2009-72

Design of an Extensible Network Testbed with Heterogeneous Components

Authors: Charlie Wiseman, Jyoti Parwatikar, Ken Wong, John DeHart, Jonathan Turner

Corresponding Author: wiseman@wustl.edu

Web Page: http://onl.wustl.edu

Abstract: Virtualized network infrastructures are currently deployed in both research and commercial contexts. The complexity of the virtualization layer varies greatly in different deployments, ranging from cloud computing environments, to carrier Ethernet applications using stacked VLANs, to networking testbeds. In all of these cases, there are many users sharing the resources of one provider, where each user expects their resources to be isolated from all other users. Our work in this area is
focused on network testbeds. In particular, we present the design of the latest version of the Open Network Laboratory (ONL) testbed. This redesign generalizes the underlying infrastructure to support resource extensibility and heterogeneity at a fundamental level. New types of resources (e.g., multicore PCs, FPGAs, network processors, etc) can be added to the testbed without modifying any testbed infrastructure software. Resource types can also be extended to support multiple distinct sets of functionality (e.g., an FPGA might act as a router, a switch, or a traffic generator). Moreover, users can dynamically add new resource extensions without any modification to the existing infrastructure.

Type of Report: Other

Design of an Extensible Network Testbed with Heterogeneous Components

Charlie Wiseman, Jyoti Parwatikar, Ken Wong, John DeHart, and Jonathan Turner
Washington University in St. Louis
wiseman@wustl.edu, {jp,kenw,jdd}@arl.wustl.edu, jon.turner@wustl.edu

Abstract

Virtualized network infrastructures are currently deployed in both research and commercial contexts. The complexity of the virtualization layer varies greatly in different deployments, ranging from cloud computing environments, to carrier Ethernet applications using stacked VLANs, to networking testbeds. In all of these cases, there are many users sharing the resources of one provider, where each user expects their resources to be isolated from all other users. Our work in this area is focused on network testbeds. In particular, we present the design of the latest version of the Open Network Laboratory (ONL) testbed. This redesign generalizes the underlying infrastructure to support resource extensibility and heterogeneity at a fundamental level. New types of resources (e.g., multicore PCs, FPGAs, network processors, etc) can be added to the testbed without modifying any testbed infrastructure software. Resource types can also be extended to support multiple distinct sets of functionality (e.g., an FPGA might act as a router, a switch, or a traffic generator). Moreover, users can dynamically add new resource extensions without any modification to the existing infrastructure.

1 Introduction

Virtualized network and system infrastructures are becoming more and more commonplace across a variety of commercial and research deployment contexts. Cloud computing environments use end system virtualization to allow many applications to share computational and storage resources owned by one cloud provider. ISPs use VLAN stacking to provide carrier Ethernet services across the wide area to large enterprise customers. Network testbeds use various types of virtualization to share network nodes and links among many concurrent virtual network experiments running on the shared testbed substrate. Fundamentally, these types of systems use virtualization to share the resources of one provider among many customers or users. More importantly, the resources assigned to each customer are isolated from those of other customers.

Configuration and management in non-virtualized infrastructures is already a difficult task. Supporting virtualization layers adds further complexity whereby providers must configure their networks to meet the isolation and performance requirements of all of their customers. Our work in this area is focused on network testbeds, although the ideas explored here could be applied in other virtualized infrastructure settings.

Researchers have come to rely extensively on testbeds for developing, testing, and evaluating new ideas. Interestingly, nearly all existing testbeds have only PCs as user configurable resources. There are clear benefits to this choice, including simpler testbed management software and user familiarity with the PC platform. However, a testbed with a diverse collection of heterogeneous resources would provide a number of benefits to the research community. Researchers would be able to conduct experiments with a variety of networking devices and technologies in a contained environment. This is also useful for educators who are interested in giving students hands-on experience with networking technologies other than PCs. Natural additions to such a testbed would include network processor systems, FPGAs, and other hardware-based or commercial devices. Access to these types of high performance, reconfigurable nodes would also allow researchers to test new protocols and ideas under realistic conditions while still operating in an isolated setting. PCs alone can certainly be used to emulate the functionality of many different devices, but they can not sustain Internet-scale throughput or match delay characteristics of specialized technology.

Most existing testbeds also provide only a low level interface to users for configuring nodes in their virtual networks. This is a reasonable choice for PC-only testbeds, although including higher level tools would reduce the manual configuration burden. In the case of a testbed

with heterogeneous components, it is necessary to include configuration interfaces that export higher level abstractions to users (e.g., routes, packet filters, queueing parameters, etc) as most users will not be familiar with the native configuration tools for resources such as FPGAs or network processors. Of course, many types of devices can be reprogrammed to support a range of networking functionality. It is thus important to support extensible resources, where users can select among the different possibilities for each device type and even add their own new functionality if necessary.

In this paper, we present the latest version of the Open Network Laboratory (ONL) testbed, which has been extended based on the above observations. Specifically, this work includes four key contributions:

1. We have deployed a testbed which provides the networking community with an experimental facility that supports a variety of heterogeneous resources.

2. We have developed a testbed architecture that is designed explicitly for extensibility. This architecture provides a general framework for configuring and interacting with heterogeneous resources that is easy enough for novices to learn while still supporting complex functionality.

3. We provide a general mechanism for specializing and extending reconfigurable resources to allow higher level interaction. For example, users can directly manage an appropriately configured FPGA card as an IPv4 router.

4. We have developed a fast and efficient resource reservation system that accommodates heterogeneous collections of resources and arbitrary virtual network configurations.

The rest of the paper is organized as follows. Section 2 gives some background about the Open Network Laboratory prior to this work. Section 3 then highlights the design considerations for an extensible testbed with heterogeneous resources. The new ONL software infrastructure is described in Section 4, including important abstractions and implementation details. An overview of how resources are scheduled in ONL is given in Section 5. Section 6 describes the resources currently in the testbed, and Section 7 shows three example ONL sessions. Section 8 contains related work. Future work is discussed in Section 9. Section 10 concludes the paper.

2 Open Network Laboratory

The Open Network Laboratory has been operating for a few years now [13]. It has been used primarily as an

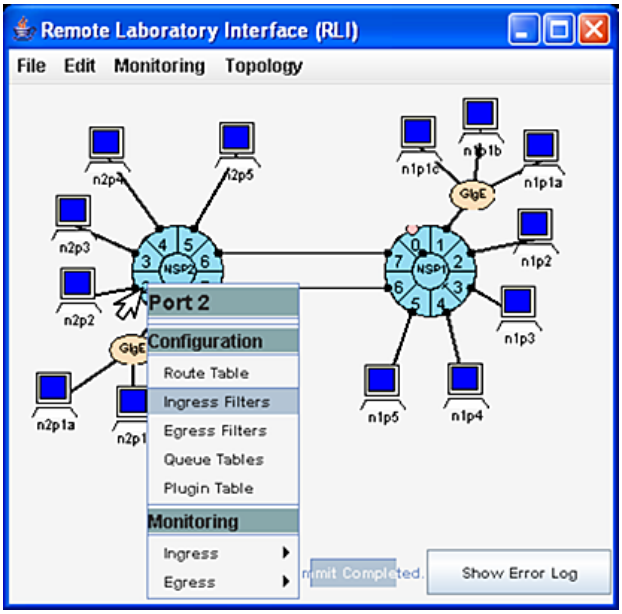

Figure 1: Original ONL RLI.

education tool in graduate and undergraduate networking courses [41][40]. Originally, ONL was designed to give users access to four locally built extensible hardware routers [10] and a few dozen PCs for traffic generation. One of the most important elements of the original design was the Remote Laboratory Interface (RLI), which is a GUI for interacting with the testbed.

The RLI is used to build arbitrary virtual network topologies and then configure and monitor nodes in that topology. All of the initial configuration can be done without ever actually being connected to the testbed. Of course, with only four routers, resource contention quickly becomes an unavoidable issue. One solution would have been to virtualize the routers. However, that is not a practical alternative for many realistic router platforms. It is also not desirable in a testbed that seeks to support high performance networking experimentation.

Instead, a simple reservation scheme was devised whereby users make a reservation in advance for the routers and PCs they need. For example, a user might request a resource reservation for two routers and twelve PCs from 3pm to 5pm. Assuming that there are enough resources available at that time, the user is then guaranteed to have access to those resources for the duration of their reservation. When the reservation time arrives, the RLI is used to connect to the testbed. The infrastructure software maps the user's virtual network onto the physical substrate of the testbed and then gives the user access to the PCs and routers in their virtual network.

The routers and PCs are all indirectly connected via commodity Ethernet switches, and VLANs are used

to isolate each virtual link in all active virtual networks. These switches will be referred to as "backbone" switches for the rest of the paper. The backbone switches are invisible to users, and the testbed software infrastructure manages them automatically. Every PC has an extra network interface that is connected to a separate control network in the testbed. This provides out-of-band access for users to log in to the PCs to start traffic sources and sinks.

Once running a session, the RLI is used for ongoing configuration and monitoring of the user's virtual network. Simple menus give users access to routing tables, packet filters, queueing subsystems, etc. A screenshot of the original RLI is shown in Figure 1.

More recently, an opportunity arose to include a new type of programmable router based on a network processor platform [39]. At the time, there was no way to easily support two different types of routers due to limitations in both the RLI and the reservation system. A temporary solution was to replicate the testbed infrastructure and run two testbeds in parallel, i.e., there were two completely independent instances of ONL running that each contained only one type of router and PCs.

Clearly, this situation was undesirable, and it served as a driving force for the redesign of the testbed software.

3 Design Considerations

Before discussing specific design choices, it will be useful to clearly define certain terms. First, a *session* is one instance of a virtual network running on the testbed. Each session begins when the user requests resources in the testbed and ends when those resources are released. Second, a *resource type*, or simply *type*, describes the properties of a particular kind of device that is available in the testbed. For example, PCs and FPGAs would be different types. Finally, a *node* is one user-allocatable instance of a type.

Most testbeds share a common set of basic design objectives. This includes the ability to support multiple concurrent sessions in the testbed and to provide some level of isolation between those sessions. Those requirements apply here as well. Of course, the first new objective is to support heterogeneous resource types. Ideally this means supporting any and all types of networking technology. The initial design presented here is focused on wired devices, although extending the model to wireless resources is possible.

> **Objective:** The testbed must support a wide range of diverse networking devices.

Naturally, many of the types that will likely be included in such a testbed are going to reprogrammable, such as FPGAs, network processors, and PCs. Users may want nodes of these types to logically function as many different network devices (e.g., routers, switches, firewalls, traffic generators, etc). It is therefore necessary for resource types to support many sets of functionality. More importantly, users may extend an existing type on their own. For example, a user might add support for new protocols to an existing router, or add traffic generation capabilities to an FPGA device.

> **Objective:** Resource types must be extensible by users.

Having many configuration options and possibilities certainly increases the flexibility of the testbed. However, it is often the case that systems with higher configurability and flexibility also have a higher barrier to entry for novice users. Some of this trade-off is intrinsic in any complex system, but a well designed testbed could maintain high levels of flexibility while still providing user interfaces that are simple and intuitive enough for inexperienced users.

> **Objective:** Session configuration must be simple enough for novices.

It is also important for testbed control and management to remain as simple as possible. This is particularly important when it comes to adding new resource types to the testbed. The testbed infrastructure should be designed to minimize the effort of adding new types, and ideally it should be handled without the need to modify any of the testbed software. This burden is entirely on the testbed managers, not the users, but reducing the time spent on ongoing management tasks frees up testbed staff to improve the testbed in more substantive ways. In other words, the testbed itself should also be easily extensible.

> **Objective:** The testbed infrastructure must be extensible to support simple management.

All testbeds provide some level of isolation between concurrent sessions. Depending on the testbed, this ranges from PC-only solutions relying on virtual machine isolation (e.g, PlanetLab [29]) to complete isolation of testbed nodes and traffic in a user virtual network (e.g., Emulab [37]). Given the previous design objectives, it is likely that the testbed will contain high performance devices. Again, this might include network processor platforms or FPGAs, but could also include other commercial or research hardware platforms. One of the benefits to having high performance types in the testbed is to allow users to conduct performance evaluations. To support this class of experimentation, it is necessary for the testbed infrastructure to provide strong isolation guarantees for every virtual network.

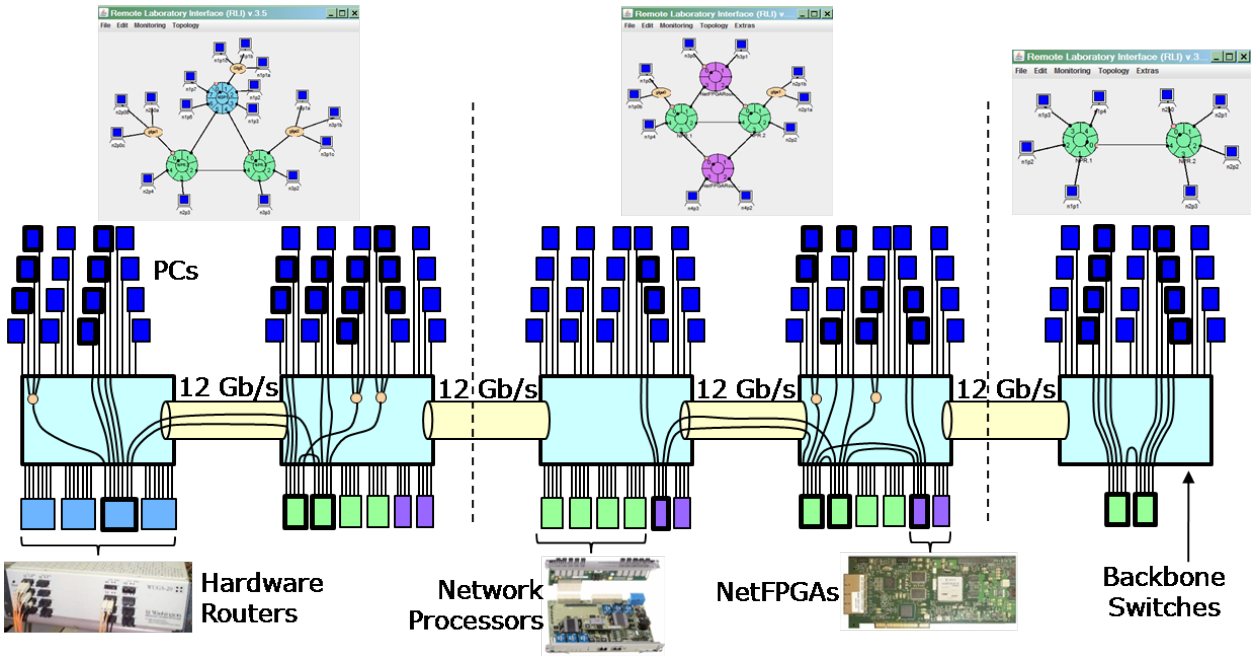

Figure 2: ONL testbed network with session mappings for three user virtual networks.

Objective: Concurrent sessions must be strongly isolated from one another.

A related consideration is how resources are shared among users. One approach is to virtualize all the nodes in the testbed and allow multiple users to share the same node simultaneously. This is clearly not a viable option here, as many networking devices do not support virtualization. PCs are one of the few types that are easily virtualized, although PC virtualization typically leads to poor performance isolation. Another approach is to allocate nodes solely to one user at a time, which fits more closely with the previous objective. Of course, the testbed may only have a small number of nodes of some resource types, so a reservation-based scheduling system should be used to allow users to share nodes easily over time. Scheduling sessions in this context is not a trivial problem, but it does provide an effective means to share testbed resources while still meeting the other design objectives. Note that virtualization is still used in the testbed substrate network in order to provide traffic and link isolation (e.g., by using VLANs on backbone switches to separate traffic on different virtual links). The testbed as a whole is, of course, a virtualized platform, whether or not individual nodes in the testbed are virtualized.

Decision: Nodes are assigned to one user at a time and are shared according to the testbed resource scheduling policy.

4 Testbed Infrastructure

The new ONL testbed infrastructure is now presented. A brief overview of the current ONL testbed network is given first along with an example that shows how the testbed is shared among multiple sessions. A description of the major software components is next, followed by the core abstractions that are used in the software to meet the design objectives from Section 3. Finally, some implementation details are discussed for key pieces of the infrastructure.

4.1 Testbed Network

There are currently 5 backbone switches in ONL. Each switch has 48x1 Gb/s ports and supports a vendor-specific 12 Gb/s stacking connection [24]. These stacking connections are used to connect the 5 switches into a simple linear backbone, where all testbed nodes connect to one of the switches. The actual layout is shown along the middle of Figure 2. The details of the resource types are given in Section 6, but as the figure shows, there are 4 hardware routers, 14 network processor systems, 6 NetFPGAs [23], and around 100 PCs. VLANs are automatically configured for every virtual link in every session. ONL also allows users to add virtual switches to their experimental topologies in the same manner as virtual links, by adding all endpoints connected to the same virtual switch to a unique VLAN.

Figure 2 also shows how three concurrent sessions

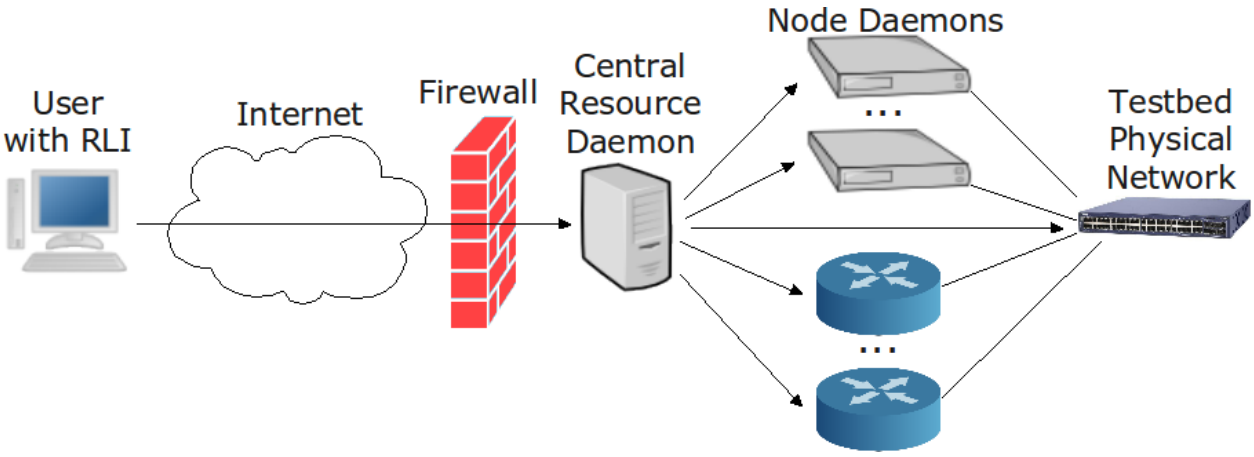

Figure 3: ONL infrastructure overview.

might be scheduled and mapped to the testbed network. The top of the figure shows RLI screenshots from each session. The node and link mappings for each session are included in the testbed network diagram below the screenshot. The colors of the different types in the RLI correspond to the colors of the blocks in the diagram. Circles in the backbone switches correspond to virtual switches in the user session. For example, the left topology has one hardware router, two network processor systems, three virtual switches, and seventeen PCs. Note that the three sessions in this example are mapped to distinct subsets of the backbone switches only for clarity. In practice, different virtual networks can and often do overlap in complicated ways when mapped onto the testbed network.

4.2 Software

A high level view of the ONL infrastructure is shown in Figure 3. There are three core software components: the Remote Laboratory Interface, the Central Resource Daemon, and the Node Daemons. These is also a testbed firewall whose primary function is to keep experimental traffic from leaving the confines of the testbed network.

The Remote Laboratory Interface (RLI) is the user interface to the testbed. It runs on the user's computer and is used to build experimental topologies, configure nodes, and monitor the user's virtual network. Clearly, the RLI must present a consistent and intuitive view of the diverse resources in the testbed.

The RLI communicates over the Internet with the Central Resource Daemon (CRD) on behalf of the user. The CRD has four main functions. First, it manages all the nodes in the testbed, ensuring that nodes are in a proper state after a session ends and granting users access to nodes in their virtual topologies. Second, the CRD manages all session state for the testbed. This includes invoking the testbed scheduler to verify that a user has permission to start a session when requested. Third, the testbed backbone switches are configured when sessions are initiated or removed. Finally, the CRD relays control messages from the RLI to other nodes in the testbed, rather than having the RLI contact nodes directly. This forces all user messages to pass through the CRD for authentication and verification.

The Node Daemons (NDs) accept control messages from the CRD and the RLI. There is one ND for each resource type, and every node of that type runs that ND. The ND understands how to interact with the node and makes any necessary changes based on the control messages received.

4.3 Abstractions

The most important abstractions used in the ONL software are the *type* abstraction and the *specialization* abstraction.

The type abstraction represents the physical device, e.g., a PC, network processor card, or NetFPGA. Each type is accompanied by a type description. Although this description could encompass many things about the hardware, there is no need for the core of the description to be complex. Indeed, the description of each type only needs to contain a unique identifier and a list of network interfaces. That is enough to accurately represent a node in a virtual network topology in the RLI and in the CRD. Figure 4 shows the XML type description for the IXP 2800 network processor platforms that are currently in ONL.

The specialization abstraction represents one set of functionality supported by a particular type. For example, an IPv4 router built on top of a network processor

```
<type id="ixp2800">
 <networkInterfaces>
  <port number="0" linkType="GigE"/>
  <port number="1" linkType="GigE"/>
  <port number="2" linkType="GigE"/>
  <port number="3" linkType="GigE"/>
  <port number="4" linkType="GigE"/>
 </networkInterfaces>
</type>
```

Figure 4: Type description for an IXP 2800 platform with five 1 Gb/s network interfaces.

card, or a traffic generator built on top of a NetFPGA. Each specialization of a type has a corresponding interface description that is used by the RLI to build the user interface for that specialization. GUI menus, tables, and monitoring displays are automatically added to the RLI based on this description so that new specializations can be added simply by providing the interface description. The RLI software does not need to be modified. Of course, the description specification needs to be fixed so that the RLI can parse it to produce the desired configuration options, but general enough to support most common user interactions with diverse networking technologies.

There are two basic ways for users to interact with a node. They can send configuration updates to the node, and they can monitor the node. Configuration updates can include anything that changes the operational state of the node. For example, a router might accept configuration requests to modify the routing state and to configure queues. In general, networking devices keep most of their operational state in tables. The interface description thus includes generic table elements that can be tailored for each specialization. Every table and general configuration command is either associated with the global node state or per-port state. For example, a router may have a separate routing table at each input port, or a global routing table shared by all input ports. Monitoring requests are used to get real-time data from the device. Continuing the router example, it may support monitoring the packet rates at each network interface as well as monitoring current queue lengths. Monitoring commands are also tied either to a particular port or to the node globally.

These two classes of interaction provide a generic framework for specifying the interface associated with each specialization of a device. Configuration updates consist of a message type and a series of parameters to accompany the request. If the update is associated with per-port state on the node, the port number is also sent with the request. The RLI automatically gets the appropriate information from the user to fill in the update re-

```
<specialization id="NPR" type="ixp2800">
 <configuration>
  <portTable name="routes">
   <param name="address" type="str"/>
   <param name="outputPort" type="int"/>
   <param name="statsIndex" type="int"/>
  </portTable>
  <portTable name="flowFilters">
   <param name="dstAddress" type="str"/>
   <param name="srcAddress" type="str"/>
   <param name="protocol" type="int"/>
   <param name="dstPort" type="int"/>
   <param name="srcPort" type="int"/>
   <param name="queue" type="int"/>
   <param name="outputPort" type="int"/>
   <param name="drop" type="bool"/>
   <param name="statsIndex" type="int"/>
   ...
  </portTable>
  <globalTable name="plugins">
   <param name="MEcore" type="int"/>
   <param name="objectFile" type="str"/>
  </globalTable>
  <portCommand name="setQueueParams">
   <param name="queue" type="int"/>
   <param name="threshold" type="int"/>
   <param name="quantum" type="int"/>
  </portCommand>
  <portCommand name="setPortRate">
   <param name="bitRate" type="int"/>
  </portCommand>
  ...
 </configuration>
 <monitor>
  <portMonitor name="rxPkt"/>
  <portMonitor name="txPkt"/>
  <portMonitor name="rxByte"/>
  <portMonitor name="txByte"/>
  <portMonitor name="queueLength">
   <param name="queue" type="int"/>
  </portMonitor>
  <globalMonitor name="flowPkt">
   <param name="statsIndex" type="int"/>
   <param name="preQueue" type="bool"/>
  </globalMonitor>
  <globalMonitor name="flowByte">
   <param name="statsIndex" type="int"/>
   <param name="preQueue" type="bool"/>
  </globalMonitor>
  <globalMonitor name="errorCnt">
   <param name="errorNum" type="int"/>
  </globalMonitor>
  ...
 </monitor>
</specialization>
```

Figure 5: Specialization description for an IPv4 router on an IXP 2800 platform.

quest and sends the update to the ND for the node. In the case of a table, the RLI generates add, remove, and update entry messages for the user as they modify the table in the RLI. Responses are often simple success or failure notifications, but could include other information to be presented to the user. Monitoring requests are represented the same way, but the responses consist of a timestamp and the requested data. Monitoring requests can also be periodic, e.g., to monitor a value in the node once a second. These values are displayed on real-time charts that allow the user to track soft state in their virtual network. Figure 5 shows part of the XML specialization interface for the Network Processor-based Router (NPR), described in Section 6, which is an IPv4 router built on the IXP 2800 type shown in Figure 4.

Each specialization is tied to one type so that the appropriate nodes can be assigned to the user, but each type may have many specializations. This separation allows the testbed to support node extensibility directly by allowing reprogrammable devices to export different configuration interfaces based on their current functionality. It also provides a means to support multiple user interfaces for a type that are targeted at different levels of expertise. For example, a specialization for novices might only support the most basic interaction with a type, while a specialization for experts would expose every possible control knob and option.

4.4 Implementation Details

The RLI parses all available type and specialization descriptions to build menus for adding each type and each specialization to a user's virtual network. Note that users can add types without any specialization if they do not need a higher level interface and plan to configure every node manually. Once added to a topology, each specialized node can be configured according to its specialization description via menus and dialogs that are accessed by clicking on the node.

By design, the RLI is not part of the trusted code base for the testbed because it is run on remote PCs under the control of users. The CRD is therefore responsible for all security checking, user authentication, and messaging protocol verification. In principle, users could run any program that conforms to the CRD interface, but, practically speaking, the RLI is a very complex piece of software that would not be easy to replicate.

Once the user is ready to start a session, the RLI sends the user's virtual topology to the CRD to instantiate the session. The CRD only uses the type description, not the specialization description, for each node. As mentioned above, the CRD has four primary functions. To keep the implementation clean, some of those functions are broken out into separate software processes as shown in Figure 6.

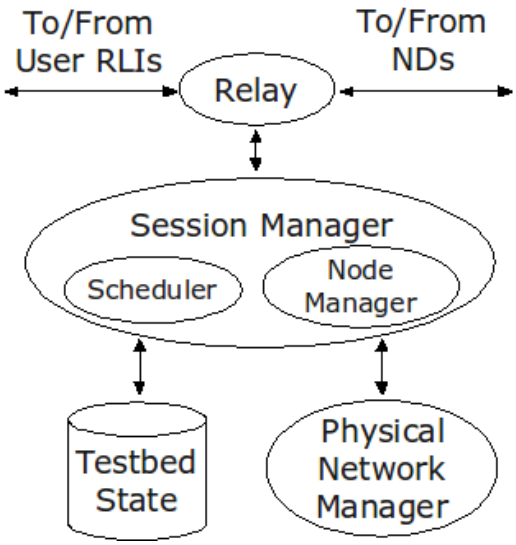

Figure 6: Software Components of the Central Resource Daemon.

The Relay accepts messages from all user RLIs and routes them either to the Node Daemons or to the Session Manager as needed. This serves as the gateway to the testbed for the RLIs. It also performs simple security and isolation checks to ensure that users attempting to send messages to a particular node actually have that node assigned to them in a current session.

The Session Manager handles the bulk of the work in the CRD and maintains all state related to the testbed. This state includes current session information, reservations, and descriptions of the nodes and physical network for the testbed. Adding new types or new nodes to the testbed is accomplished by adding new entries into tables in the testbed database and requires no modifications to the Session Manager or any other ONL software components. The scheduler subsystem is invoked to add and remove reservations. It is also called to verify that the user has a valid reservation when a new session request arrives. The Session Manager then initiates the new session by performing any initial configuration for all the nodes and links in the user's virtual network.

Configuring the underlying physical testbed network is handled by the Physical Network Manager (PNM). The PNM supports two basic functions: adding/removing a virtual link and adding/removing a virtual switch. The actual mapping from virtual links and switches in a user virtual network to the physical links in the testbed network is handled by the scheduler in the Session Manager. The PNM simply adds or removes the mappings as requested. In the case of ONL, this means contacting the backbone switches (via SNMP) to add or

remove VLANs. This functionality is not complex, but it is tied fairly closely to the types of backbone switches used in ONL. Separating it from the Session Manager allows this infrastructure to be more easily adapted to changes in the physical network as only the PNM would need modification.

The Node Manager subsystem of the Session Manager contacts the appropriate Node Daemons for any initial configuration. The Node Daemons are split into two parts, corresponding to the type and specialization of the node. The *Type* Node Daemon (TND) is responsible for initializing the node at the beginning of a session and for bringing the node back to a known state at the end of a session. It also starts the appropriate *Specialization* Node Daemon (SND) at the end of its initialization. The SND interacts with the RLI to respond to configuration and monitoring requests defined in the specialization description.

This split enforces a clean separation between the specializations for every type. More importantly, it allows users to add their own specializations for a type by providing the specialization interface description and the SND associated with it. For example, a user might add a firewall specialization to a PC by writing a simple SND and interface description for adding and removing firewall filtering rules. The SND is a standard socket program and so users are free to use any programming language to build one, but at the moment we only provide C++ templates that handle the details of the messaging protocol. None of the testbed software infrastructure needs to be modified to handle the new specialization.

5 Resource Scheduling

ONL utilizes a resource scheduler to map user virtual networks onto the physical resources available in the testbed. As was discussed in Section 3, each physical node is given to at most one user at any time. The scheduling policy is thus used to determine how nodes should be shared when the demand is higher than the capacity for any particular resource type. The full description of the problem and the solution used in ONL is beyond the scope of this paper, but a brief overview is given here. See [38] for details.

There are two basic scheduling approaches: resources are either given on-demand or reserved in advance. In the former case, the scheduler looks strictly at the physical resources that are not in use by any other current session, as in standard admission control. The latter case is somewhat more complicated. The scheduler must keep a time line of *reservations* that determines which resources are available at any given time. In addition to the virtual network topology, user requests include a period of time when that virtual network should be active. When a

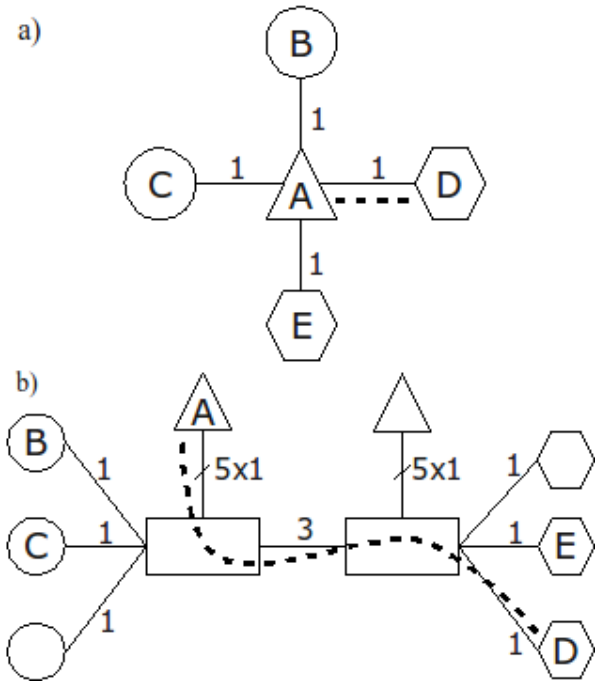

Figure 7: a) Example user virtual network, and b) example testbed physical network.

new request is made, all previously accepted reservations with overlapping times are considered. Any reservations whose start time has not come (i.e., are not yet active) could potentially be remapped to a new set of physical resources, if necessary, to "make room" for the new reservation. Maintaining a schedule of network mappings is a generalization of pure on-demand admission control. Given the nature of ONL and the design objectives, the more general reservation-based approach is used here.

Scheduling virtual networks in testbeds has been studied previously [31][32][3]. It is a variant of the general network embedding problem, long known to be NP-hard. As such, ONL uses heuristic schedulers.

The scheduler takes a virtual network request and attempts to find a mapping from the virtual network onto the available physical resources. If a mapping is found, the physical resources in that mapping are added to a reservation for the user. The scheduler ensures that each physical node is mapped to no more than one virtual node across all virtual networks. It also ensures that there is enough capacity in the underlying testbed network to support every virtual link. Of course, the scheduler has to consider every previously accepted reservation when finding a new mapping. Figure 7 shows a simple example mapping. In the figure, different shapes represent different types, with rectangles representing the backbone switches. The edge labels are edge capacities. Node labels show an example mapping from nodes in the user

network to nodes in the testbed network. The dashed lines show one mapping from an edge in the user network to the corresponding path in the testbed network.

The current ONL scheduler utilizes Mixed Integer Programs (MIPs) to find mappings that maximize the potential of accepting future requests. When a new request comes in, the scheduler finds all existing reservations that overlap with the requested period of time. All resources (nodes and link capacities) used in those reservations are removed from the set of available resources. The remaining testbed resources and the user's virtual network are encoded in a MIP that computes a mapping, if possible, between the two. The objective function of the MIP is set to minimize the use of link capacity between backbone switches in the testbed network.

Our experience so far with this scheduler, and verified in simulation results given in [38], has been that response times to reservation requests are nearly always under 50 ms. The scheduler also accepts a large percentage of reasonable requests, even under medium to high loads. It is not yet clear how well this approach scales as the size of the testbed network increases. Our initial results indicate that the testbed could increase to several times its current size while maintaining sub-second response times, but larger and more complex testbeds could require faster (and more approximate) mapping algorithms.

6 Current Resources

The Network Services Platform (NSP) [10] is the custom-built IP router from the original version of ONL. It is designed to operate in a similar fashion to larger, scalable router platforms. Each of the eight 1 Gb/s ports houses an FPGA and a general-purpose processor. The ports are connected via a 2 Gb/s cell switch. All of the standard packet processing tasks are handled by the FPGA. Users write *plugins* for the general-purpose processor to dynamically extend the router's functionality. ONL currently contains four NSPs.

The second router added to ONL was the Network Processor-based Router [39] (NPR), which is an IPv4 router built on Intel IXP 2800s [1]. Each IXP 2800 has five 1 Gb/s ports. The NPR is a specialization of the IXP 2800 type. The router software is itself extensible through user-written plugins as in the NSP. Here, plugins are run on five of the sixteen MicroEngine cores on the IXP. The router supports complex flow filters that are used to direct specific packet flows in the router to a destination other than the matching route. This includes sending packet flows to plugins for processing. Now that ONL directly supports many specializations of the same type, we are building extensible Ethernet switch and OpenFlow [21] switch specializations for the IXP 2800 boards. There are currently 14 of these IXPs in ONL.

The newest addition to ONL is the NetFPGA [23]. The NetFPGA is a relatively new FPGA-based device that has seen substantial use in both research and educational contexts. There are a number of projects readily available including IPv4 routers, Ethernet switches, OpenFlow switches [22], and packet generators [12]. Some of these projects already have specializations available in ONL, and more should be available soon. There are currently six NetFPGAs in ONL.

Linux PCs are used as standard end points in ONL. Each PC has two network interfaces. The first is a 1 Gb/s data interface used in experimental topologies. The second is an out-of-band control and management interface. Users can select from a small set of pre-built Linux kernel images to boot, or supply their own if needed. Approximately 100 PCs are currently available in ONL.

7 Example Sessions

ONL can be used to conduct networking experiments over a wide range of areas. Three examples follow to illustrate some of these possibilities.

7.1 TCP Dynamics

The first example is a simple session to study TCP dynamics over a shared bottleneck link. A screenshot is shown in Figure 8. The topology configuration window is on the left, and two real-time charts are on the right. The 8 port device at the bottom right of the central square is an NSP. The 4 port device at the top left of the square is a NetFPGA running as an IPv4 router. The two 5 port devices at the other corners are NPRs. The small ovals are virtual switches, and the other symbols represent PCs.

Two TCP flows are started at the same time, one from a PC in the top left of the topology to a PC in the bottom right, and the other from a PC in the bottom left to a second PC in the bottom right. Network routes are configured statically so that both flows share the bottom link in the central square as a bottleneck link. The link capacity is set to 100 Mb/s. Initially, both flows share a 100 KB queue at the bottleneck. Half way through the flows, the router is reconfigured to map one of the flows to a different queue.

The results are shown on the right of Figure 8. The top chart shows the bandwidth of each flow, and the bottom chart shows the queue lengths of the two queues at the bottleneck. When the flows share a queue, they do not converge to their fair share of 50 Mb/s each. This follows because each flow is attempting to use packet drops as feedback to adjust its sending rate, but the shared queue leads to uneven drops for each flow. When one

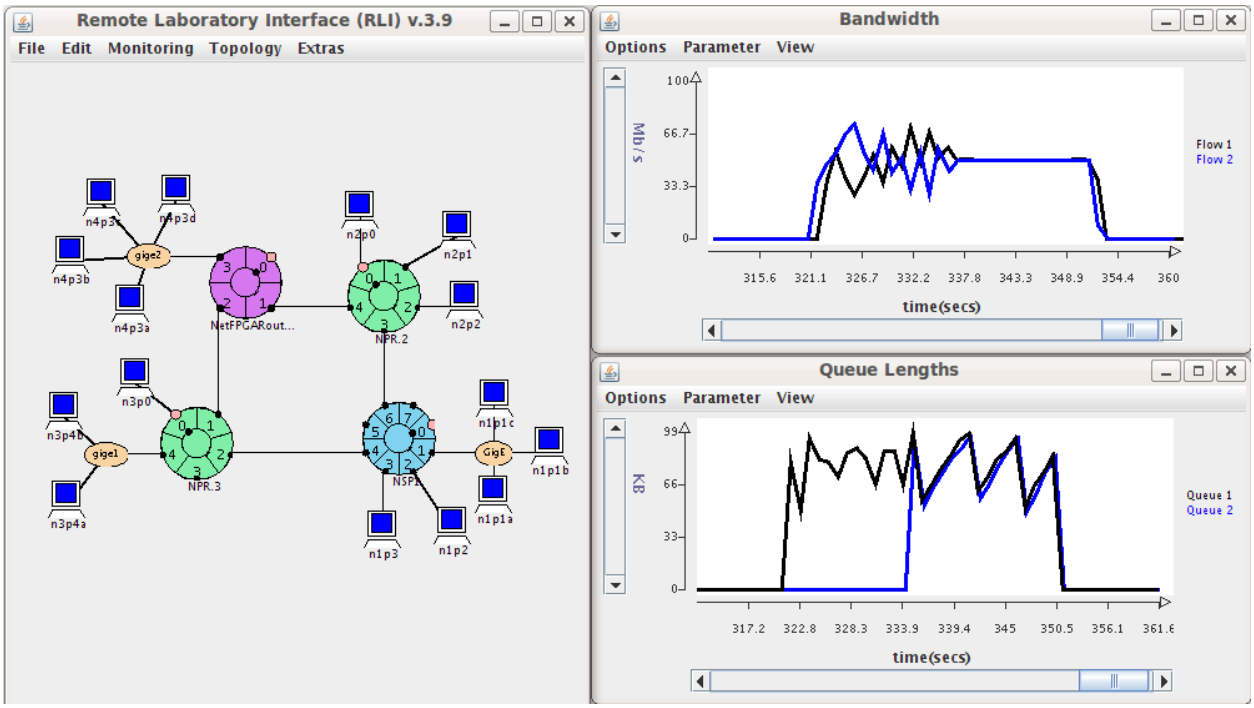

Figure 8: ONL session studying TCP dynamics.

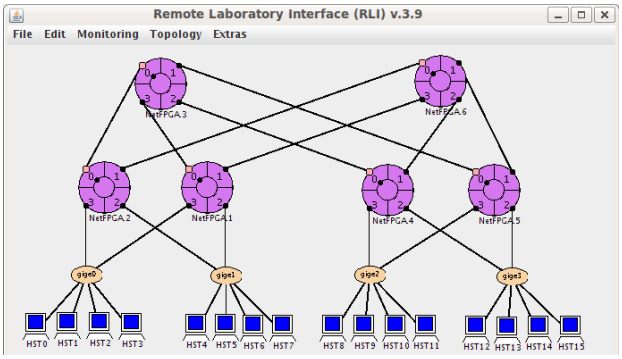

Figure 9: ONL topology for a small-scale data center.

of the flows is remapped so that each flow is in a separate queue, the standard TCP queueing behavior is seen. Weighted Deficit Round Robin scheduling is used at the bottleneck, and each queue has the same quantum. As a result, the two flows quickly converge to their expected fair share of the link.

7.2 Data Center Networking

The second example uses ONL to replicate one segment of a data center network. A screenshot of the ONL topology is shown in Figure 9. This session uses six NetFPGAs as data center switches. In this case, they are running as OpenFlow switches, and the OpenFlow controller, NOX [18], is running on one of the PCs at the bottom of the hierarchy. This network is organized similarly to a fat tree (e.g., as in PortLand [25]), but with twice as many PCs connected to each edge switch as in a standard fat tree.

ONL allows users to build data center networks like this one quickly and easily. The result is that it is much faster to study a broad range of possibilities than it would be without a testbed infrastructure. In this example, ONL is being used to study slight modifications to fat tree topologies, but it could also be used to explore completely different data center topologies without any extra overhead.

7.3 Overlay Networking

The final example session is an overlay network scenario, where the overlays are built on top of a virtual network in ONL. This particular overlay network is based on the Forest [19] architecture that is designed to support highly interactive virtual worlds. Forest networks are built around provisioned tree-structured communication channels called comtrees. Comtrees support both unicast and multicast packet delivery. In a typical Forest application, such as a First Person Shooter game, each comtree is associated with a separate game session or world instance. Within that comtree, then, each multicast address is associated with the state of some object

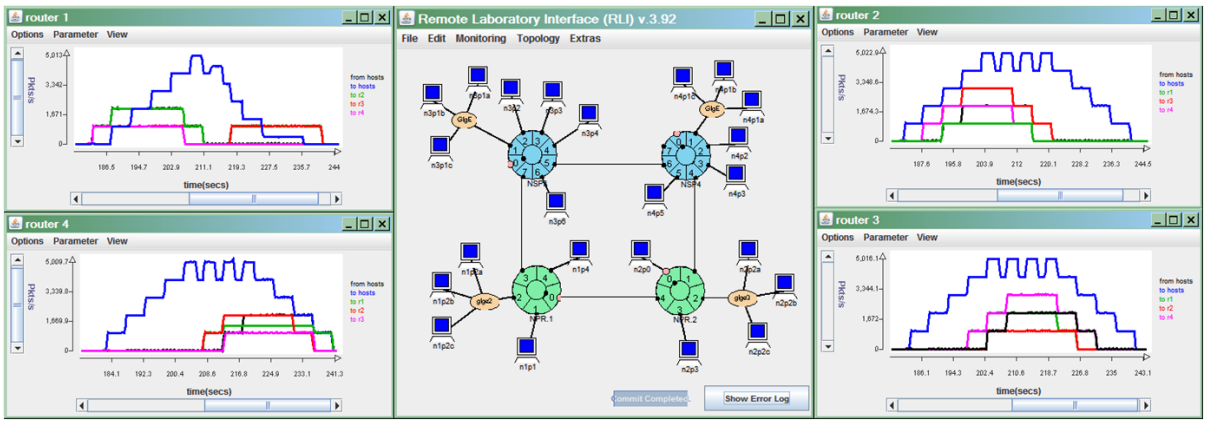

Figure 10: ONL session for PC-based overlay networking.

in the game, such as a player's avatar. The multicast mechanism in Forest is extremely light-weight in order to support the high levels of churn in state subscription changes within such virtual worlds.

Figure 10 shows a screenshot of a Forest overlay operating within ONL. The four PCs in the middle of the topology implement Forest routers and are connected to one another by a full mesh of overlay links (implemented by the IP routers in the ONL context). The PCs around the periphery represent end systems in a Forest network and each of them is connected by an overlay link to one of the four Forest routers. In this sample session, the end systems are simply sending artificial traffic which is passing over pre-configured multicasts spanning several different comtrees. The four real-time charts in the figure show traffic rates (in packets/s) at each of the four Forest routers.

8 Related Work

Of all the current networking testbeds, Emulab [37] is the most closely related to ONL. Emulab is one of the most widely used testbeds in operation today, for both networking research and education. The Emulab testbed provides users with access to a large number of PCs. Sessions are strongly isolated from one another, as in ONL, by using a set of Ethernet switches that indirectly connect all the PCs in the testbed. VLANs are then configured to isolate traffic along user virtual links. Emulab PCs also have at least two network interfaces, one for out-of-band control access and the others for use in experimental topologies. Most of the PCs have two to four interfaces which allows them to more readily emulate other devices like routers or switches. Unfortunately, Emulab is limited to PCs, and most configuration of nodes is done at a low level by logging in to the nodes individually to run system utilities.

The Emulab software has been made available so that other groups can use it to run their own testbeds. A number of these Emulab-powered testbeds are currently operating, although most of them are not open to the public. One exception is the DETER testbed [8] 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 sessions. The Wisconsin Advanced Internet Laboratory (WAIL) [36] is another testbed that utilizes the Emulab software base. WAIL is unique among the Emulab testbeds in that they have extended the testbed software to support commercial routers as fundamental resources available to researchers. The documentation on the website indicates that users do not have write access to router configurations by default, and, unfortunately, there is not yet any detailed documentation that discusses how they have modified the Emulab model to support these heterogeneous resources.

Another widely used network testbed is PlanetLab [29], which is an Internet overlay testbed that has been in operation for many years. At its core, PlanetLab simply consists of a large number of PCs with Internet connections scattered around the globe. At the time of this writing there are over 1000 PlanetLab nodes at over 475 sites [30]. Each PlanetLab node can support multiple concurrent sessions by instantiating one virtual machine on the node for each active session. The PlanetLab control software takes user requests to run a session on a set of nodes and contacts the individual nodes to add the virtual machines for the user. Researchers use PlanetLab to debug and refine their new services and applications in the Internet context. Unfortunately, PlanetLab's success has resulted in large numbers of active sessions (particularly before major conference deadlines) which results in

extremely poor performance for any individual session. Each active session is competing for both processor cycles and the limited network bandwidth available on the node, and there is no admission control or limit on the number of sessions on each node.

There have been a few efforts that aim to enhance PlanetLab through the design of new nodes while still operating in the PlanetLab context. VINI [7] is at the vanguard of these efforts. VINI nodes are currently deployed at sites in the Internet2, National LambdaRail, and CESNET backbones [35], and have dedicated bandwidth between them. The VINI infrastructure gives researchers the ability to deploy protocols and services in a realistic environment in terms of network conditions, routing, and traffic, as well as some stronger guarantees on node and traffic isolation between sessions. The Supercharged PlanetLab Platform (SPP) [34] is another project which is working to provide new resource types for PlanetLab. SPP nodes consist of a mix of general-purpose and network processors. Users can run existing PlanetLab software unmodified on the general-purpose systems and then push down core networking code to the network processors to achieve much higher and more consistent performance. A small number of SPP nodes are currently being deployed to the Internet2 backbone as part of the GENI project [15].

SatelliteLab [14] is another Internet overlay testbed. It does provide some support for heterogeneous edge devices. The testbed contains desktop PCs, laptop PCs, and handheld devices that are connected to the Internet over a range of link types, including broadband, ISDN, Wi-Fi, and cellular connections. Users can then test their applications across widely varied network conditions. Of course, it would be difficult to have user-written code running on devices like cell phones, so they break their architecture into two tiers. Standard PlanetLab nodes form the backbone of the testbed where users run their applications. The edge devices only forward traffic with pre-built code from the testbed operators. As a result, SatelliteLab does provide users with network heterogeneity that is lacking in other testbeds, but does not actually allow users to experiment with code on anything other than PlanetLab PCs.

There are a few other areas of networking research that are related to testbeds. Cloud computing has recently become a popular topic of discussion both in the public and among researchers. The actual definition of cloud computing has been much debated [6], but cloud computing infrastructures share many properties with testbed infrastructures. For example, Eucalyptus [26] and Seattle [9] are both research-driven cloud computing systems. The architectures of these systems are similar in many ways to the testbed architecture described in this paper. There are also a number of commercial cloud computing solutions such as Amazon's EC2 [2] and Google's AppEngine [16]. Few details have been published about the design internals of these commercial clouds. However, it is interesting to see the differences in how the cloud is configured by users. For example, EC2 gives users access as close to the bare metal as possible. AppEngine is on the other end of the spectrum, providing a very high level interface where the user has little control but can also build and deploy applications much more quickly.

Finally, there is overlap between testbed management and general operational network management. In each case, there are potentially large numbers of distributed resources that have to be managed by the single provider. The interesting research questions for the two areas may differ somewhat, but the mechanisms used to address network control and management plane issues are similar. For example, testbeds typically support end-user-driven experiments that utilize some set of the testbed resources, while an operational network is configured by the network provider in response to a set of high level policy decisions. Both cases, however, rely on easy deployment of changes to the underlying networks. This, in turn, requires a software infrastructure that can present all the relevant information to the user/operator and then enact changes across the entire network. Most network management solutions are tied to the actual network equipment such as Cisco's IOS [11] or HP's OpenView [20]. There are other commercial solutions like OpenNMS [28] which are less hardware-dependent but generally do not scale well. Some research has also been done to try to simplify the entire management plane by redesigning the control protocols from the ground up, e.g., as in the 4D architecture [17].

9 Future Work

All of the types currently in ONL are built on 1 Gb/s Ethernet technology, but 10 Gb/s Ethernet is becoming more and more commonplace. As such, we are in the early phases of a 10 Gb/s Ethernet expansion for the testbed. The existing ONL infrastructure already allows us to add new types with a minimum of effort, so we plan to incorporate multicore PCs with 10 Gb/s NICs, newer versions of the NetFPGA, and next generation network processors cards. We also will expand the backbone Ethernet network to include 10 Gb/s switches (e.g., [5]). This will require modifying the Physical Network Manager to manage the new testbed network.

Using VLANs to isolate every instantiated virtual link is a proven approach. One implication, however, is that users can not configure their own VLANs for use in their experimental networks because they might conflict with the VLANs used by the testbed. Until recently this has not been a serious limitation as the available types in

the testbed led most sessions to be focused on IPv4 and above concerns rather than Ethernet issues. Particularly with the coming 10 Gb/s expansion, it is much more reasonable for users to want to manage their own VLANs, e.g., in data center research. Fortunately, many Ethernet switches now support VLAN stacking. When packets arrive at a switch, a VLAN header can be added even if the packet has an existing VLAN header. This "outer" VLAN header can similarly be removed as packets leave a switch. We plan on utilizing these features to allow users to configure and manage their own VLANs in their virtual networks.

A recent trend in networking research is to build high performance systems out of many distinct components [4][33]. For example, trying to build a 10 Gb/s router out of 10 PCs each with a 1 Gb/s NIC. ONL is particularly well suited for this research already, and we are currently working on mechanisms to enhance user interactions with these types of systems. Users will be able to provide "aggregate" or "cluster" descriptions that are composed of other types connected in some topology. As with specializations now, users will also provide an aggregate daemon that handles messages from the RLI to either relay messages to existing Node Daemons or to configure all the nodes in the aggregate directly. The RLI will only display a single icon to represent the aggregate so that users can configure it as if it were a single entity.

10 Conclusion

We have presented the newest version of the Open Network Laboratory testbed, which is an experimental facility for networking and systems researchers and educators. ONL supports a wide variety of heterogeneous resources from which users build virtual topologies. The Remote Laboratory Interface provides an intuitive user interface that is easy for novice users to learn while still providing complex configuration options for expert users. The testbed infrastructure was designed to support both resource heterogeneity and extensibility from the ground up. Adding new types of resources requires no modification to the existing testbed software. Each resource type can support many different specializations, either to extend the functionality of a type or to provide different interfaces targeted at users with different levels of expertise. Moreover, users can add new specializations with no modification to the testbed infrastructure.

The ideas developed in this work can also be applied to other virtualized infrastructures where many users are sharing the resources of a single provider. They could also be applied to other testbeds. We hope that our experiences with ONL will be used to help build a better GENI [15], where heterogeneous and extensible resources will play a fundamental role.

ONL is available now and is open to all researchers, educators, and students. New users can sign up for accounts at the ONL website [27]. The website also contains tutorials and other documentation.

References

[1] ADILETTA, M., ROSENBLUTH, M., BERNSTEIN, D., WOLRICH, G., AND WILKINSON, H. The next generation of intel ixp network processors. *Intel Technology Journal 6* (2002).

[2] AMAZON. Elastic compute cloud website. `http://aws.amazon.com/ec2/`.

[3] ANDERSEN, D. G. Theoretical approaches to node assignment. 2002.

[4] ARGYRAKI, K., BASET, S., CHUN, B.-G., FALL, K., IANNACCONE, G., KNIES, A., KOHLER, E., MANESH, M., NEDEVSCHI, S., AND RATNASAMY, S. Can software routers scale? In *PRESTO '08: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow* (New York, NY, USA, 2008), ACM, pp. 21–26.

[5] ARISTA. Arista 7100 series switches. `http://www.aristanetworks.com/en/7100Series`.

[6] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A., STOICA, I., AND ZAHARIA, M. Above the clouds: A berkeley view of cloud computing. Tech. rep., University of California, Berkeley, 2009.

[7] BAVIER, A., FEAMSTER, N., HUANG, M., PETERSON, L., AND REXFORD, J. 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* (New York, NY, USA, 2006), ACM, pp. 3–14.

[8] BENZEL, T., BRADEN, R., KIM, D., NEUMAN, C., JOSEPH, A., SKLOWER, K., OSTRENGA, R., AND SCHWAB, S. 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* (Berkeley, CA, USA, 2007), USENIX Association, pp. 1–1.

[9] CAPPOS, J., BESCHASTNIKH, I., KRISHNAMURTHY, A., AND ANDERSON, T. Seattle: a platform for educational cloud computing. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education* (New York, NY, USA, 2009), ACM, pp. 111–115.

[10] CHOI, S., DEHART, J., KANTAWALA, A., KELLER, R., KUHNS, F., LOCKWOOD, J., PAPPU, P., PARWATIKAR, J., RICHARD, W. D., SPITZNAGEL, E., TAYLOR, D., TURNER, J., AND WONG, K. Design of a high performance dynamically extensible router. In *Proceedings of the DARPA Active Networks Conference and Exposition* (May 2002).

[11] CISCO. Ios website. `http://www.cisco.com/en/US/products/sw/iosswrel/products_ios_cisco_ios_software_category_home.html`.

[12] COVINGTON, G. A., GIBB, G., LOCKWOOD, J., AND MCKEOWN, N. A packet generator on the netfpga platform. In *The 17th Annual IEEE Symposium on Field-Programmable Custom Computing Machines* (5–7 April 2009).

[13] DEHART, J., KUHNS, F., PARWATIKAR, J., TURNER, J., WISEMAN, C., AND WONG, K. The open network laboratory. In *SIGCSE '06: Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2006), ACM, pp. 107–111.

[14] DISCHINGER, M., HAEBERLEN, A., BESCHASTNIKH, I., GUMMADI, K. P., AND SAROIU, S. Satellitelab: adding heterogeneity to planetary-scale network testbeds. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication* (New York, NY, USA, 2008), ACM, pp. 315–326.

[15] GENI. Global environment for network innovations website. `http://www.geni.net`.

[16] GOOGLE. Appengine website. `http://code.google.com/appengine/`.

[17] GREENBERG, A., HJALMTYSSON, G., MALTZ, D. A., MYERS, A., REXFORD, J., XIE, G., YAN, H., ZHAN, J., AND ZHANG, H. A clean slate 4d approach to network control and management. *SIGCOMM Comput. Commun. Rev. 35*, 5 (2005), 41–54.

[18] GUDE, N., KOPONEN, T., PETTIT, J., PFAFF, B., CASADO, M., MCKEOWN, N., AND SHENKER, S. Nox: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev. 38*, 3 (2008), 105–110.

[19] HAITJEMA, M., PATNEY, R., TURNER, J., WISEMAN, C., AND DEHART, J. Performance-engineered network overlays for high quality interaction in virtual worlds. Tech. Rep. WUCSE-2009-18, Washington University in St. Louis, June 2009.

[20] HP. Openview website: `http://www.managementsoftware.hp.com`.

[21] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. Openflow: enabling innovation in campus networks. vol. 38, ACM, pp. 69–74.

[22] NAOUS, J., ERICKSON, D., COVINGTON, G. A., APPENZELLER, G., AND MCKEOWN, N. 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* (New York, NY, USA, 2008), ACM, pp. 1–9.

[23] NAOUS, J., GIBB, G., BOLOUKI, S., AND MCKEOWN, N. Netfpga: reusable router architecture for experimental research. In *PRESTO '08: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow* (New York, NY, USA, 2008), ACM, pp. 1–7.

[24] NETGEAR. Gsm7532s website. `http://netgear.com/Products/Switches/FullyManaged10_100_1000Switches/GSM7352S.aspx`.

[25] NIRANJAN MYSORE, R., PAMBORIS, A., FARRINGTON, N., HUANG, N., MIRI, P., RADHAKRISHNAN, S., SUBRAMANYA, V., AND VAHDAT, A. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication* (New York, NY, USA, 2009), ACM, pp. 39–50.

[26] NURMI, D., WOLSKI, R., GRZEGORCZYK, C., OBERTELLI, G., SOMAN, S., YOUSEFF, L., AND ZAGORODNOV, D. The eucalyptus open-source cloud-computing system. In *Proc. 9th IEEE/ACM International Symposium on Cluster Computing and the Grid CCGRID '09* (May 18–21, 2009), pp. 124–131.

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

[28] OPENNMS. Opennms website. `http://www.opennms.org`.

[29] PETERSON, L., ANDERSON, T., CULLER, D., AND ROSCOE, T. A blueprint for introducing disruptive technology into the internet. In *Proceedings of HotNets–I* (Princeton, New Jersey, October 2002).

[30] PLANETLAB. Planetlab website. `http://www.planet-lab.org`.

[31] RICCI, R., ALFELD, C., AND LEPREAU, J. A solver for the network testbed mapping problem. *SIGCOMM Comput. Commun. Rev. 33*, 2 (2003), 65–81.

[32] RICCI, R., OPPENHEIMER, D., LEPREAU, J., AND VAHDAT, A. Lessons from resource allocators for large-scale multiuser testbeds. *SIGOPS Oper. Syst. Rev. 40*, 1 (2006), 25–32.

[33] SHEVADE, U., KOKKU, R., AND VIN, H. M. Run-time system for scalable network services. In *Proc. INFOCOM 2008. The 27th Conference on Computer Communications. IEEE* (Apr. 13–18, 2008), pp. 1813–1821.

[34] TURNER, J. S., CROWLEY, P., DEHART, J., FREESTONE, A., HELLER, B., KUHNS, F., KUMAR, S., LOCKWOOD, J., LU, J., WILSON, M., WISEMAN, C., AND ZAR, D. Supercharging planetlab: a high performance, multi-application, overlay network platform. In *SIGCOMM '07: Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (New York, NY, USA, 2007), ACM, pp. 85–96.

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

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

[37] WHITE, B., LEPREAU, J., STOLLER, L., RICCI, R., GURUPRASAD, S., NEWBOLD, M., HIBLER, M., BARB, C., AND JOGLEKAR, A. An integrated experimental environment for distributed systems and networks. In *OSDI '02: Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (New York, NY, USA, 2002), ACM, pp. 255–270.

[38] WISEMAN, C., AND TURNER, J. The virtual network scheduling problem for heterogeneous network emulation testbeds. Tech. Rep. WUCSE-2009-68, Washington University in Saint Louis, September 2009.

[39] WISEMAN, C., TURNER, J., BECCHI, M., CROWLEY, P., DEHART, J., HAITJEMA, M., JAMES, S., KUHNS, F., LU, J., PARWATIKAR, J., PATNEY, R., WILSON, M., WONG, K., AND ZAR, D. 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* (New York, NY, USA, 2008), ACM, pp. 20–29.

[40] WISEMAN, C., WONG, K., WOLF, T., AND GORINSKY, S. Operational experience with a virtual networking laboratory. In *SIGCSE '08: Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2008), ACM, pp. 427–431.

[41] WONG, K., WOLF, T., GORINSKY, S., AND TURNER, J. Teaching experiences with a virtual network laboratory. In *SIGCSE '07: Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2007), pp. 481–485.