Using the NetFPGA in the Open Network Laboratory

Charlie Wiseman, Jonathan Turner, John DeHart, Jyoti Parwatikar,
Ken Wong, David Zar

Department of Computer Science and Engineering
Washington University in St. Louis
{cgw1,jst,jdd,jp,kenw,dzar}@arl.wustl.edu

ABSTRACT

The Open Network Laboratory is an Internet-accessible network testbed that provides access to a large set of heterogeneous networking resources for research and educational pursuits. Those resources now include the NetFPGA. ONL makes it easy for NetFPGA users to integrate multiple NetFPGAs into heterogeneous experimental networks, using a simple graphical user interface. The testbed software infrastructure automatically manages all of the details, including mapping the user's topology to actual hardware and time-sharing of resources via a standard reservation mechanism. The inclusion of NetFPGAs into the testbed allows users just getting started with NetFPGAs to conduct interesting research quickly without the need to set up and manage the NetFPGAs themselves. For more experienced users, the testbed provides an easy path to larger and more diverse experimental configurations.

1. INTRODUCTION

Networking and systems researchers have come to rely on a wide range of tools and platforms to conduct their experiments. This includes specific types of technology as well as simulation and testbed environments. One such technology that has recently seen wide adoption is the NetFPGA [11][9][15], which is a relatively inexpensive reprogrammable hardware platform. NetFPGAs are now also available as part of the Open Network Laboratory (ONL) testbed [6].

ONL is an Internet-accessible network testbed which features resources based on a variety of networking technologies. Users configure arbitrary network topologies with a simple GUI tool and then run interactive experiments using that topology. The user is granted sole control of the physical components that make up their topology for the duration of their experiment. Sharing is accomplished via a standard reservation mechanism. ONL is open to all researchers and educators (at no cost, of course).

There are advantages to using NetFPGAs in ONL for the entire range of NetFPGA users. Clearly, it opens the way for NetFPGA use by those who do not have the means to acquire, set up, and manage their own NetFPGA installations. This barrier is already low for many users due to the low cost of the platform and substantial available documentation. However, the ongoing overheads to manage many NetFPGAs can be high in certain contexts. One example is a course where many students have to coordinate to share a small number of NetFPGAs. ONL enables this sharing naturally with minimal additional overhead to both the educator and the students. More generally, ONL removes the management overhead when there is substantial sharing of NetFPGA resources.

For existing NetFPGA users, ONL provides an easy means to experiment with NetFPGAs in a variety of elaborate configurations. ONL currently has nearly 20 routers and over 100 PCs in addition to 6 NetFPGAs. This allows researchers to build experimental topologies that reflect many real world scenarios. Moreover, it only takes a few minutes with the GUI to configure an entirely different topology. It would require a substantial investment of money and time to reproduce this capability in a local environment. The ability to quickly run experiments over diverse network configurations is particularly useful in conjunction with the NetFPGA because of its highly flexible nature.

The rest of the paper is organized as follows. Section 2 provides background information about ONL including brief descriptions of the other resources currently available in the testbed. Section 3 describes how to use NetFPGAs as part of ONL along with three examples. Related work is covered in Section 4, and some future work is discussed in Section 5. Finally, Section 6 contains closing thoughts.

2. ONL

The Open Network Laboratory testbed provides a diverse set of user-configurable resources for network experimentation. This includes both research and educational contexts [23]. The Remote Laboratory Interface (RLI) is a simple Java GUI run on the user's computer to configure experimental topologies and monitor network traffic within the topology in real time. An example experiment consisting of a small dumbbell topology is shown in Figure 1. The top of the figure is the topology configuration window and the bottom is a real time chart with three data sets showing bandwidth at various points in the topology.

Users first configure their network topology with the RLI by selecting various components from the menus and linking them together. Once the topology is ready, a reservation is made to ensure that the required resources will be available for the duration of the user's experiment. Specifically, the reservation request consists of a length and a range of times that the user chooses during which they could run their experiment. The ONL software will either grant the reservation for a specific interval during the given range of times, or reject the reservation if there are not enough resources available during that time. The resource availability for the near future can be found on the ONL website [16] to aid in finding open time slots. Also note that each user may have many outstanding reservations. Users are guaran-

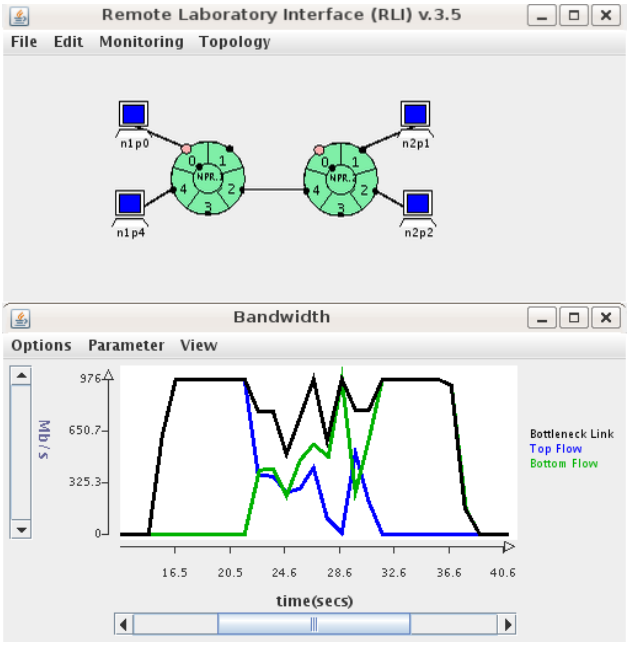

Figure 1: Example ONL configuration with real time charts.

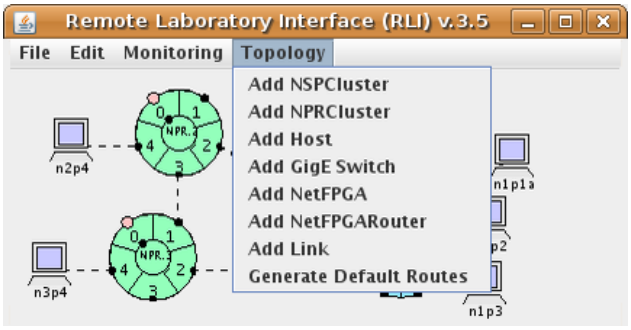

Figure 2: RLI Topology Menu.

teed to be able to run their experiment without interference from any other experiments for their reservation duration. When the time for the reservation arrives, the RLI is used to start the experiment as well as to monitor and configure the resources in the experiment topology.

Before moving on the NetFPGA, a brief description is given of the other ONL resources.

The Network Services Platform (NSP) [4] is a custom-built IP router designed in the style of larger, scalable router platforms. Each of the eight ports is an independent entity and consists of a 1 Gb/s interface, a Field Programmable Port Extender (FPX) [12], and a Smart Port Card (SPC) [7]. A 2 Gb/s cell switch connects the ports together. The FPX handles all the common packet processing tasks such as classification, queueing, and forwarding. The SPC contains a general purpose processor which is used to run *plugins*. Plugins are user developed code modules which can be dynamically loaded on to the router ports to support new or specialized packet processing. There are currently 4 NSPs in ONL.

Next is the Network Processor-based Router (NPR) [22], which is an IP router built on Intel IXP 2800s [1]. The NPR has five 1 Gb/s ports and a Ternary Content Addressable Memory (TCAM) which is used to store routes and packet filters. User written plugins are supported as in the NSP. In the case of the NPR, five of the sixteen MicroEngine cores on the IXP are dedicated to running plugin code and the user has the ability to dynamically map different plugins to different MicroEngines. Packet filters are used to direct specific packet flows to the plugins, and the plugin can forward the packet to outbound queues, to another plugin, back to the classification engine, or the packet can be dropped. There are currently 14 NPRs in ONL.

Standard Linux machines are used as traffic sources and sinks. Every host has two network interfaces: a 1 Gb/s data interface which is used in the experimental topology, and a separate management interface. Once a user's experiment is active, they are granted SSH access through the management interface to each host in their topology. The hosts are all installed with a range of utilities and traffic generators, and users are welcome to install their own software in their user directory (which is shared across all the hosts). The only limitation in the environment is that users are not given a root shell, so any specific actions requiring root privileges are granted through normal sudo mechanisms. There are currently over 100 hosts in ONL.

All of the ONL resources are indirectly connected through a set of configuration switches. Virtual Local Area Networks (VLANs) are used extensively in the switches to enforce isolation among different experiments. All of the configuration of these switches takes place automatically and invisibly from a user's perspective. The reservation system is responsible for ensuring that the configuration switches have sufficient capacity to guarantee that no experiment could ever interfere with any other experiment. VLANs also provide a way for ONL to support standard Ethernet switches in experimental topologies.

3. USING NETFPGAS IN ONL

The are currently 6 NetFPGAs available in ONL, each installed in a separate Linux host. The hosts have the base NetFPGA software distribution installed, as well as many of the other NetFPGA project distributions. Similar to the standard ONL hosts, users are granted SSH access to the NetFPGA host for each NetFPGA in their experimental topology so that they can run the utilities to load and configure the hardware. There are a few of these utilities that require root privileges to run, so sudo is employed as on the other ONL hosts. After each experiment ends, the hosts are rebooted automatically and the NetFPGAs are prepared for user programming.

Most of the other ONL resources directly support monitoring and configuration in the RLI via software daemons that are tightly coupled with the individual resources. No such daemon exists for the NetFPGA because the interfaces for monitoring and configuration change depending on what is currently running on the hardware. As such, all configuration is done via the utilities and scripts that are already provided with the existing projects. Monitoring can be done in the RLI indirectly by means of a simple mechanism supported by the ONL software. Specifically, the RLI can monitor and plot data that is put into a file on the host (the file

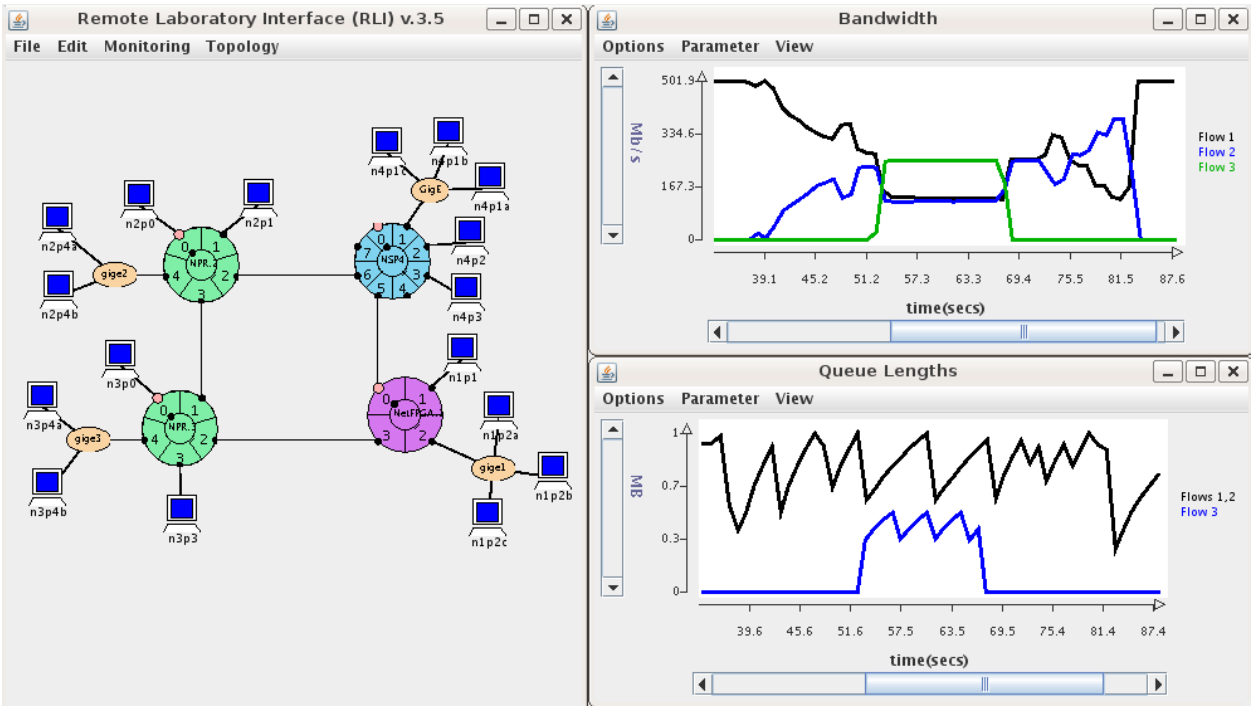

Figure 3: Configuration using many different types of resources with bandwidth and queue length charts.

location is specified in the RLI). Users write scripts to populate these files with whatever data they choose. For the NetFPGA, that will typically mean reading registers that contain packet or byte counts in order to display packet rates, bandwidths, or queue lengths in the RLI.

NetFPGAs are added to a configuration in the RLI similarly to other types of resources, with one exception. The core function of other resources is fixed in nature, i.e., a resource is either a router or something else such as a switch, or a traffic source or sink. This is important when the ONL software configures each resource, as hosts need to know their default gateway, and routers need to know if they are connected to other routers in order to build network routes correctly. NetFPGAs, however, can fill either role and so each NetFPGA is added either as a router node or as a non-router node. A screenshot of the actual RLI menu is shown in Figure 2.

Three examples are given next to illustrate how NetFPGAs are typically used in ONL.

3.1 IP Router

The first example is something that might be seen in an introductory networking course. The topology configuration is shown on the left of Figure 3. The 4 port device at the bottom right of the central "square" is a NetFPGA acting as an IP router. The 8 port device above it is an NSP and the 5 port devices to the left are NPRs. The small ovals connected to each of the routers are Ethernet switches. The others symbols are hosts.

The user configures the NetFPGA by logging in to the PC that hosts the NetFPGA assigned to this experiment, which can be determined by clicking on the NetFPGA in the RLI or by referencing shell variables that are automatically added to the user's ONL shell. For this experiment, the reference IP router bit file is loaded from the command line via the nf2_download utility. Then the software component of the reference router, SCONE, is started. In order to configure the router appropriately, SCONE must be given the Ethernet and IP information for each of the NetFPGA ports as well as the static routes for the router. Normally this information would be sent directly from the RLI to a software daemon that understands how to manage each type of resource. As mentioned above, the NetFPGA interface changes depending on the situation, so the user must build this information manually. We are currently working on ways to ease this burden.

Once the router hardware and software are running, the NetFPGA is acting as a standard IP router and is treated as such by the user. In this example, the user is studying TCP dynamics over a shared bottleneck link. Three TCP flows are sent from hosts in the bottom left of the topology to hosts in the bottom right across the link from the NPR to the NetFPGA. Two of the flows share an outgoing queue at the bottleneck link and the third flow has its own queue. The link capacity is set at 500 Mb/s, the shared queue is 1 MB in size, and the non-shared queue is 500 KB in size. The first flow is a long-lived flow, the second is a medium length flow, and the third is a short flow.

The top right of Figure 3 shows the throughput seen by each flow and the bottom right shows the queue lengths at the bottleneck. The first flow consumes the entire bottleneck capacity in the absence of other traffic. Once the second flow begins, the two attempt to converge to a fair share of the link, but the third flow begins before they reach it. The NPR is using a typical Weighted Deficient Round Robin scheduler. The user has configured the two queues to receive an equal share of the capacity by setting their scheduling quanta to be the same. The result is that the third flow

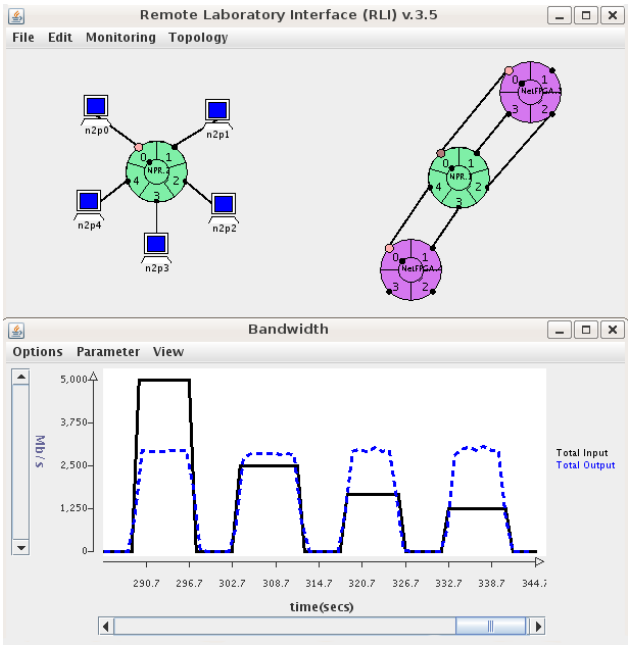

Figure 4: Using NetFPGAs as packet generators to stress test NPR multicast.

gets 250 Mb/s of the 500 Mb/s link because it is the only flow in its queue, and the first two share the remaining 250 Mb/s because they are sharing a queue.

3.2 Traffic Generation

The second example utilizes NetFPGAs as packet generators [5]. In general, it is quite difficult to produce line rate traffic for small packets from general purpose PCs. The ONL solution in the past was to either use many hosts to aggregate traffic on a single link or to use special purpose plugins in the routers that made many copies of each input packet and thus multiply the input traffic up to the line rate. Each option required the use of many resources and produced results that were difficult to control precisely. The NetFPGA packet generator is a much cleaner solution as it can produce line rate traffic on all four ports of any size packet. The packets are specified in the standard PCAP file format.

For this example, NetFPGA packet generators are used to stress test the NPR capabilities. The configuration used is shown in the top of Figure 4. There are actually two separate networks in this configuration. The one on the left is used to generate the packet traces needed for the packet generator, and the one on the right is used to perform the actual stress test. To generate the traces, the hosts send packets via normal socket programs and the user runs tcpdump on the nodes to record the packet streams.

The bottom of Figure 4 shows the results from one particular stress test. The NPR natively supports IP multicast (details in [22]), and this test is meant determine if the packet *fanout* has any effect on peak output rate for minimum size packets. The fanout is simply the replication factor for each incoming packet. Four sets of traffic are sent to the NPR in succession. The first set has a fanout of one, meaning that each input packet is sent out one other port.

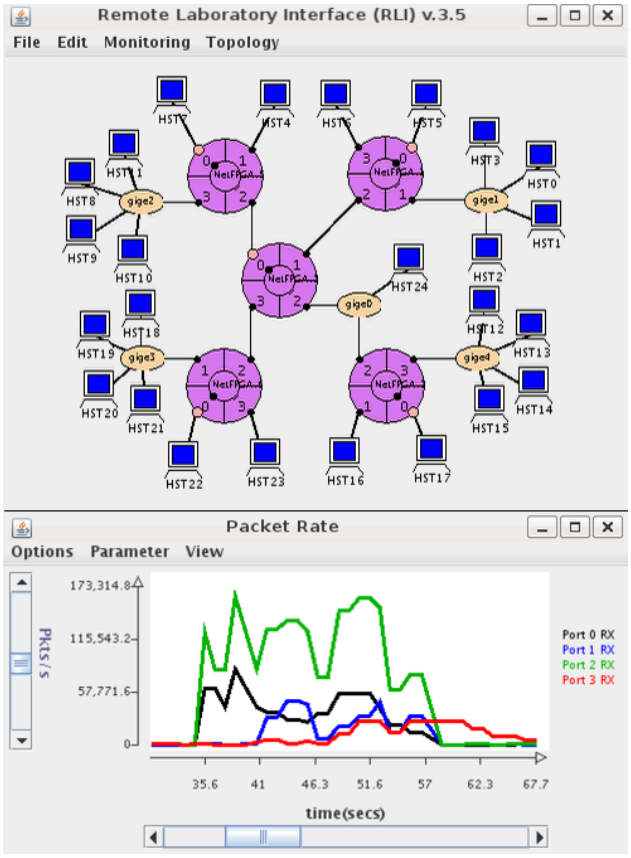

Figure 5: An OpenFlow network with real time charts generated from counters on one OpenFlow switch.

The second set has a fanout of two, the third has a fanout of three, and the last has a fanout of four. In each case the total input rate is set so that the total output rate would be 5 Gb/s if the router dropped no packets, and the input and output rates are each shared equally by all 5 ports. The chart shows the aggregate input rate (solid line) and output rate (dashed line) at the router. In each test, the output rate is around 3 Gb/s, showing that the fanout has little effect on the eventual output rate. Other tests confirm that the peak forwarding rate for minimum size packets is around 3 Gb/s for unicast traffic as well.

3.3 OpenFlow

The last example is of an OpenFlow [13] network. There is an existing NetFPGA OpenFlow switch implementation which supports the Type 0 specification [14]. A description of OpenFlow is out of scope for this paper, but more details can be found on the OpenFlow website [17].

The top of Figure 5 shows an OpenFlow network topology with 5 NetFPGAs acting as OpenFlow switches, 5 Ethernet switches, and 25 hosts. This example demonstrates OpenFlow switches inter-operating with non-OpenFlow (i.e, standard Ethernet) switches. Every OpenFlow network needs a controller that is responsible for interacting with and configuring the OpenFlow switches. The OpenFlow software distribution comes with a basic controller that allows each OpenFlow switch to operate as a normal learning Ether-

net switch. Nox [10] is another OpenFlow controller which is substantially more complex and configurable. Both controllers are available on all ONL hosts so that any ONL host can act as an OpenFlow controller.

The bottom of Figure 5 shows a chart monitoring packet rates through the OpenFlow switch in the middle of the topology when there are many flows traversing the network. The chart is generated as described above by parsing the output of an OpenFlow utility which reads per-flow counters in the switch. The results are placed in a file which allows them to be displayed in the RLI.

4. RELATED WORK

There are a few network testbeds that are generally similar to ONL. Of these, ONL is closest in nature to Emulab [21]. The Emulab testbed has been widely used for research and can often be found as part of the experimental evaluation in papers at top networking conferences. Emulab provides access to a large number of hosts which can be used to emulate many different types of networks. As in ONL, every host has at least two network interfaces. One is used for control and configuration, and the others are used as part of the experimental topology. Also as in ONL, Emulab uses a small number of switches and routers to indirectly connect all the hosts in the testbed. Emulab has many useful features which make it an attractive choice for conducting research. For example, users can configure individual link properties such as delay and drop rate, and the Emulab software automatically adds an additional host which is used as a traffic shaper for that link. According to the Emulab website [8], they have also added six NetFPGAs as testbed resources.

The Emulab software has also been made available so that other groups can use it to run their own testbeds. Many of these Emulab-powered testbeds are currently operating, although most of them are not open to the public. One exception is the DETER testbed [3] that is focused on security research. The control infrastructure is identical to Emulab, but additional software components were added to ensure that users researching security holes and other dangerous exploits remain both contained in the testbed and isolated from other experiments. The Wisconsin Advanced Internet Laboratory (WAIL) [20] is another such testbed that utilizes the Emulab software base. WAIL is unique among the Emulab testbeds in that they have extended the software to support commercial routers as fundamental resources available to researchers.

Another widely used testbed is PlanetLab [18]. At its core, PlanetLab simply consists of a large number of hosts with Internet connections scattered around the globe. At the time of this writing there are over 1000 PlanetLab nodes at over 480 sites. Each PlanetLab node can support multiple concurrent experiments by instantiating one virtual machine on the node for each active experiment. The PlanetLab control software takes user requests to run an experiment on a set of nodes and contacts the individual nodes to add the virtual machines for the user. Researchers use PlanetLab to debug, refine, and deploy their new services and applications in the context of the actual Internet. Unfortunately, PlanetLab's success has resulted in large numbers of active experiments, particularly before major conference deadlines. This often leads to poor performance for any individual experiment because each active experiment is competing for both processor cycles and the limited network bandwidth available on the host.

There have been a few efforts that aim to enhance PlanetLab through the design of new nodes that enable better and more consistent performance while still operating in the PlanetLab context. VINI [2] is at the front of these efforts. VINI nodes are currently deployed at sites in the Internet2, National LambdaRail, and CESNET backbones [19], and have dedicated bandwidth between them. The VINI infrastructure gives researches the ability to deploy protocols and services in a realistic environment in terms of network conditions, routing, and traffic.

With the exception of WAIL, all of these testbeds provide direct access only to hosts. WAIL does provide access to commercial routers, although the website indicates that it is read-only access, i.e., users do not have the ability to configure the routers themselves. In contrast, ONL does provide user-driven control of many different types of networking technology, and we hope to continue increasing the resource diversity in the future.

5. FUTURE WORK

As discussed earlier, most ONL resources are configured primarily through the RLI. Configuration updates are sent to the daemon that manages the resource and knows how to enact the updates. This is not possible with the NetFPGA because the types of configuration change depending on how it is being used. Moreover, two different implementations of the same functionality may export different configuration interfaces. For example, there are already two IP routers available that have different interfaces for adding routes. The result is that all configuration of NetFPGAs is done manually. Simple scripts help to reduce the time spent doing manual configuration, but they can not remove the need for it completely.

An alternative that is being considered is to have different NetFPGA options available in the RLI that correspond to specific NetFPGA implementations. This would allow a user to add, for example, a NetFPGA IP router using the NetFPGA reference router specifically. The RLI would then load the correct bit file, automatically compute and add routes, set up the NetFPGA port addresses, and be able to monitor values in the router directly. In fact, being able to monitor packet counters directly via the RLI would make such a feature valuable even to a project like the reference Ethernet switch which requires no user configuration. The largest obstacle is that there has to be an ONL software daemon for each NetFPGA project which can receive these configuration and monitoring requests. While some of the most common projects could be supported relatively easily by ONL staff, a more scalable approach would allow any user to dynamically provide the portions of code required to actually carry out the requests. In this way, users could leverage the ONL infrastructure even for their own private NetFPGA projects.

6. CONCLUSION

NetFPGAs are a particularly versatile networking platform which can be used as routers, switches, traffic generators, and anything else that NetFPGA developers can build. Deploying such a platform in a network testbed like the Open Network Laboratory benefits both existing testbed

users and NetFPGA users. Six NetFPGAs have been added to the ONL resource set and more could be added if there is sufficient interest and use. This allows those interested in using NetFPGAs to get started immediately without the need to set up and manage the NetFPGAs themselves. Educators in particular are likely to benefit from this. It also provides a means for existing NetFPGA users to test and experiment with their own projects in much broader and more diverse configurations than are likely available in a local lab. In general, we believe that the inclusion of NetFPGAs in ONL will help both projects to reach broader audiences. The ONL source code is available on request.

7. REFERENCES

[1] M. Adiletta, M. Rosenbluth, D. Bernstein, G. Wolrich, and H. Wilkinson. The next generation of intel ixp network processors. *Intel Technology Journal*, 6, 2002.

[2] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In vini veritas: Realistic and controlled network experimentation. In *SIGCOMM '06: Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 3–14, New York, NY, USA, 2006. ACM.

[3] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab. Design, deployment, and use of the deter testbed. In *DETER: Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, pages 1–1, Berkeley, CA, USA, 2007. USENIX Association.

[4] S. Choi, J. Dehart, A. Kantawala, R. Keller, F. Kuhns, J. Lockwood, P. Pappu, J. Parwatikar, W. D. Richard, E. Spitznagel, D. Taylor, J. Turner, and K. Wong. Design of a high performance dynamically extensible router. In *Proceedings of the DARPA Active Networks Conference and Exposition*, May 2002.

[5] G. A. Covington, G. Gibb, J. Lockwood, and N. McKeown. A packet generator on the netfpga platform. In *The 17th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 5–7 April 2009.

[6] J. DeHart, F. Kuhns, J. Parwatikar, J. Turner, C. Wiseman, and K. Wong. The open network laboratory. In *SIGCSE '06: Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, pages 107–111, New York, NY, USA, 2006. ACM.

[7] J. D. DeHart, W. D. Richard, E. W. Spitznagel, and D. E. Taylor. The smart port card: An embedded unix processor architecture for network management and active networking. Technical report, Washington University, August 2001.

[8] Emulab. Emulab website. http://www.emulab.net.

[9] G. Gibb, J. W. Lockwood, J. Naous, P. Hartke, and N. McKeown. Netfpga: Open platform for teaching how to build gigabit-rate network switches and routers. 51(3):364–369, Aug. 2008.

[10] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, 2008.

[11] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. Netfpga–an open platform for gigabit-rate network switching and routing. In *Proc. IEEE International Conference on Microelectronic Systems Education MSE '07*, pages 160–161, 3–4 June 2007.

[12] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor. Reprogrammable network packet processing on the field programmable port extender (fpx). In *FPGA '01: Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*, pages 87–93, New York, NY, USA, 2001. ACM.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. volume 38, pages 69–74, New York, NY, USA, 2008. ACM.

[14] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown. Implementing an openflow switch on the netfpga platform. In *ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 1–9, New York, NY, USA, 2008. ACM.

[15] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. Netfpga: reusable router architecture for experimental research. In *PRESTO '08: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, pages 1–7, New York, NY, USA, 2008. ACM.

[16] ONL. Open network laboratory website. http://onl.wustl.edu.

[17] OpenFlow. Openflow website. http://openflowswitch.org.

[18] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *Proceedings of HotNets–I*, Princeton, New Jersey, October 2002.

[19] VINI. Vini website. http://www.vini-veritas.net.

[20] WAIL. Wisconsin advanced internet laboratory website. http://www.schooner.wail.wisc.edu/.

[21] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI '02: Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pages 255–270, New York, NY, USA, 2002. ACM.

[22] C. Wiseman, J. Turner, M. Becchi, P. Crowley, J. DeHart, M. Haitjema, S. James, F. Kuhns, J. Lu, J. Parwatikar, R. Patney, M. Wilson, K. Wong, and D. Zar. A remotely accessible network processor-based router for network experimentation. In *ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 20–29, New York, NY, USA, 2008. ACM.

[23] K. Wong, T. Wolf, S. Gorinsky, and J. Turner. Teaching experiences with a virtual network laboratory. In *SIGCSE '07: Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, pages 481–485, New York, NY, USA, 2007.