The Design of a Programmable Edge Node for GENI

University of Massachusetts Lowell
September 2009

Abstract

A high performance and flexible programmable edge node (PEN) is an important element to the GENI sites federating into the GENI global network and to the users carrying out experiments. The PENs provide virtual network interface cards (NICs) and virtual containers to support concurrent GENI experiments on the shared substrate. Experiment measurement can be performed at the NIC level, being isolated from the experiments running on the host CPUs. In this document, we present the design of a PEN prototype based on multi-core x86 processors and network processors (NPs).

1. Introduction

GENI is a highly virtualized network infrastructure being established for supporting large scale network experiments. The users of GENI will request resources from GENI aggregates and deploy their experiments on the allocated slices of resources. At a micro level, users execute experiments on physical hosts with virtualization capabilities, and measure the performance of the experiments. A Programmable Edge Node (PEN) connects a set of components at a GENI site to the outside network and presents virtual resources to the researchers. Our project focuses on the prototyping, analysis and optimization of PEN with the following motivations:

(1) A high-performance and cost-effective PEN can enable resource providers with limited budget to participate in GENI, and a familiar development platform enables early user trials. The scale of a GENI aggregate can vary and over-architecting needs to be avoided. One of our goals is to provide a quantitative performance analysis of the most popular server architecture (x86 multi-core and PCI-express) used to support virtual routers in a PEN, in comparison with some high-end programmable core routers based on ATCA architectures. The performance analysis will be valuable for GENI architects to understand the performance potentials and budget their sites appropriately.

(2) Isolation of GENI experiments from the measurement and diagnosis of GENI substrate is required. We need a system architecture that can provide hardware support for experiment isolation, fault-tolerance, measurability and diagnosis of the substrate. With the proposed PEN architecture, we can decouple the experiments and measurements into two set of processors isolated with the PCIe bus.

The x86 server architecture is one of the most popular computing platforms. And the trend is that multiple cores are integrated to a chip to provide scalable performance. The number of cores on a chip reaching tens to hundreds is the direction. With respect to the interconnection on board, PCI-Express is now universal in all the PCs and x86 servers. Chip vendors have released specifications of high-speed low-latency interconnections such as Hyper-Transport and QuickPath (QPI) to achieve throughput of 40GB/s and beyond to and from the microprocessor.

Our project follows the advancement path of these emerging technologies. We focus on x86 multi-core architecture, which will be benefited from more cores on a chip in the coming five

years. We develop virtual NICs and packet-processing acceleration on Netronome's NFE-i8000 NP card that employs a multi-core based network processor (NP). The NP interfaces with the host CPU via PCI-e bus currently. Netronome has licensed Intel's QPI technology to connect host CPU with their next-generation of Netronome packet processing engines. Thus, our proposed PEN platform can evolve with processor and interconnection technologies.

This document intends to describe the design of the PEN to help the readers understand the internals of the PEN. We hope it can facilitate the deployment and further development of PEN by interested early adopters. However, the readers should be aware that this document is a live document that is being constantly refined, due to the prototyping nature of the work. Please refer to PEN project website (http://cans.uml.edu/index.php?n=Research.PEN) for the most up-to-date information.

In the rest of this document, we introduced the hardware and software architecture of the PEN prototype, in Section 2 and 3, respectively. We summarize our work in Section 4 and give the readers the pointers to PEN-related materials in Section 5.

2. Hardware Architecture

We will implement a PEN prototype with an x86 multi-core server and Intel IXP network processors illustrated in Figure 1 (left: conceptual architecture, right: hardware architecture.)

The right portion of Figure 1 illustrates the hardware architecture built upon the off-the-shelf components including two quad-core Intel Xeon processors, 8GB memory, 400GB storage and a Netronome NFE-i8000 card with an Intel IXP 28xx network processor onboard. The NP card has four 1Gbps Ethernet interfaces connected to up to four clusters, which can be, for example, part of the PlanetLab, Emulab or other future GENI clusters. More NP cards can be added to scale up the number of computing clusters connected (shown as dashed box on the right of Figure 1.)

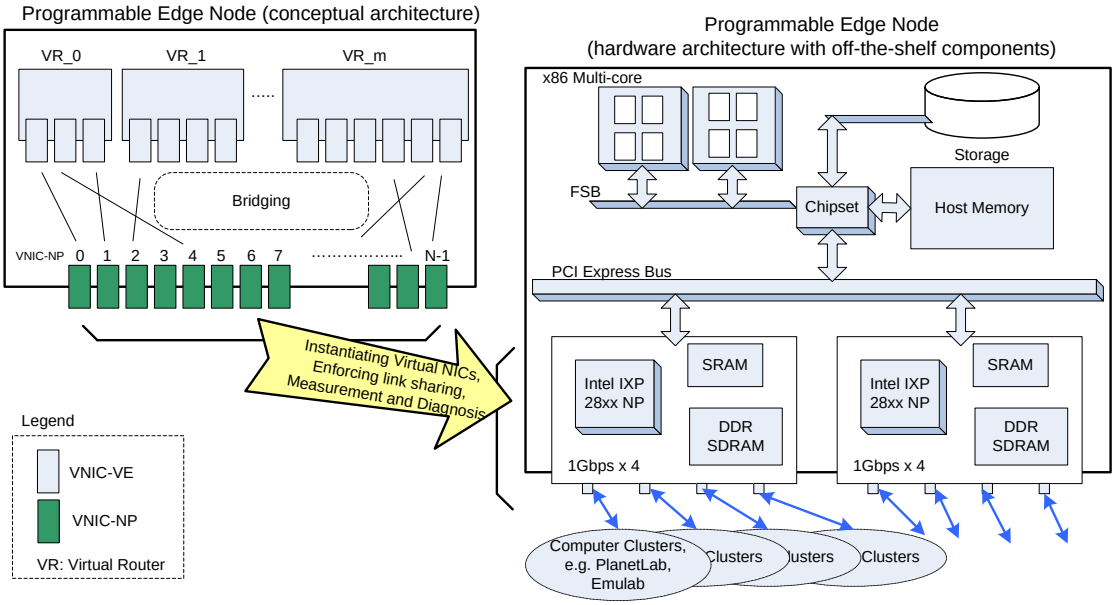

Figure 1. Architecture of Programmable Edge Node on an x86 plus network processor platform.

Netronome NFE-i8000 network processor based acceleration card used in the PEN prototype. NFE-i8000 is a PCI express card with four 1Gbps Ethernet ports. The card is equipped with an Intel IXP2855 network processor, 768MB RDRAM, 40MB QDR SRAM and 9Mb TCAM. The NP has 16 programmable cores running at 1.4GHz. Each core is independently programmable and supports hardware multithreading (8 threads each). The card is plugged into a multicore x86 server and functions as an intelligent NIC.

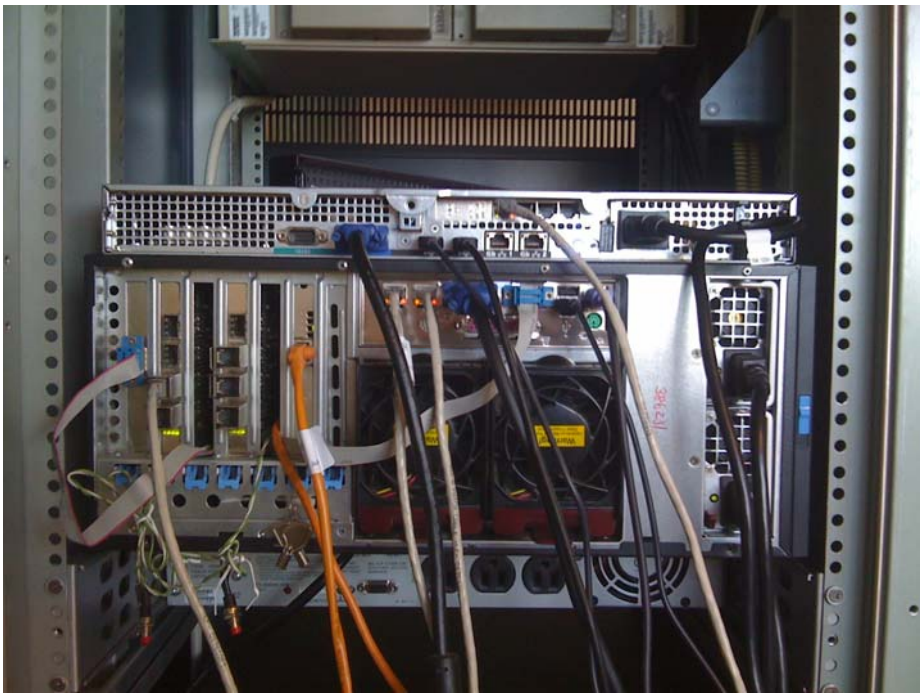

Figure 2. A Rack-mounted PEN Prototype in Operation at UMass Lowell CANS Lab

We have acquired the hardware for PEN, including a multicore x86 server (vendor: Supermicro, dual quad core, 8GB memory, 400GB hard disk) and a Netronome NFE-i8000 network processor card. The PEN prototype has been fully assembled and located in the CANS lab at UMass Lowell. Figure 2 shows the rear view of the rack-mounted PEN prototype in operation.

3. Software Architecture

3.1 Overview

Our prototyping work is mainly to realize the conceptual architecture of the PEN through software development. In the PEN, virtual routers can be created per user requests, acquiring specified CPU, storage, and network link resources. Virtual NICs are allocated to the virtual routers for connecting to outside networks. Additional functionalities such as experiment measurement are implemented at the NIC level with network processor cards.

We install CentOS 5 with OpenVZ enabled Linux kernel. We create configuration scripts and customized templates of programmable router to instantiate virtual routers using OpenVZ's Virtual Environment (VE). Each VE contains a number of virtual network interfaces (denoted as VNIC-VE) one-to-one mapped to virtual interfaces on the host (also called Host Node or HN).

We denote the latter VNICs as VNIC-HN, which are bridged with "real" network interfaces (e.g. eth0) of the HN. That is, VEs communicate with outside network in the following path:

VNIC-VE within VE ←→ VNIC-HN on HN ←→ "real" interfaces of HN ←→ outside

We have two configurations depending on what "real" interfaces of HN are actually used:

Configuration 1: two regular onboard NICs are used as "real" interfaces. This is the baseline configuration where the network link allocation is managed by the host CPU. We plan to evaluate the effectiveness of the network link bandwidth allocation, especially in the cases when VE's workloads are heavy and vary.

Configuration 2: Netronome NP card(s) are used as "real" interfaces. We use VNIC-NP to denote the network interfaces presented to host Linux, but in fact created and managed by NP. The arrow in Figure 1 shows the instantiation of VNIC-NPs by the NP.

The communication path in Configuration 2 will be as follows:

VNIC-VE in VE ←→ VNIC-HN on HN ←→ ***VNIC-NP on HN ←→ Intel IXP NP*** *←→ outside*

Compared to Configuration 1, we use VNIC-NP interfaces to replace the regular NICs. The NP is programmed (through NP side microcode, host and NP device drivers) to present VNIC-NP interfaces to the host as if they are regular NICs. The actual packet transfer goes through Intel IXP NP system on the Netronome card.

The benefits of this design (**Configuration 2**) are:

(1) GENI experiment measurement and diagnosis can be achieved on NP instead of host CPU. This improves the isolation of GENI experiments, and fault-tolerance and measurability of GENI substrate.
(2) The NP offloads traffic shaping from host CPU. This alleviates the workload of host CPU which is to be utilized by virtual routers requested by GENI users. As a result, the link sharing can be guaranteed even when the router workloads become heavy or vary.
(3) The number of VNIC-NP is configurable. The number of VEs and the number of their Virtual NICs change with GENI user's request and release of resources. With a configurable number of VNIC-NP, virtual NICs on VEs can be one-to-one mapped to a VNIC-NP so that the bridging overhead can be avoided.

3.2 OpenVZ Virtualization on PEN

OpenVZ is free, open source software utilized to achieve the virtualization features provided by the PEN. The OpenVZ software provides container based virtualization. Each container is a secure, isolated virtual machine which exists on the same server. The isolated nature of each container allows for greater utilization of host resources while preventing application conflicts across virtual machines. Each container functions as a standalone machine, which means that a container can be rebooted, started, or terminated without affecting other live virtual containers. When logged into a virtual container all aspects appear as a standard Linux distribution – containers possess root access, users, memory, processes, IP addresses, etc.

For a user to activate and use a container on the host machine, the containers must first be created. An administrator must create container templates which detail the characteristics of available virtual machines. These templates will contain such characteristics as operating system, file system, packages, and users. When a container is activated, it is created from a defined template. Presently, when a PEN hosted machine is selected, it is created based upon a Centos 5 template. We have also created virtual router templates using Quagga. Quagga is an open source routing software suite, allowing computers running Linux to act as routers. It provides the functionality for many routing protocols including RIP, OSPF, and BGP. We have set up a container template with Quagga preinstalled so that it is ready for immediate use after a container is created.

The virtual machine may be activated with any number of network interfaces, but these interfaces must be mapped to a physical interface in order to be properly utilized. In order to facilitate the mapping of virtual network interface cards (NICs) the PEN node possesses a NIC containing an Intel network processor, manufactured by Netronome. This NIC possesses four physical ports which are capable of being mapped to 256 virtual ports through the utilization of either the Netronome Flow Driver (NFD) or the Netronome Flow Manager (NFM). The NFD is an API which allows for the development of user applications and kernel modules so that the card may be used as desired. The NFM is a tool provided by Netronome which exemplifies some of the capabilities of the Intel network processor. This software is used for assigning virtual NICs to the physical card.

In the allocation sequence of a virtual machine the code will first select a template from which to boot a container. The memory space will be allocated, the file system will be established, and the container will activate. Once the container is live, the scripts use the NFM to assign a physical interface on the Netronome card to each of the virtual interfaces within the container. As each interface is assigned to a virtual port, an IP address is granted which correlates with the protoGENI experiment responsible for the creation of the container.

When the container is no longer needed, the de-allocation sequence will call for its destruction. This sequence is the reverse of the allocation sequence. First, the scripts will use the NFM to remove each virtual interface and destroy the connections to the physical ports. Once these connections have been terminated, OpenVZ commands are issued to stop the container. This is the equivalent of shutting down the machine. After the container has been stopped, the space it occupied is destroyed, thus completing the de-allocation sequence.

3.3 Virtual NIC based on NP Acceleration Card

The essence of our PEN prototype is the virtual NICs provided by the NP acceleration card. The acceleration card presents to the host OS up to 256 virtual NICs, which operate on top of the four 1Gbps Ethernet ports on the card. The virtual NICs are made possible with the NP side microcode and the host side device drivers. Two implementation options are available for creating virtual NICs: closed-source Netronome NFM software and the open-source virtual NIC drivers we have developed.

NFM is licensed software provided by Netronome, the vendor of the NP acceleration card. The software package contains NP-side executable (a .uof file), host-side device drivers, and tools to map the virtual NICs to the physical Ethernet ports on the card. While the NFM software is running, VNICs can be added dynamically with the "nfmvnic" command. This command takes command line inputs that specify how many VNICs are going to be created (or deleted), and which physical port each will be mapped to. The new interfaces are given unique sequential MAC addresses, and are named "nsnetX" (where X is a unique number beginning at 0). Once these interfaces are created, they can be configured in the same manner as any real interface, using "ifconfig" to assign IP addresses and netmasks. Furthermore, these interfaces can be assigned to specific VE's. OpenVZ allows each VE to have exclusive control over some resources, including network interface cards. In this manner, one of the VNICs created by NFM can be immediately assigned to a specific VE, so that no other VE (or the HN itself) can use it. NFM supports up to 256 virtual NICs and measurement capabilities on the NP card. For example, one of the tools provided with NFM can report the number of packets/bytes received per port and the distribution of packets types (TCP, UDP, etc.)

Although the NFM software meets some of our requirements (e.g. virtual NICs and basic measurement), it does not provide the extensibility for us to augment new functionalities on the NP card. In the near future, we plan to add clock synchronization and packet capture functions at the NIC level. The closed-source nature of NFM prevents us from adding such functionalities.

We have developed open-source virtual NIC software for the Netronome NP card. Our software consists of two parts: the device driver on the host and the microcode programs on the network processor. The device driver on the host is designed as an Ethernet device driver for Linux. The device driver talks to the microcode program executed on the NP card through NFD, low-level open-source software provided by Netronome.

NFD software provides basic message passing functions between the Netronome card and the host over PCIe bus. The software has an open source NP-side sample program that receives packets from interfaces, transmits packets to outgoing interfaces and optionally forwards the packets to the host side CPUs. This sample program is important to our research and development as we can reprogram the NP on the card by compiling the sample program with a free software development kit. We leverage the message passing APIs of NFD to develop the host-side device driver and the NP-side program. The open-source software we implement currently supports four virtual NICs and we are extending it to support up to 256 virtual NICs.

4. Conclusion

We have designed a prototype programmable edge node and installed it in CANS lab at UMass Lowell campus. OpenVZ virtualization software runs on the PEN to provide virtual container for the users. We instantiate virtual NICs for these containers based on the support of Netronome NP acceleration card. Basic measurement capability is

supported at the NIC level. Such a PEN is deployable to the GENI aggregates. The open source software we develop is made available at the project website and will enable us and the early adopters to extend the functionalities on the PEN.

5. Further Readings

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

For information about the Netronome NFE-i8000 NP acceleration card and NFM/NFD software, please refer to http://www.netronome.com/

For information about the internals of PEN, please refer to the document entitled "*The Design of a Programmable Edge Node for GENI*", available at the project website.

For information about how the PEN fits in GENI, please refer to the document entitled "*The Integration of Programmable Edge Node with ProtoGENI Control Framework*", available at the project website.