical Resources

Some resources, such as CPU and bandwidth, are easily shared by multiple slices by applying conventional scheduling techniques. However, some logical resources are not easily shared in time, requiring the GENI backbone to provide gateway software that serializes the access to the resource and enforces isolation across experiments. We envision that the gateway software could run on the programmable router and communicate with the slices that need to employ the gateway service. Thus far, we have identified two such logical resources: the BGP sessions with neighboring ASes and the signaling interface to the optical switch.

Experiments with new control planes may need to exchange BGP messages with neighboring ASes in the legacy Internet. However, neighboring ASes may not be willing to have separate BGP sessions with each slice. Instead, we envision that the programmable router would run a BGP gateway that has BGP sessions with neighboring ASes, as well as separate BGP sessions to individual slices. The BGP gateway would combine the BGP update messages sent by the slices into a single stream of messages to each neighboring AS, and relay the neighbor's update messages to each slice. The gateway ensures that an experiment that sends malformed messages or crashes frequently does not crash the real BGP sessions to the neighboring ASes. The gateway can also filter BGP update messages to ensure that a slice does not send announcements for address blocks it does not own or send update messages at an excessive rate. The gateway can also apply policies that provide a slice with a customized view of its BGP connectivity. For example, one slice might want to connect to AS A, while another experiment might want to connect to both AS A and B.

Another class of experiments need the ability to dynamically set up and tear down virtual circuits. Yet, an experiment should not be permitted to set up and tear down circuits at an excessive rate, or to create a circuit that violates the resources limits for that slice. However, existing optical components, such as SONET cross-connects, do not support virtualization of the command-line interface or signaling plane. Instead, we envision having a gateway that lies between the slices and the optical component to provide the illusion of a separate virtual switch for experiments that require this functionality. The gateway interacts with the underlying optical component (perhaps via a vendor-specific command-line interface, or via a signaling protocol such as GMPLS) and also performs the necessary book-keeping and admission control to ensure isolation between slices. [**Note**: The backbone group needs to dig into the optical control issues in more detail to understand the kinds of signaling interfaces supported by existing commercial devices. Also, we may need to take special care to retain control over the routing of virtual circuits, and to prevent automatic rerouting after a failure, should an experiment want to handle these failures directly.]

4.3 Software to Lower Barrier to Experimentation

In addition to supporting virtualization, isolation, and programmability, GENI could provide software libraries to lower the barrier to experimentation. For example, the distributed services working group is defining a variety of basic services, such as distributed storage and topology discovery. The GENI backbone can also provide software libraries that provide basic functionality that would be useful for a wide class of experiments and needs to run at a high speed. Many of these functions relate to how the data plane forwards packets:

- **Shared IPv4 data plane for distributed services:** Many slices will be running experiments with new distributed services that carry conventional IPv4 data packets across the GENI backbone to other hosts. Devoting a separate network processor to each (low-bandwidth) slice would be very wasteful. Instead, we envision that a single network processor could forward IPv4 data packets across the backbone on behalf of multiple slices, while enforcing the necessary bandwidth limits on each slice. The forwarding-table entries may be created by conventional IP routing protocols running on the GENI backbone.

- **Shared IPv4 data plane for alternate control planes:** Other slices will be running experiments with new control planes that need to generate their own data-plane state (such as forwarding-table entries and access-control lists). Still, many of these experiments may assume conventional IPv4 packet forwarding, allowing them to share a single network processor that implements IPv4 packet forwarding and filtering. As such, we envision that the programmable router would have software that would allow a single network processor to perform IPv4 packet forwarding with separate forwarding tables and access-control lists for the participating slices.

- **Forwarding based on flat addresses:** Another common class of network architectures rely on flat addresses, such as MAC addresses. The GENI backbone could conceivably provide software libraries that support packet forwarding based on flat addresses, by exploiting the content-addressable memory (CAM) functionality on the network processors. On one extreme, the library could support a single packet format (such as Ethernet frames); on the other extreme, the library could allow the researcher to specify a bit mask for the fields in the packet that are used to index the CAM.

- **Packet sampling:** Many researchers may want to collect measurements of the packets in their slice that flow between components inside of GENI. (Note: This is distinct from the monitoring of IP packets that enter and leave GENI, as needed for accountability and security.) Providing libraries for packet sampling, either for IPv4 packets or configurable packet formats, would lower the barrier for researchers to collect detailed traces at high speeds.

Of these four scenarios, the first item (on shared IPv4 data planes for distributed services) is the most important, since this is necessary to make efficient use of the network processors to support the large number of experiments with new distributed services. We could arguably forego providing libraries for the other three items, since researchers could write these modules themselves, and presumably share them with others who need similar functionality. Still, there

is value in providing hardened, optimized, and well-supported software that would run on the network processors, to lower the barrier to evaluating new network architectures. Also, having these libraries written, debugged, and supported in a professional manner may be preferable to having many researchers depend on "research code" that is not as well written or debugged. In the end, the decision of which libraries to support (if any) may depend on the available resources and the likelihood that such functionality would be likely to emerge from the research community anyway.

In supporting these functions, we advocate applying existing APIs wherever possible, to obviate the need for a substantial design activity to define new APIs. In addition, using existing APIs may enable the use of existing software implementations, and would allow researchers to design their experiments based on the APIs described in existing documents. Examples of such APIs for IPv4 packet forwarding include the ForCES interface (defined by the IETF to enable a separation of data and control planes in IP routers) and the Forwarding Element Abstraction (defined by the Click open-source forwarding engine and used by the XORP open-source router). Another example in the context of packet monitoring is the psamp (packet sampling) activity at the IETF.

4.4 Capitalizing on Open-Source Software Whenever Possible

Much of the key software functionality required for the backbone is already available, at least in part, in open-source software. For example, GENI can support communication with the legacy Internet by exploiting

- data-plane functionality in Click (e.g., packet forwarding and NAT).
- routing-protocol implementations in XORP, Quagga, or OpenBGPd (e.g., for experimenters to evaluate and extend routing protocols, and for use in building the BGP gateway).
- tunneling using OpenVPN.

These issues are discussed in more detail in the VINI paper "In VINI Veritas: Realistic and Controlled Network Experimentation" at SIGCOMM'06. We may also be able to exploit open-source software for other purposes, such as packet monitoring (e.g., tcpdump) and tools for developing code for network processors and FPGAs.

4.5 Separate Identifier Spaces for Different Functions

Each component needs an id – a control id. Each sliver—an instantiation of a slice on a single physical resource—needs an id. Each slice has an id. Between GENI nodes, include the slice id in the packets. Should this be an MPLS label or VLAN tag, possibly stacked? [Note that stacking may be a very nice way to support federation, where some experiments may span multiple infrastructures.] Should these labels/tags be globally unique to make trace-back and troubleshooting easier, or swapped at each hop to preserve ID space? [Side question: does LDP allow globally unique labels?] What about limitations on ID space, e.g., just 4K VLANs? How do end users identify the slices they want to "opt in" to? [Some DNS support? Is Distributed Services already looking at this problem?] Other question: should GENI provide some support

for helping slices manage their IP address space?)

5 Topics for Discussion

This is a laundry list of issues that need to be investigated and discussed. Many of these issues span multiple working groups.

5.1 Edge Device at Local Sites

The tail circuits need a device on the remote end at the local edge sites. We have identified a set of elements at the edge sites to be provided by the GENI infrastructure (PER, FED, PAP). We have identified some of the required functionality of these devices but we still need to debate these features as well as the different combinations and configurations of these devices. For each of these devices, we need to identify the hardware platform. Can the PER be a scaled down version of the PCR? How many ports do we need to support? What technologies and speeds? For the FEDs and PAPs, will PC platform be sufficient. Can we assume that the FED clusters and the PAPS will have already shaped/scheduled the traffic, obviating the need for the PER to have QoS functionality? [**Note**: E-mail discussion has already ensured on this topic across the backbone, wireless, and distributed-services groups, but we have not yet reached a common understanding]. We have also identified the CER as the point of entry to the edge site. In this case, do we expect the tail circuit (for example, an MPLS tunnel) to be terminated at this CER or would it be terminated at the PER. If it is terminated at the CER, how would QoS be enforced on the traffic between the CER and the PER (can we use VLAN tags?). Also, what type of configuration is required at the CER and how would this be enabled?

5.2 Configuration of a slice

The issue pertains to the GMC (GENI Management Core). How would an experimenter configure the different elements within the GENI PoP and the Edge Sites to makeup a slice? This requires different components in the experiment to be identified, slivers in each of these components be reserved with the required capabilities (CPU, memory, network bandwidth etc.) and the slices be glued together and made consistent. There are two ways to approach this problem. One is through direct configuration of each of the elements using the GMC (which could consist of management components that are hierarchically organized). The other approach is using signaling where all the elements in the path (or a set of paths which could form a tree) that an experimenters expects packets to flow within the experiment are signaled from a root node using some variant of RSVP (extended to include support for CPU and memory allocation within the components, as well as allocating any other resources). This configuration versus signaling issue needs further discussion.

5.3 Constructing End-to-End Experiments

How do we construct end-to-end experiments, in terms of gluing all the pieces of a slice together. What does the end host need to be running? Might we want to virtualize the end host

itself for certain classes of experiments (e.g., experiments with new end-to-end architectures)?

5.4 Monitoring

Monitoring is important for **(i)** determining if GENI itself is buggy (e.g., not providing isolation), **(ii)** providing measurements for experimenters to evaluate their ideas, (iii) detecting attacks launched from within GENI and tracing them back to the offending slice, and (iv) detecting attacks on GENI from the external Internet.

Paul Barford has been thinking about these issues in general. We have some questions about the monitoring capabilities required for the backbone links—both the intra-GENI links (for the benefit of experimenters, and for trace-back within a slice) and edge links (as traffic goes to/from the legacy Internet). The high speed of some of the links may be a constraint we need to consider, especially for full-packet capture. Also, will the programmable router need a disk to (temporarily) store the traces, or will a separate monitor store this? We may also need to log NAT state to be able to interpret data collected before/after address translation has been applied.

5.5 Inter-Slice Communication

How do we envision inter-slice communication taking place? This issue must also come up in the distributed services working group.

5.6 Bootstrapping and Configuration

How should we boot-strap and manage the individual nodes (e.g., the various blades in the programmable router)?

What about discovery and dissemination within the GENI backbone, to allow the backbone to "boot" and for researchers to communicate with the slivers in their slices. Initially, we could assume that all of the backbone components are accessible via the legacy Internet, but we may want a more auto-configuring and secure solution for the long-term. Hui Zhang will be thinking about these issues, drawing on his work on the 4D architecture.

Are there challenges with auto-configuring legacy commercial devices that were not designed with that in mind? Perhaps the auto-configuration and discovery functionality could be built in the gateways that we already need for adding virtualization (e.g., of the signaling/command-line interface to the optical components).

5.7 Alarm Propagation

When equipment fails, virtual components fail, too. Some experiments will need to be notified that their virtual components have failed, whereas others may really run on an overlay that hides these kinds of annoyances. For the "raw" experiments that want to see the details, we need to produce an "up call" of sorts to let the each affected sliver know that its (say) virtual link has failed. This might involve supporting certain SNMP MIBs that are (relatively) device independent. This general problem of alarm propagation seems to be a general GENI-wide issue, and not unique to the backbone group, though the backbone group may need to take

special care to make sure GENI had visibility into layer-2 equipment failures.

5.8 NAT/Firewall Per Slice

We probably need NAT and firewall state per slice in communicating with the external Internet. What about the difficulties of supporting P2P applications in the presence of NATs and firewalls? (Think, for example, of the challenges conventional P2P applications like Skype face when both end points are running NATs and firewalls. Sometimes it is unavoidable to have to direct traffic through a third party that each of the other two nodes can contact. Will GENI need to do this, too, for cases where two end users are NATted addresses?

5.9 Tunneling Over the GENI Backbone

What if an end user is on a campus with a tail circuit to GENI and wants to connect to a network architecture that is not running on the nearby PoP? Should the traffic be routed via GENI itself to the nearest PoP that is part of that slice? How? Do we need the ability to tunnel through the GENI backbone to reach a slice? That is, does the GENI backbone need to provide a general reacahbility service.

5.10 Dynamic Circuit Set-Up

What kinds of set-up speeds are feasible with today's hardware and software? If the hardware can support fast set-up, but the software interfaces don't (as seems to be the case), how much are we willing to pay to have more flexible software with faster set-up? Is circuit set-up on packet time scales a requirement for GENI? If so, why? If so, how far is existing signaling software from achieving these goals, and how much would it cost to get new software that is better?

How do we support policies on a per-slice basis (this relates to the gateway/mux function)? Do we want to allow slices to do explicit routing or actually have the illusion of running their own signaling and path-selection software at each hop?