GENI Backbone Run-Time Software for Experimenters

GDD-06-36

GENI: Global Environment for Network Innovations

November 30, 2006

Status: Draft (Version 0.1)

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

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

Revision History:

Version	Changes log	Date
v0.1	Original version posted	11/30/06

Table of Contents

1 Introduction

The GENI backbone consists of a collection of Programmable Core Nodes (PCNs) connected via an underlying fiber plant, with tail circuits to edge sites. In addition to the component manager, the PCN needs specialized software for three main purposes:

- **Capitalizing on high-speed packet forwarding**: The PCN should provide application programming interfaces (APIs) to enable experiments to exploit the high-speed packet-forwarding hardware described in "A proposed architecture for the GENI backbone platform" (GENI Design Document GDD-06-09).

- **Connecting experiments to the legacy Internet**: Researchers using the PCN need software modules for interfacing with the legacy Internet, including network address translation (NAT), tunneling to IP-connected end hosts, and maintaining Border Gateway Protocol (BGP) sessions with neighboring domains.

- **Controlling the optical equipment within an experiment**: To enable experiments to dynamically control the optical equipment, the PCN needs to include software that allows researchers to configure or signal the optical equipment, while maintaining isolation between slices.

The current version of this document focuses on the first two items, deferring the discussion of the software for controlling the optical equipment till a future revision. This document is part of a three-document series on the GENI software that obsoletes the earlier "Backbone Software Architecture" document (GDD-06-25). Besides the current document, the other two documents describe (i) the hardware-specific software for the programmable router (e.g., the software that runs on the network processors and FPGA, as well as development tools for experimenters to program these devices) and (ii) the component manager and management aggregate for the Programmable Core Nodes. *Note: At the time of this writing, these other documents are under preparation; the material on hardware-specific software will likely appear in an update of GDD-06-09.*

Before describing the main functionality provided by the backbone software, we discuss the key principles that drive our thinking about these software components:

- **Lowering the barrier for creating 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 logic (e.g., to run on a network processor or FPGA), GENI should provide implementations of high-speed packet forwarding, with suitable APIs for experimenters to install forwarding state (e.g., forwarding-table entries and access-control lists), without programming the network processor or FPGA directly. Similarly, we envision that some experiments would need to exchange reachability information with neighboring domains via the Border Gateway Protocol (BGP). Rather than having each research team form relationships with Internet Service Providers (ISPs) to arrange their own BGP connectivity, we envision that the

GENI facility would arrange such connectivity and enable multiple experiments to share a single BGP session from the GENI substrate to each neighboring router.

- **Multiplexing access to logical resources that are not easily shared in time**: 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 domains and the signaling or configuration interface to the dynamic optical switch.

- **Capitalizing on open-source software wherever possible**: Much of the key software functionality required for the GENI backbone is already available, at least in part, in open-source software. Exploiting open-source software would lower the cost for developing and maintaining the GENI backbone 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 building the BGP gateway), tunneling using OpenVPN, and tcpdump for collecting IP packet traces.

- **Capitalizing on standard protocols and APIs wherever possible:** Standards bodies, such as the IETF, have defined protocols and APIs that overlap substantially with some of the functionality needed in the GENI backbone. Drawing on these standards, where possible, capitalizes on the substantial work involved in defining the functionality and increases the likelihood of alignment with commercial hardware and software development. For example, ForCES (created by the IETF) defines an API for installing forwarding state, such as forwarding-table entries and access-control lists; similary, the psamp working group is defining standard support for packet sampling to collect traffic statistics. In some cases, the GENI backbone may be able to rely on *de facto* standards, such as the forwarding element abstraction (FEA) used by the Click modular router or the Linux API for installing forwarding tables in the kernel, in much the same way.

In the next two sections, we describe the software for capitalizing on the high-speed hardware for packet forwarding and connecting to the legacy Internet, respectively.

2 Capitalizing on High-Speed Packet Forwarding

The programmable router consists of a switching fabric that interconnects line cards and processing elements, including general-purpose processors and specialized hardware, such as network processors (NPs) and field programmable gate arrays (FPGAs). Experiments can capitalize on the specialized hardware for high-speed packet processing. Although some researchers may want to program an NP or FPGA in arbitrary ways, many experiments would use these devices to offload conventional packet-handling functions, such as

- *Packet forwarding* by indexing a table based on some bits in the packet header (e.g., based on a destination IP address or MAC address),

- *Access control* by discarding packets that match rules based on some bits in the packet header (e.g., based on a destination address or the IP five-tuple of source address, destination address, source port number, destination port number, and protocol), and

- *Queuing and rate limiting* based on a packet shaper that ensures an experiment does not send packets to a particular line card beyond the configured rate (e.g., based on a constant bit rate or, more generally, a leaky-bucket traffic shaper).

Rather than dedicate a separate NP or FPGA to each experiment, we envision that many experiments can share a single NP or FPGA that implements these packet-handling functions. Sharing an NP and FPGA makes more efficient use of the resources and also obviates the need for the researchers to program the specialized hardware directly. The shared NPs and FPGAs need not be limited to IPv4 or MAC-based forwarding. Rather, they may support a more general forwarding model based on a configurable set of bits in the packet header, allowing different experiments to have different header formats and different address/label sizes for indexing the packet-handling state.

When using a shared NP or FPGA, the researcher would write software, running (say) on a general-purpose processor, that installs packet-handling state (e.g., forwarding-table entries or access-control lists) in the shared NP or FPGA. The NPs and FPGAs will have a particular configuration interface for instantiating the state and ensuring that an experiment does not try to instantiate more state than expected. This logic would run directly on the NPs and FPGAs. Although the researchers' experimental software could directly generate the commands to the NPs and FPGAs, we envision providing a more familiar, higher-level API that hides these low-level details (including variations across NPs and FPGAs in how packet-handling state is installed). In particular, we envision that an experiment may employ one of these three APIs:

- **Forwarding Element Abstraction (FEA) in Click**: Many researchers building prototypes of networked systems use the Click modular router 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.

- **Forwarding and Control Element Separation (ForCES) standard**: ForCES is an emerging IETF standard 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.

- **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. Allowing these experiments to capitalize on the dedicated packet-forwarding hardware transparently would offer a substantial advantage to experimenters. As such, we envision supporting updates to the kernel forwarding table as a standard API for using the shared NPs and FPGAs. For example, a library could implement the main system calls for accessing the kernel forwarding tables, while transparently copying the table entries to an NP or FPGA. Alternatively, periodic polling of the kernel forwarding tables would provide a way to mirror the forwarding state without modifying the system calls.

In practice, these APIs may be more general than what the shared NP or FPGA could easily support. As such, we envision defining a restricted subset of these APIs that accurately captures the capabilities of the NP and FPGA hardware. (*Note: we need to clearly identify these restrictions.*) Also, we note that any researcher that wants a different API always has the option of using the "raw" interface to the shared NP or FPGA, such as writing table entries directly into a ternary content addressable memory (TCAM).

3 Connecting Experiments to the Legacy Internet

Many experiments will need to interact with the legacy Internet in some way, such as tunneling to end hosts, receiving return traffic from external services, and maintaining BGP adjacencies with neighboring domains. *Note: We still need to reconcile these requirements with the services already envisioned by the Distributed Services Working Group. Also, we need to determine whether each PCN should run one instance of these services vs. run an instance per sliver.*

3.1 Tunneling to End Hosts

End hosts connected to the legacy Internet may opt in to an experiment running on a GENI backbone node. As such, the GENI backbone node must provide a way to terminate a tunnel. We envision that the programmable router would run a server, such as OpenVPN, to terminate tunnels to these end hosts. Although the tunnel would be implemented using IP (for backwards compatibility), the packets sent over the tunnel, and delivered to the experiment on a virtual interface, may have any arbitrary format chosen by the experimenter.

Note: We need to resolve whether the tunnel should traverse the Internet to reach PCN running the sliver, or whether the tunnel can partially traverse the GENI backbone itself, including PCNs that might not be part of the slice of interest. If the tunnel is permitted to traverse GENI components, then the GENI backbone would need to provide a basic reachability service to direct the packets to the chosen sliver. This may be realized by a thin, best-effort sliver (that is part of the experiment's slice, though perhaps not visible to the experimenter) that serves only to direct the tunneled packets to the appropriate sliver that terminates the tunnel.

3.2 Receiving Return Traffic from External Services

The GENI backbone nodes will have connections to the Internet for reaching external services, such as Web sites. These sites are not necessarily "opting in" to a GENI experiment. In some cases, the external site may not know how to reach the IP address of the user that initiated the communication, e.g., because the user has a private IP address for his end of the tunnel to

GENI. Even if the user has a public IP address, the experiment may not want the return traffic to reach the user via the legacy Internet. Instead, the GENI backbone may need to ensure that return traffic from these sites goes through the GENI backbone and reaches the appropriate sliver. Network address translation (NAT) at the GENI/Internet boundary can provide control over whether and where the return traffic reaches GENI. We envision that conventional NAPT functionality, perhaps implemented in a network processor or FPGA for high speed, would suffice for this purpose.

Note: This NAPT model assumes that the slice, or an end host connected to the slice, initiates the communication with the external server. Some applications, like peer-to-peer file sharing, do not match this model, since the external host (not participating in GENI) may want to initiate communication. This is identical to the challenges today of supporting peer-to-peer communication across NATs and firewalls. When this style of communication is necessary, the experiment may need public IP addresses of its own to enable external, legacy hosts to initiate communication.

3.3 Maintaining BGP Adjacencies with Neighboring Domains

Some experiments connected to the legacy Internet may need control over how they direct traffic to external hosts, and how legacy hosts reach them. For example, a researcher evaluating a new routing protocol may need to exchange reachability information with neighboring domains (in the legacy Internet) via the Border Gateway Protocol (BGP). These experiments may have their own IP address blocks (i.e., prefixes) to announce to the Internet, and may want to receive BGP announcements for externally-reachable prefixes. However, a neighboring domain might not permit its routers to have separate BGP sessions with many slivers running in GENI, due to concerns about scalability or malfunctioning/misbehaving experiments. In addition, multiple slivers cannot easily share a single BGP session through conventional multiplexing techniques, such as dividing access to the session over time.

Instead, we envision that the programmable router would run a BGP gateway that has BGP sessions with neighboring domains, as well as separate BGP sessions to individual slices. The BGP gateway would combine the BGP update messages sent by the slivers (each with its own set of IP prefixes) into a single stream of messages to each neighboring router, 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 domain. The gateway can also filter the 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 domain A, while another experiment might want to connect to both domains A and B.

Implementing a BGP gateway does not seem like a particularly difficult task. The gateway functionality is very similar to a route server or route reflector, with some extra care needed to provide transparency to the experiments. Ideally, the experiment would think it has dedicated BGP sessions to the neighboring domains, rather than passing messages back and forth through the gateway. Also, for performance and scalability reasons, only the BGP messages should travel through the gateway; regular data packets should travel directly from the experiment's sliver to the link connecting to the neighboring domain. We believe that the gateway function can be implemented using existing open-source routing software, such as Xorp, Quagga, or

OpenBGPd. For example, the BGP gateway may run one BGP process for each external neighbor, with separate sessions to each of the participating experiments. Careful configuration of the routing software and the underlying environment can give the experiments the illusion of direct BGP connectivity to the external neighbor.