Virtualized Reconfigurable Hardware Resources in the SAVI Testbed

Stuart Byma(✉), Hadi Bannazadeh, Alberto Leon-Garcia, J. Gregory Steffan, and Paul Chow

University of Toronto, Toronto, Ontario, Canada
{bymastua,steffan,pc}@eecg.toronto.edu,
{hadi.bannazadeh,alberto.leongarcia}@utoronto.ca

Abstract. Reconfigurable hardware can allow acceleration of compute intensive tasks, provide line-rate packet processing capabilities, and in short, expand the range of experiments and applications that can be run on a testbed. Few large-scale networking testbeds have made any concerted effort towards the inclusion of virtualized reconfigurable devices, such as FPGAs, into their systems as allocatable resources. This changes with the SAVI testbed. In this paper, we present the current state of heterogeneous, reconfigurable hardware resources in the SAVI testbed, as well as how they are virtualized and facilitated to end-users through the Control and Management system. In addition, we present several use cases that show how beneficial these resources can be, including an in-network multicore multithreaded network processor programmable in C, and network-connected custom hardware modules.

Keywords: Testbeds · Reconfigurable hardware · Virtualization

1 Introduction

There are now quite a number of large-scale research testbeds in use or being developed [1]. Many of these have architectures that virtualize resources in some fashion, allowing researchers and users to have their own private subset of testbed resources. Often absent from these resources however, are reconfigurable devices, such as *Field Programmable Gate Arrays (FPGAs)*. It is highly desirable to incorporate these devices into testbeds, as there are many compute-intensive and high-speed processing tasks that they excel at. Many testbeds are focused on Future Internet or other networking themes where FPGAs can be very useful – they are capable of line-rate packet processing, being used in commercial equipment like routers and switches all the time, and their programmability allows the researcher to tailor their design to whatever paradigm or protocol necessary for their experiment.

In this paper, we introduce the different types of virtualized reconfigurable resources in the testbed of the *Smart Applications on Virtual Infrastructure (SAVI)* network [2,3]. The purpose of the SAVI network is to investigate future application platforms that rely on virtualized, flexible infrastructure. This infrastructure

V.C.M. Leung et al. (Eds.): TridentCom 2014, LNICST 137, pp. 54–64, 2014.
DOI: 10.1007/978-3-319-13326-3_6

is able to deploy large-scale, distributed systems that utilize the resources (wireless and wired networks, computing, devices) to deliver applications.

The SAVI testbed is a realization of this infrastructure, and is meant to be a testing ground for SAVI research in application platforms and Future Internet. The SAVI testbed implements a controlled and managed multi-tier cloud, consisting of Core and Smart Edge nodes connected by virtual networks over a large geographic region in Canada.

We describe in this paper how the heterogeneous resources in the Smart Edge nodes are enabled, and then present the different types of reconfigurable hardware resources in the SAVI testbed, and how each is controlled and managed. We present several use cases for these resources, showing how they allow researchers to run experiments and applications that were previously impossible, and how virtualized reconfigurable resources can easily outperform applications run in software on Virtual Machines (VMs).

We organize this paper as follows. Section 2 explores related and prior work in the research testbed field, dealing specifically with reconfigurable resources. In Section 3, we describe the SAVI testbed, it's architecture and capabilities, and examine it's software-defined infrastructure manager called *Janus*, describing how the system uses modifications to OpenStack to enable heterogeneous, non-Virtual Machine resources. Section 4 describes the different virtualized reconfigurable hardware resources in the testbed and how they are enabled and managed. In Section 5 we examine some use cases for these resources in the SAVI testbed, in particular network-connected custom hardware accelerators, and FPGA-based network processors. Section 6 looks at future work we hope to accomplish, and Section 7 concludes the paper.

2 Related Work

There has not been a significant amount of work on including reconfigurable hardware and FPGAs into research testbeds. The NetFPGA [4] has seen some use within GENI [5,6] and Internet2 [7,8], however it is unclear as to whether these are allocatable to end users as resources that are fully programmable and managed on the same level as VMs. The precursor to the SAVI testbed, *Virtualized Application Networking Infrastructure (VANI)* [9], integrated bare-metal servers with FPGA cards as resources, and the SAVI testbed builds on what began with VANI.

As far as we are aware, the SAVI testbed represents the first major push towards inclusion of reconfigurable hardware as resources on par with VMs, managed under the same system.

3 SAVI Testbed Control and Management

The SAVI testbed consists of several main components: Core data center nodes with traditional cloud computing resources (VMs, storage, network), Smart Edge nodes that complement traditional cloud resources with heterogeneous resources (bare-metal servers, FPGAs, GPUs), Access Nodes that provide wireless connectivity, the SAVI testbed network that interconnects all components, and a Control Center to orchestrate applications and experiments.

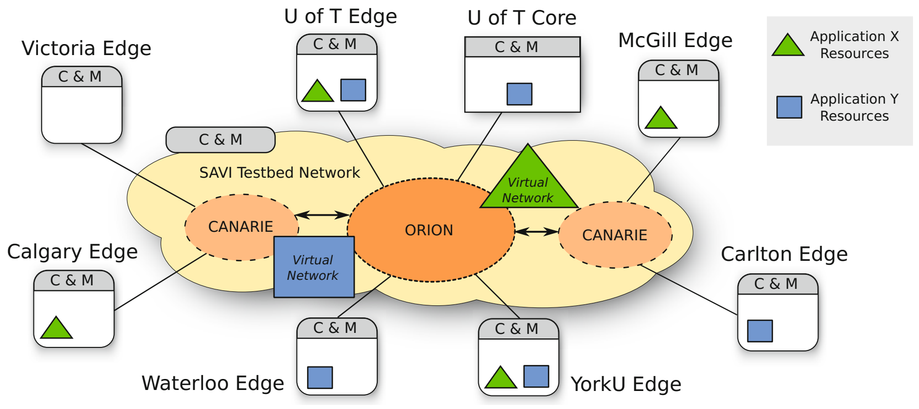


Fig. 1. The SAVI testbed. The ORION [10] and CANARIE [11] networks connect all components over a large geographic area of Canada. Experiments and applications can leverage virtualized resources from anywhere in the testbed.

Figure 1 shows the current state of the SAVI testbed. The components of the testbed are architected into a Control and Management (C & M) plane, and an Applications and Experiments plane. Our discussion will mostly be limited to the C & M plane, as we wish to describe how resources are controlled and managed in the system. We will also mainly limit our discussion to the Smart Edge node, as this is the component that contains the heterogeneous resources. A detailed overview of the entire SAVI testbed system is available in [2].

Figure 2 shows a diagram of the SAVI testbed Smart Edge. Resources in the system are virtualized using OpenStack [12], an open source cloud computing framework. OpenStack management forms the Smart Edge C & M plane in conjunction with the Software-Defined Infrastructure manager, called *Janus*. *Janus* offloads certain tasks from OpenStack, such as network control and resource scheduling, and also performs configuration management and orchestration of the testbed's OpenFlow-based *Software-Defined Network* (SDN). *Janus* uses *FlowVisor* (FV) to virtualize the network into slices, and users can run their own OpenFlow controller to manage their own private network slice. C & M services are all reachable through RESTful [13] APIs. OpenStack *Keystone* and *Glance* provide authentication and a global image registry respectively.

Of particular interest to this paper in Figure 2 is the *Nova* component of OpenStack, which is the part that allocates resources. The standard *Nova* only supports processor virtualization, where Virtual Machines (VMs) are booted on top of hypervisors that abstract away the physical hardware. The vision of the Smart Edge however, incorporates heterogeneous resources in addition to VMs. Thus *Nova* in the SAVI testbed is extended to enable it to manage these new resources.

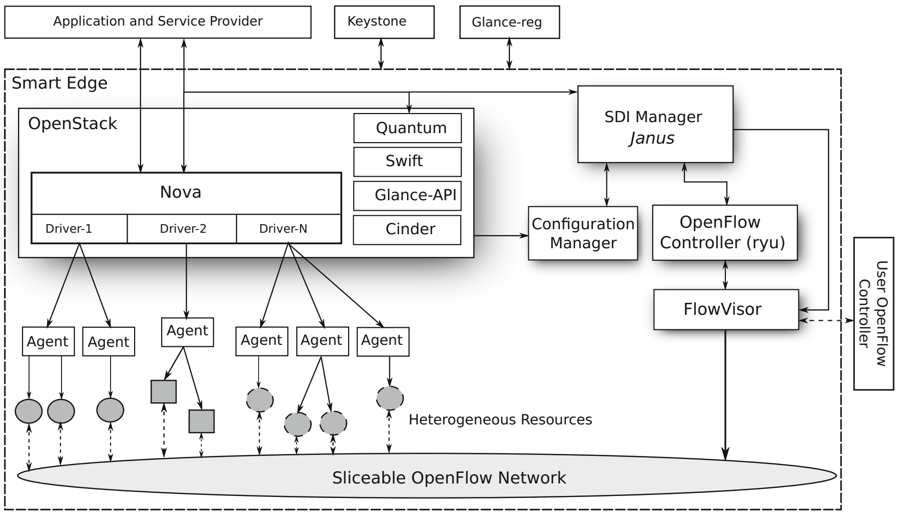


Fig. 2. The SAVI testbed Smart Edge node

3.1 Enabling Heterogeneous Resources

For OpenStack to manage different types of resources, they must all *appear* homogeneous in nature. To accomplish this, we use a *Driver-Agent* system. A driver for any resource implements required OpenStack management API methods, such as *boot, reboot, start, stop* and *release*. The driver then communicates these OpenStack management commands to an Agent, which carries them out directly on the resource, via a hypervisor or otherwise. In this fashion, OpenStack can manage all resources through the same interface. Figure 3 shows a diagram of the Driver-Agent system.

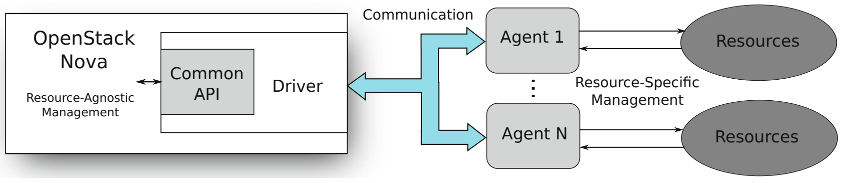


Fig. 3. The *Driver-Agent* abstraction used in the SAVI testbed OpenStack system

If a user desires to allocate a resource, they need to be able to specify what resource type they want – we extend the OpenStack notion of resource *flavor* to enable this. Usually, resource flavor refers to the number of virtual processors and amount of RAM to allocate to a VM. Here we extend *flavor* to also include resource *type*. The SAVI testbed currently has several of these additional resource types including GPUs, bare-metal servers, and reconfigurable hardware.

To be made aware of their existence, OpenStack must have resource references placed in its database – one for each allocatable resource. This is done using the *nova-manage* tool. The resource database entry includes the address of the Agent that provides the resource, a type name that can be associated with a flavor, and how many physical network interfaces the resource has. A flavor is created for each unique resource type.

3.2 Heterogeneous Resource Boot Sequence

When a boot command is received by OpenStack, it resolves which resource type is required from the flavor specified by the user. Scheduling is the process of figuring out which Agent (there may be multiple for one resource type) will host this particular resource instance – in the SAVI testbed, this may be offloaded to *Janus*. *Janus* also takes care of networking for the resource – some heterogeneous resources in the testbed can have several network interfaces, and *Janus* allows users to connect each interface to a different network, even their own virtual network slice with their own OpenFlow controller. Eventually OpenStack calls the *boot* API method in the driver associated with the required resource type and passes several parameters: the address of the Agent, the user-specified image, and the network information generated by *Janus* that belongs to the resource. The Agent takes the required steps to boot the resource and set up network connectivity, whatever they may be for the particular type, and acknowledges the driver request. A reference for the resource is then returned to the user.

4 Reconfigurable Devices as Resources

In the SAVI testbed, we use the *Driver-Agent* method to enable FPGA-based reconfigurable hardware resources as well. The following subsections describe the different FPGA resources available in the SAVI testbed.

4.1 BEE2 Board FPGAs

The SAVI testbed has a number of BEE2 systems [14]. The BEE2 is equipped with five Xilinx FPGAs, with one used to control the others. In the testbed, an Agent runs on an embedded system on the control FPGA, and manages the other FPGAs as resources that can be allocated. Each FPGA resource has four 10G-capable CX4 interfaces that connect to the testbed SDN, allowing the user to send and receive data from their hardware on the FPGA.

Since the user simply gets the entire device as a resource, they are responsible for designing and compiling their hardware using vendor tools, ensuring that their hardware ports match the correct pin locations on the BEE2, and ensuring that the hardware will function correctly. Once they generate a bitstream file for programming the FPGA, it is uploaded through the OpenStack *Glance* API as an image.

Note that we are again extending the definition of a concept in OpenStack. Normally, an "image" refers only to an Operating System (OS) image, however *Glance* allows any file type to be uploaded as an image. Therefore, for a BEE2 FPGA resource, the image will be a bitstream generated by the FPGA tools. For the BEE2 resource, the Agent will receive this image from the OpenStack controller via the driver, and simply configures it onto an unused FPGA. OpenStack sees the FPGA as any other resource thanks to the *Driver-Agent* abstraction, and the user can now make use of custom hardware acceleration in the SAVI testbed.

4.2 PCIe-Based FPGA Cards

To increase the range of different FPGA applications available to researchers, it is useful to have FPGAs closely coupled to processors so that the reconfigurable hardware can accelerate compute-intensive portions of software. The SAVI testbed provides several PCI-Express-based FPGA boards connected to physical servers: The NetFPGA, the NetFPGA10G [4] and the DE5Net [15]. The boards have varying FPGA device sizes and on-board memory, but have in common four network interfaces that are connected to the testbed SDN. The NetFPGA has four 1G Ethernet ports, while the NetFPGA10G and DE5Net have four 10G Ethernet ports. A researcher can now design custom hardware that can accelerate software tasks, provide line-rate packet processing, or a combination of both.

In addition to these boards, the testbed also contains MiniBEE [16] resources. The MiniBEE contains a conventional processor and an on-board FPGA connected through PCIe. It also has 10G network interfaces, a large amount of memory and an expansion port for additional FPGA peripherals.

Since the PCIe boards are required to be mounted inside physical servers, the SAVI testbed provides the server itself with the FPGA card attached as a resource. In the case of the MiniBEE, the entire system is also offered as a resource.

4.3 Fully Virtualized Hardware

With the BEE2 and PCIe-based SAVI testbed resources, the FPGAs are not as fully virtualized as they could be – OpenStack manages the resource, but a user still gets the entire physical device. This may not be quite as scalable or flexible as a fully virtualized approach, and also may not make full use of large FPGA's reconfigurable fabric. Therefore, we wish to virtualize FPGAs to a greater extent, in order to more closely match conventional cloud computing models. We have developed in the SAVI testbed a system that uses FPGA *partial reconfiguration* (PR), a technique to reconfigure only a portion of an FPGA at a time, to split the device into several virtual pieces [17]. Another custom driver and Agent allows OpenStack to manage each of these PR regions as a resource. We call these regions *Virtualized FPGA Resources*. Some hardware on the FPGA that is not partially reconfigured (called the *static logic*) forms an embedded system that interacts with the Agent, facilitating safe partial reconfiguration and setting up VFR networking. Buffering and arbitration in the static logic results in a three-cycle latency penalty for packet data streams into the VFRs, however throughput is only affected by a one-cycle stall per packet.

Figure 4 shows a diagram of this system. The system is implemented on one or several of the NetFPGA10G resources, showing how one resource in SAVI

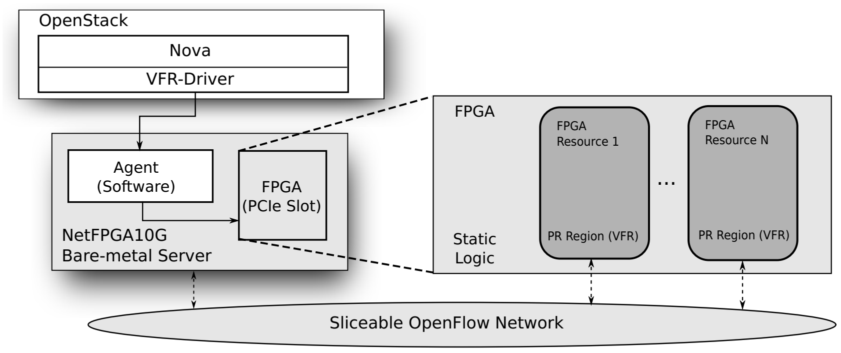


Fig. 4. Virtualized FPGA Resources in the SAVI testbed

can be used to provide additional, new resource types. Each VFR is connected through an arbiter in the static logic to the board's 10Gb Ethernet ports, and thus the testbed SDN. Researchers can make use of template Verilog HDL files and a script-based compile system to generate custom hardware that matches the interfaces to the VFRs and generate images of this hardware that can be uploaded via *Glance* and booted through OpenStack. The VFRs can be booted very quickly relative to VMs, taking around 2.6 seconds on average to get to a state where they are fully configured and able to process data. Because of this, VFR-based systems can scale extremely rapidly.

The system also significantly simplifies hardware design for the user. All chip level I/O, Ethernet interfacing and memory interfacing is done in the static logic of the system. The static logic therefore removes several complex, difficult integration tasks for users, and leaves them with a few standard, well-defined interfaces with which to build their system. This also makes it much easier to use tools like High-Level Synthesis instead of HDL design entry. Design and test time is greatly reduced, and researchers can set up prototypes and experiments much more quickly.

5 Use Cases

Researchers using the SAVI testbed now have access to FPGA-based hardware acceleration, either in-network, CPU-coupled over PCIe, or a combination of both. In this section we describe two use cases for the FPGA resources in the testbed.

5.1 A Multicore, Multithreaded Network Processor

NetThreads10G [18] is a port and expansion of the original NetThreads system by Martin Labrecque et al. [19]. Designed for the NetFPGA10G, it is a soft multicore, multithreaded network processor. Figure 5 shows a diagram of the system. Each of the four cores implements a modified MIPS instruction set, has a private instruction cache, and executes four-way multithreading. The four cores share access to a data cache, and a 20 packet capacity buffer, which is filled with incoming packets by hardware connected to the NetFPGA10G's 10Gb Ethernet stream interfaces. A set of 16 hardware locks enables safe sharing of data between threads, and the NetFPGA10G on-board RLDRAM provides 64MB of system memory.

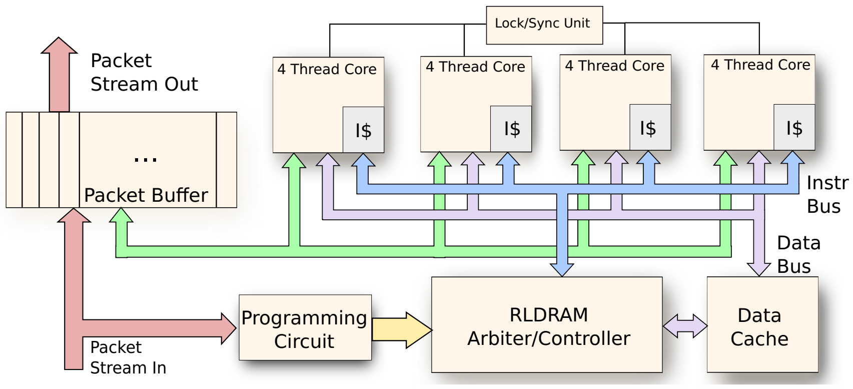


Fig. 5. The NetThreads architecture

The NetThreads framework also includes a MIPS gcc cross-compiler, and library providing rudimentary functions to read and write the packet buffer, allocate memory, and get and set the hardware locks for parallel programming. The NetThreads10G hardware contains a dedicated programming circuit that operates over Ethernet, meaning the system is programmable from anywhere in the testbed network.

The SAVI testbed researcher now has access to a gigabit-line-rate network processor that they can program easily in C – no hardware design necessary. Using the system is simple – since the hardware is already synthesized, placed, and routed, a user need only use the OpenStack API to allocate a NetFPGA10G resource and then program the NetThreads bitstream onto the device. A programmer application takes the output files of the cross-compiler and sends them over the network to the NetThreads system, whose programming circuit loads the software into memory and starts the processor system that can run many applications at line-rate.

5.2 A VFR-Based Load Balancer

Load balancing is an important part of large-scale cloud applications. In this section we demonstrate how an arbitrary protocol load balancer [17] can be implemented using the SAVI testbed's *Virtualized FPGA Resources.*

The load balancer is designed using a template Verilog file whose ports match those defined by the VFR system static logic. The hardware is designed to match a hypothetical protocol running on top of UDP, and distribute incoming packets to a number of servers. Servers send update packets to the load balancer, which tracks available server addresses in a memory. The balancer cycles through this memory as packets arrive, sending them to servers in a round-robin fashion. Figure 6 shows the VFR system compile and boot sequence.

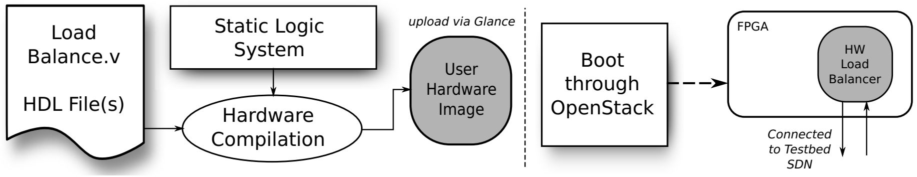


Fig. 6. Compiling and using VFRs. Hardware Design Language (HDL) files are compiled in conjunction with the static logic system using FPGA vendor tools. The generated image containing FPGA programming files can be "booted" through OpenStack, and the user's hardware is partially reconfigured into a VFR on the fly.

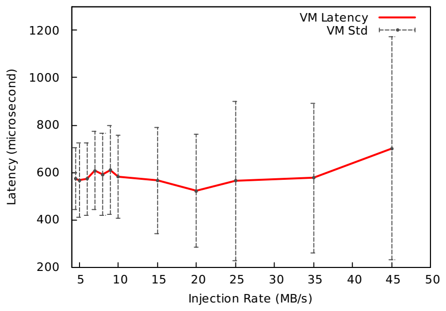
We compare the hardware load balancer to a software implementation run on a VM in the SAVI testbed. A client VM sends packets to the load balancer to be distributed amongst servers, and the servers send a direct response back to the client after receiving a packet from the load balancer. The round trip time is measured at the client and averaged over 10000 packets. Other VMs are used to inject additional traffic so that we can measure the approximate throughput capability of the software and hardware load balancers. A VM load balancer can only handle up to around 25MB/s before dropping packets and performing unpredictably. The VFR load balancer could handle over 100MB/s, even with the presence of the static logic virtualization layer, and did not drop a single packet. Figure 7 shows latency versus additional injection rate for software and hardware load balancers. Since each point is an average of 10000 packets, we show standard deviation as well.

This example shows how users of the SAVI testbed can offload network-based processing to VFRs and get a substantial performance gain through the simplified hardware design flow provided by the virtualization system.

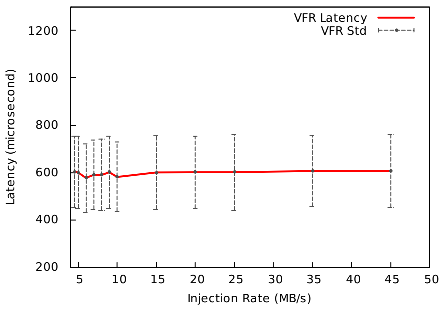
6 Future Work

There is a significant amount of future work to be done with the reconfigurable resources in the SAVI testbed. We plan to continue adding to the number of physical FPGA resources in the system, and expand these resources to all Smart Edge nodes in the testbed.

We will also continue exploring the concept of *Virtualized FPGA Resources*, to see how closely they can be fit within the cloud computing model. This involves



(a)



(b)

Fig. 7. (a) Latency through VM Load Balancer. (b) Latency through VFR hardware load balancer.

making them capable of more VM-like tasks, such as migration among physical machines. We also plan to investigate methods of chaining VFRs and other resources over the network at the system level – creating heterogeneous processing chains for arbitrary tasks.

7 Conclusion

We have presented the different types of reconfigurable resources in the SAVI testbed, and how they are enabled by the testbed's *Driver-Agent* abstraction for heterogeneous resources. Researchers using the SAVI testbed can use familiar management commands to access network-coupled and CPU-coupled FPGAs as cloud resources, and make use of either predefined or custom-designed hardware. These reconfigurable hardware resources will enable a new range of applications and experiments that were previously unavailable in the SAVI testbed, and the networking testbed community at large.

References

1. Pan, J., Paul, S., Jain, R.: A Survey of the Research on Future Internet Architectures. Communications Magazine, IEEE **49**(7), 26–36 (2011)
2. Joon-Myung K., Bannazadeh, H., Rahimi, H., Lin, T., Faraji, M., Leon-Garcia, A.: Software-Defined Infrastructure and the Future Central Office. In: IEEE International Conference on Communications Workshops (ICC), pp. 225–229 (2013)
3. Smit, M., Ng, J., Litoiu, M., Iszali, G., Leon-Garcia, A.: Smart Applications on Virtual Infrastructure. In: Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research, CASCON 2011, pp. 381–381, Riverton (2011)
4. NetFPGA. NetFPGA 10G (2014). http://netfpga.org/
5. GENI. Global Environment for Networking Innovations (GENI) Project (2014). http://geni.net/

6. Emulab. ProtoGENI Nodes (2014). https://wiki.emulab.net/wiki/pgeniNodes
7. Internet2 (2014). http://www.internet2.edu/
8. Lockwood, J.: NetFPGA Update at GEC4. Presented at NSF GENI Engineering Conference (2009)
9. Redmond, K., Bannazadeh, H., Chow, P., Leon-Garcia, A.: Development of a Virtualized Application Networking Infrastructure Node. In: IEEE GLOBECOM Workshops, pp. 1–6 (2009)
10. ORION. Ontario Research and Innovation Optical Network (2014). http://www.orion.on.ca/
11. CANARIE. Canada's Advanced Research and Innovation Network (2014). http://www.canarie.ca/
12. OpenStack (2013). http://www.openstack.org/
13. Fielding, R.T.: REST: Architectural Styles and the Design of Network-Based Software Architectures. PhD thesis, University of California, Irvine (2000)
14. Chang, C., Wawrzynek, J., Brodersen, R.W.: BEE2: A High-End Reconfigurable Computing System. Design Test of Computers, IEEE **22**(2), 114–125 (2005)
15. Terasic Technologies Inc. DE5Net (2013). http://de5-net.terasic.com/
16. BEECube Inc. miniBEE - Research in a Box (2014). http://www.beecube.com/products/miniBEE.asp
17. Byma, S., Steffan, J.G., Bannazadeh, H., Leon-Garcia, A., Chow, P.: FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack. In: 22nd Internation Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE (2014)
18. Byma, S., Steffan, J.G., Chow, P.: NetThreads-10G: Software Packet Processing on NetFPGA-10G in a Virtualized Networking Environment Demonstration Abstract. In: 23rd International Conference on Field Programmable Logic and Applications (FPL). IEEE (2013)
19. Labrecque, M., Steffan, J.G., Salmon, G., Ghobadi, M., Ganjali, Y.: NetThreads: Programming NetFPGA with Threaded Software. In: NetFPGA Developers Workshop, vol. 9 (2009)